Approximate Lineage for Probabilistic Databases

Christopher Ré
chrisre@cs.washington.edu
Department of Computer Science & Engineering
University of Washington, Seattle
Box 352350

Dan Suciu
suciu@cs.washington.edu
Department of Computer Science & Engineering
University of Washington, Seattle
Box 352350

ABSTRACT

In probabilistic databases, *lineage* is fundamental to both query processing and understanding the data. Current systems s.a. Trio or Mystiq use a *complete approach* in which the lineage for a tuple t is a Boolean formula which represents all derivations of t. In large databases lineage formulas can become huge: in one public database (the Gene Ontology) we often observed 10MB of lineage (provenance) data for a single tuple. In this paper we propose to use *approximate lineage*, which is a much smaller formula keeping track of only the most important derivations, which the system can use to process queries and provide explanations. We discuss in detail two specific kinds of approximate lineage: (1) a conservative approximation called sufficient lineage that records the most important derivations for each tuple, and (2) polynomial lineage, which is more aggressive and can provide higher compression ratios, and which is based on Fourier approximations of Boolean expressions. In this paper we define approximate lineage formally, describe algorithms to compute approximate lineage and prove formally their error bounds, and validate our approach experimentally on a real data set.

1. INTRODUCTION

In probabilistic databases, *lineage* is fundamental to processing probabilistic queries and understanding the data. Many state-of-the-art systems use a *complete approach*, *e.g.* Trio [7] or Mystiq [16, 46], in which the lineage for a tuple t is a Boolean formula which represents all derivations of t. In this paper, we observe that for many applications, it is often unnecessary for the system to painstakingly track every derivation. A consequence of ignoring some derivations is that our system may return an approximate query probability such as 0.701 ± 0.002, instead of the true value of 0.7. An application may be able to tolerate this difference, especially if the approximate answer can be obtained significantly faster. A second issue is that although a complete lineage approach explains all derivations of a tuple, it does not tell us which facts are the most influential in that derivation. In large data sets, a derivation may become extremely large because it aggregates together a large number of individual facts. This makes determining which individual facts are influential an important and non-trivial task.

With these observations in mind, we advocate an alternative to complete lineage called *approximate lineage*. Informally, the spirit of approximate lineage is to compress the data by tracking only the most influential facts in the derivation. This approach allows us to both efficiently answer queries, since the data is much smaller, and also to directly return the most important derivations. We motivate our study of approximate lineage by discussing two application domains: (1) large scientific databases and (2) similarity data. We show that approximate lineage can compress the data by up to two orders of magnitude, *e.g.* 100s of MB to 1MB, while providing high-quality explanations.

Application (1): Large Scientific databases In large scientific databases, lineage is used to integrate data from several sources [12]. These sources are combined by both large consortia, *e.g.* [14], and single research groups. A key challenge faced by scientists is that facts from different sources may not be trusted equally. For example, the Gene Ontology Database (GO) [14] is a large (4GB) freely available database of genes and proteins that is integrated by a consortium of researchers. For scientists, the most important data stored in GO is a set of associations between proteins and their functions. These associations are integrated by GO from many sources, such as PubMed articles [45], raw experimental data, data from SWISS-PROT [9], and automatically inferred matchings. GO tracks the provenance of each association, using what we call *atoms*. An atom is simply a tuple that contains a description of the source of a statement. An example atom is *"Dr. X's PubMed article PMID:12593804"*. Tracking provenance is crucial in GO because much of the data is of relatively low quality: approximately 96% of the more than 19 million atoms stored in GO are automatically inferred. To model these trust issues, our system associates each atom with a probability whose value reflects our trust in that particular annotation. Fig. 1 illustrates such a database.

Example 1.1 A statement derivable from GO is, "Dr. X claimed in PubMed PMID:12593804 that the gene Argonaute2 (AGO2) is involved in cell death"[26]. In our model, one way to view this is that there is a fact, *the gene Argonaute2 is involved in cell death* and there is an atom, *Dr. X made the claim in PubMed PMID:12593804*. If we trust Dr. X, then we assign a high confidence value to this atom. This is reflected in Fig. 1 since the atom, x_1, has a high probability, $\frac{3}{4}$. More complicated annotations can be *derived*, *e.g.* via query processing. An example is the view `V` in Fig. 1, that asks for gene products that share a process with the gene 'Aac11'. The tuple, AGO2 (t_6), that appears in V is derived from the facts that both AGO2 and Aac11 are involved in "cell death" (t_1 and t_4) and "embryonic development" (t_2 and t_5); these tuples use the atoms x_1 (twice), x_2 and x_3 shown in the Annotations table.

A benefit of annotating the data with confidence scores is that

VLDB '08, August 24-30, 2008, Auckland, New Zealand

Process (P)

	Gene Product	Process	λ
(t_1)	AGO2	"Cell Death"	x_1
(t_2)	AGO2	"Embryonic Development"	x_1
(t_3)	AGO2	"Gland development"	x_2
(t_4)	Aac11	"Cell Death"	x_2
(t_5)	Aac11	"Embroynic Development"	x_3

Annotations (Atoms)

Atom	Code	Description	**P**
x_1	TAS	"Dr. X's PubMed PMID:12593804"	$\frac{3}{4}$
x_2	NAS	"Dr. Y's RA Private Communication"	$\frac{1}{4}$
x_3	IEA	"Inferred from Computational Similarity"	$\frac{1}{8}$

Level I DB (Complete Lineage)

$V(y)$:– P(x, y), P('Aac11', y), $x \neq$ 'Aac11'

	Gene Product	λ
(t_6)	AGO2	$(x_1 \wedge x_2) \vee (x_1 \wedge x_3)$

Level II Lineage (Approximate)

Type	Lineage Formula
Sufficient	$\tilde{\lambda}^S_{t_6} = x_1 \wedge x_2$
Arithmetization	$\tilde{\lambda}^A_{t_6} = x_1(1-(1-x_2)(1-x_3))$
Polynomial	$\tilde{\lambda}^P_{t_6} = \frac{33}{128} + \frac{21}{32}(x_2 - \frac{1}{4}) + \frac{9}{16}(x_3 - \frac{1}{8})$

Figure 1: `Process` (P) relates each gene product to a process, *e.g.* AG02 is involved in "cell death". Each tuple in `Process` has an annotation from the set of atoms. An atom, x_i for $i = 1, 2, 3$, is a piece of evidence that has an associated probability, *e.g.* x_1 is the proposition that we trust *"Dr. X.'s PubMed article PMID:12593804"*, which we assign probability $\frac{3}{4}$. V is a view that asks for *"Gene Products that share a process with a product 'Aac11"'*. Below V's definition is its output in the original database with a complete approach. At the right examples of approximate lineage functions we consider are listed. The compressed database is obtained by replacing λ with one of these functions, *e.g.* $\tilde{\lambda}^S_{t_6}$. This database is inspired by the Gene Ontology (GO) database [14].The terms (*Level I*) and (*Level II*) are specific to our approach and defined in (Sec. 1.1).

the scientist can now obtain the *reliability* of each query answer. To compute the reliability value in a complete approach, we may be forced to process all the lineage for a given tuple. This is challenging, because the lineage can be very large. This problem is not unique to GO. For example, [13] reports that a 250MB biological database has 6GB of lineage. In this work, we show how to use *approximate lineage* to effectively compress the lineage more than two orders of magnitude, even for extremely low error rates. Importantly, our compression techniques allow us to process queries directly on the compressed data. In our experiments, we show that this can result in up to two orders of magnitude more efficient processing than a complete approach.

An additional important activity for scientists is understanding the data; the role of the database in this task is to provide interactive results to hone the scientist's knowledge. As a result, we cannot tolerate long delays. For example, the lineage of even a single tuple in the gene ontology database may be 9MB. Consider a scientist who finds the result of V in Fig 1 surprising: One of her goals may be to find out why t_6 is returned by the system, *i.e.* she wants a sufficient explanation as to why AGO2 was returned. The system would return that the *most likely explanation* is that we trust Dr.X that AGO2 is related to cell death (t_1) and Dr.Y's RA that Aac11 is also related to cell death (t_4). An alternative explanation uses t_1 and the automatic similarity computation (t_5). However, the first explanation is more likely, since the annotation associated with t_4 (x_2) is more likely than the annotation of t_5 (x_3), here $\frac{1}{4} = p(x_2) \geq p(x_3) = \frac{1}{8}$.

A scientist also needs to understand the effect of her trust policy on the reliability score of t_6. Specifically, she needs to know which atom is the most *influential* to computing the reliability for t_6. In this case, the scientist is relatively sure that AGO2 is associated with cell death, since it is assigned a score of $\frac{3}{4}$. However, the key new clement leading to this surprising result is that Aac11 is also associated "cell death", which is supported by the atom x_2, the statement of Dr. Y's RA. Concretely, x_2 is the most influential atom because changing x_2's value will change the reliability of t_6 more than changing any other atom. In our experiments, we show that we can find sufficient explanations with high precision, *e.g.* we can find the top 10 influential explanations with between 70% and 100% accuracy. Additionally, we can find influential atoms with high precision (80% – 100% of the top 10 influential atoms). In both cases, we can conduct these exploration tasks without directly accessing the raw data.

Application (2): Managing Similarity Scores Applications that manage similarity scores can benefit from approximate lineage. Such applications include managing data from object reconciliation procedures [3, 34] or similarity scores between users, such as iLike.com. In iLike, the system automatically assigns a music compatibility score between friends. The similarity score between two users, *e.g.* Bob and Joe, has a lineage: It is a function of many atomic facts, *e.g.* which songs they listen to and how frequently, which artists they like, *etc.* All of these atomic facts are combined into a single numeric score which is then converted into *quantitative* buckets, *e.g.* high, medium and low. Intuitively, to compute such rough buckets, it is unnecessary to precisely maintain every bit of lineage. However, this painstaking computation is required by a complete approach. In this paper, we show how to use approximate lineage to effectively compress object reconciliation data in the IMDB database [35].

1.1 Overview of our Approach

At a high level, both of our example applications, large scientific data and managing similarity scores, manage data that is annotated with probabilities. In both applications, we propose a two-level architecture: The *Level I* database is a large, high-quality database that uses a complete approach and is queried infrequently. The *Level II* database is much smaller, and uses an approximate lineage system. A user conducts her query and exploration tasks on the *Level II* database, which is the focus of this paper.

The key technical idea of this work is **approximate lineage**, which is a strict generalization of complete lineage. Abstractly, lineage is a function λ that maps each tuple t in a database to a Boolean formula λ_t over a fixed set of Boolean atoms. For example in Fig. 1, the lineage of the tuple t_6 is $\lambda_{t_6} = (x_1 \wedge x_2) \vee (x_1 \wedge x_3)$. In this paper, we propose two instantiations of approximate lineage: a conservative approximation, **sufficient lineage**, and a more aggressive approximation, **polynomial lineage**.

In **sufficient lineage**, each lineage function is replaced with a smaller formula that logically implies the original. For example, a sufficient lineage for t_6 is $\tilde{\lambda}^S_{t_6} = x_1 \wedge x_2$. The advantage of suffi-

cient lineage is that it can be much smaller than standard lineage, which allows query processing and exploration takes to proceed much more efficiently. For example, in our experiments processing a query on an uncompressed data took 20 hours, while it completed in 30m on a database using sufficient lineage. Additionally, understanding query reliability is easy with sufficient lineage: the reliability computed for a query q is always less than or equal to the reliability computed on the original Level I database. However, only monotone lineage functions can be represented by a sufficient approach.

The second generalization is **polynomial lineage** which is a function that maps each tuple t in a database to a *real-valued polynomial* on Boolean variables, denoted $\tilde{\lambda}_t^P$. An example polynomial lineage is $\tilde{\lambda}_{t_6}^P$ in Fig. 1. There are two advantages of using real-valued polynomials instead of Boolean-valued functions: (1) powerful analytic techniques already exist for understanding and approximating real-valued polynomials, *e.g.* Taylor series or Fourier Series, and (2) *any* lineage function can be represented by polynomial approximate lineage. Polynomial lineage functions can allow a more accurate semantic than sufficient lineage in the same amount of space, *i.e.* , the difference in value between computing q on the Level I and Level II database is small. In Sec. 5 we demonstrate a view in GO such that polynomial lineage achieves a compression ratio of 171 : 1 and sufficient lineage achieves 27 : 1 compression ratio with error rate less than 10^{-3} (Def. 2.10).

Although polynomial lineage can give better compression ratios and can be applied to a broader class of functions, there are three advantages of sufficient lineage over polynomial lineage: (1) sufficient lineage is syntactically identical to complete lineage, and so can be processed by existing probabilistic relational databases without modification, *e.g.* Trio and Mystiq. (2) The semantic of sufficient lineage is easy to understand since the value of a query is a lower bound of the true value, while a query may have either a higher or lower value using polynomial lineage. (3) Our experiments show that sufficient lineage is less sensitive to skew, and can result in better compression ratios when the probability assignments to atoms are very skewed.

In both lineage systems, there are three fundamental technical challenges: creating it, processing it and understanding it. In this paper, we study these three fundamental problems for both forms of approximate lineage.

1.2 Contributions, Validation and Outline

We show that we can (1) efficiently construct both types of approximate lineage, (2) process both types of lineage efficiently and (3) use approximate lineage to explore and understand the data.

- In Sec. 2, we define the semantics of approximate lineage, motivate the technical problems that any approximate lineage system must solve and state our main results. The technical problems are: creating approximate lineage (Prob. 1); explaining the data, *i.e.* finding sufficient explanations (Prob. 2), finding influential variables (Prob. 3); and query processing with approximate lineage (Prob. 4).
- In Sec. 3, we define our implementation for one type of approximate lineage, *sufficient lineage*. This requires that we solve the three problems above: we give algorithms to construct it (Sec. 3.2), to use it to understand the data (Sec. 3.3), and to process further queries on the data (Sec. 3.4).
- In Sec. 4, we define our proposal for *polynomial approximate lineage*; our proposal brings together many previous results in the literature to give algorithms to construct it (Sec. 4.2), to understand it (Sec. 4.3) and to process it.
- In Sec. 5, we provide experimental evidence that both approaches work well in practice; in particular, we show that approximate lineage can compress real data by orders of magnitude even with very low error, (Sec. 5.2), provide high quality explanations (Sec. 5.3) and provide large performance improvements (Sec. 5.4). Our experiments use data from the Gene Ontology database [14, 52] and a probabilistic database of IMDB [35] linked with reviews from Amazon.

We discuss related work in Sec. 6 and conclude in Sec. 7.

2. STATEMENT OF RESULTS

We first give some background on lineage and probabilistic databases, and then formally state our problem with examples.

2.1 Preliminaries: Queries and Views

In this paper, we consider conjunctive queries and views written in a datalog-style syntax. A query q is a conjunctive rule written $q :- g_1, \ldots, g_n$ where each g_i is a subgoal, that is, a relational predicate. For example, $q_1 :- R(x), S(x, y, \text{'a'})$ defines a query with a join between R and S, a variable y that is projected away, and a constant 'a'. For a relational database W, we write $W \models q$ to denote that W entails q.

2.2 Lineage and Probabilistic Databases

In this section, we adopt a viewpoint of lineage similar to *c*-tables [29, 36]; we think of lineage as a constraint that tells us which worlds are possible. This viewpoint results in the standard *possible worlds semantics* for probabilistic databases [16, 20, 29].

DEFINITION 2.1 (LINEAGE FUNCTION). *An **atom** is a Boolean proposition about the real world, e.g. Bob likes Herbie Hancock. Fix a relational schema σ and a set of atoms $\mathcal{A}$. A lineage function, λ, assigns to each tuple t conforming to some relation in σ, a Boolean expression over $\mathcal{A}$, which is denoted λ_t. An **assignment** is a function $\mathcal{A} \to \{0, 1\}$. Equivalently, it is a subset of $\mathcal{A}$, denoted A, consisting of those atoms that are assigned true.*

Fig. 1 illustrates tuples and their lineages. The atoms represent propositions about data provenance. For example, the atom x_1 in Fig. 1 represents the proposition that we trust "Dr. X's PubMed PMID:12593804". Of course, atoms can also represent more coarsely grained propositions, "A scientist claimed it was true" or finely-grained facts "Dr. X claimed it in PubMed 18166081 on page 10". In this paper, we assume that the atoms are given; we briefly discuss this at the end the current section..

To define the standard semantics of lineage, we define a *possible world* W through a two-stage process: (1) select a subset of atoms, A, *i.e.* an assignment, and (2) For each tuple t, if $\lambda_t(A)$ evaluates to true then t is included in W. This process results in an unique world W for any choice of atoms A.

Example 2.2 If we select $A_{13} = \{x_1, x_3\}$, that is, we trust Dr. X and Dr. Y's RA, but distrust the similarity computation, then $W_{1256} = \{t_1, t_2, t_5, t_6\}$ is the resulting possible world. The reason is that for each $t_i \in W_{1256}$, λ_{t_i} is satisfied by the assignment corresponding to A_{13} and for each $t_j \notin W_{1256}$, λ_{t_j} is false. In contrast, $W_{125} = \{t_1, t_2, t_5\}$ is *not* a possible world because in W_{125}, we know that AGO2 and Aac11 are both associated with Cell Death, and so AGO2 should appear in the view (t_6). In symbols, $\lambda_{t_6}(W_{125}) = 1$, but $t_6 \notin W_{125}$.

We capture this example in the following definition:

DEFINITION 2.3. *Fix a schema σ. A **world** is a subset of tuples conforming to σ. Given a set of atoms A and a world W, we say that W is a **possible world** induced by A if it contains exactly those tuples consistent with the lineage function, that is, for all tuples t, $\lambda_t(A) \iff t \in W$. Moreover, we write $\lambda(A, W)$ to denote the Boolean function that takes value 1 if W is a possible world induced by A. In symbols,*

$$\lambda(A, W) \stackrel{\text{def}}{=} \bigwedge_{t:t\in W} \lambda_t(A) \bigwedge_{t:t\notin W} (1 - \lambda_t(A)) \tag{1}$$

Eq. 1 is important, because it is the main equation that we generalize to get semantics for approximate lineage.

We complete the construction of a probabilistic database as a distribution over possible worlds. We assume that there is a function p that assigns each atom $a \in \mathcal{A}$ to a probability score denoted $p(a)$. In Fig. 1, x_1 has been assigned a score $p(x_1) = \frac{3}{4}$, indicating that we are very confident in Dr. X's proclamations. An important special case is when $p(a) = 1$, which indicates absolute certainty.

DEFINITION 2.4. *Fix a set of atoms $\mathcal{A}$. A **probabilistic assignment** p is a function from $\mathcal{A}$ to $[0, 1]$ that assigns a probability score to each atom $a \in \mathcal{A}$. A **probabilistic database** $\mathcal{W}$ is a probabilistic assignment p and a lineage function λ that represents a distribution μ over worlds defined as:*

$$\mu(W) \stackrel{\text{def}}{=} \sum_{A\subseteq\mathcal{A}} \lambda(A, W) \left(\prod_{i:a_i\in A} p(a_i)\right)\left(\prod_{j:a_j\notin A} (1 - p(a_j))\right)$$

Given any Boolean query q on $\mathcal{W}$, the marginal probability of q denoted $\mu(q)$ is defined as

$$\mu(q) \stackrel{\text{def}}{=} \sum_{q:W\models q} \mu(W) \tag{2}$$

i.e. the sum of the weights over all worlds that satisfy q.

Since for any A, there is a unique W such that $\lambda(A, W) = 1$, μ is a probability measure. In all of our semantics, the semantic for queries will be defined similarly to Eq. 2.

Example 2.5 Consider a simple query on our database:

$$q_1() :- \mathtt{P}(x, \text{'Gland Development'}), \mathtt{V}(x)$$

This query asks if there exists a gene product x, that is associated with 'Gland Development', and also has a common function with 'Aac11', that is it also appears in the output of V. On the data in Fig. 1, q_1 is satisfied on a world W if and only if (1) *AGO2 is associated with Gland development* and (2) *AGO2 and Aac11 have a common function*, here, either Embryonic Development or Cell Death. The subgoal requires that t_3 be present and the second that t_6 be present. The formula $\lambda_{t_3} \wedge \lambda_{t_6}$ simplifies to $x_1 \wedge x_2$, *i.e.* we must trust both Dr.X and Dr.Y's RA to derive q_1, which has probability $\frac{3}{4} \cdot \frac{1}{4} = \frac{3}{16} \approx 0.19$.

We now generalize the standard (complete) semantics to give approximate semantics; the approximate lineage semantics are used to give semantics to the compressed Level II database.

Sufficient Lineage Our first form of approximate lineage is called *sufficient lineage*. The idea is simple: Each λ_t is replaced by a Boolean formula $\tilde{\lambda}^S_t$ such that $\tilde{\lambda}^S_t \implies \lambda_t$ is a tautology. Intuitively, we think of $\tilde{\lambda}^S_t$ as a good approximation to λ_t if λ^S_t and λ_t agree on most assignments. We define the function $\tilde{\lambda}^S(A, W)$ following Eq.1:

$$\tilde{\lambda}^S(A, W) \stackrel{\text{def}}{=} \bigwedge_{t:t\in W} \tilde{\lambda}^S_t(A) \bigwedge_{t:t\notin W} (1 - \tilde{\lambda}^S_t(A)) \tag{1s}$$

The formula simply replaces each tuple t's lineage, λ_t with sufficient lineage, $\tilde{\lambda}^S_t$ and then checks whether W is a possible world for A given the sufficient lineage.This, in turn, defines a new probability distribution on worlds $\tilde{\mu}^S$:

$$\tilde{\mu}^S(W) \stackrel{\text{def}}{=} \sum_{A\subseteq\mathcal{A}} \tilde{\lambda}^S(A, W) \left(\prod_{i:a_i\in A} p(a_i)\right)\left(\prod_{j:a_j\notin A} (1 - p(a_j))\right)$$

Given a query q, we define $\tilde{\mu}^S(q)$ exactly as in Eq. 2, with μ syntactically replaced by $\tilde{\mu}^S$, *i.e.* as a weighted sum over all worlds W satisfying q. Two facts are immediate: (1) $\tilde{\mu}^S$ is a probability measure and (2) for any a conjunctive (monotone) query q, $\tilde{\mu}^S(q) \leq \mu(q)$. Sufficient lineage is syntactically the same as standard lineage. Hence, it can be used to process queries with existing relational probabilistic database systems, such as Mystiq and Trio. If the lineage is a large DNF formula, then any single disjunct is a sufficient lineage. However, there is a trade off between choosing sufficient lineage that is small and lineage that is a good approximation. In some cases, it is possible to get both. For example, the lineage of a tuple may be less than 1% of the original lineage, but still be a very precise approximation.

Example 2.6 We evaluate q from Ex. 2.5. In Fig. 1, a sufficient lineage for tuple t_6 is trusting Dr. X and Dr. Y's RA, that is $\tilde{\lambda}^S_{t_6} = x_1 \wedge x_2$. Thus, q is satisfied exactly with this probability which is ≈ 0.19. In this example, the sufficient lineage computes the exact answer, but this is not the general case. In contrast, if we had chosen $\tilde{\lambda}^S_{t_6} = x_1 \wedge x_3$, *i.e.* our explanation was trusting Dr.X and the matching, we would have computed that $\tilde{\mu}^S(q) = \frac{3}{4} \cdot \frac{1}{8} = \frac{3}{32} \approx 0.09 \leq 0.19$.

One can also consider the dual form of sufficient lineage, *necessary lineage*, where each formula λ_t is replaced with a Boolean formula $\tilde{\lambda}^N_t$, such that $\lambda_t \implies \tilde{\lambda}^N_t$ is a tautology. Similar properties hold for necessary lineage: For example, $\tilde{\mu}^N$ is an upper bound for μ, which implies that using necessary and sufficient lineage in concert can provide the user with a more robust understanding of query answers. For the sake of brevity, we shall focus on sufficient lineage for the remainder of the paper.

Polynomial Lineage In contrast to both standard and sufficient lineages that map each tuple to a Boolean function, polynomial approximate lineage maps each tuple to a *real-valued function*. This generalization allows us to leverage approximation techniques for real-valued functions, such as Taylor and Fourier series.

Given a Boolean formula λ_t on Boolean variables $x_1, \ldots, x_n$ an *arithmetization* is a real-valued polynomial $\lambda^A_t(x_1, \ldots, x_n)$ in real variables such that (1) each variable x_i has degree 1 in λ^A_t and (2) for any $x_1, \ldots, x_n \in \{0, 1\}^n$, we have $\lambda_t(x_1, \ldots, x_n) = \lambda^A_t(x_1, \ldots, x_n)$ [42, p. 177]. For example, an arithmetization of $xy \vee xz$ is $x(1-(1-y)(1-z))$ and an arithmetization of $xy \vee xz \vee yz$ is $xy + xz + yz - 2xyz$. Fig. 1 illustrates an arithmetization of the lineage formula for t_6, which is denoted $\lambda^A_{t_6}$.

In general, the arithmetization of a lineage formula may be exponentially larger than the original lineage formula. As a result, we do not use the arithmetization directly; instead, we approximate it. For example, an approximate polynomial for λ_{t_6} is $\tilde{\lambda}^P_{t_6}$ in Fig. 1.

To define our formal semantics, we define $\tilde{\lambda}^P(A, W)$ generalizing Eq. 1 by allowing $\tilde{\lambda}^P$ to assign a real-valued, as opposed to Boolean, weight.

$$\tilde{\lambda}^P(A, W) \stackrel{\text{def}}{=} \prod_{t:t\in W} \tilde{\lambda}^P_t(A) \prod_{t:t\notin W} (1 - \tilde{\lambda}^P_t(A)) \tag{1p}$$

In addition to assigning real-valued weights to worlds, as opposed to Boolean weights, Eq. 1p maps an assignment of atoms, A, to *many worlds* by polynomial lineage, instead of to only a single world, as is done in the standard approach and sufficient approaches.

Example 2.7 In Fig. 1, W_{1256} is a possible world since $\lambda(A, W_{1256}) = 1$ for $A = \{x_1, x_3\}$. In contrast, $\tilde{\lambda}^P(A, W_{1256}) \neq 1$. To see this, $\tilde{\lambda}^P(A, W_{1256})$ simplifies to $\tilde{\lambda}^P_{t_6}(A, W_{1256})$, since all other lineage functions have $\{0, 1\}$ values. Evaluating $\tilde{\lambda}^P_{t_6}(A)$ gives $\frac{33}{128} + \frac{21}{32}(0 - \frac{1}{4}) + \frac{9}{16}(1 - \frac{1}{8}) = \frac{75}{128} \approx 0.58$. Further, approximate lineage functions may assign non-zero mass even to worlds which are not possible. For example W_{125} is not a possible world, but $\tilde{\lambda}^P(A, W_{13}) = 1 - \lambda_{t_6}(A)(1 - \frac{75}{128}) \neq 0$.

The second step in the standard construction is to define a probability measure μ (Def. 2.4); In approximate lineage, we define a function $\tilde{\mu}^P$ – which may not be a probability measure – that assigns arbitrary real-valued weights to worlds. Here, $p_i = p(a_i)$ where p is a probability assignment as in Def. 2.4:

$$\tilde{\mu}^P(W) \stackrel{\text{def}}{=} \sum_{A \subseteq \mathcal{A}} \tilde{\lambda}(A, W) \left(\prod_{i:a_i \in A} p_i \right) \left(\prod_{j:a_j \notin A} (1 - p_j) \right) \tag{3}$$

Our approach is to search for $\tilde{\lambda}^P$ that is a good approximation, that is if for any q, we have $\tilde{\mu}(q) \approx \mu(q)$, *i.e.* the value computed using approximate lineage is close to the standard approach. Similar to sufficient lineage, we get a query semantic by syntactically replacing μ by $\tilde{\mu}^P$ in Eq. 2. However, the semantics for polynomial lineage is more general than the two previous semantics, since an assignment is allowed map to *many* worlds.

Example 2.8 Continuing Ex. 2.7, in the original data, $\mu(W_{1256}) = \frac{9}{128}$. However, $\tilde{\mu}^P$ assigns W_{1256} a different weight:

$$\tilde{\mu}^P(W_{1256}) = \tilde{\lambda}^P(W_{1256})(\frac{3}{4})(\frac{1}{8})(1 - \frac{1}{4}) = \frac{75}{128}\frac{9}{128}$$

Recall q_1 from Ex. 2.5; its value is $\mu(q_1) \approx 0.19$. Using Eq. 3, we can calculate that the value of q_1 on the Level II database using polynomial linage in Fig. 1 is $\tilde{\mu}^P(q_1) \approx 0.17$. In this case the error is ≈ 0.02. If we had treated the tuples in the database independently, we would get the value $\frac{1}{4} * \frac{11}{32} \approx 0.06$ – an error of 0.13, which is an order of magnitude larger error than an approach using polynomial lineage. Further, $\tilde{\lambda}^P$ is smaller than the original Boolean formula.

2.3 Problem Statements and Results

In our approach, the original Level I database, that uses a complete lineage system, is lossily compressed to create a Level II database, that uses an approximate lineage system; we then perform all querying and exploration on the Level II database. To realize this goal, we need to solve three technical problems (1) create a "good" Level II database, (2) provide algorithms to explore the data given the approximate lineage and (3) process queries using the approximate lineage.

Internal Lineage Functions Although our algorithms apply to general lineage functions, many of our theoretical results will consider an important special case of lineage functions called *internal lineage functions* [7]. In internal linage functions, there are some tables (base tables) such that every tuple is annotated with a single atom, *e.g.* P in Fig. 1. The database also contains derived tables (views), *e.g.* V in Fig. 1. The lineage for derived tables is derived using the definition of V and tuples in base tables. For our purposes, the significance of internal lineage is that all lineage is a special kind of Boolean formula, a k-monotone DNFs (k-mDNF). A Boolean formula is a k-mDNF if it is a disjunction of monomials each containing at most k literals and no negations. The GO database is caputred by an internal lineage function.

PROPOSITION 2.9. *If t is a tuple in a view V such that, when unfolded, references k (not necessarily distinct) base tables, then the lineage function λ_t is a k-mDNF.*

One consequence of this is that k is typically small. And so, as in data complexity [1], we consider k a small constant. For example, an algorithm is considered efficient if it is at most polynomial in the size of the data, but possibly exponential in k.

2.3.1 Creating Approximate Lineage

Informally, approximate lineage is good if (1) for each tuple t the function $\tilde{\lambda}_t$ is a close approximation of λ_t, *i.e.* λ_t and $\tilde{\lambda}_t$ are close on many assignments, and (2) the size of $\tilde{\lambda}_t$ is small for every t. Here, we write $\tilde{\lambda}_t$ (without a superscript) when a statement applies to either type of approximate lineage.

DEFINITION 2.10. *Fix a set of atoms $\mathcal{A}$. Given a probabilistic assignment p for $\mathcal{A}$, we say that $\tilde{\lambda}_t$ is an ε-approximation of λ_t if*

$$\mathbf{E}_p[(\tilde{\lambda}_t - \lambda_t)^2] \leq \varepsilon$$

where $\mathbf{E}_p$ denotes the expectation over assignments to atoms induced by the probability function p.

Our goal is to ensure that the lineage function for *every* tuple in the database has an ε-approximation. Def. 2.10 is used in computational learning, *e.g.* [40, 50], because an ε-approximation of a function disagrees only a few inputs:

Example 2.11 Let y_1 and y_2 be atoms such that $p(y_i) = 0.5$ for $i = 1, 2$. Consider the lineage function for some t, $\lambda_t(y_1, y_2) = y_1 \vee y_2$ and an approximate lineage function $\tilde{\lambda}^S_t(y_1, y_2) = \tilde{\lambda}^P_t(y_1, y_2) = y_1$. Here, λ_t and $\tilde{\lambda}^S_t$ (or $\tilde{\lambda}^P_t$) differ on precisely one of the four assignments, *i.e.* $y_1 = 0$ and $y_2 = 1$. Since all assignments are equally weighted, $\tilde{\lambda}^S_t$ is a 1/4-approximation for λ. In general, if λ_1 and λ_2 are Boolean functions on atoms $A = \{y_1, \ldots, y_n\}$ such that $p(y_i) = 0.5$ for $i = 1, \ldots, n$, then λ_1 is an ε approximation of λ_2 if λ_1 and λ_2 differ on less than an ε fraction of assignments.

Our first problem is constructing lineage that has arbitrarily small error approximation and occupies a small amount of space.

PROBLEM 1 (CONSTRUCTING LINEAGE). *Given a linage function λ_t and an input parameter ε, can we efficiently construct an ε-approximation for λ_t that is small?*

For internal lineage functions, we show how to construct approximate lineage efficiently that is provably small for both sufficient lineage (Sec. 3.2) and polynomial lineage (Sec. 4.2), under the technical assumption that the atoms have probabilities bounded away from 0 and 1, *e.g.* we do *not* allow probabilities of the form n^{-1} where n is the size of the database. Further, we experimentally verify that sufficient lineage offers compression ratios of up to 60 : 1 on real datasets and polynomial lineage offers up to 171 : 1 even with stringent error requirements, *e.g.* $\varepsilon = 10^{-3}$.

2.3.2 Understanding Lineage

Recall our scientist from the introduction, she is skeptical of an answer the database produces, *e.g.* t_6 in Fig. 1, and wants to understand why the system believes that t_6 is an answer to her query. We informally discuss the primitive operations our system provides to help her understand t_6 and then define the corresponding formal problems.

Sufficient Explanations She may want to know the possible *explanations* for a tuple, *i.e.* a sufficient reason for the system to return t_6. There are many explanations and our technical goal is to find the top-k *most likely* explanations from the Level II database.

Finding influential atoms Our scientist may want to know which atoms contributed to returning the surprising tuple, t_6. In a complicated query, the query will depend on many atoms, but some atoms are more *influential* than others. Informally, an atom x_1 is influential if it there are many assignments such that it is the "deciding vote", *i.e.* changing the assignment of x_1 changes whether t_6 is returned. Our technical goal is to return the most influential atoms directly from the Level II database, without retrieving the much larger Level I database.

Intuitively, sufficient lineage supports sufficient explanations better than polynomial lineage because the lineage formula *is* a set of good sufficient explanations. In contrast, our proposal for polynomial lineage supports finding the influential tuples more naturally. We now discuss these problems more formally:

Sufficient Explanations An **explanation** for a lineage function λ_t is a minimal conjunction of atoms τ_t such that for any assignment $\boldsymbol{a}$ to the atoms, we have $\tau(\boldsymbol{a}) \implies \lambda_t(\boldsymbol{a})$. The probability of an explanation, τ, is $\mathbf{P}[\tau]$. Our goal is to retrieve the *top-k* explanations, ranked by probability, from the lossily-compressed data.

PROBLEM 2. *Given a tuple t, calculate the top-k explanations, ranked by their probability using only the Level II database.*

This problem is straightforward for sufficient lineage, but challenging for polynomial lineage. The first reason is that polynomials seem to throw away information about monomials. For example, $\tilde{\lambda}^P_{t_6}$ in Fig. 1 does not mention the terms of *any* monomial. Further complicating matters is that even computing the expectation of $\tilde{\lambda}^P_{t_6}$ may be intractable, and so we have to settle for approximations which introduce error. As a result, we must resort to statistical techniques to guess if a formula τ_t is a sufficient explanation. In spite of these problems, we are able to use polynomial lineage to retrieve sufficient explanations with a precision of up to 70% for $k = 10$ with error in the lineage, $\varepsilon = 10^{-2}$.

Finding Influential Atoms The technical question is: Given a formula, *e.g.* λ_{t_6}, which atom is most influential in computing λ_{t_6}'s value? We define the **influence** of x_i on λ_t, denoted $\text{Inf}_{x_i}(\lambda_t)$, as:

$$\text{Inf}_{x_i}(\lambda_t) \stackrel{\text{def}}{=} \mathbf{P}[\lambda_t(A) \neq \lambda_t(A \oplus \{i\})] \tag{4}$$

where $\oplus$ denotes the symmetric difference. This definition, or a closely related one, has appeared has appeared in wide variety of work, *e.g.* underling causality in the AI literature [31, 44], influential variables in the learning literature [40], and critical tuples in the database literature [41, 47].

Example 2.12 What influence does x_2 have on tuple t_6 presence, *i.e.* what is the value of $\text{Inf}_{x_1}(\lambda_{t_6})$? Informally, x_2 can only change the value of λ_{t_6} if x_1 is true and x_3 is false. This happens with probability $\frac{3}{4}(1-\frac{1}{8}) = \frac{21}{32}$, which is not coincidentally the coefficient of x_2 in $\tilde{\lambda}_{t_6}$.

The formal problem is to find the top k most influential variables, *i.e.* the variables with the k highest influences:

PROBLEM 3. *Given a tuple t, efficiently calculate the k most influential variables in λ_t using only the level II database.*

This problem is challenging because the Level II database is a lossily-compressed version of the database and so some information needed to exactly answer Prob. 3 is not present. The key observation for polynomial lineage is that the coefficients we retain are the coefficients of influential variables; this allows us to compute the influential variables efficiently in many cases. We show that we can achieve an almost-perfect average precision for the top 10. For sufficient lineage, we are able to give an approach with bounded error to recover the influential coefficients.

2.3.3 Query Processing with Approximate Lineage

Our goal is to efficiently answer queries directly on the Level II database, using sampling approaches:

PROBLEM 4. *Given an approximate lineage function $\tilde{\lambda}$ and a query q, efficiently evaluate $\tilde{\mu}(q)$ with low-error.*

Processing sufficient lineage is straightforward using existing complete techniques; However, we are able to prove that the error will be small. We verify experimentally that we can answer queries with low-error 10^{-3}, 2 orders of magnitude more quickly than a complete approach. For polynomial lineage, we are able to directly adapt techniques form the literature, such as [8].

2.4 Discussion

The acquisition of atoms and trust policies is an interesting future research direction. Since our focus is on large databases, it is impractical to require users to label each atom manual. One approach is to define a language for specifying trust policies. Such a language could do double duty, by also specifying correlations between atoms. We consider the design of a policy language to be important future work. In this paper, we assume that the atoms are given, the trust policies are explicilty specified, and all atoms are independent.

3. SUFFICIENT LINEAGE

We define our proposal for sufficient lineage that replaces a complicated lineage formula λ_t, by a simpler (and smaller) formula $\tilde{\lambda}^S_t$. We construct $\tilde{\lambda}^S_t$ using several sufficient explanations for λ_t.

3.1 Sufficient Lineage Proposal

Given an internal lineage function for a tuple t, that is, a monotone k-DNF formula λ_t, our goal is to efficiently find a sufficient lineage $\tilde{\lambda}^S_t$ that is small and is an ε-approximation of λ_t (Def. 2.10). This differs from *L-minimality* [6] that looks for a formula that is equivalent, but smaller. In contrast, we look for a formula that may only approximate the original formula. More formally, the size of a sufficient lineage $\tilde{\lambda}^S_t$ is the number of monomials it contains, and so is small if it contains few monomials. The definition of ε-approximation (Def. 2.10) simplifies for sufficient lineage and gives us intuition how to find good sufficient lineage.

PROPOSITION 3.1. *Fix a Boolean formula λ_t and let $\tilde{\lambda}^S_t$ be a sufficient explanation for λ_t, that is, for any assignment A, we have $\tilde{\lambda}^S_t(A) \implies \lambda_t(A)$. In this situation, the error function simplifies to $\mathbf{E}[\lambda_t] - \mathbf{E}[\tilde{\lambda}^S_t]$; formally, the following equation holds $\mathbf{E}[(\lambda_t - \tilde{\lambda}^S_t)^2] = \mathbf{E}[\lambda_t] - \mathbf{E}[\tilde{\lambda}^S_t]$*

Prop. 3.1 tells us that to get sufficient lineage with the low error, it is enough to look for sufficient formula $\tilde{\lambda}_t$ with *high* probability.

PROOF SKETCH. The formula $(\lambda_t - \tilde{\lambda}^S_t)^2$ is non-zero only if $\lambda_t \neq \tilde{\lambda}^S_t$, which means that $\lambda_t = 1$ and $\tilde{\lambda}^S_t = 0$, since $\tilde{\lambda}^S_t(A) \implies \lambda_t(A)$ for any A. Because both λ_t and $\tilde{\lambda}^S_t$ are Boolean, $(\lambda_t - \tilde{\lambda}^S_t)^2 \in \{0, 1\}$ and simplifies to $\lambda_t - \tilde{\lambda}^S_t$. We use linearity of $\mathbf{E}$ to conclude. □

Scope of Analysis In this section, our theoretical analysis considers only internal lineage functions with constant bounded probability distributions; a distribution is *constant bounded* if there is a constant β such that for any atom a, $p(a) > 0$ implies that $p(a) \geq \beta$. To justify this, recall that in GO, the probabilities are computed based on the type of evidence: For example, a citation in PubMed is assigned 0.9, while an automatically inferred matching is assigned 0.1. Here, $\beta = 0.1$ and is independent of the size of the data. In the following discussion, β will always stand for this bound and k will always refer to the maximum number of literals in any monomial of the lineage formula. Further, we shall only consider sufficient lineage which are subformulae of λ_t. This choice guarantees that the resulting formula is sufficient lineage and is also simple enough for us to analyze theoretically.

3.2 Constructing Sufficient Lineage

The main result of this section is an algorithm (Alg. 3.2.1) that constructs good sufficient lineage, solving Prob. 1. Given an error term, ε, and a formula λ_t, Alg. 3.2.1 efficiently produces an approximate sufficient lineage formula $\tilde{\lambda}_t^S$ with error less than ε. Further, Thm. 3.2 shows that the size of the formula produced by Alg. 3.2.1 depends only on ε, k and β – not on the number of variables or number of terms in λ_t; implying that the formula is theoretically small.

Algorithm 3.2.1 Suff(λ_t, ε) constructs sufficient lineage

Input: A monotone k+1-DNF formula λ_t and an error $\varepsilon > 0$
Output: $\tilde{\lambda}_t^S$, a small sufficient lineage ε-approximation.

1: Find a matching M, greedily. (*A subset of monomials*)
2: **if** $\mathbf{P}[\lambda_t^S] - \mathbf{P}[M] \leq \varepsilon$ **then** (*If λ_t is a 1-mDNF always true*)
3: Let $M = m_1 \vee \cdots \vee m_l$ s.t. $i \leq j$ implies $\mathbf{P}[m_i] \geq \mathbf{P}[m_j]$
4: **return** $M_r \stackrel{\text{def}}{=} m_1, \ldots, m_r$, r is min s.t. $\mathbf{P}[\lambda_t] - \mathbf{P}[M_r] \leq \varepsilon$.
5: **else**
6: Select a small cover $C = \{x_1, \ldots, x_c\} \subseteq \mathbf{var}(M)$
7: Arbitrarily assign each monomial to a $x_c \in C$ that covers it
8: **for each** $x_i \in C$ **do**
9: $\lambda_i^S \leftarrow \mathbf{Suff}(\lambda_i[x_i \rightarrow 1], \varepsilon/c)$. (*$\lambda_i[x_i \rightarrow 1]$ is a k-DNF*)
10: **return** $\bigvee_{i=1,\ldots,n} \lambda_i^S$.

Algorithm Description Alg. 3.2.1 is a recursive algorithm, whose input is a k-mDNF λ_t and an error $\varepsilon > 0$, it returns $\tilde{\lambda}_t^S$, a sufficient ε-approximation. For simplicity, we assume that we can compute the expectation of monotone formula exactly. In practice, we estimate this quantity using sampling, *e.g.* using Luby-Karp [38]. The algorithm has two cases: In case (I) on lines 2-4, there is a large matching, that is, a set of monomials M such that distinct monomials in M do not contain common variables. For example, in the formula $(x_1 \wedge y_1) \vee (x_1 \wedge y_2) \vee (x_2 \wedge y_2)$ a matching is $(x_1 \wedge y_1) \vee (x_2 \vee y_2)$. In Case (II) lines 6-10, there is a small cover, that is a set of variables $C = \{x_1, \ldots, x_c\}$ such that every monomial in λ_t contains some element of C. For example, in $(x_1 \wedge y_1) \vee (x_1 \wedge y_2) \vee (x_1 \wedge y_3)$, the singleton $\{x_1\}$ is a cover. The relationship between the two cases is that if we find a maximal matching smaller than m, then there is a cover of size smaller than km (all variables in M form a cover).

Case I: (lines 2-4) The algorithm greedily selects a maximal matching $M = \{m_1, \ldots, m_l\}$. If M is a good approximation, *i.e.* $\mathbf{P}[\lambda_t^S] - \mathbf{P}[\bigvee_{m \in N} m] \leq \varepsilon$ then we trim M to be as small as possible so that it is still a good approximation. Observe that $P[\bigvee_{m \in M} m]$ can be computed efficiently since the monomials in M do not share variables, and so are independent. Further, for any size l the subset of M of size l with the highest probability is exactly the l highest monomials.

Case II: (lines 6-10) Let $\mathbf{var}(M)$ be the set of all variables in the maximal matching we found. Since M is a maximal matching, $\mathbf{var}(M)$ forms a cover, $x_1, \ldots, x_c$. We then arbitrarily assign each monomial m to one element that covers m. For each x_i, let λ_i be the set of monomials associated to an element of the cover, x_i. The algorithm recursively evaluates on each λ_i, with smaller error, ε/c, and returns their disjunction. We choose ε/c so that our result is an ε approximate lineage.

THEOREM 3.2 (SOLUTION TO PROB. 1). *For any $\varepsilon > 0$, Alg. 3.2.1 is a randomized algorithm that computes small ε-sufficient lineage with linear data complexity. Formally, the output of the algorithm, $\tilde{\lambda}_t$ satisfies two properties: (1) $\tilde{\lambda}_t^S$ is an ε-approximation of λ_t and (2) the number of monomials in $\tilde{\lambda}_t$ is less than $k!\beta^{-\binom{k}{2}} \log^k(\frac{1}{\varepsilon}) + O\left(\log^{k-1}(\frac{1}{\delta})\right)$, which is independent of the size of λ_t.*

PROOF SKETCH. The running time follows immediately, since no monomial is replicated and the depth of the recursion at most k, the running time is $O(k|\lambda_t|)$. The algorithm is randomized because we need to evaluate $\mathbf{E}[\lambda_t]$. Claim (1) follows from the preceding algorithm description.

To prove claim (2), we inspect the algorithm. In Case I, the maximum size of a matching is upper bounded by $\beta^{-k} \log(\frac{1}{\varepsilon})$ since a matching of size m in a k-dnf has probability at least $1-(1-\beta^k)^m$; if this value is greater than $1-\varepsilon$, we can trim terms; combining this inequality and that $1 - x \leq e^{-x}$ for $x \geq 0$, completes Case I. In Case II, the size of the cover c satisfies $c \leq k\beta^{-k} \log(\frac{1}{\varepsilon})$. If we let $S(k+1, \varepsilon)$ denote the size of our formula at depth $k+1$ with parameter ε, then it satisfies the recurrence $S(k+1, \varepsilon) = (k+1)\beta^{-(k+1)} \log(\frac{1}{\varepsilon}) \cdot S(k, \varepsilon/c)$, which grows no faster than the claimed formula. □

Completeness Our goal is to construct lineage that is small as possible; one may wonder if we can efficiently produce substantially smaller lineage with a different algorithm. We give evidence that no such algorithm exists by showing that the key step in Alg. 3.2.1 is intractable ($\mathcal{NP}$-hard) even if we restrict to internal lineage functions with 3 subgoals, that is $k = 3$. This justifies our use of a greedy heuristic above.

PROPOSITION 3.3. *Given a k-mDNF formula λ_t, finding a subformula $\tilde{\lambda}_t^S$ with d monomials such that $\tilde{\lambda}_t^S$ has largest probability among all subformula of λ_t is $\mathcal{NP}$-Hard, even if $k = 3$.*

The reduction is from of finding a matching in a k-uniform k-regular hypergraph. The greedy algorithm is essentially an optimal approximation for this hypergraph matching [33]. Since our problem appears to be more difficult, this suggests – but does not prove – that our greedy algorithm may be close to optimal.

3.3 Understanding Sufficient Lineage

Both Prob. 2, finding sufficient explanations, and Prob. 3, finding influential variables deal with understanding the lineage functions: Our proposal for sufficient lineage makes Prob. 2 straightforward: Since λ_t^S is a list of sufficient explanations, we simply return the highest ranked explanations contained in $\tilde{\lambda}_t^S$. As a result, we focus on computing the influence of a variable given only sufficient lineage. The main result is that we can compute influence with only a small error using sufficient lineage. We do not discuss finding the top-k efficiently; for which we can use prior art, *e.g.* [46]. We restate the definition of influence in a computationally friendly form (Prop. 3.4) and then prove bounds on the error of our approach.

PROPOSITION 3.4. *Let x_i be an atom with probability $p(x_i)$ and $\sigma_i^2 = p(x_i)(1-p(x_i))$. If λ_t is a monotone lineage formula:*

$$\text{Inf}_{x_i}(\lambda_t) = \sigma_i^{-2}\mathbf{E}[\lambda_t(x_i - p(x_i))]$$

The use of Prop. 3.4 is that to show that we can compute influence from sufficient lineage with small error:

PROPOSITION 3.5. *Let $\tilde{\lambda}_t^S$ be a sufficient ε-approximation of λ_t, then for any $x_i \in \mathcal{A}$ s.t. $p(x_i) \in (0,1)$, we have the following pair of inequalities*

$$\text{Inf}_{x_i}(\tilde{\lambda}_t^S) - \frac{\varepsilon}{\sigma^2}p(x_i) \leq \text{Inf}_{x_i}(\lambda_t) \leq \text{Inf}_{x_i}(\tilde{\lambda}_t^S) + \frac{\varepsilon}{\sigma^2}(1-p(x_i))$$

This proposition basically says that we can calculate the influence for *uncertain* atoms. With naïve random sampling, we can estimate the influence of sufficient lineage to essentially any desired precision. The number of relevant variables in sufficient lineage is small, so simply evaluating the influence of each variable and sorting is an efficient solution to solve Prob. 3.

3.4 Query Processing

Existing systems such as Mystiq or Trio can directly process sufficient lineage since it is syntactically identical to standard (complete) lineage. However, using sufficient lineage in place of complete lineage introduces errors during query processing. In this section, we show that the error introduced by query processing is at most a constant factor worse than the error in a single sufficient lineage formula.

Processing a query q on a database with lineage boils down to building a lineage expression for q by combining the lineage functions of individual tuples, *i.e. intensional evaluation* [22, 46]. For example, a join producing a tuple t from t_1 and t_2 produces lineage for t, $\lambda_t = \lambda_{t_1} \wedge \lambda_{t_2}$. We first prove that the error in processing a query q is upper bounded by the number of lineage functions combined by q (Prop. 3.6). Naïvely applied, this observation would show that the error grows with the size of the data. However, we observe that the lineage function for a conjunctive query depends on at most constantly many variables; from these two observations it follows that the query processing error is only a constant factor worse.

PROPOSITION 3.6. *If $\tilde{\lambda}_1^S$ and $\tilde{\lambda}_2^S$ are sufficient ε approximations for $\tilde{\lambda}_1$ and $\tilde{\lambda}_2$ then, both $\tilde{\lambda}_1^S \wedge \tilde{\lambda}_2^S$ and $\tilde{\lambda}_1^S \vee \tilde{\lambda}_2^S$ are 2ε sufficient approximations.*

This proposition is essentially an application of a union bound [42]. From this proposition and the fact that a query q that produces n tuples and has k subgoals has kn logical operations, we can conclude that if all lineage functions are ε_S approximations, then $\mu(q) - \tilde{\mu}^S(q) \leq \varepsilon_S kn$. This bound depends on the size of the data. We want to avoid this, because it implies that to answer queries as the data grows, we would need to continually refine the lineage. We can do much better using essentially the same idea as in Sec. 3.2:

LEMMA 3.7. *Given a database with sufficient approximate lineage such that $\tilde{\lambda}_t$ is a ε-approximation of λ_t for every tuple t and a query q with k subgoals, the error of q is only a constant factor worse than ε. Formally, for any $\delta > 0$ we have:*

$$\mu(q) - \tilde{\mu}^S(q) \leq k\varepsilon f(k,\delta) + \delta$$

where $f(k,\delta) \stackrel{\text{def}}{=} k!\beta^{-\binom{k}{2}}\log^k(\frac{1}{\delta}) + O\left(\log^{k-1}(\frac{1}{\varepsilon})\right)$ and β is the constant of the bounded distribution.

This shows that sufficient lineage can be effectively utilized for query processing, solving Prob. 4.

4. POLYNOMIAL LINEAGE

In this section, we propose an instantiation of polynomial lineage based on sparse low-total-degree polynomial series. We focus on the problems of constructing lineage and understanding lineage, since there are existing approaches, *e.g.* [8], that solve the problem of sampling from lineage, which is sufficient to solve the query evaluation problem (Prob. 4).

4.1 Sparse Fourier Series

Our goal is to write a Boolean function as a sum of smaller terms; this decomposition is similar to Taylor and Fourier series decompositions in basic calculus. We recall the basics of Fourier Series on the Boolean Hypercube[1].

In our discussion, we fix a set of independent random variables $x_1, \ldots, x_n$, *e.g.* the atoms, where $p_i = \mathbf{E}[x_i]$ (the expectation) and $\sigma_i^2 = p_i(1-p_i)$ (the variance). Let $\mathcal{B}$ be the vector space of real-valued Boolean functions on n variables; a vector in this space is a function $\lambda : \{0,1\}^n \to \mathbb{R}$. Rather than the standard basis for $\mathcal{B}$, we define the Fourier basis for functions. To do so we equip $\mathcal{B}$ with an inner product that is defined via expectation, that is, $\langle \lambda_1, \lambda_2 \rangle \stackrel{\text{def}}{=} \mathbf{E}[\lambda_1 \cdot \lambda_2]$. This inner product induces a norm, $\|\lambda_t\|^2 \stackrel{\text{def}}{=} \langle \lambda_t, \lambda_t \rangle$. This norm captures our error function (see Def. 2.10) since $\mathbf{E}[(\lambda_t - \tilde{\lambda}_t^P)^2] = \left\|\tilde{\lambda}_t - \tilde{\lambda}_t^P\right\|^2$. We can now define an orthonormal basis for the vector space using the set of *characters*:

DEFINITION 4.1. *For each $\boldsymbol{z} \in \{0,1\}^n$, the **character** associated with $\boldsymbol{z}$ is a function from $\{0,1\}^n \to \mathbb{R}$ denoted $\phi_{\boldsymbol{z}}$ and defined as:*

$$\phi_{\boldsymbol{z}} \stackrel{\text{def}}{=} \prod_{i:z_i=1}(x_i - p_i)\sigma_i^{-1}$$

Since the set of all characters is an orthonormal basis, we can write any function in $\mathcal{B}$ as a sum of the characters. The coefficient of a character is given by projection on to that character, as we define below.

DEFINITION 4.2. *The **Fourier transform** of a function λ_t is denoted $\mathcal{F}_{\lambda_t}$ and is a function from $\{0,1\}^n \to \mathbb{R}$ defined as:*

$$\mathcal{F}_{\lambda_t}(\boldsymbol{z}) \stackrel{\text{def}}{=} \langle \lambda_t, \phi_{\boldsymbol{z}} \rangle = \mathbf{E}[\lambda_t \phi_{\boldsymbol{z}}]$$

*The **Fourier series** of f is defined as $\sum_{\boldsymbol{z}\in\{0,1\}^n} \mathcal{F}_{\lambda_t}(\boldsymbol{z})\phi_{\boldsymbol{z}}(A)$.*

The Fourier series captures λ_t, that is, for any assignment A, $f(A) = \sum_{\boldsymbol{z}\in\{0,1\}^n} \mathcal{F}_{\lambda_t}(\boldsymbol{z})\phi_{\boldsymbol{z}}(A)$. An important coefficient is $\mathcal{F}_{\lambda_t}(\boldsymbol{0})$, which is the probability (expectation) of λ_t. We give an example of to illustrate the computation of Fourier series:

Example 4.3 Let $\lambda_t = x_1 \vee \cdots \vee x_n$, that is, the logical or of independent n random variables. The arithmetization for λ_t is $1 - \prod_{i=1,\ldots,n}(1-x_i)$. Applying Def. 4.2, $\mathcal{F}_{\lambda_t}(\boldsymbol{0}) = \mathbf{E}[\lambda_t] = 1 - \prod_{i=1,\ldots,n}(1-p(x_i))$ and for $\boldsymbol{z} \neq \boldsymbol{0}$:

$$\begin{aligned}
\mathcal{F}_{\lambda_t}(\boldsymbol{z}) &= \mathbf{E}\left[\phi_{\boldsymbol{z}}\left(1 - \textstyle\prod_{i=1,\ldots,n}(1-x_i)\right)\right] \\
&= \mathbf{E}\left[\phi_{\boldsymbol{z}} - \left(\textstyle\prod_{i:z_i=1}\phi_{e_i}(1-x_i)\right)\left(\textstyle\prod_{j:z_j=0}(1-x_j)\right)\right] \\
&= \left(\textstyle\prod_{i:z_i=1}\sigma_i\right)\left(\textstyle\prod_{j:z_j=0}(1-p(x_j))\right)
\end{aligned}$$

where for $i = 1, \ldots, n$, $\sigma_i^2 = p(x_i)(1-p(x_i))$ (the variance of x_i).

Our goal is to get a small, but good approximation; we make this goal precise using sparse Fourier series:

[1]For more details, see [40, 43]

DEFINITION 4.4. *An s-**sparse series** is a Fourier series with at most s non-zero coefficients. We say λ has an (s, ε) approximation if there exists an s-sparse approximation $\tilde{\lambda}_t^P$ such that $\left\|\lambda_t - \tilde{\lambda}_t^P\right\|^2 \leq \varepsilon$. A **best** s-sparse series for a function λ is the s-sparse series that minimizes ε.*

Our approach for polynomial lineage is to approximate the lineage for a tuple t, λ_t, by a sparse Fourier series $\tilde{\lambda}_t^P$, ideally an (s, ε)-sparse approximation for small s and ε. Additionally, we want $\tilde{\lambda}_t^P$ to have low total degree (constant) so we can describe its coefficients succinctly (in constant space).

Selecting an approximation The standard approach to approximation using series is to keep only the largest coefficients, which is optimal in this case:

PROPOSITION 4.5. *For any Boolean function λ_t and any $s > 0$, a best s-spare approximation for λ_t is the s largest coefficients in absolute value, ties broken arbitrarily.*

4.2 Constructing Lineage

We construct polynomial lineage by searching for the largest coefficients using the KM algorithm [39]. The KM algorithm is complete in the sense that if there is an (s, ε) sparse approximation it finds an only slightly worse $(s, \varepsilon + \varepsilon^2/s)$ approximation. The key technical insight, is that k-DNFs do have sparse (and low-total-degree) Fourier series, [40, 50]. This implies we only need to keep around a relatively few coefficients to get a good approximation. More precisely,

THEOREM 4.6 ([39, 40, 50]). *Given a set of atoms $\mathcal{A} = \{x_1, \ldots, x_n\}$ and a probabilistic assignment p, let $\beta = \min_{i=1,\ldots,n}\{p(x_i), 1 - p(x_i)\}$ and λ_t be a (not necessarily monotone) k-DNF function over $\mathcal{A}$, then there exists an (s, ε)-approximation $\tilde{\lambda}_t^P$ where $s \leq k^{O(k\beta^{-1}\log(\frac{1}{\varepsilon}))}$ and the total degree of any term in $\tilde{\lambda}_t^P$ is bounded by $c_0\beta^{-1}k\log(\frac{1}{\varepsilon})$ where c_0 is a constant. Further, we can construct $\tilde{\lambda}_t^P$ in randomized polynomial time.*

The KM algorithm is an elegant recursive search algorithm. However, a key practical detail is at each step it requires that we use a two-level estimator, that is, the algorithm requires that at each step, we estimate a quantity y_1 via sampling; to compute each sample of y_1, we must, in turn, estimate a second quantity y_2 via sampling. This can be very slow in practice. This motivates us to purpose a cheaper heuristic: For each monomial m, we estimate the coefficient corresponding to each subset of variables of m. For example, if $m = x_1 \wedge x_2$, then we estimate $\mathbf{0}$, $\boldsymbol{e}_1$, $\boldsymbol{e}_2$ and $\boldsymbol{e}_{12}$. This heuristic takes time $2^k|\lambda_t|$, but can be orders of magnitude more efficient in practice, as we show in our evaluation section (Sec. 5.2). This is linear with respect to data complexity.

4.3 Understanding Approximate Lineage

Our goal in this section is to find sufficient explanations and influential variables, solving Prob. 2 and Prob. 3, respectively.

Sufficient Explanations Let λ_t be a lineage formula such that $\mathbf{E}[\lambda_t] \in (0, 1)$ and $\tilde{\lambda}_t^P$ be a polynomial approximation of λ_t. Given a monomial m, our goal is to test if m is a sufficient explanation for λ_t. The key idea is that m is a sufficient explanation if and only if $\mathbf{P}[\lambda_t \wedge m] = \mathbf{P}[m]$, since this implies the implication holds for every assignment.

If $\tilde{\lambda}_t^P$ is exactly the Fourier series for λ_t, then we can compute each value in time $O(2^k)$, since

$$\mathbf{E}[\tilde{\lambda}_t^P m] = \sum_{\mathbf{Z}: z_i = 1 \implies i \in m} \mathcal{F}_{\lambda_t}(\boldsymbol{z}) \left(\prod_{i \in m: \mathbf{Z}_i = 1} \sigma\right)\left(\prod_{j \in m: \mathbf{Z}_j = 0} \mu_j\right) \quad (5)$$

However, often λ_t is complicated, which forces us to use sampling to approximate the coefficients of $\tilde{\lambda}_t^P$. Sampling introduces noise in the coefficients. To tolerate noise, we relax our test:

DEFINITION 4.7. *Let $\tau > 0$, the tolerance, and $\delta > 0$, the confidence, then we say that a monomial m is a (τ, δ) **sufficient explanation** for $\tilde{\lambda}_t^P$ if:*

$$\mathbf{P}_{\mathcal{N}}[|\mathbf{E}[\underbrace{\tilde{\lambda}_t^P \cdot m}_{(\dagger)}] - \mathbf{E}[m]| \leq \tau] \geq 1 - \delta \quad (6)$$

where $\mathcal{N}$ denotes the distribution of the sampling noise.

The intuition is that we want that $\mathbf{E}[\tilde{\lambda}_t^P m]$ and $\mathbf{E}[m]$ to be close with high probability. For independent random sampling, the $\mathcal{N}$ is a set of normally distributed random variables, one for each coefficient. Substituting Eq. 5 into Eq. 6 shows that (†) is a sum of 2^k normal variables, which is again normal; we use this fact to estimate the probability that (†) is less than τ.

Our heuristic is straightforward, given a tolerance τ and a confidence δ: For each monomial m, compute the probability in Eq. 6, if it is within δ then declare m a sufficient explanation. Finally, rank each sufficient explanation by the probability of that monomial.

Influential tuples The key observation is that the influence of x_i is determined by its coefficient in the expansion [40, 50]:

PROPOSITION 4.8. *Let λ_t be an internal lineage function, x_i an atom and $\sigma_i^2 = p(x_i)(1 - p(x_i))$ then*

$$\mathrm{Inf}_{x_i}(\lambda_t) = \sigma_i^{-1}\mathcal{F}_{\lambda_t}(e_i)$$

This gives us a simple algorithm for finding influential tuples using polynomial lineage, simply scale each $\mathcal{F}_{\lambda_t}(e_i)$, sort them and return them. Further, the term corresponding to e_i in the transform is $\mathcal{F}_{\lambda_t}(e_i)\phi_{e_i} = \mathrm{Inf}_{x_i}(\lambda_t)(x_i - p(x_i))$, as was shown in Fig. 1.

5. EXPERIMENTS

In this section, we answer three main questions about our approach: (1) In Sec. 5.2, do our lineage approaches compress the data? (2) In Sec. 5.3, to what extent can we recover explanations from the compressed data? (3) In Sec. 5.4, does the compressed data provide a performance improvement while returning high quality answers? To answer these questions, we experimented with the Gene Ontology database [14] (GO) and similarity scores from a movie matching database [35, 46].

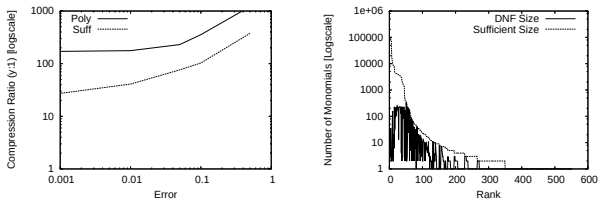
5.1 Experimental Details

Primary Dataset The primary dataset is GO, that we described in the introduction. We assigned probability scores to evidence tuples based on the type of evidence. For example, we assigned a high reliability score (0.9) to a statement in a PubMed article, while we assigned a low score (0.1) to an automated similarity match. Although many atoms are assigned the same score, they are treated as independent events. Additionally, to test the performance of our algorithms, we generated several probability values that were obtained from more highly skewed distributions, that are discussed in the relevant sections.

Primary Views We present four views which are taken from the examples and view definitions that accompany the GO database [14]. The first view `V1` asks for all evidence associated with a fixed pair of gene products. `V2` looks for all terms associated with a fixed gene product. `V3` is a view of all annotations associated with the Drosophila fly (via FlyBase [21]). `V4` is a large view of all gene products and associated terms. Fig. 2 summarizes the relevant parameters for each view: (1) the number of tables in the view definition (2) the number of sources evidence, that is, how many times it

Query	Tables	# Evidence	# Tuples	Avg. Lin. Size	Size
V1	8	2	1	234	12k
V2	6	2	1119	1211	141M
V3	6	1	295K	3.36	104M
V4	7	1	28M	7.68	31G

Figure 2: Query statistics for the GO DB [14].



(a) (b)

Figure 3: (a) Compression ratio as error increases in log scale for query V2. (b) Distribution of size of DNF for V2 with and without compression, x-axis is sorted by size, *e.g.* $x = 1$ is the largest DNF (823k).

joins with the evidence table (3) the number of tuples returned (4) the average of the lineage sizes for each tuple, and (5) the storage size of the result.

Secondary Dataset To verify that our results apply more generally than the GO database, we examined a database that (fuzzily) integrated movie reviews from Amazon [2] that have been integrated with IMDB (the Internet Movie Database) [35]. This data has two sources of imprecision: matches of titles between IMDB and Amazon, ratings assigned to each movie by automatic sentiment analysis, that is, a classifier.

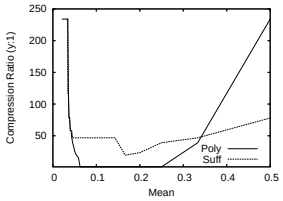
Experimental Setup All experiments were run on a Fedora core Linux machine (2.6.23-14 SMP) with Dual Quad Core 2.66GHz 16Gb of RAM. Our prototype implementation of the compression algorithms was written in approximately 2000 lines of Caml. Query performance was done using a modified C++/caml version of the Mystiq engine[10] backed by databases running SQL Server 2005. The implementation was not heavily optimized.

5.2 Compression

We verify that our compression algorithms produce small approximate lineage, even for stringent error requirements. We measured the compression ratios and compression times achieved by our approaches for both datasets at varying errors.

Compression Ratios Fig. 3(a) shows the compression ratio versus error trade-off achieved by polynomial and sufficient lineage for V2. Specifically, for a fixed error on the x-axis the y axis shows the compression ratio of the lineage (in log scale). As the graph illustrates, in the best case, V2, the compression ratio for the polynomial lineage is very large. Specifically,even for extremely small error rates, 10^{-3}, the compressed ratio $171:1$ for polynomial lineage versus $27:1$ times smaller for sufficient lineage. In contrast, V3 is our worst case. The absolute maximum our methods can achieve is a ratio of $3.36:1$, which is the ratio we would get by keeping a single monomial for each tuple. At an error $\varepsilon = 0.01$, polynomial lineage achieves a $1.8:1$ ratio, while sufficient lineage betters this with a $2.1:1$ ratio.

The abundance of large lineage formula in V2 contain redundant information, which allows our algorithms to compress them efficiently. Fig. 3(b) shows the distribution of the size of the original lineage formulae and below it the size after compression. There are



(a)

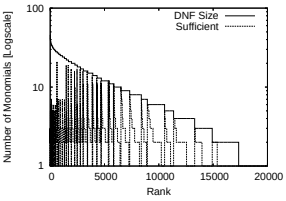
Q	Suff PT	Poly PT
V1	0.23s	0.50s
V2	3.5h	3.5h
V3	10.3m	24.3m
V4	50.3h	67.4h

(b)

Figure 4: (a) The compression ratio versus the mean of the distribution for V1. Sufficient is more stable, though the polynomial lineage can provide better approximation ratios. (b) The compression time for each view, the processing time (PT).

ε	Suff	Poly
0.1	4.0	4.5
0.05	2.7	2.6
0.01	1.4	1.5
0.001	1.07	1.3

(a)



(b)

Figure 5: (a) Compression Ratio ($\frac{|\text{Original}|}{|\text{Compressed}|}$) (b) The distribution of lineage size in IMDB view, by rank.

some very large sources in the real data; the largest one contains approximately 823k monomials. Since large DNFs have probabilities very close to one, polynomial lineage can achieve an ε-approximation can use the constant 1. In contrast, sufficient lineage cannot do this.

Effect of Skew We investigate the effect of skew, by altering the probabilistic assignment, that is, the probability we assigned to each atom. Specifically, we assigned an atom a score drawn from a skewed probability distribution. We then compressed V1 with the skewed probabilities. V1 contains only a single tuple with moderate sized lineage (234 monomials). Fig. 4(a) shows the compression ratio as we vary the skew from small means, 0.02, to larger means, 0.5. More formally, the probability we assign to an atom is drawn from a Beta distribution with $\beta = 1$ and α taking the value on the x axis. Sufficient lineage provides lower compression ratios for extreme means, that is close to 0.02 and 0.5, but is more consistent in the less extreme cases.

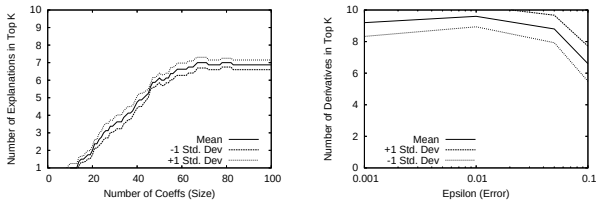
Compression Time Fig. 4(b) shows the processing time for each view we consider. For views V2, V3 and V4, we used 4 dual-core CPUs and 8 processes simultaneously. The actual end-to-end running times are about a factor of 8 faster, *e.g.*, V2 took less than 30m to compress. It is interesting to to note that the processor time for V2 is much larger than the comparably sized V3, the reason is that the complexity of our algorithm grows non-linearly with the largest DNF size. Specifically, the increase is due to the cost of sampling.

The compression times for polynomial lineage and sufficient lineage are close; this is only true because we are using the heuristic of Sec. 4.2. The generic algorithm is orders of magnitude slower: It could not compress V1 in an hour, compared to only 0.5s using the heuristic approach. Our implementation of the generic search algorithm could be improved, but it would require orders of magnitude improvement to compete with the efficiency the simple heuristic.

IMDB and Amazon dataset Using the IMDB movie data, we compressed a view of highly rated movies. Fig. 5(a) shows the

compression ratio for versus error rate. Even for stringent error requirements, our approach is able to obtain good compression ratios for both instantiations of approximate lineage. Fig. 5(b) shows the distribution of the lineage size, sorted by rank, and its sufficient compression size. Compared to Fig. 3, there are relatively few large lineage formulae, which means there is less much opportunity for compression. On a single CPU, the time taken to compress the data was always between 180 and 210s. This confirms that our results our more general than a single dataset.

5.3 Explanations



(a) (b)

Figure 6: (a) Shows the precision of the top-k explanations versus the number of terms in the polynomial expansion (c) The number (precision) of influential variables in the top 10 **returned using sufficient lineage that are in the top** 10 **of the uncompressed function.**

We assess how well approximate lineage can solve the explanation tasks in practice, that is finding sufficient explanations (Prob. 2) and finding influential variables (Prob. 3). Specifically, we answer two questions: (1) How well can sufficient lineage compute influential variables? (2) How well can polynomial lineage generate sufficient explanations?

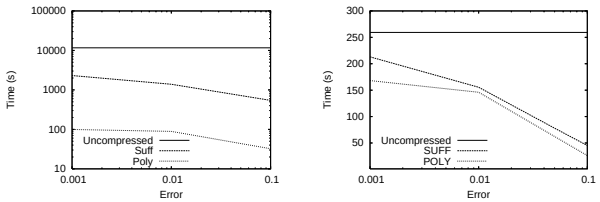
To answer question (1), we created 10 randomly generated probabilistic assignment for the atoms in `V1`; we ensured that the resulting lineage formula had non-trivial reliability, *i.e.* , in $(0.1, 0.9)$. We then tested precision: Out of the top 10 influential variables, how many were returned in the top 10 using sufficient lineage (Sec. 3.3)? Fig. 6(b) shows that for high error rates, $\varepsilon = 0.1$, we still are able to recover 6 of the top 10 influential variables and for lower error rates, $\varepsilon = 0.01$, we do even better: the average number of recovered top 10 values is 9.6. The precision trails-off for very small error rates due to small swaps in rankings near the bottom of the top 10, *e.g.*, all top 5 are within the top 10.

To answer question (2), we used the same randomly generated probabilistic assignments for the atoms in `V1` as in the answer to question (1). Fig. 6(a) shows the average number of terms in the top k explanations returned by the method of Sec. 4.3 that are actual sufficient explanations versus the number of terms retained by the formula. We have an average recall of approximately 0.7 (with low standard deviation), while keeping only a few coefficients. Here, we are using the heuristic construction of polynomial lineage. Thus, this experiment should be viewed as a lower bound on the quality of using polynomial lineage for providing explanations.

These two experiments confirm that both sufficient and polynomial lineage are able to provide high quality explanations of the data directly on the compressed data.

5.4 Query Performance

Fig. 7 shows the effect of compression on execution time of `V2`; The query asks to compute each tuple in the view. The y-axis is



(a) (b)

Figure 7: Query performance on (a) `V2`**. (b) IMDB data.**

in log scale, it takes just under 20 hours to run this query on the uncompressed data. On data compressed with sufficient lineage at $\varepsilon = 0.001$, we get an order of magnitude improvement; the query takes approximately 35m to execute. Using the data compressed with polynomial lineage, we get an additional order of magnitude; the query now runs in 1.5m.

Fig. 7(b) shows the effect of compression on query performance for the IMDB movie dataset where the compression was not as dramatic. Again our query was to compute the lineage for each tuple in the view. The time taking is to perform Monte Carlo sampling on the now much smaller query. As expected, the data with higher error, and so smaller, allows up to a five time performance gain. In this example both running times scale approximately with the size of compression.

6. RELATED WORK

Lineage systems and provenance are important topics in data management, [12, 13, 17, 28]. Compressing lineage is cited as an important techinque to scaling these systems [13]. Of these, only [30] considers probabilistic data, but not approximate semantics.

There is long, successful line of work that compresses (deterministic) data to speed up query processing [18, 23, 25, 51, 53]. In wavelet approaches, probabilistic techniques are used to achieve a higher quality synopses, [18]. In contrast, lineage in our setting contains probabilities, which must be captured. The fact that the lineage is probabilistic raises the complexity of compression. For example, the approach of Garofalakis *et al.* [23] assumes that the entire wavelet transform can be computed efficiently. In our work, the transform size is exponential in the size of the data. Probabilistic query evaluation can be reduced to calculating a single coefficient of the transform, which implies exact computation of the transform is intractable [16, 27]. Aref *et al.* [19] advocate an approach to operate directly on compressed data to optimze queries on Biological sequences. However, this approach is not lineage aware and so cannot extract explanations from the compressed data.

In probabilistic databases, lineage is used for query processing in Mystiq [16, 46] and Trio [54]. However, neither considers approximate lineage. Ré *et al.* [46] consider approximately computing the probability of a query answer, but do not consider the problem of storing the lineage of a query answer. These techniques are orthogonal: We can use the techniques of [46] to compute the top-k query probabilities from the Level II database using sufficient lineage. Approximate lineage is used to materialize views of probabilistic data; this problem has been previously considered [47], but only with an exact semantics.

Sen *et al.* [49] consider approximate processing of relational queries using graphical models, but not approximate lineage. In the graphical model literature [15, 37] approximate representation is considered, where the goal is to compress the model for improved performance. However, the data and query models of the our ap-

proaches is different. Specifically, our approach leverages the fact that lineage is database is often *internal*.

Our approach to computing polynomial lineage is based on computational learning techniques, such as the seminal paper by Linial *et al.* [40], and others, [8, 11, 43]. A key ingredient underlying these results are *switching lemmata*, [5, 32, 48].

So far, learning techniques have only been applied to compressing the data, but have not compressed the lineage [4, 24]. A difference between our approach and this prior art is that we do not discard any tuples, but may discard lineage.

Explanation is a well-studied topic in the Artificial Intelligence community, see [31, 44]. The definition of explanation of a fact is a formula that is a minimal and sufficient to explain a fact – which is similar to our definition – but they additionally require that the formula be *unknown* to the user. We do not model the knowledge of users, but such a semantic would be very useful for scientists.

7. CONCLUSION

In this paper, we have proposed two instantiations of approximate lineage, a conservative approximation called sufficient lineage and a more aggressive approximation called polynomial lineage. The intuition behind both approaches is to keep track of only the most important explanations or correlations. We provided fundamental algorithms to create, explore and understand, and process approximate lineage. Our approach acheives high compression ratios, high-quality explanations and good query performance.

Acknowledgement This work was partially supported by an NSF graduate research fellowship, NSF Grants IIS-0454425, IIS-0513877, IIS-0627585, IIS-0713576, and Gifts from Microsoft including a gift under the SensorMap RFP.

8. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley Publishing Co, 1995.

[2] Amazon.com: `http://www.amazon.com/`.

[3] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *VLDB*, pages 586–597, 2002.

[4] S. Babu, M. Garofalakis, and R. Rastogi. Spartan: A model-based semantic compression system for massive data tables. In *SIGMOD*, pages 283–294, 2001.

[5] P. Beame. A switching lemma primer. Technical Report 95-07-01, University of Washington, Seattle, WA, 1995.

[6] O. Benjelloun, A. Das Sarma, A. Y. Halevy, M. Theobald, and J. Widom. Databases with uncertainty and lineage. *VLDB J.*, 17(2):243–264, 2008.

[7] O. Benjelloun, A. Das Sarma, C. Hayworth, and J. Widom. An introduction to ULDBs and the Trio system. *IEEE Data Eng. Bull*, 29(1):5–16, 2006.

[8] A. Blum, M. L. Furst, J C. Jackson, M J. Kearns, Y. Mansour, and S. Rudich. Weakly learning dnf and characterizing statistical query learning using fourier analysis. In *STOC*, pages 253–262, 1994.

[9] B. Boeckmann, A. Bairoch, R. Apweiler, M. C. Blatter, A. Estreicher, E. Gasteiger, M. J. Martin, K. Michoud, C. O'Donovan, I. Phan, S. Pilbout, and M. Schneider. The swiss-prot protein knowledgebase and its supplement trembl in 2003. *Nucleic Acids Res*, 31(1):365–370, January 2003.

[10] J. Boulos, N .Dalvi, B. Mandhani, S. Mathur, C. Re, and D. Suciu. Mystiq: A system for finding more answers using probabilities (demo). In *SIGMOD*, 2005.

[11] N. Bshouty and C. Tamon. On the fourier spectrum of monotone functions. *J. ACM*, 43(4):747–770, 1996.

[12] P. Buneman, A. Chapman, and J. Cheney. Provenance management in curated databases. In *SIGMOD*, pages 539–550, 2006.

[13] A. Chapman and H. V. Jagadish. Issues in building practical provenance systems. *IEEE Data Eng. Bull.*, 30(4):38–43, 2007.

[14] The Gene Ontology Consortium. Gene ontology: tool for the unification of biology. In *Nature Genet.*, pages 25–29, (2000).

[15] R. G. Cowell, S. L. Lauritzen, A. P. David, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[16] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, Toronto, Canada, 2004.

[17] Special issue on data provenance. *IEEE Data Eng. Bull.*, 30(4), 2007.

[18] A. Deligiannakis, M. Garofalakis, and N. Roussopoulos. Extended wavelets for multiple measures. *ACM Trans. Database Syst.*, 32(2):10, 2007.

[19] M. Eltabakh, M. Ouzzani, and W. G. Aref. bdbms - a database management system for biological data. In *CIDR*, pages 196–206, 2007.

[20] R. Fagin, J. Y. Halpern, and N. Megiddo. A logic for reasoning about probabilities. *Information and Computation*, 87(1/2):78–128, 1990.

[21] `http://flybase.bio.indiana.edu/`.

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

[23] M. Garofalakis and P. Gibbons. Probabilistic wavelet synopses. *ACM Trans. Database Syst.*, 29:43–90, 2004.

[24] L. Getoor, B. Taskar, and D. Koller. Selectivity estimation using probabilistic models. In *SIGMOD*, pages 461–472, 2001.

[25] J. Goldstein, R. Ramakrishnan, and U. Shaft. Compressing relations and indexes. In *ICDE*, pages 370–379, 1998.

[26] S.M. Gorski, S. Chittaranjan, E.D. Pleasance, J.D. Freeman, C.L. Anderson, R.J. Varhol, S.M. Coughlin, S.D. Zuyderduyn, S.J. Jones, and M.A. Marra. A SAGE approach to discovery of genes involved in autophagic cell death. *Curr. Biol.*, 13:358–363, Feb 2003.

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

[28] T. Green, G. Karvounarakis, N. E. Taylor, O. Biton, Z. G. Ives, and V. Tannen. Orchestra: facilitating collaborative data sharing. In *SIGMOD*, pages 1131–1133, 2007.

[29] T. Green and V. Tannen. Models for incomplete and probabilistic information. *IEEE Data Engineering Bulletin*, 29(1):17–24, March 2006.

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

[31] J. Halpern and J. Pearl. Causes and explanations: A structural-model approach - part II: Explanations. In *IJCAI*, pages 27–34, 2001.

[32] J. Håstad. *Computational limitations for small depth circuits*. M.I.T Press, Cambridge, Massachusetts, 1986.

[33] E. Hazan, S. Safra, and O. Schwartz. On the hardness of approximating k-dimensional matching. *ECCC*, 10(020), 2003.

[34] M. Hernandez and S. Stolfo. The merge/purge problem for large databases. In *SIGMOD*, pages 127–138, 1995.

[35] Internet movie database: *http://www.imdb.com/*.

[36] T. Imielinski and W. Lipski. Incomplete information in relational databases. *Journal of the ACM*, 31:761–791, October 1984.

[37] M. Jordan, Z. Ghahramani, T. Jaakkola, and L. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, 1999.

[38] R. Karp and M. Luby. Monte-Carlo algorithms for enumeration and reliability problems. In *Proceedings of the annual ACM symposium on Theory of computing*, 1983.

[39] E. Kushilevitz and Y. Mansour. Learning decision trees using the fourier spectrum. *SIAM J. Comput.*, 22(6):1331–1348, 1993.

[40] N. Linial, Y. Mansour, and N. Nisan. Constant depth circuits, fourier transform, and learnability. *J. ACM*, 40(3):607–620, 1993.

[41] G. Miklau and D. Suciu. A formal analysis of information disclosure in data exchange. *J. Comput. System Sci.*, 73(3):507–534, 2007.

[42] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, August 1995.

[43] R. W. O'Donnell. *Computational Applications of Noise Sensitivity*. PhD thesis, M.I.T., 2003.

[44] J. Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, Cambridge, UK, 2000.

[45] `http://www.pubmed.gov`.

[46] C. Ré, N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, 2007.

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

[48] N. Segerlind, S. R. Buss, and R. Impagliazzo. A switching lemma for small restrictions and lower bounds for k-dnf resolution. *SIAM J. Comput.*, 33(5):1171–1200, 2004.

[49] P. Sen and A. Deshpande. Representing and querying correlated tuples in probabilistic databases. In *ICDE*, 2007.

[50] R. Servedio. On learning monotone dnf under product distributions. *Inf. Comput.*, 193(1):57–74, 2004.

[51] M. Stonebraker, D. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, P. O'Neil, A. Rasin, N. Tran, and S. Zdonik. C-store: A column-oriented dbms. In *VLDB*, pages 553–564, 2005.

[52] Go database v. go200801.

[53] J. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *SIGMOD*, pages 193–204, 1999.

[54] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, pages 262–276, 2005.