Provenance and Probabilities in Relational Databases: From Theory to Practice

Pierre Senellart
DI ENS, ENS, CNRS, PSL Research University
& Inria Paris
& LTCI, Télécom ParisTech
Paris, France
pierre@senellart.com

ABSTRACT

We review the basics of data provenance in relational databases. We describe different provenance formalisms, from Boolean provenance to provenance semirings and beyond, that can be used for a wide variety of purposes, to obtain additional information on the output of a query. We discuss representation systems for data provenance, circuits in particular, with a focus on practical implementation. Finally, we explain how provenance is practically used for probabilistic query evaluation in probabilistic databases.

1. INTRODUCTION

The central task in data management is *query evaluation*: given a database instance and a query, compute the results of that query on that instance. But what if we want something more than just the query result? We might want to know:

- *why* this specific result was obtained;
- *where* values in the result come from;
- *how* the result was produced from the query;
- how the result would change if some of the input *tuples* were *missing*;
- *how many times* each query result was obtained;
- what *probability* the result has, given a probability distribution on the input data;
- what *minimal security clearance* is needed to see the result, given some security information on the input data;
- what the *most economical way* to obtain this result is, in terms of number of data accesses.

All these questions, and more, can be answered using the tool of *data provenance* [11,12], which is some additional bookkeeping information maintained during query evaluation, that allows answering a large number of such meta-questions on the output of a query. The precise nature and form this provenance information should take depends on the question we want to answer, and on the query language we consider.

We focus in this paper on the setting of relational databases, though provenance and its applications apply as well and are equally important in other settings, such as scientific workflows [19], knowledge graphs [30], or hierarchical data [11].

There has been a large amount of work on the foundations of data provenance in relational database systems: early definitions and implementations of data provenance (called *lineage* at the time) for specific applications [13,48]; the influential framework of where- and why-provenance introduced by Buneman, Khanna, and Tan [11]; the seminal paper on provenance semirings [27] by Green, Karvounarakis, and Tannen; and further extensions thereof [7, 8, 22, 24]. Provenance has also been a particularly useful tool in the area of probabilistic databases [3, 23, 32], where the use of provenance is dubbed the *intensional approach* to probabilistic query evaluation.

The goal of this paper is to review some of the most important definitions of provenance, unifying them in a single framework as much as possible, and to address some of the concrete issues that arise in building a modern provenance-aware database management, and in implementing the intensional approach to probabilistic query evaluation.

Example 1. We will use throughout this paper a very simple running example of a single-table database, in Table 1, that contains information on the personal of a (fictitious) intelligence agency, to illustrate various aspects of data provenance. Note in Table 1 the t_i annotation on each tuple. These will be used in the following as *provenance tokens* associated with the tuple, which means that they will contain the elementary provenance information associated to the tuple. You can think of them for now as simple tuple identifiers.

We will consider the following two example queries

Table 1: Table Personal for the personal of an intelligence agency, used as a running example

id	name	position	city	classification	
1	John	Director	New York	unclassified	t_1
2	Paul	Janitor	New York	restricted	t_2
3	Dave	Analyst	Paris	confidential	t_3
4	Ellen	Field agent	Berlin	secret	t_4
5	Magdalen	Double agent	Paris	top_secret	t_5
6	Nancy	HR	Paris	restricted	t_6
7	Susan	Analyst	Berlin	secret	t_7

on this database. The first query, Q_1, asks for the cities referenced in the Personal table which host at least two different employees of the agency:

```
SELECT DISTINCT P1.city
FROM Personal P1, Personal P2
WHERE P1.city = P2.city AND P1.id < P2.id
```

Obviously, the answer to this query, disregarding provenance, is a single-attribute table, containing the three cities New York, Paris, and Berlin. Our second query, Q_2, asks for the cities that host exactly one employee of the agency:

```
  SELECT DISTINCT city FROM Personal
EXCEPT
  SELECT DISTINCT P1.city
  FROM Personal P1, Personal P2
  WHERE P1.city = P2.city AND P1.id < P2.id
```

Its output, once again disregarding provenance, is the empty table. □

The paper is organized as follows: in Section 2, we first introduce the simple setting of *Boolean provenance* that is in particular used in probabilistic databases and has the advantage of being definable for an arbitrary query language. We move in Section 3 to *semiring provenance*, which captures more information than Boolean provenance but is defined for a specific, monotone, query language. We explore formalisms that go beyond semiring provenance in Section 4. In Section 5, we address the concrete problem of which formalism to use to represent data provenance, and settle on circuits as compact formalism. We explain in Section 6 how probabilistic query evaluation in probabilistic databases can be solved using provenance, and which tools can be used to do this efficiently.

2. BOOLEAN PROVENANCE

Boolean provenance is one of the simplest forms of provenance, while having a major conceptual advantage: it can be defined independently of a specific query language. The notion of Boolean provenance is implicit in the work of Imieliński and Lipski on conditional tables [31], although this predates the notion of provenance itself and the query language was limited to the relational algebra. The term *Boolean provenance* was introduced specifically in the setting of provenance semirings [26], though this was restricted to the monotone case. The generalization to non-monotone queries we give below is straightforward, and was made, e.g., in [3].

We fix a finite set $X = \{ x_1, \ldots, x_n \}$, the elements of which we call *Boolean events* (i.e., variables that can be either $\top$ or $\bot$).

As in [31], we let provenance tokens (the annotations attached to tuples of the input databases) be Boolean functions over X, that is, functions of the form $\varphi : (X \to \{\top, \bot\}) \to \{\top, \bot\}$. They are interpreted under a *possible-world semantics*: every valuation $\nu : X \to \{\top, \bot\}$ denotes a *possible world* of the database; in this possible world, a tuple with annotation φ exists if and only if $\varphi(\nu) = \top$. For a given database D, we denote $\nu(D)$ the set of tuples t with annotation φ_t such that $\varphi_t(\nu) = \top$. It is a subset of the database D.

Example 2. Consider again Table 1 and assume that, for each i, t_i is the indicator function of a distinct Boolean event x_i, i.e., the function that maps a valuation ν to $\top$ if and only if $\nu(x_i) = \top$. Then the set of possible worlds of Personal is the set of all subrelations of Personal. For instance, if ν maps x_1, x_3, x_5, and x_7 to $\top$ and all other variables to $\bot$, then ν(Personal) is the subrelation of Personal where only the tuples of id 1, 3, 5, and 7 survive. □

Let Q be an arbitrary query, i.e., a function that takes as input a finite relational database over a fixed database schema, and produces as output a finite relation over a fixed relational schema. Then the *Boolean provenance* of Q over a database D, denoted $\text{prov}_{Q,D}$, is a function that maps a tuple t of the

output relational schema to the Boolean function that maps a valuation ν to $\top$ if $t \in Q(\nu(D))$, and to $\bot$ otherwise. In other words, given provenance annotations on input database tuples, we obtain as output of a query a new database, namely,

$$\bigcup_{\nu:X\to\{\top,\bot\}} Q(\nu(D)),$$

with new provenance annotations on each tuple t, namely $\text{prov}_{Q,D}(t)$.

Note that if Q is *monotone*, i.e., if $D \subseteq D' \Rightarrow Q(D) \subseteq Q(D')$, then $\bigcup_{\nu:X\to\{\top,\bot\}} Q(\nu(D)) \subseteq Q(D)$, but this is not true for arbitrary queries.

Example 3. We will denote in this example Boolean functions using propositional formulas. See Section 5 for an alternate representation. To simplify, we assume that, once again, every t_i is an indicator function of a different Boolean event, and we write simply t_i for this event instead of x_i.

One can check that the Boolean provenances of Q_1 and Q_2 on Personal are, respectively:

city	
New York	$t_1 \wedge t_2$
Paris	$(t_3 \wedge t_5) \vee (t_3 \wedge t_6) \vee (t_5 \wedge t_6)$
Berlin	$t_4 \wedge t_7$

and:

city	
New York	$(t_1 \wedge \neg t_2) \vee (t_2 \wedge \neg t_1)$
Paris	$(t_3 \wedge \neg(t_5 \vee t_6)) \vee (t_5 \wedge \neg(t_3 \vee t_6)) \vee (t_6 \wedge \neg(t_3 \vee t_5))$
Berlin	$(t_4 \wedge \neg t_7) \vee (t_7 \wedge \neg t_4)$

□

One of the major applications of Boolean provenance is query evaluation in probabilistic databases. Assume that each Boolean event x_i comes with an independent probability $\Pr(x_i)$ of being true. Then we can define the probability of a valuation $\nu : X \to \{\top, \bot\}$ as:

$$\Pr(\nu) = \prod_{\nu(x_i)=\top} \Pr(x_i) \prod_{\nu(x_i)=\bot} (1 - \Pr(x_i)).$$

From there, it is natural to define, for any given Boolean function φ over X:

$$\Pr(\varphi) = \sum_{\substack{\nu:X\to\{\top,\bot\} \\ \varphi(\nu)=\top}} \Pr(\nu).$$

In particular, this defines a probability distribution $\Pr(\text{prov}_{Q,D}(t))$ on provenance annotations of output tuples given a probability distribution on the provenance annotations of input tuples. This observation was first made by Green and Tannen in [28]. When t_i's are indicator functions, one gets the simple model of *tuple-independent databases* [14, 23, 36] that has been widely studied.

Example 4. Assume again every t_i is an indicator function of a different Boolean function, and comes with an independent probability $\Pr(t_i)$ of being true.

Then $\Pr(\text{New York} \in Q_1(\text{Personal})) = \Pr(t_1 \wedge t_2) = \Pr(t_1) \times \Pr(t_2)$.

Note that we were able to compute the probability this way because the Boolean formula was simple enough. We discuss in Section 6 options when Boolean functions are more complex. □

Though Boolean provenance can be formally defined for any query, what we need in practice is efficient algorithms for computing the provenance of a query in a given query language. We also want to capture more with provenance than what Boolean provenance can do. This is what provenance semirings, and extensions thereof, offer.

3. SEMIRING PROVENANCE

Provenance semirings have been introduced in [27] as a formalism for data provenance that has been shown [34] to cover and generalize, using a clean mathematical framework, previous formalisms such as *why-provenance* [11], lineages used in view maintenance [13], or the lineage used by the Trio uncertain management system [9].

A *semiring* $(K, \mathbb{0}, \mathbb{1}, \oplus, \otimes)$ is a set K with distinguished elements $\mathbb{0}$ and $\mathbb{1}$, along with two binary operators:

- $\oplus$, an associative and commutative operator, with identity $\mathbb{0}$;
- $\otimes$, an associative and commutative[1] operator, with identity $\mathbb{1}$.

We further require $\otimes$ to distribute over $\oplus$, and $\mathbb{0}$ to be annihilating for $\otimes$.

Examples of semirings include [27, 28, 34]:

- $(\mathbb{N}, 0, 1, +, \times)$: the *counting* semiring;
- $(\{\bot, \top\}, \bot, \top, \vee, \wedge)$: the *Boolean* semiring;
- $(\{\text{unclassified}, \text{restricted}, \text{confidential}, \text{secret}, \text{top secret}\}, \text{top secret}, \text{unclassified}, \min, \max)$: the *security* semiring;
- $(\mathbb{N} \cup \{\infty\}, \infty, 0, \min, +)$: the *tropical* semiring;

[1]It is almost always required in the literature [27, 34] that the semiring be commutative, which means that $\otimes$ must be commutative. Note that this is only necessary if the cross product operator of the relational algebra is assumed to be commutative, which is the case in the *named* perspective, but not in the *unnamed* one [1]. This assumption has some technical impact, e.g., on the universality of $\mathbb{N}[X]$, but is actually not critical to implement provenance support.

- ($\{$positive Boolean funct. over $X\}, \bot, \top, \vee, \wedge$): the semiring of *positive Boolean functions* over X;
- $(\mathbb{N}[X], 0, 1, +, \times)$: the semiring of *integer polynomials* with variables in X (also called *how*-semiring or *universal* semiring, see further);
- $(\mathcal{P}(\mathcal{P}(X)), \emptyset, \{\emptyset\}, \cup, \Cup)$: *why*-semiring over X ($A \Cup B \coloneqq \{a \cup b \mid a \in A, b \in B\}$).

Now, given a fixed semiring $(K, \mathbb{0}, \mathbb{1}, \oplus, \otimes)$, semiring provenance works as follows: we assume provenance tokens are all in K. We consider a query Q from the *positive relational algebra* [1] (selection, projection, renaming, cross product, union). We define a semantics for the provenance of a tuple $t \in Q(D)$ inductively on the structure of Q, informally as follows (formal definitions can be found in [27]):

- selection and renaming do not affect provenance annotations;
- in the set semantics, the provenance annotations of tuples that are identical after projection are $\oplus$-ed; in the bag semantics [29] that more closely models SQL, projection does not affect provenance annotations, but *duplicate elimination* $\oplus$-es the annotations of merged tuples;
- the provenance annotations of unioned tuples are $\oplus$-ed;
- the provenance annotations of tuples combined in a cross product are $\otimes$-ed.

Example 5. Consider the security semiring and query Q_1, which can be rewritten in the relational algebra as:

$$\Pi_{\mathsf{city}}(\sigma_{\mathsf{id}<\mathsf{id2}}(\Pi_{\mathsf{id},\mathsf{city}}(\mathsf{Personal}) \bowtie \rho_{\mathsf{id}\to\mathsf{id2}}(\pi_{\mathsf{id},\mathsf{city}}(\mathsf{Personal})))))$$

(the join operator $\bowtie$ being a combination of a cross product, selection, and projection). Using the inductive definition of the provenance of a tuple in a query result, and assuming that the initial provenance tokens t_i are equal to the classification attribute of the tuple, one can compute the provenance of the output of the query as:

city	
New York	restricted
Paris	confidential
Berlin	secret

Similarly, if we consider the counting semiring and query Q_1, assuming the initial provenance tokens t_i are equal to the id attribute of the tuple, one can compute the provenance of the output of the query as:

city	
New York	2
Paris	63
Berlin	28

□

Indeed, simply using this inductive definition of semiring provenance, one can use different semirings to compute different meta-information on the output of a query, with polynomial-time overhead in data complexity:

counting semiring: the number of times a tuple can be derived;

Boolean semiring: if a tuple exists when a subdatabase is selected;

security semiring: the minimum clearance level required to get a tuple as a result;

tropical semiring: minimum-weight way of deriving a tuple (as when computing shortest paths in a graph);

positive Boolean functions: Boolean provenance, *as previously defined*;

integer polynomials: universal provenance, see further;

why-semiring: why-provenance of [11], set of combinations of tuples needed for a tuple to exist.

However, [27] makes two important observations that lead to a different way to compute provenance annotations, instead of doing it one semiring at a time. First, semiring homomorphisms *commute* with provenance computation: if there is a homomorphism from semiring K to semiring K', then one can compute the provenance in K, apply the homomorphism, and obtain the same result as when computing provenance in K'. Second, the integer polynomial semiring $\mathbb{N}[X]$ is the unique *universal* semiring in the following sense: there exists a unique homomorphism to any other commutative semiring that respects a given valuation of the variables. Combining these two facts, we have that *all computations can be performed in the universal semiring*, with homomorphisms only applied when the provenance for a given semiring is required. This suggests a way to implement provenance computation in a DBMS, discussed in Section 5.

Note that two queries that are equivalent in the usual sense [1] can have different semiring provenance, as semiring provenance captures more than logical equivalence. Indeed, two queries are logically equivalent if and only if they have the same *Boolean provenance* on every database.

Provenance semirings only capture the positive relational algebra, a relatively small fragment of SQL. We next discuss how to go beyond this fragment,

and investigate if all interesting forms of provenance are captured by provenance semirings.

4. BEYOND SEMIRING PROVENANCE

We now discuss some extensions of the provenance semiring framework.

Semirings with monus. Semiring provenance can only be defined for the positive fragment of the relational algebra, excluding non-monotone operations such as difference. However, some semirings can be straightforwardly equipped with a *monus* operator $\ominus$ [6, 24], that captures non-monotone behavior. Such an operator must verify the following properties, for all $a, b, c \in K$:

- $a \oplus (b \ominus a) = b \oplus (a \ominus b)$;
- $(a \ominus b) \ominus c = a \ominus (b \oplus c)$;
- $a \ominus a = \mathbb{0} - a = \mathbb{0}$.

This is the case for the Boolean function semiring, which, equipped with the monus operator $a \ominus b = a \wedge \neg b$, forms a *semiring with monus*, or *m-semiring* for short. This is also the case [7] for the why-semiring with set difference, the integer polynomial and counting semirings with truncated difference on scalar values ($a \ominus b = \max(0, a - b)$), etc. Indeed, most natural semirings (though not all [5]) can be extended to m-semirings.

Once such an m-semiring is defined, provenance of the full relational algebra can be captured in that m-semiring. For Boolean functions, it coincides with the Boolean provenance introduced in Section 4.

Note, however, that sometimes some seemingly natural axioms, such as distributivity of $\otimes$ over $\ominus$, fail over m-semirings [7], which implies that two very similar queries may return different provenances.

Another important difference between m-semirings and semirings is that $\mathbb{N}[X]$ is not a universal m-semiring [24]. There does indeed exist a unique universal m-semiring [24], but it is simply the *free m-semiring*, i.e., the m-semiring of free terms constructed using $\oplus$, $\otimes$, $\ominus$, quotiented by the equivalence relations imposed by the m-semiring structure.

We will illustrate in Section 5 how computation is performed using m-semirings.

Provenance for aggregates. One of the most natural ways to extend the relational algebra is to add aggregation capabilities [37]. There have been attempts at defining provenance formalisms for aggregate queries [8, 22]. This is feasible in the case of associative and commutative aggregation, though it requires moving from annotations at the level of tuples to annotations at the level of values, as elements of a *semimodule* that combines provenance semiring annotation with scalar values from the range of the aggregation function. We believe the representation systems of Section 5 could be extended to provide somewhat compact representations of these semimodule elements, but leave this for future work.

Where-provenance. One notable provenance formalism that was introduced early on [11] is where-provenance. The where-provenance is a bipartite graph that connects values in the output relation to values in the input relation to indicate where a specific value may come from in the input. It was shown [12] that where-provenance *cannot* be captured by semiring provenance: there is no semiring for which semiring provenance allows reconstructing the where-provenance of a query. This is, intuitively, for two reasons:

- since where-provenance is assigned to individual values instead of tuples, it is affected either by renaming (in the named perspective) or by projection (in the unnamed perspective), as there needs to be a way to keep track of which value of a given tuple has which where-provenance;
- where-provenance is affected by joins differently as by a combination of cross product, selection, and projection: a value that results from a join of two relations has where-provenance pointing to the joined value in both relations.

However, a system keeping track of semiring provenance could be relatively straightforwardly extended to keep track of where-provenance: instead of dealing with values in a semiring (or in an m-semiring), just maintain value in a free algebra of terms, whose operator includes, in addition to $\oplus$, $\otimes$, and perhaps $\ominus$, operators to record projections (or renaming) and joined values.

Recursive queries. Query languages considered so far are unable to express recursive queries, such as shortest distance in a graph. It is also possible to define provenance notions for such queries as extension to semiring provenance, as long as the provenance formalism can express cycles. This was done in the original work on provenance semirings [27] for *ω-continuous semirings*, showing that the semiring $\mathbb{N}^\infty[[X]]$ of formal power series with integer coefficients is a universal ω-continuous semiring. An alternative approach is the use of semirings with Kleene stars [21], such as k-closed semirings [39], for which efficient algorithms for provenance computation can be designed [39]. We leave details for further work, though we note that the representation systems we are introducing next – circuits – need to

be amended in the case of recursive queries, using either equation systems as in [27] or *cycluits* (cyclic circuits) as in [2].

5. PROVENANCE CIRCUITS

We now discuss concrete representations for provenance annotations. As we have seen, leaving the case of provenance for aggregates, where-provenance, and recursive queries, to future work, provenance annotations can be Boolean functions (see Section 2) useful for probabilistic databases, semiring values (see Section 3), or m-semiring values (see Section 4). In addition, positive Boolean provenance is a special case of semiring provenance, and non-monotone Boolean provenance is a special case of m-semiring provenance. Finally, there exist a universal semiring and a universal m-semiring.

In some semirings (the Boolean, counting, and security semirings, for instance), provenance annotations are *elementary*, i.e., they are easily representable with enumerations or native types. Other semirings, such as $\mathbb{N}[X]$ or the Boolean function semirings have complex annotations, for which a compact representation needs to be found.

In many previous works [27, 31, 45], provenance annotations have been represented as formulas, e.g., propositional formulas for Boolean provenance. But this leads to suboptimal representations, as (Boolean) formulas can be less compact than (Boolean) *circuits* [4, 47]. We therefore argue in favor of using *provenance circuits*, arithmetic circuits whose gates are the operators of the (m-)semiring as in [3, 20], as a compact representation system for provenance.

Example 6. Consider queries Q_1 and Q_2 on Personal. We can represent the provenance annotations of their output as references to gates in the universal m-semiring circuit shown in Figure 1. The output of Q_1 is as follows:

city	
New York	g_1
Paris	g_2
Berlin	g_3

while that of Q_2 is:

city	
New York	g_4
Paris	g_5
Berlin	g_6

Indeed, by developing the circuit, one can for instance verify that the provenance of "New York" for Q_2 on Personal is $(t_1 \oplus t_2) \ominus (t_1 \otimes t_2)$. As can be seen on Figure 1, a significant amount of sharing can be obtained for provenance within and across queries by using provenance circuits. □

This suggests a practical way for computing provenance of queries over a relational database: inductively construct a provenance circuit over input tuples for each operation performed in a query, reusing parts of the circuit that have been constructed by subqueries. By constructing this circuit in the universal m-semiring, it then becomes easy to instantiate it to a wide variety of semirings and m-semirings.

Example 7. Consider the query Q_1 on Personal, for which we want to compute the security and counting semiring annotations. Since we have already computed in Figure 1 a circuit for this query in the universal m-semiring, we can directly obtain a circuit whose evaluation returns the provenance of Q_1 in either of these semirings by applying the appropriate semiring homomorphisms. This is what is shown in Figures 2 and 3. One can verify that the evaluation of these queries returns the provenance annotations already computed in Example 5.

Similarly, one can compute the Boolean circuit of Figure 4 by applying the m-semiring homomorphism from the universal m-semiring of Figure 1 to the Boolean function semiring. □

This approach of incremental provenance computation in the universal m-semiring, with specialization to arbitrary semirings and m-semirings on demand, is that taken by ProvSQL [43], a lightweight add-on to the PostgreSQL database management system for support of (m-)semiring provenance computation on relational databases. To our knowledge, this is the only publicly available system for management of data with semiring provenance, with support of a large subset of the SQL query language. The closest such software may be orchestra [25], which is unfortunately unavailable.

6. PROBABILITY EVALUATION

As discussed in Section 2, Boolean provenance is a very important tool to compute the probability of a query in probabilistic databases, an intractable (#P-hard) problem in general [45]. It allows separating the concerns between query evaluation on the one hand, which produces a Boolean provenance annotation in polynomial time, and probability evaluation of the provenance annotation on the other hand, itself a #P-hard problem.

Let us thus assume we have obtained a Boolean circuit of the data provenance of a query over a probabilistic database. What can, then, be done to evaluate the probability of the provenance annotation, given the intractability of the problem?

Brute-force algorithms. The first possible way is to resort to an exponential-time enumera-

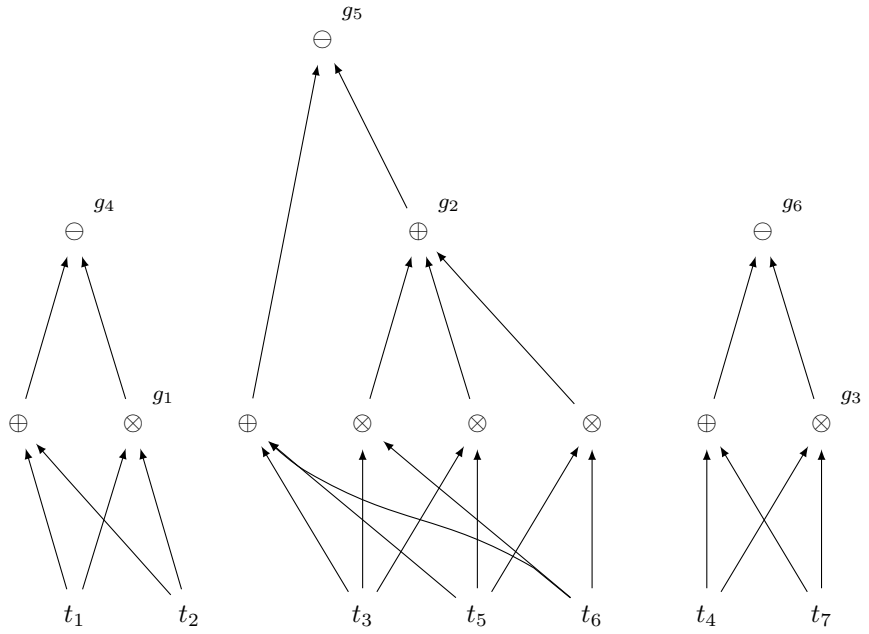

Figure 1: Provenance circuit for queries Q_1 and Q_2 in the universal m-semiring

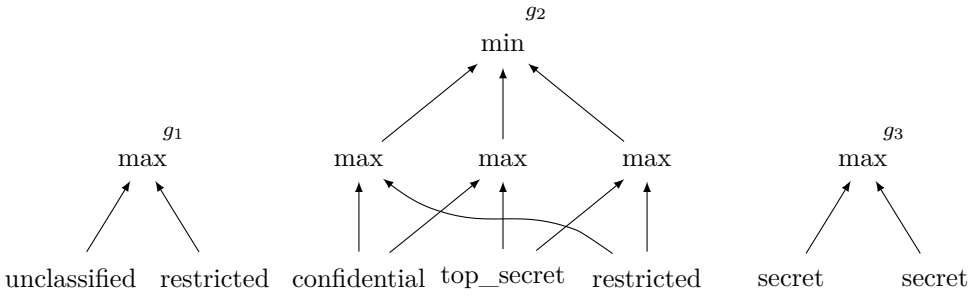

Figure 2: Provenance circuit for query Q_1 in the security semiring

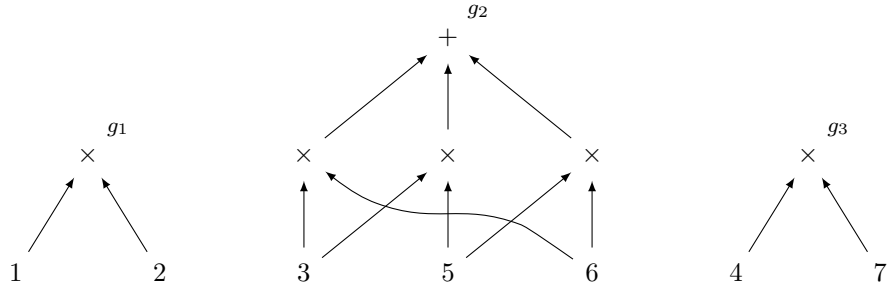

Figure 3: Provenance circuit for query Q_1 in the counting semiring

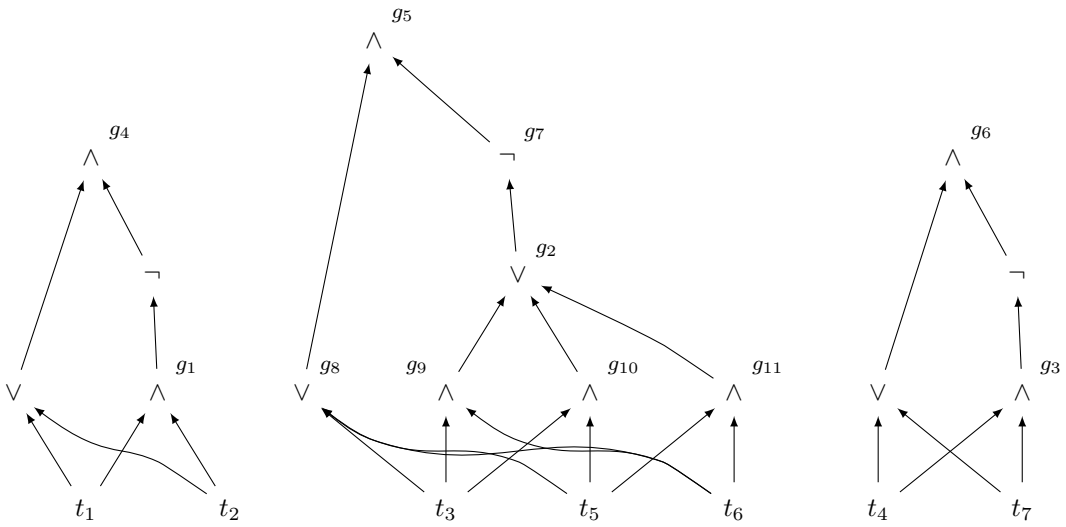

Figure 4: Boolean circuit for queries Q_1 and Q_2

tion of all possible valuations of the Boolean events occurring in the provenance annotation, and summing the probabilities of valuations mapped to $\top$ by the provenance annotation to compute the overall probability. This is rarely feasible, but note that it is at least more efficient than enumerating all possible worlds of the initial database.

Approximations. One can always resort to approximating the probability of the query, either by Monte-Carlo sampling, which is always feasible but fairly slow, or by more refined approximation techniques, as in [42, 44].

Exploiting the query structure. Jha and Suciu [33] have shown that, when queries have specific forms, it is possible to construct Boolean provenance circuits of certain types, that allow for efficient probability evaluation. In particular, if a union of conjunctive queries (UCQ) is *inversion-free*, an ordered binary diagram (OBDD [10]) for it can be obtained. If some more general property is satisfied, then it admits a deterministic decomposable negation normal form (d-DNNF [16]). Both OBDDs and the more general class of d-DNNFs allow for efficient (linear-time) probabilistic query evaluation. Of course, this approach is inapplicable if the query is not of the specific form required. Note also that this approach is specific to Boolean provenance, which means it precludes computation of provenance in a more general (m-)semiring before specializing to the Boolean function case.

Exploiting the data structure. An alternative is to exploit the fact that the structure of the data is not arbitrary. Indeed, if the data has the structure of a tree, or has a low treewidth, meaning that its structure is close to that of a tree, it has been shown [2, 3] that a bounded-treewidth provenance circuit can be constructed, which in turn supports tractable query evaluation. This line of technique has been successfully applied to synthetic [40] and real-world [38] data, for specific kinds of queries.

When none of this is feasible, one can resort to general *knowledge compilation* techniques [18]. Knowledge compilation is the problem of transforming Boolean functions of a certain form into another, more tractable, form. Over the years, a wide variety of techniques, results, heuristics, and tools have emerged from the knowledge compilation community. In particular, tools such as C2D [17], DSHARP [41], and D4 [35] compile arbitrary formulas in *conjunctive normal form* into d-DNNFs.

One practical approach for probabilistic query evaluation is therefore to produce a Boolean provenance circuit, transform it into a conjunctive normal form in linear time using the standard Tseitin transformation [46], and feed it to a knowledge compiler. This approach is used in PROVSQL.

Example 8. Consider the middle connected component of the Boolean circuit of Figure 4, and, in particular, gate g_5 which yields the Boolean provenance of "Paris" in the output of query Q_2 on Personal. One can transform this part of the circuit into the following equivalent conjunctive normal form, where variables are inputs of the circuit along with its

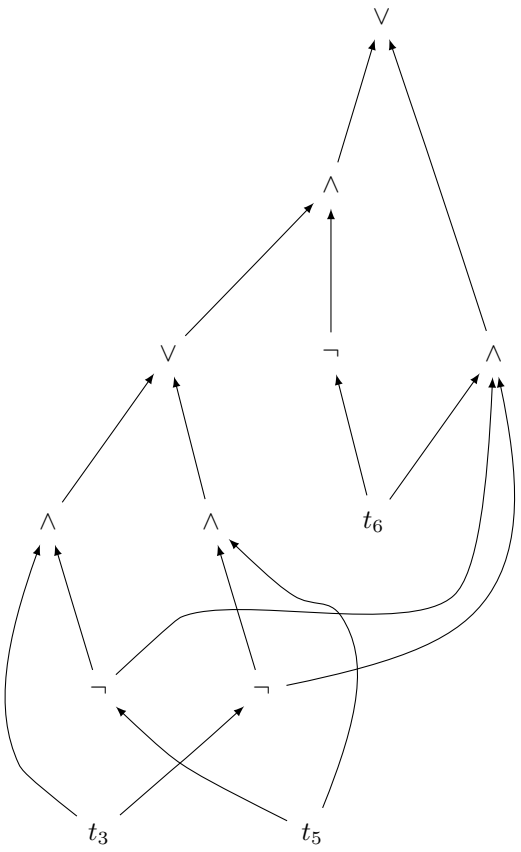

Figure 5: d-DNNF for tuple "Paris" in the output of query Q_2

internal gates, using Tseitin transformation:

$$
\begin{array}{llll}
 & g_5 \vee \bar{g}_8 \vee \bar{g}_7 & \wedge & \bar{g}_5 \vee g_8 \\
\wedge & \bar{g}_5 \vee g_7 & \wedge & \bar{g}_8 \vee t_3 \vee t_5 \vee t_6 \\
\wedge & g_8 \vee \bar{t}_3 & \wedge & g_8 \vee \bar{t}_5 \\
\wedge & g_8 \vee \bar{t}_6 & \wedge & g_7 \vee g_2 \\
\wedge & \bar{g}_7 \vee \bar{g}_2 & \wedge & \bar{g}_2 \vee t_9 \vee t_{10} \vee t_{11} \\
\wedge & g_2 \vee \bar{t}_9 & \wedge & g_2 \vee \bar{t}_{10} \\
\wedge & g_2 \vee \bar{t}_{11} & \wedge & g_9 \vee \bar{t}_3 \vee \bar{t}_6 \\
\wedge & \bar{g}_9 \vee t_3 & \wedge & \bar{g}_9 \vee t_6 \\
\wedge & g_{10} \vee \bar{t}_3 \vee \bar{t}_5 & \wedge & \bar{g}_{10} \vee t_3 \\
\wedge & \bar{g}_{10} \vee t_5 & \wedge & g_{11} \vee \bar{t}_6 \vee \bar{t}_6 \\
\wedge & \bar{g}_{11} \vee t_6 & \wedge & \bar{g}_{11} \vee t_6 \\
\wedge & g_5 & &
\end{array}
$$

Once such a formula obtained, it can be given as input to a knowledge compiler. For example, D4 outputs the d-DNNF in Figure 5. A d-DNNF is a special case of a Boolean circuit, where every ¬-gate is directly connected to an input, every ∧-gate has children with disjoint sets of descendant leaves, and every ∨-gate is such that only one of its child can be true in any possible world. These restrictions make it possible to compute the probability of a gate in linear-time, given a probability distribution on input gate: ∧-gates become products, while ∨-gates become sums. The computation of $\Pr(g_5)$ from the d-DNNF in Figure 4 is shown in Figure 6. □

Note that this intensional (provenance-based) approach to probabilistic query evaluation is not the only one. The *extensional approach*, which directly manipulates probabilities as the query is evaluated, without an intermediate provenance representation, has also been successfully used [14, 15]. It is an open problem [15, 33] whether there are cases where the extensional approach succeeds but no compact d-DNNF is obtainable.

7. CONCLUSION

Data provenance is a major tool to obtain additional information on query output. It allows answering questions about:
why in the why-semiring;
where using where-provenance;
how using integer polynomials;
missing tuples using Boolean provenance;
how many times in the counting semiring;
probability using probability evaluation of Boolean provenance;
minimal security clearance in the security semiring;
most economical way in the tropical semiring.

Practical implementation of provenance management is very much possible, since it introduces a relatively low overhead. One avenue for practical implementations is to perform all computations in a universal structure, such as the universal m-semiring, and only specialize when needed. Using these provenance representations, it is also possible to perform query evaluation on probabilistic databases, for instance using knowledge compilation to obtain Boolean provenance representations on which probabilistic evaluation is efficient.

8. REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
[2] Antoine Amarilli, Pierre Bourhis, Mikaël Monet, and Pierre Senellart. Combined tractability of query evaluation via tree automata and cycluits. In *ICDT*, 2017.
[3] Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Provenance circuits for trees and treelike instances. In *ICALP*, 2015.

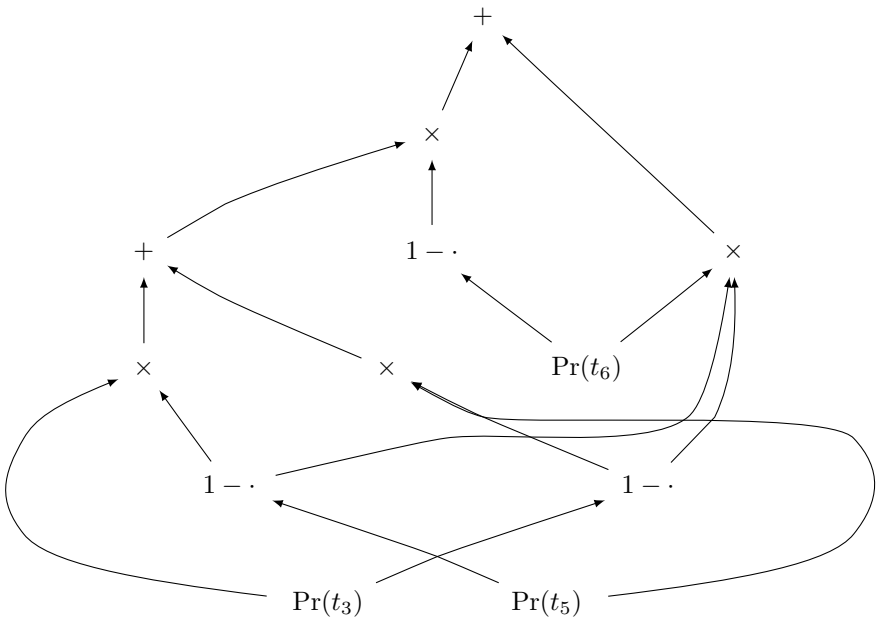

Figure 6: Probability computation following the d-DNNF of Figure 5

[4] Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Tractable lineages on treelike instances: Limits and extensions. In *PODS*, 2016.

[5] Antoine Amarilli and Mikaël Monet. Example of a naturally ordered semiring which is not an m-semiring. `http://math.stackexchange.com/questions/1966858`, 2016.

[6] K. Amer. *Algebra Universalis*, 18(1), 1984.

[7] Yael Amsterdamer, Daniel Deutch, and Val Tannen. On the limitations of provenance for queries with difference. In *TaPP*, 2011.

[8] Yael Amsterdamer, Daniel Deutch, and Val Tannen. Provenance for aggregate queries. In *PODS*, 2011.

[9] Omar Benjelloun, Anish Das Sarma, Alon Halevy, and Jennifer Widom. ULDBs: Databases with uncertainty and lineage. In *VLDB*, 2006.

[10] Randal E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3), 1992.

[11] Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. Why and where: A characterization of data provenance. In *ICDT*, 2001.

[12] James Cheney, Laura Chiticariu, and Wang Chiew Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4), 2009.

[13] Yingwei Cui and Jennifer Widom. Practical lineage tracing in data warehouses. In *ICDE*, 2000.

[14] Nilesh Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. *The VLDB Journal*, 16(4), 2007.

[15] Nilesh Dalvi and Dan Suciu. The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM*, 59(6), 2012.

[16] Adnan Darwiche. On the tractable counting of theory models and its application to truth maintenance and belief revision. *J. Applied Non-Classical Logics*, 11(1-2), 2001.

[17] Adnan Darwiche. New advances in compiling CNF to decomposable negation normal form. In *ECAI*, 2004.

[18] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *J. Artificial Intelligence Research*, 17(1), 2002.

[19] Susan B Davidson and Juliana Freire. Provenance and scientific workflows: challenges and opportunities. In *SIGMOD*, 2008.

[20] Daniel Deutch, Tova Milo, Sudeepa Roy, and Val Tannen. Circuits for Datalog provenance. In *ICDT*, 2014.

[21] Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of weighted automata*. Springer Science & Business Media, 2009.

[22] Robert Fink, Larisa Han, and Dan Olteanu.

Aggregation in probabilistic databases via knowledge compilation. *PVLDB*, 5(5), 2012.
[23] Norbert Fuhr and Thomas Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Transactions on Information Systems*, 15(1), 1997.
[24] Floris Geerts and Antonella Poggi. On database query languages for K-relations. *J. Applied Logic*, 8(2), 2010.
[25] Grigoris Green, Todd J. and Karvounarakis, Zach Ives, and Val Tannen. Provenance in ORCHESTRA. *IEEE Data Eng. Bull*, 33(3), 2010.
[26] Todd J Green. Containment of conjunctive queries on annotated relations. *Theory of Computing Systems*, 49(2), 2011.
[27] Todd J Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, 2007.
[28] Todd J. Green and Val Tannen. Models for incomplete and probabilistic information. *IEEE Data Eng. Bull.*, 29(1), 2006.
[29] Stéphane Grumbach and Tova Milo. Towards tractable algebras for bags. *J. Computer and System Sciences*, 52(3), 1996.
[30] Olaf Hartig. Provenance information in the web of data. In *LDOW*, 2009.
[31] Tomasz Imielinski and Witold Lipski Jr. Incomplete information in relational databases. *J. ACM*, 31(4), 1984.
[32] Abhay Jha, Dan Olteanu, and Dan Suciu. Bridging the gap between intensional and extensional query evaluation in probabilistic databases. In *EDBT*, 2010.
[33] Abhay Jha and Dan Suciu. Knowledge compilation meets database theory: compiling queries to decision diagrams. *Theory of Computing Systems*, 52(3), 2013.
[34] Grigoris Karvounarakis and Todd J Green. Semiring-annotated data: queries and provenance? *ACM SIGMOD Record*, 41(3), 2012.
[35] Jean-Marie Lagniez and Pierre Marquis. An improved decision-DNNF compiler. In *IJCAI*, 2017.
[36] Laks VS Lakshmanan, Nicola Leone, Robert Ross, and Venkatramanan Siva Subrahmanian. Probview: A flexible probabilistic database system. *ACM Transactions on Database Systems*, 22(3), 1997.
[37] Leonid Libkin and Limsoon Wong. Query languages for bags and aggregate functions. *J. Computer and System sciences*, 55(2), 1997.
[38] Silviu Maniu, Reynold Cheng, and Pierre Senellart. An indexing framework for queries on probabilistic graphs. *ACM Transactions on Database Systems*, 42(2), 2017.
[39] Mehryar Mohri. Semiring frameworks and algorithms for shortest-distance problems. *J. Automata, Languages and Combinatorics*, 7(3), 2002.
[40] Mikaël Monet. Probabilistic evaluation of expressive queries on bounded-treewidth instances. In *SIGMOD/PODS PhD Symposium*, 2016.
[41] Christian J Muise, Sheila A McIlraith, J Christopher Beck, and Eric I Hsu. Dsharp: Fast d-DNNF compilation with sharpSAT. In *Canadian Conference on AI*, 2012.
[42] Dan Olteanu, Jiewen Huang, and Christoph Koch. Approximate confidence computation in probabilistic databases. In *ICDE*, 2010.
[43] Pierre Senellart. ProvSQL. `https://github.com/PierreSenellart/provsql`, 2017.
[44] Asma Souihli and Pierre Senellart. Optimizing approximations of DNF query lineage in probabilistic XML. In *ICDE*, 2013.
[45] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Morgan & Claypool, 2011.
[46] G Tseitin. On the complexity of derivation in propositional calculus. *Studies in Constrained Mathematics and Mathematical Logic*, 1968.
[47] Ingo Wegener. *The complexity of Boolean functions*. Wiley, 1987.
[48] Allison Woodruff and Michael Stonebraker. Supporting fine-grained data lineage in a database visualization environment. In *ICDE*, 1997.