ULDBs: Databases with Uncertainty and Lineage*

Omar Benjelloun[1] Anish Das Sarma[1] Alon Halevy[2] Jennifer Widom[1]

[1]Stanford University
{benjello,anish,widom}@cs.stanford.edu

[2]Google
halevy@google.com

Abstract

This paper introduces ULDBs, an extension of relational databases with simple yet expressive constructs for representing and manipulating both *lineage* and *uncertainty*. Uncertain data and data lineage are two important areas of data management that have been considered extensively in isolation, however many applications require the features in tandem. Fundamentally, lineage enables simple and consistent representation of uncertain data, it correlates uncertainty in query results with uncertainty in the input data, and query processing with lineage and uncertainty together presents computational benefits over treating them separately.

We show that the ULDB representation is *complete*, and that it permits straightforward implementation of many relational operations. We define two notions of ULDB minimality—*data-minimal* and *lineage-minimal*—and study minimization of ULDB representations under both notions. With lineage, derived relations are no longer self-contained: their uncertainty depends on uncertainty in the base data. We provide an algorithm for the new operation of extracting a database subset in the presence of interconnected uncertainty. Finally, we show how ULDBs enable a new approach to query processing in probabilistic databases.

ULDBs form the basis of the *Trio* system under development at Stanford.

* This work was supported by the National Science Foundation under grants IIS-0324431, IIS-1098447, and IIS-9985114, by DARPA Contract #03-000225, and by a grant from the Boeing Corporation.

VLDB '06, September 12-15, 2006, Seoul, Korea.

1 Introduction

The problems faced when managing *uncertain data*, and those associated with tracking *data lineage*, have been addressed in isolation in the past (e.g., [2, 5, 19, 23, 27, 28, 31, 36, 39] for uncertain data and [11, 15, 16, 17, 34, 35] for data lineage). Motivated by a diverse set of applications including data integration, deduplication, scientific data management, information extraction, and others, we became interested in the combination of uncertainty and lineage as the basis for a new type of data management system [40].

Intuitively, an uncertain database is one that represents multiple *possible instances*, each corresponding to a single possible state of the database. Lineage identifies a data item's *derivation*, in terms of other data in the database, or outside data sources. One relationship between uncertainty and lineage is that lineage can be used for understanding and resolving uncertainty. To draw a loose analogy with web search, answers returned by a search engine are uncertain, reflected by their ranking. Search engines typically provide lineage information including at least a URL and text snippet, and users tend to consider both ranking and lineage to determine which links to follow. More generally, any application that integrates information from multiple sources may be uncertain about which data is correct, and the original source and derivation of data may offer helpful additional information.

Lineage is also important for uncertainty within a single database. When users pose queries against uncertain data, the results are uncertain too. Lineage facilitates the correlation and coordination of uncertainty in query results with uncertainty in the input data. For example, suppose we know that either one set of base data is correct or another one is, but not both. Then we don't want to produce any query results that are derived by mixing data from the two sets, directly or indirectly, now or later. Lineage is a particularly convenient and intuitive mechanism for encoding the complex uncertainty relationships that can arise among base and derived data.

Beyond the conceptual relationships between uncertainty and lineage, this paper presents several tangible representational and computational benefits derived from their combination. We begin by describing a representation for uncertainty and lineage that extends the relational model with *tuple alternatives* (a set of possible values for each tuple), *maybe tuples* (tuples that may be present

or absent), and a *lineage function* mapping tuple alternatives to the data from which they were derived. We call databases in this scheme ULDBs, for *Uncertainty-Lineage Databases*. We show that because we represent lineage along with uncertainty, ULDBs are *complete*, i.e., they can represent all finite sets of possible instances. In contrast, complete models for uncertainty without lineage are more complex, e.g., [21, 28].

Next, we study problems related to querying ULDBs. First, we show that ULDBs permit straightforward and efficient implementation of many relational operations. We then consider the problem of extracting one or more relations from a ULDB: creating a "projection" of a ULDB onto a subset of its relations, without changing the possible instances of the relations. Extracting relations is tricky because when data in a relation R is derived from data in R', then the possible instances of R may correlate with the possible instances of R' (even when R' is not be included in the projection), which may in turn correlate with possible instances of other relations. Finally, for both querying and extraction we are interested in operating on ULDBs that satisfy some notions of minimality. We define *data minimality* and *lineage minimality* of ULDBs, and we present results on minimizing ULDBs.

ULDBs also open up an interesting alternative approach to query processing in *probabilistic databases*, which are captured by a simple extension of basic ULDBs to include *confidence values*. Previous work [19] suggests special techniques for constructing query plans that ensure correctness for probabilistic data. It turns out that when lineage is tracked, special considerations are no longer needed: Query execution initially proceeds without computing probabilities, so any query plan may be used. Probabilities are then computed from lineage as needed in a separate step.

In summary, this paper makes the following contributions:

- We define ULDBs—uncertain databases with lineage—and show that they are complete (Sections 2 and 3).
- We give algorithms for relational operations in ULDBs (Section 4.2).
- We define data-minimality and lineage-minimality for ULDBs, and discuss both types of minimization (Section 4.3).
- We define the new problem of extracting data from a ULDB, and we present an algorithm for it (Section 4.4).
- We describe how ULDBs can be extended with confidence values, and we show how they offer an alternative solution to query processing in probabilistic databases (Section 5).

We discuss related work in Section 6 and conclude with future directions in Section 7. Note that ULDBs as presented in this paper form the basis of the *Trio* system under development at Stanford. An overview of Trio is given in [7].

2 Preliminaries

We begin by describing databases with lineage, which we call LDBs, and then we describe uncertain databases. In Section 3 we present ULDBs, which combine the two formalisms.

LDBs and ULDBs extend the relational model. A database D is comprised of a set of relations $\bar{R} = R_1, \ldots, R_n$, where each R_i is a multiset of tuples. We attach a unique identifier to each tuple in the database, and $I(\bar{R})$ denotes all identifiers in relations $R_1, \ldots, R_n$.

2.1 Databases with Lineage

In the terminology of [11], in LDBs we focus on "where lineage": the lineage of a tuple identifies the data from which it was derived. Some tuples in an LDB are derived from other LDB tuples, e.g., as a result of queries. The lineage of derived tuples consists of references to other tuples in the LDB, via their unique identifiers. Base tuples in some cases are derived from entities outside the LDB, such as an external data set or a sensor feed. For the latter case we introduce *external* lineage, which is formalized in this section along with *internal* lineage, but not discussed in any detail until Section 4. External lineage refers to a set of external symbols we denote by E. Thus, the set of symbols known by an LDB is $S = I(\bar{R}) \cup E$.

Definition 2.1. *(Database with lineage)*: An LDB D is a triple $(\bar{R}, S, \lambda)$, where $\bar{R}$ is a set of relations, S is a set of symbols containing $I(\bar{R})$, and λ is a *lineage function* from S to 2^S. □

Example 2.2. We introduce as a running example a highly simplified "crime-solver" database. Consider LDB relations `Drives(person,car)` and `Saw(witness,car)` representing driver information and crime-vehicle sightings respectively. Consider also a relation `Accuses(witness,person)` produced by the query $\pi_{\texttt{witness,person}}(\texttt{Saw} \bowtie \texttt{Drives})$. Here is some sample data:

Saw

ID	witness	car
21	Amy	Mazda
22	Amy	Toyota
23	Betty	Honda

Drives

ID	person	car
31	Jimmy	Mazda
32	Jimmy	Toyota
33	Billy	Mazda
34	Billy	Honda

Accuses

ID	witness	person	
41	Amy	Jimmy	λ (41) = {21, 31}
42	Amy	Jimmy	λ (42) = {22, 32}
43	Amy	Billy	λ (43) = {21, 33}
44	Betty	Billy	λ (44) = {23, 34}

The ID column denotes the tuple identifiers, and empty lineage is omitted. □

Our basic formalism places no restrictions on the lineage function λ. However, when operations are performed there is often an obvious lineage function for the

tuples in the result. The above example demonstrates a natural lineage function for joins: lineage of a tuple t in the result of a join is the set of tuples, one from each of the joined relations, that were combined to form t, e.g., `(Amy, Billy)` is obtained from `(Amy, Mazda)` and `(Billy, Mazda)`. Some operations, such as negation, duplicate-elimination, and aggregation, have less obvious lineage functions. For discussion of lineage functions see, e.g., [8, 11, 16, 17, 32]. The operations we consider in this paper all have simple lineage functions, and furthermore they preserve a notion of *well-behaved* lineage that we formalize later in the paper.

In an LDB, query results include lineage that refers to other tuples in the database. Hence, in our formalism the result of applying a query Q to database D includes the original relations $\bar{R}$ and a new relation for Q's answer with the appropriate lineage function. Thus, an important aspect of LDBs is that we cannot consider each relation in the database in isolation. We explore this point further in Section 4.4.

Note that while a relation may contain duplicates, each tuple has its own lineage. For example, tuples 41 and 42 in Example 2.2 have the same data, but each one has a different derivation, and therefore different lineage. Extending our model to set semantics requires more complex lineage functions than we consider in this paper, and is a subject of follow-on work.

2.2 Uncertain Databases

An uncertain database represents a set of *possible instances*, each of which is one possible state of the database. A number of different formalisms have been proposed for representing sets of possible instances, e.g., [1, 5, 21, 26, 28, 31]. One difference among these formalisms is in their expressive power: which sets of possible instances can be represented in the formalism. In what follows we introduce *x-relations*, a specific formalism for uncertain databases. Conceivably, we could have considered the combination of any uncertainty formalism with lineage, but we found x-relations to be a good starting point and a good fit for applications we are considering.

Definition 2.3. An *x-tuple* is a multiset of one or more tuples, called *alternatives*. An x-tuple may be annotated with a '?', in which case it is called a *maybe x-tuple*. An *x-relation* is a multiset of x-tuples. □

Alternatives of an x-tuple represent mutually exclusive values for the tuple, leading to the following definition of possible instances.

Definition 2.4. An x-relation R represents the set of possible instances P that can be constructed as follows: choose exactly one alternative from each x-tuple in R that is not a maybe x-tuple, and choose zero or one alternative from each x-tuple in R that is a maybe x-tuple. □

Example 2.5. The following x-relation represents an uncertain version of relation `Saw` from Example 2.2:

ID	Saw(witness, car)	
21	(Amy,Mazda) ‖ (Amy,Toyota)	?
23	(Betty,Honda)	

Here, Amy may have seen a Mazda, a Toyota, or no car at all, and the relation has three possible instances. □

A formalism for representing uncertainty is said to be *complete* if it can represent any finite set of possible instances. *c-tables* [28] is the prototypical complete formalism for uncertainty. x-relations are not a complete formalism. For example, the join `Accuses` of the x-relation `Saw` above with `Drives` from Example 2.2 cannot be represented as an x-relation: x-tuples are independent, so they cannot express the fact that if Amy accuses Jimmy (due to the Mazda), then she must accuse Billy as well.

Studies of completeness in various models for uncertainty can be found in, e.g., [2, 21, 26, 28, 31]. We will soon see (Section 3.1) that although x-relations alone are incomplete as shown above, adding lineage makes them complete.

3 Combining Lineage and Uncertainty

We now present ULDBs, a representation that captures both lineage and uncertainty. ULDBs extend the LDBs of Section 2.1 with the x-relations of Section 2.2.

Definition 3.1. A ULDB D is a triple $(\bar{R}, S, \lambda)$, where $\bar{R}$ is a set of x-relations, S is a set of symbols containing $I(\bar{R})$, and λ is a lineage function from S to 2^S. □

Identifiers in $I(\bar{R})$ now correspond to tuple alternatives. $I(\bar{R})$ thus contains pairs (i, j), where i identifies the x-tuple and j is an index for one of its alternatives. When we refer to an arbitrary symbol in the set S, we use $s_{(i,j)}$, denoting either $(i, j) \in I(\bar{R})$ or an external symbol. We will later see in Section 4.4 why (i, j) subscripts on external symbols are useful.

Example 3.2. We combine the uncertain `Saw` x-relation from Example 2.5 with the earlier `Drives` relation to create a new version of `Accuses` that has both uncertainty and lineage:

ID	Saw(witness, car)	
21	(Amy,Mazda) ‖ (Amy,Toyota)	?
23	(Betty,Honda)	

ID	Drives(person, car)
31	(Jimmy,Mazda)
32	(Jimmy,Toyota)
33	(Billy,Mazda)
34	(Billy,Honda)

ID	Accuses(witness, person)		
41	(Amy,Jimmy)	?	λ(41,1)={(21,1),(31,1)}
42	(Amy,Jimmy)	?	λ(42,1)={(21,2),(32,1)}
43	(Amy,Billy)	?	λ(43,1)={(21,1),(33,1)}
44	(Betty,Billy)	?	λ(44,1)={(23,1),(34,1)}

We now define the semantics of a ULDB as a set of possible instances, where each instance is an LDB. The main technical challenge in the definition is to ensure that each possible LDB is based on consistent lineage. Recall that alternatives of an x-tuple are mutually exclusive in a given instance (0 or 1 of them are chosen), so we need to ensure that a possible LDB does not have two tuples whose lineages are from distinct alternatives of the same x-tuple. Recall $s_{(i,j)}$ denotes both internal identifiers $(i,j) \in I(\bar{R})$ and external symbols.

Definition 3.3. Let $D = (\bar{R}, S, \lambda)$ be a ULDB. A possible LDB D_k of D is obtained as follows. Pick a set of symbols $S_k \subseteq S$ such that:

1. If $s_{(i,j)} \in S_k$, then for every $j' \neq j$, $s_{(i,j')} \notin S_k$.
2. $\forall s_{(i,j)} \in S_k$, $\lambda(s_{(i,j)}) \subseteq S_k$.
3. If for some x-tuple t_i there does not exist a $s_{(i,j)} \in S_k$, then t_i is a maybe x-tuple, and $\forall s_{(i,j)} \in t_i, \lambda(s_{(i,j)}) = \emptyset$ or $\lambda(s_{(i,j)}) \not\subseteq S_k$.

The possible LDB D_k is the triple $(\bar{R}_k, S_k, \lambda_k)$ where $\bar{R}_k$ includes exactly the alternatives of x-tuples in $\bar{R}$ such that $s_{(i,j)} \in S_k$, and λ_k is the restriction of λ to S_k. □

Intuitively, the first condition in Definition 3.3 says that alternatives of the same x-tuple are mutually exclusive, i.e., at most one of them may appear in each possible instance. The second condition enforces the semantics of lineage: if an alternative is present in a possible instance, so must be the alternatives it was derived from. Observe that this implication is in one direction only. The third condition says that an x-tuple must yield a tuple in a possible instance unless: (i) it is a maybe x-tuple, and (ii) none of its alternatives has a nonempty lineage that would have been consistent with condition 2.

Example 3.4. We explain the possible instances of the ULDB in Example 3.2. Consider the choices for x-tuple 21 of Saw, which has two alternatives and is a maybe x-tuple. The possible instance that picks (21,1) must also have (41,1) and (43,1) to satisfy condition 3 in Definition 3.3, and it cannot have (42,1) or condition 2 would be violated. Similarly, the possible instance that picks (21,2) must have (42,1) but not (41,1) or (43,1). The possible instance that doesn't pick any alternative for x-tuple 21 has neither of (41,1) or (42,1), nor (43,1) by condition 2. Note that since (23,1) and (34,1) are always present, all possible instances have tuple (44,1) to satisfy condition 3. This gives us the three possible instances we expect. Note in particular that not all combinations of the maybe x-tuples in Accuses are included in the possible instances. □

3.1 Completeness

As discussed earlier, *completeness* is one of the important measures for the expressive power of a formalism for uncertainty. In general, a formalism is complete if it is possible to represent any set of possible instances within the formalism. Extending the traditional notion of completeness for ULDBs, we consider a stronger definition that includes both uncertainty and lineage. The following theorem shows that ULDBs are indeed complete.

Theorem 3.5. Given any set of possible LDBs $P = \{P_1, P_2, \ldots, P_m\}$ over relations $\overline{R} = \{R_1, R_2, \ldots, R_n\}$, there exists a ULDB $D = (\overline{R}, S, \lambda)$ whose possible LDBs are P. □

A formal proof for this theorem (and for all other theorems in the paper) are presented in the appendix.

3.2 Well-Behaved Lineage

Although the formal definition of a ULDB allows an arbitrary lineage function λ, in practice tuples are derived as results of queries, data imports, and other activities. Therefore, we expect λ to have a restricted structure and not be an arbitrary function. As a simple example, we don't expect to have a tuple t_1 derived from t_2 and also t_2 derived from t_1.

We define an interesting restricted class of lineage that we call *well-behaved* lineage. We will see that this class is closed under many relational operations, and its properties yield efficient algorithms for them. Let λ^* denote the transitive closure of lineage function λ.

Definition 3.6 (Well-Behaved Lineage)**.** The lineage of an x-tuple t_i is *well-behaved* if it satisfies the following three conditions:

1. **Acyclic:** $\forall s_{(i,j)}$, $s_{(i,j)} \notin \lambda^*(s_{(i,j)})$
2. **Deterministic:** $\forall s_{(i,j)}, s_{(i,j')}$, if $j \neq j'$ then either $\lambda(s_{(i,j)}) \neq \lambda(s_{(i,j')})$ or $\lambda(s_{(i,j)}) = \emptyset$
3. **Uniform:** $\forall s_{(i,j)}, s_{(i,j')}$, $B(s_{(i,j)}) = B(s_{(i,j')})$, where $B(s_{(i,j)}) = \{t_k | \exists s_{(k,l)},\ s_{(k,l)} \in \lambda(s_{(i,j)})\}$

We say that a ULDB $D = (\overline{R}, S, \lambda)$ is well-behaved if all its x-tuples have well-behaved lineage. □

Informally, Definition 3.6 says lineage is well-behaved when: (1) there are no cycles; (2) all alternatives of an x-tuple have distinct lineage; and (3) their lineage points to alternatives of the exact same set of x-tuples.

Let *base x-tuples* be defined as all x-tuples with empty lineage. An interesting and useful property of well-behaved lineage is that the possible instances of a well-behaved ULDB are determined entirely by the base x-tuples. That is, selecting a set of alternatives for base x-tuples determines which alternatives are selected for all x-tuples derived from them.

Theorem 3.7 (Well-Behaved ULDB)**.** For two possible instances D_1 and D_2 of a well-behaved ULDB $D = (\bar{R}, S, \lambda)$, $D_1 = D_2$ if and only if D_1 and D_2 have the same set of alternatives chosen for all base x-tuples. □

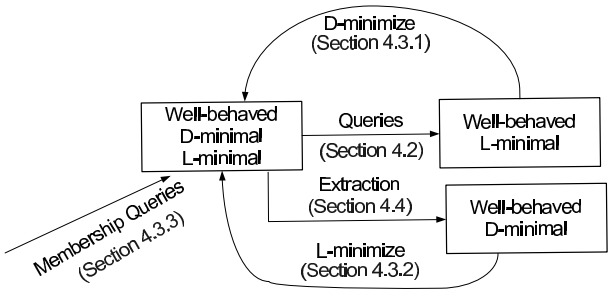

Figure 1: ULDB States and Queries

Recall that proofs of all theorems appear in the appendix.

Unless otherwise specified, we assume well-behaved ULDBs for the rest of the paper. We will soon see that if we start from a well-behaved ULDB and perform a standard set of relational operations creating the natural lineage for the results, the ULDB remains well-behaved.

4 Querying ULDBs

In this section we consider queries and operations we can perform on ULDBs. We begin (Section 4.2) by considering the case in which the result of a query also includes the original database, and we describe standard relational operations under this assumption. As noted earlier, because we are tracking lineage, we cannot look at an x-relation in a ULDB in isolation of others. Hence, we consider the *extraction* problem (Section 4.4), where the goal is to return only the relation that is the answer to the query (or more generally, a set of x-relations), without the original database. The challenge here is to extract the appropriate lineage along with the result x-relation, so that the correct set of possible instances is preserved.

The computation and representation of query answers (though not the possible instances) can depend on whether the input and the output are minimal. In Section 4.3 we define two notions of minimality for ULDBs: (1) *D-minimality*, guaranteeing that a ULDB does not contain extraneous data, and (2) *L-minimality*, guaranteeing that a ULDB does not contain extraneous lineage. We discuss both types of minimization, and we typically apply our query operations on the minimal forms. We show how minimization enables efficient answering of *membership queries*, where the goal is to determine whether a particular tuple (or set of tuples) is guaranteed to be in some (or all) possible instances of a ULDB.

Figure 1 summarizes the different operations (querying, extraction, and minimization) we consider for ULDBs, and the possible transitions between states of the ULDB. The remainder of this section proceeds as follows. In Section 4.1 we introduce the class of queries we consider, and in Section 4.2 we explain how these queries are processed against a ULDB. Section 4.3 defines ULDB minimality and discussesalgorithms for minimization. Finally, Section 4.4 explains how to correctly extract a set of x-relations from a ULDB.

4.1 DL-Monotonic Queries

We will restrict our discussion to queries that are *monotonic* with respect to data and lineage. To define monotonicity, we must first define containment of LDBs. Intuitively, for an LDB D to be contained in D', every data element and its transitive "lineage graph" in D should also be in D'.

Definition 4.1. Let $D = (\bar{R}, S, \lambda)$ and $D' = (\bar{R}', S', \lambda')$ be two LDBs, where $\bar{R}$ and $\bar{R}'$ have the same schemas. We say that D is *contained* in D', denoted $D \subseteq D'$, if:

1. $S \subseteq S'$
2. $\bar{R}$ is contained in $\bar{R}'$, i.e., if $t \in R_i$ then $t \in R'_i$, with the same tuple identifier
3. For every symbol $s_1 \in S$, if $s_2 \in \lambda(s_1)$, then $s_2 \in \lambda'^*(s_1)$. □

Note that $\subseteq$ is not exactly a partial order on LDBs because it is not antisymmetric. Specifically, $D \subseteq D'$ and $D' \subseteq D$ only implies that $\lambda^* = \lambda'^*$, not necessarily that $\lambda = \lambda'$.

Based on Definition 4.1, we define the class of *DL-monotonic queries*. In the definition, given a query Q and an LDB D, $Q(D)$ is an LDB that extends D with one x-relation R_q and with lineage λ_{R_q} from R_q to $I(\bar{R})$. We write $Q(D) = D + (R_q, I(R_q), \lambda_{R_q})$.

Definition 4.2. Let D be an LDB. Let $D|_I$ denote the restriction of D to the tuples identified in set I, and the lineage among them. A DL-monotonic query is a function Q from LDBs to LDBs that satisfies the following conditions:

1. $\forall t \in R_q, Q(D|_{\lambda(t)}) = D|_{\lambda(t)} + (t, I(t), \lambda(t))$, and no strict subset of $D|_{\lambda(t)}$ produces t.
2. $\forall D, D'$ such that $D \subseteq D'$, $Q(D) \subseteq Q(D')$. □

The first condition constrains the lineage of a result tuple to be a minimal subset of the database that produces exactly that tuple, and the second condition enforces monotonicity on both data and lineage.

Example 4.3. In Example 2.2, the query $\texttt{Accuses} = \pi_{\texttt{witness,person}}(\texttt{Saw} \bowtie \texttt{Drives})$ is DL-monotonic. In particular the reader may verify that the lineage associated with the four x-tuples of Accuses satisfies Definition 4.2 above. Note that the lineage of each of the two (Amy,Jimmy) tuples must have a distinct combination of base tuples so that condition 2 of Definition 4.2 is satisfied. □

Intuitively, any operation that can produce its results in a "tuple-by-tuple" fashion is DL-monotonic. Considering the standard relational operations, multiset selection, projection, join, and union are all DL-monotonic, and so are any queries composed from them. Aggregation, duplicate-elimination, and some set operators are not DL-monotonic. In the remainder of this section, we assume all

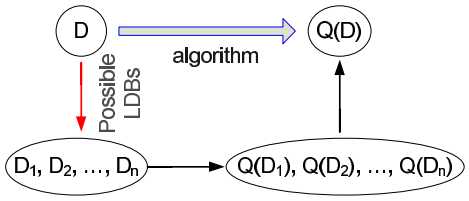

Figure 2: Semantics of Queries on ULDBs.

input: a ULDB D with x-relations $\{R_1, \ldots, R_n\}$, and a query Q on D
output: a ULDB $D' = Q(D)$
1: $R_q \leftarrow \emptyset$; $\lambda_{R_q} \leftarrow$ undefined function
2: Let $\overline{D} = \overline{R_1}, \ldots, \overline{R_n}$ be the LDB such that $\forall i$, $\bar{R}_i = \{$tuples $s_{(i,j)} | s_{(i,j)}$ is an alternative in $R_i\}$
3: Compute $Q(\bar{D}) = \bar{D} + (\bar{R}_q, I(\bar{R}_q), \lambda_{\bar{R}_q})$
4: Group the tuples in $\bar{R}_q$ by the x-tuple identifiers corresponding to the tuples in their lineage
5: **for** each group of x-tuple identifiers $t_1, \ldots, t_n$ **do**
6: create a maybe x-tuple t_l in R_q with all the tuples of the group as alternatives
7: $\forall s_{(l,k)}$ alternative of t_l, set $\lambda_{R_q}(s_{(l,k)})$ as in $\lambda_{\bar{R}_q}$
8: **end for**
9: **return** $D' = D + (R_q, I(R_q), \lambda_{R_q})$

Algorithm 1: Query Evaluation

queries Q to be DL-monotonic. In follow-on work we are extending our approach to other operations, as discussed briefly in Sections 5.3 and 7.

4.2 Applying a Query to a ULDB

We consider the problem of applying a query Q to a ULDB D, where the result $Q(D)$ is defined to include the original database and the answer relation. Query semantics are defined in terms of possible instances (see Figure 2): $Q(D)$'s possible LDBs are logically obtained by applying Q to each of the $D_1, \ldots, D_n$ possible instances of D. We now present an algorithm for evaluating Q directly on the ULDB representation, shown as the broad arrow in Figure 2.

Algorithm 1 (see figure) proceeds in two phases. First (lines 4–5), it performs a "standard" evaluation of the query Q on an LDB $\overline{D}$ that contains all the alternatives of the base x-relations. The resulting relation $\overline{R_q}$ and its lineage λ_{R_q} are then used to: (a) construct one x-tuple t_l in R_q for each combination $t_1, \ldots, t_n$ of x-tuples in D that produced tuples through Q (lines 6–8); and (b) generate lineage for t_i's alternatives (line 9). Note that although t_l is defined as a maybe x-tuple, it may still contribute a tuple in every possible LDB of $Q(D)$. We discuss elimination of extraneous '?'s in Section 4.3.1.

Theorem 4.4. Given a ULDB D and a query Q:

1. Algorithm 1 returns $Q(D)$.
2. If D is a well-behaved ULDB, then so is $Q(D)$. □

Observe that our algorithm is based on evaluating Q over a conventional database $\overline{D}$. Since the size of $\overline{D}$ is the same as the size of x-relations $R_1, \ldots, R_n$, complexity does not increase due to uncertainty. More importantly, we can implement Algorithm 1 readily using a standard relational DBMS, without having to build a special-purpose query engine for ULDBs. In fact, our initial implementation of the *Trio* prototype ULDB has taken exactly this approach [3, 7]. Of course special-purpose techniques also may be interesting in order to maximize performance of query processing on ULDBs.

4.3 ULDB Minimality

We now define two notions of minimality for ULDBs: *data minimality* and *lineage minimality*.

4.3.1 Data Minimality

As the following example illustrates, a ULDB may contain extraneous data, including "impossible" alternatives in an x-tuple, or x-tuples unnecessarily marked with '?'. As a special case, an entire x-tuple is extraneous if all its alternatives are extraneous.

Example 4.5. In Example 3.2, the '?' on x-tuple 44 is extraneous because the alternative (24,1) is present in every possible LDB. As an example of an extraneous alternative (entire x-tuple in this case), consider the following x-relations, where `Car1` and `Car2` represent separate lists of possible crime vehicles.

ID	Saw(witness, car)
1	(Carol,Acura) || (Carol,Lexus)

ID	Car1(car)
2	Acura

ID	Car2(car)
3	Lexus

Suppose we perform `Saw1 = (Car1 ⋈ Saw)` and `Saw2 = (Car2 ⋈ Saw)` to get sightings related to the two car lists:

ID	Saw1(witness,car)
4	(Carol,Acura)

λ(4,1)={(1,1),(2,1)}

ID	Saw2(witness,car)
5	(Carol,Lexus)

λ(5,1)={(1,2),(3,1)}

Finally, suppose we compute $(\texttt{Saw1} \bowtie_{\texttt{witness}} \texttt{Saw2})$ to find pairs of car sightings in `Car1` and `Car2` by the same witness:

ID	(witness,car1,car2)
6	(Carol,Acura,Lexus)

? λ(6,1)={(4,1),(5,1)}

There is no possible instance of the database with alternative (6,1). Intuitively, Carol saw either an Acura or a Lexus, while both sightings would be necessary to derive x-tuple `(Carol,Acura,Lexus)`. Thus, `(Carol,Acura,Lexus)` is extraneous. □

We now define data minimality formally.

Definition 4.6 (D-Minimality)**.** An alternative (i, j) of an x-tuple t_i in a ULDB D is said to be *extraneous* if removing it from the x-relation does not change the possible instances of D. Similarly, a '?' on an x-tuple in D is said to be extraneous if removing it does not change the possible instances of D. A ULDB D is *D-minimal* if it does not include any extraneous alternatives or '?'s. □

The following theorems provide conditions on ULDBs that enable us to detect extraneous data.

Theorem 4.7 (Extraneous Alternative)**.** Let D be a well-behaved ULDB. An alternative with identifier (k, l) (in x-tuple t_k) in D is extraneous if and only if there exist $s_{(i,j_1)}, s_{(i,j_2)} \in \lambda^*(s_{(k,l)})$, with $j_1 \neq j_2$. □

In other words, an alternative is extraneous if and only if it has contradictory lineage.

In the next theorem, let $\eta(t_i)$ denote the number of alternatives in x-tuple t_i that are not extraneous. Let $h(t_i)$ denote the set of base x-tuples from which t_i is derived, i.e., $t_j \in h(t_i)$ if $\exists s_{(i,k)}, s_{(j,l)}$ such that $s_{(j,l)} \in \lambda^*(s_{(i,k)})$ and $\forall m, \lambda(s_{(j,m)}) = \emptyset$.

Theorem 4.8 (Extraneous '?')**.** Let D be a well-behaved ULDB. A '?' on an x-tuple $t \in D$ is extraneous if and only if:

1. No x-tuple in $h(t)$ has a '?'
2. $\eta(t) = \prod_{t' \in h(t)} \eta(t')$ □

We can now use Theorems 4.7 and 4.8 to D-minimize ULDB representations. Minimization needs to work on the transitive closure λ^* of the lineage, which presents two approaches to D-minimization: (1) a *lazy* approach in which λ^* is computed during minimization, and (2) an *eager* approach in which the algorithm for operations maintains λ^* and also the D-minimal form. Algorithm 2 presents the lazy approach for D-minimizing a ULDB D; the eager approach uses the same idea but performs the computation incrementally with operations. It is easy to see that the algorithm returns the D-minimal representation.

4.3.2 Lineage Minimality

A second notion of minimality has to do with lineage. For ULDB $D = (\bar{R}, S, \lambda)$, let its *internal lineage* be the restriction of λ to only symbols in $I(\bar{R})$. (Recall the domain of symbols $S = I(\bar{R}) \cup E$ also includes external symbols E.)

Definition 4.9 (L-minimal ULDB)**.** A ULDB $D = (\bar{R}, S, \lambda)$ is *L-minimal* if for any $D' = (\bar{R}, S', \lambda')$ over the same x-relations $\bar{R}$ such that:

1. $S' \subseteq S$, $\lambda'^* \subseteq \lambda^*$
2. D and D' have the same internal lineage

D' has the same possible instances as D only if $S' = S$ and $\lambda'^* = \lambda^*$. □

```
1: input: ULDB D
2: output: equivalent but D-minimized version of D
3: for each x-relation R in D do
4:     Skip if R has been D-minimized
5:     Recursively perform Steps 3-8 to D-minimize all
       x-relations {R_1, R_2, ..., R_n} that contain lineage
       of data in R.
6:     Compute λ* for each alternative of R using the al-
       ready computed λ* for each R_i
7:     Delete all extraneous alternatives using the condi-
       tion of Theorem 4.7
8:     Compute η(t) for all x-tuples t in R and for all x-
       tuples in h(t)
9:     Use the condition in Theorem 4.8 to delete any ex-
       traneous '?'s
10:    Mark R as D-minimized
11: end for
12: return D
```

Algorithm 2: Lazy Algorithm for D-minimization

We have the following main theorem about L-minimality.

Theorem 4.10 (L-minimality of Algorithm 1)**.** Given a well-behaved L-minimal ULDB D and a query Q, the result $Q(D)$ of Algorithm 1 is an L-minimal ULDB. □

The above theorem guarantees that query processing preserves L-minimality. In general, "L-minimizing" a ULDB D, i.e., finding an L-minimal D' that coincides with D on data and internal lineage, is a tractable problem. However, the result of L-minimization is not unique. It is still open whether we can efficiently find a "global minimum" among all possible L-minimizations, with respect to the size of their representation. We plan to investigate this question in future work.

4.3.3 Membership Queries

One useful side-effect of minimization is that it helps us answer *membership queries* [2, 21, 26, 27, 28]: determining whether a particular tuple or relation is present in some (or every) possible instance of an uncertain database. In the context of ULDBs, these problems are defined as follows.

Definition 4.11 (Membership Queries)**.**

- *Tuple Membership (resp. Certainty):* Given a ULDB D containing a relation R, and given a tuple t, determine whether $t \in R$ in some (resp. all) possible instance(s) of D.
- *Instance Membership (resp. Certainty):* Given a ULDB D containing a relation R, and a multiset T of tuples, determine whether R contains exactly the tuples of T in some (resp. all) possible instance(s) of D. □

The following theorem shows that it is tractable to answer both of the tuple-membership problems. The algorithms to do so (included in the proof) build directly

on D-minimization. However, as is true of all complete uncertainty models [21] including ULDBs, the instance-membership problems are intractable.

Theorem 4.12. Let D be a well-behaved ULDB.

1. The tuple-membership and tuple-certainty problems are solvable in polynomial time in the size of D.
2. The instance-membership and instance-certainty problems are NP-hard. □

4.4 Extraction

Typically, after issuing a query to a database, users are interested in seeing only the result relation, not the entire database. More generally, given a ULDB, we may want to extract a subset of its relations, but in a way that preserves the possible instances of the extracted subset. In principle, whenever a database includes constraints across relations, extracting a subset of the database is an interesting question; otherwise, the meaning of every relation is independent of the others, and therefore extraction is trivial.

Definition 4.13 (Extraction)**.** Let D be a well-behaved ULDB with x-relations $\overline{R}$ and possible instances P, and let $\bar{X}$ be a subset of $\overline{R}$. The problem of extracting $\bar{X}$ from $\overline{R}$ is to return a well-behaved ULDB D' with $\overline{R'} = \bar{X}$ and possible instances P', such that the restriction of P to $\bar{X}$ equals P' with respect to data and internal lineage.

Simply removing the relations in $\overline{R} - \bar{X}$ and their symbols does not give a correct extracted result. For instance, if the x-relation Accuses from Example 3.2 is extracted without any lineage, x-tuple 43 may now occur without x-tuple 41, which is not allowed by any of the possible instances of the original ULDB.

The following short but dense algorithm produces the correct extraction.

1: **input:** ULDB $D = (\bar{R}, S, \lambda)$, and $\bar{X} \subseteq \bar{R}$
2: **output:** a ULDB $D' = (\bar{X}, S', \lambda')$
3: $S' = I(\bar{X}) \cup (\bigcup_{x \in I(\bar{X})} \lambda^*(x))$
4: $\lambda' = \lambda|_{S'}$, the restriction of λ to S'
5: **return** D'

Effectively, the algorithm works by identifying all lineage that is necessary to ensure that the possible instances of the extracted relations are preserved. Lineage that is not within the extracted relations is converted from internal (identifiers (i, j) in $I(\bar{R})$) to external (the corresponding symbols $s_{(i,j)}$). Note that by our definitions, the mutual exclusion of x-tuple alternatives carries over to what are now external symbols. One subtlety is that we must associate a logical '?' with each set of external symbols that were created from an x-tuple having a '?'.

Consider again the Accuses example discussed above. If we extract Accuses from the database shown in Example 3.2, we retain the lineage on the x-tuples of Accuses, except it now refers to external symbols. By doing so, Definition 3.3 of possible instances correctly prohibits a possible instance containing one but not the other of x-tuples 41 and 43.

We have the following theorem about our extraction algorithm.

Theorem 4.14. Let $D = (\bar{R}, S, \lambda)$ be a well-behaved D-minimal ULDB, and consider any $\bar{X} \subseteq \bar{R}$.

1. The extraction algorithm returns a correct extraction D'.
2. The extraction algorithm runs in polynomial time in the size of D.
3. The result D' is D-minimal. □

5 Confidences and Probabilistic Data

As a final contribution, we show how ULDBs can be extended to include *confidence values* and probabilistic query processing. With confidences, ULDBs subsume the typical notion of *probabilistic databases*, which assign a confidence value to tuples, without alternatives or lineage [5, 12, 19, 31]. A noteworthy feature of probabilistic query processing using ULDBs is that we can decouple the computation of data in query results from the computation of the data's probability (confidence) values. This decoupling enables more freedom with query plan selection than is typically available for probabilistic query processing [20], and it allows confidence values to be computed selectively as needed.

5.1 Confidence Values

In the remainder of this section we assume ULDBs to be well-behaved and D-minimized. If we consider the semantics of x-relations probabilistically, then without lineage different alternatives of the same x-tuple represent *disjoint* events, while different x-tuples represent *independent* events. Recall from Section 3.2 that in well-behaved ULDBs, the possible instances are determined entirely by the choices for the base x-tuples; the choices for derived x-tuples are determined by their lineage.

We preserve this intuition when extending ULDBs with confidences. Now, each base alternative a has an associated *confidence value* $c(a)$. For each base x-tuple t, the sum $\sigma(t)$ of the confidence values of its alternatives must be at most 1, and exactly 1 if t has no '?'. The confidence of '?' for any x-tuple is $(1 - \sigma(t))$. When we map to possible instances, each instance has a *probability* of being the "correct" instance, based on confidences in the data comprising the instance: The probability of a possible instance is the product of the confidences of the base alternatives and '?' chosen in it.

Example 5.1. Suppose Amy sighted an Acura with confidence 0.8, while Betty is sure she saw either an Acura or a Mazda with confidences 0.4 and 0.6 respectively. Furthermore, Hank drives an Acura with confidence 0.6. We have:

ID	Saw(witness, car)	
11	(Amy,Acura):0.8	?
12	(Betty,Acura):0.4 ‖ (Betty,Mazda):0.6	

ID	Drives(person, car)	
51	(Hank,Acura):0.6	?

This database has eight possible instances, since each of the three x-tuples has two possible choices. For example, the possible instance where Amy saw an Acura, Betty saw a Mazda, and Hank does not drive an Acura has confidence $0.8 * 0.6 * (1 - 0.6) \approx 0.20$. □

It can be shown that for any well-behaved D-minimal ULDB with confidences, the following desirable properties hold.

1. The sum of probabilities of its possible instances is 1.
2. The confidence of a base alternative a (resp. '?' on an x-tuple t) equals the sum of the confidences of the possible instances where a (resp. no alternative of t) is picked.

5.2 Query Processing

The presence of lineage allows us to decouple ULDB query processing with confidences into two steps:

1. *Data computation*, in which we compute the data and lineage in query results, just as in ULDBs without confidences
2. *Confidence computation*, in which we compute confidence values for query results based on their lineage (and confidence values on base data)

We first motivate why this decoupling works. Then we briefly discuss confidence computation in Section 5.3 and data computation in Section 5.4. Overall, the topic of ULDB query processing with confidences is a rich and interesting one, and the subject of considerable ongoing work.

Suppose we have a derived x-tuple t, and consider one of its alternatives a. With well-behaved lineage, a appears in a possible instance if and only if all of the base x-tuple alternatives in the transitive closure of a's lineage appear in the instance. Furthermore, these base x-tuple alternatives are *independent*, since they have no lineage of their own and cannot be alternatives of the same x-tuple. Thus, the confidence of a is computed as the product of the confidences of the base-tuple alternatives in the transitive closure of its lineage. For an x-tuple t with a '?', confidence for the '?' is $(1 - \sigma(t))$, where $\sigma(t)$ is the sum of the confidences of t's alternatives.

Thus, the confidence value for every result alternative a is a function of the confidence values for the base alternatives reachable by a's transitive lineage. Hence we need not compute confidence values during query processing—we can compute them afterwards using the lineage on query results together with the original base data confidences.

Next, we show how decoupling data and confidence computation overcomes a previously identified shortcoming of query processing in probabilistic databases, and we briefly discuss efficient confidence computation in the decoupled scenario.

5.3 Confidence Computation

Dalvi and Suciu [19] show that naive propagation of confidences during query processing—essentially assuming independence of tuples in intermediate results—may lead to incorrect confidences in the result. We illustrate the problem with an example, and also show how our decoupled technique operates (correctly) on the same example.

Example 5.2. Let us simplify the data in Example 5.1 to:

ID	Saw(witness,car)	
11	(Amy,Acura):0.8	?
12	(Betty,Acura):0.4	?

ID	Drives(person,car)	
51	(Hank,Acura):0.6	?

Suppose we want the list of accused persons with confidences: $\texttt{Accused} = \Pi_{\texttt{person}}\,\texttt{(Saw} \bowtie \texttt{Drives)}$. Here we are using a duplicate-eliminating projection. We consider three ways of executing this query: two query plans that compute confidences as part of operator execution, and a third method showing our decoupled approach.

Query Plan 1 (correct): Evaluating the query using the following plan gives the correct confidences in the result:

$$\Pi_{\texttt{person}}(\Pi_{\texttt{car}}(\texttt{Saw}) \bowtie \texttt{Drives})$$

In $\Pi_{\texttt{car}}(\texttt{Saw})$, there is just one tuple (Acura) whose confidence is given by:

$$\begin{aligned}&\Pr((11,1) \vee (12,1))\\ &= \Pr((11,1)) + \Pr((12,1)) - \Pr((11,1) \wedge (12,1))\end{aligned}$$

Since alternatives $(11,1)$ and $(12,1)$ are independent, $\Pr((\texttt{Acura}))$ evaluates to $0.8 + 0.4 - (0.8 * 0.4) = 0.88$. Now joining (Acura) with x-tuple 51, we get the confidence of the result (Hank,Acura) to be $0.88 * 0.6 = 0.528$. In the final step, projecting onto person, the confidence remains 0.528.

Query Plan 2 (incorrect): Suppose instead we use plan:

$$\Pi_{\texttt{person}}\,\texttt{(Saw} \bowtie \texttt{Drives)}$$

Now we get an incorrect result, because the intermediate x-tuples (Amy,Acura,Hank) and (Betty,Acura,Hank) from (Saw ⋈ Drives) are not independent. Let these tuples have IDs $(61,1)$ and $(62,1)$ respectively. The confidence of (Amy,Acura,Hank) is:

$$\Pr((61,1)) = \Pr((11,1) \wedge (51,1))$$

giving $0.8 * 0.6 = 0.48$. Similarly, the confidence of (Betty,Acura,Hank) is $0.6*0.4 = 0.24$. Now the x-tuple Hank after projecting onto person has confidence given by

$$\Pr((61,1) \vee (62,1))$$
$$= \Pr((61,1)) + \Pr((62,1)) - \Pr((61,1) \wedge (62,1))$$

Assuming independence of tuples $(61,1)$ and $(62,1)$, the confidence evaluates to $0.48 + 0.24 - 0.48 * 24 = 0.6048$, which is incorrect. See [19] for further discussion of these issues. □

Query Plan 3 (decoupled approach): In our approach, we first compute the query result using any execution plan. We get the one x-tuple (Hank); let its identifier be $(71,1)$. Because of the duplicate-elimination operator, which is not DL-monotonic, $\lambda((71,1))$ is no longer a set of tuple alternatives (indicating conjunction), but rather a boolean formula over alternatives. (Disjunctive and negative lineage is required once we go beyond the DL-monotonic operations; details are the subject of ongoing work.) Specifically, $\lambda((71,1)) = ((51,1) \wedge ((11,1) \vee (12,1)))$.

Now, we compute the confidence of the (Hank) tuple based on its lineage formula and confidence values for the (independent) base alternatives:

$$\Pr((71,1)) = \Pr(((51,1) \wedge ((11,1) \vee (12,1)))$$

With $\Pr((51,1)) = 0.6$, $\Pr((11,1)) = 0.8$, and $\Pr((12,1)) = 0.4$, we obtain the correct result $\Pr((71,1)) = 0.528$. □

Our decoupled approach has two important advantages: First, the data computation step has the flexibility to use the most efficient execution plan, without worrying about plans that produce incorrect confidences as illustrated above. Second, in the case where confidence values may not be required for all data in all query results, the values can be computed selectively and on-demand. Further discussion of both of these points appears in the next subsection.

Of course we do incur some overhead when confidences are finally computed, particularly if we follow the most naive approach of tracing the entire lineage of each result x-tuple alternative to obtain the base data confidences. We have several ideas for optimizing the confidence computation:

- The confidence value for a derived alternative can be computed from confidence values for a set of "closest independent descendents" (CIDs) for the alternative, rather than from confidence values on base data. Roughly, the CID of an alternative a is a minimal set S of alternatives in a's transitive lineage such that the alternatives in S do not share a common base alternative in their transitive lineage. It can be shown that CIDs are unique, and for more complex types of lineage, recursive computation of confidence values based on CIDs can be much cheaper than not using CIDs.
- CIDs also enable *memoization*, which avoids performing redundant confidence computations. Memoization can be useful within the computation for a single alternative, as well as across confidence computations, as long as intervening updates don't alter the relevant lineage or confidences.
- If transitive lineage λ^* is already being maintained for eager D-minimization (Section 4.3.1), it can then also be applied to considerably speed up confidence computations.
- So far we have discussed computing the confidence value for a single alternative. In the case where we wish to compute confidences for an x-tuple or an entire x-relation, batch techniques can be used based on the structure guaranteed by well-behaved lineage.

All of these topics are the subject of ongoing work.

5.4 Data Computation

To avoid the erroneous confidence calculations as exhibited in Example 5.2, reference [19] characterizes logical query plans that are guaranteed to propagate confidences correctly, and restricts their evaluation strategies to such plans. In our decoupled approach, we have the luxury of a wider space of plans, which can be shown to result in arbitrarily large performance improvements (confidence computation included) in extreme cases. Consider a query Q that produces an empty result. Our approach does not need to perform any confidence computation for Q since there are no result x-tuples. The alternative approach computes confidences during query execution until finally the result is discovered to be empty. Furthermore, an expensive plan may need to be used in order to correctly compute confidence values that are eventually thrown away.

More concretely, suppose we have $2n$ large relations, $R_1(X)$, ..., $R_n(X)$ and $S_1(Z)$, ..., $S_n(Z)$, and two small relations $A(X,Y)$ and $B(Y,Z)$. Consider a query $Q(Y)$ that computes the natural join of all the relations and projects onto Y, and suppose $A \bowtie B$ is empty. With simple statistics any standard optimizer will choose to perform $A \bowtie B$ first. However, in the plans permitted by [19] (or any other plans that require independence of tuples for confidence propagation), $A \bowtie B$ must be performed last. In these plans, we can make the cost of computing $R_1 \bowtie \ldots \bowtie R_n$ and $S_1 \bowtie \ldots \bowtie S_n$ arbitrarily large.

Of course this example was contrived, and reference [19] shows that for some queries, computing results with confidences has #P-hard data complexity, regardless. In such situations, our decoupled approach offers a practical solution: Answers without confidence values give an approximation of the result, and their lineage can be used to selectively compute confidence values for tuples of interest. If the latter is still too expensive, we can use

approximate techniques like the Monte-Carlo simulations proposed in [30] to estimate the confidences.

6 Related Work

In [40] we described the original motivation that led to the work in this paper: development of a general-purpose database management system that incorporates data, lineage, and uncertainty. In [21] we explored the space of incomplete and complete models for uncertainty, without considering lineage. In [22] we posed and solved a number of new theoretical problems with respect to representation schemes for uncertainty, again without lineage.

We are not aware of any previously proposed formal data representation that integrates both lineage and uncertainty. We briefly overview some of the work that addresses uncertainty and lineage independently.

Representation schemes and query answering for uncertain databases has been studied extensively, e.g., [2, 5, 6, 10, 23, 26, 27, 28, 31, 39]. Much of this previous work is theoretical, but there has been recent interest in building systems, e.g., [9, 13, 40] for uncertainty, and [29, 37] for integrating inconsistent data sources. Query answering in probabilistic databases has seen considerable progress and efficient solutions have been proposed [18, 19, 20]. We build on that work in this paper, showing how lineage can further improve query processing.

Approximate query answering has also received significant attention over the last decade [4, 25, 24, 38], but we focus on exact queries over uncertain data rather than inexact queries over certain data. However, the simple representation of uncertainty in ULDBs is likely to facilitate approximate querying, and we plan to investigate this avenue of future work.

Integrating lineage (also known as *provenance*) has been proposed for relational databases, e.g., [11, 34, 35], and for data warehouses, e.g., [15, 16, 17]. It has been observed that there are various choices in defining lineage, and in this paper we use a definition similar to the *where lineage* of [11]. Analysis of possible lineage information was also used for optimizing query evaluation and determining independence of queries from updates [33]. A recent system being developed around data provenance is described in [8, 14].

7 Conclusions and Future Work

We introduce ULDBs as a representation for databases with both lineage and uncertainty. With simple extensions to the relational model (tuple alternatives, maybe tuples, and lineage functions), ULDBs can represent any finite set of possible instances containing data and lineage, and ULDBs are amenable to efficient query processing using standard relational techniques. ULDBs can be extended naturally to represent and query probabilistic data; moreover, because lineage enables query evaluation to be decoupled from the computation of confidences, substantial performance gains may be achieved over computing query operators and confidences in tandem.

In this paper we focused on a specific class of DL-monotonic queries and their lineage. We are extending our techniques and results to a larger set of operations, e.g., duplicate-elimination, aggregation, and negation. Doing so primarily entails extending the types of lineage allowed, e.g., adding disjunctive and negative lineage, as briefly shown in Section 5.3.

We are building a system called *Trio* based on ULDBs, currently implemented on top of a standard relational DBMS [3, 7]. Through simple rewriting techniques, Trio evaluates DL-monotonic queries on ULDBs without altering any system internals. However, new techniques are required if we are to handle all aspects of ULDBs covered in this paper, e.g., keeping a ULDB D-minimized as query results are added, and efficiently L-minimizing the result of an extraction.

We are currently exploring a number of other challenges related to query processing in ULDBs with confidences. In particular, we are studying various algorithms and optimizations when computing confidences, such as memoization and minimizing lineage traversal. We are also studying eager versus on-demand confidence computation, incremental propagation of confidence updates, and "top-K" and ordering queries based on confidences.

There are a number of other current and future directions of work in ULDBs:

- **Updates:** We are currently identifying a set of update primitives for ULDBs, and considering the design of efficient update algorithms.
- **Implementation:** ULDBs introduce several new physical design issues, such as data layout, indexing, partitioning, and materialized views, and their integration into query optimization. Fully exploring these topics is likely to entail modifying our prototype to operate inside (instead of on top of) a DBMS.
- **Theory:** There are numerous interesting theoretical problems to work on. We can reconsider nearly every topic in relational database theory in the context of ULDBs, e.g., dependency theory, query containment, and sampling and statistics.
- **Long-Term Goals:** Our agenda for the overall Trio project [40] includes several features not yet present in ULDBs, such as uncertainty in the form of continuous distributions, incomplete relations, and versioning of data, uncertainty, and lineage.

Acknowledgments

We thank Parag Agrawal, Dan Suciu, Jeff Ullman, and the entire Trio group for helpful discussions, and Chris Hayworth for creating an initial prototype implementation of ULDBs.

References

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

[2] S. Abiteboul, P. Kanellakis, and G. Grahne. On the Representation and Querying of Sets of Possible Worlds. *Theoretical Computer Science*, 78(1), 1991.

[3] P. Agrawal, O. Benjelloun, A. Das Sarma, C. Hayworth, S. Nabar, T. Sugihara, and J. Widom. Trio: A system for data, uncertainty, and lineage. In *Proc. of VLDB*, 2006. Demonstration description.

[4] S. Agrawal, S. Chaudhuri, G. Das, and A. Gionis. Automated Ranking of Database Query Results. In *Proc. of CIDR*, 2003.

[5] D. Barbará, H. Garcia-Molina, and D. Porter. The Management of Probabilistic Data. *IEEE Trans. Knowl. Data Eng.*, 4(5), 1992.

[6] R. S. Barga and C. Pu. Accessing Imprecise Data: An Approach Based on Intervals. *IEEE Data Engineering Bulletin*, 16(2), 1993.

[7] O. Benjelloun, A. Das Sarma, C. Hayworth, and J. Widom. An Introduction to ULDBs and the Trio System. *IEEE Data Engineering Bulletin*, 29(1), 2006.

[8] D. Bhagwat, L. Chiticariu, W. Tan, and G. Vijayvargiya. An annotation management system for relational databases. In *Proc. of VLDB*, 2004.

[9] J. Boulos, N. Dalvi, B. Mandhani, S. Mathur, C. Re, and D. Suciu. MYSTIQ: a system for finding more answers by using probabilities. In *Proc. of ACM SIGMOD*, 2005.

[10] B. P. Buckles and F. E. Petry. A Fuzzy Model for Relational Databases. *International Journal of Fuzzy Sets and Systems*, 7, 1982.

[11] P. Buneman, S. Khanna, and W. Tan. Why and where: A charaterization of data provenance. In *Proc. of ICDT*, 2001.

[12] R. Cavallo and M. Pittarelli. The theory of probabilistic databases. In *Proc. of VLDB*, 1987.

[13] R. Cheng, S. Singh, and S. Prabhakar. U-DBMS: A database system for managing constantly-evolving data. In *Proc. of VLDB*, 2005.

[14] L. Chiticariu, W. Tan, and G. Vijayvargiya. DBNotes: a post-it system for relational databases based on provenance. In *Proc. of ACM SIGMOD*, 2005.

[15] Y. Cui and J. Widom. Practical lineage tracing in data warehouses. In *Proc. of ICDE*, 2000.

[16] Y. Cui and J. Widom. Lineage tracing for general data warehouse transformations. *VLDB Journal*, 12(1), 2003.

[17] Y. Cui, J. Widom, and J. L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM TODS*, 25(2), 2000.

[18] N. Dalvi, G. Miklau, and D. Suciu. Asymptotic Conditional Probabilities for Conjunctive Queries. In *Proc. of ICDT*, 2005.

[19] N. Dalvi and D. Suciu. Efficient Query Evaluation on Probabilistic Databases. In *Proc. of VLDB*, 2004.

[20] N. Dalvi and D. Suciu. Answering Queries from Statistics and Probabilistic Views. In *Proc. of VLDB*, 2005.

[21] A. Das Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working Models for Uncertain Data. In *Proc. of ICDE*, 2006.

[22] A. Das Sarma, S. Nabar, and J. Widom. Representing uncertain data: Uniqueness, equivalence, minimization, and approximation. Technical report, Stanford InfoLab, 2005. Available at http://dbpubs.stanford.edu/pub/2005-38.

[23] N. Fuhr and T. Rölleke. A Probabilistic NF2 Relational Algebra for Imprecision in Databases. *Unpublished Manuscript*, 1997.

[24] N. Fuhr and T. Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM TOIS*, 14(1), 1997.

[25] Norbert Fuhr. A Probabilistic Framework for Vague Queries and Imprecise Information in Databases. In *Proc. of VLDB*, 1990.

[26] G. Grahne. Dependency Satisfaction in Databases with Incomplete Information. In *Proc. of VLDB*, 1984.

[27] G. Grahne. Horn Tables - An Efficient Tool for Handling Incomplete Information in Databases. In *Proc. of ACM PODS*, 1989.

[28] T. Imielinski and W. Lipski Jr. Incomplete Information in Relational Databases. *Journal of the ACM*, 31(4), 1984.

[29] Z. G. Ives, N. Khandelwal, A. Kapur, and M. Cakir. Orchestra: Rapid, collaborative sharing of dynamic data. In *Proc. of CIDR*, 2005.

[30] R. M. Karp and M. Luby. Monte-carlo algorithms for enumeration and reliability problems. In *Proc. of FOCS*, 1983.

[31] L. V. S. Lakshmanan, N. Leone, R. Ross, and V.S. Subrahmanian. ProbView: A Flexible Probabilistic Database System. *ACM TODS*, 22(3), 1997.

[32] A. Y. Levy, R. E. Fikes, and S. Sagiv. Speeding up inferences using relevance reasoning: A formalism and algorithms. *Artificial Intelligence*, 97(1-2), 1997.

[33] Alon Y. Levy and Yehoshua Sagiv. Queries independent of updates. In *Proc. of VLDB*, 1993.

[34] W. Tan P. Buneman, S. Khanna. Data provenance: Some basic issues. In *Proc. of FSTTCS*, 2000.

[35] W. Tan P. Buneman, S. Khanna. On propagation of deletions and annotations through views. In *Proc. of ACM PODS*, 2002.

[36] Y. Tao, R. Cheng, X. Xiao, W. K. Ngai, B. Kao, and S. Prabhakar. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In *Proc. of VLDB*, 2005.

[37] Nicholas E. Taylor and Zachary G. Ives. Reconciling while tolerating disagreement in collaborative data sharing. In *Proc. of ACM SIGMOD*, 2006.

[38] A. Theobald and G. Weikum. The XXL Search Engine: Ranked Retrieval of XML Data Using Indexes and Ontologies. In *Proc. of ACM SIGMOD*, 2002.

[39] M. Y. Vardi. Querying logical databases. In *Proc. of ACM PODS*, 1985.

[40] J. Widom. Trio: A System for Integrated Management of Data, Accuracy, and Lineage. In *Proc. of CIDR*, 2005.

A Proofs

Proof of Theorem 3.5: First construct $\overline{R}$ with x-relations S_1 through S_n, corresponding to R_1 through R_n, and an extra relations *PW(n)* that encodes the possible instances. PW contains exactly one x-tuple: `(1)|(2)|...|(m)`. Intuitively, the possible instance of D in which this tuple takes the value `(j)` encodes P_j.

Each S_i is constructed as follows. For every P_j, each tuple t in R_i forms a maybe x-tuple with just one alternative with value t. Duplicates within and across possible instances are preserved in S_i. We add `(j)` in *PW* to the lineage of alternatives in tuples copied from P_j. This now exactly encodes the data in each of the possible instances. The correct lineage is obtained as follows. We look at the lineage λ_j in P_j and mimic it in the x-tuples it contributes in S_1 through S_n. For example, if $\lambda_j(t_1) = \{t_2\}$ in P_j, where $t_1 \in R_1$ and $t_2 \in R_2$, the x-tuple that t_2 gave in S_2 is added to the lineage of the x-tuple from t_1 in S_1.

As a final step, we remove the extra relation *PW* but retain its symbols as external lineage. Therefore, each possible LDB of D now has the same schema as each P_j, and represents exactly the same data and internal lineage. □

Proof of Theorem 3.7: Define the distance of each alternative from base data as the maximum number of lineage links that can be traversed before reaching base data. The acyclicity property of well-formed lineage ensures that this distance of each alternative is finite.

Since $D_1 \neq D_2$, there exists an alternative (from a derived tuple) in D_1 that is not present in D_2. Consider one such alternative $s_{(i,j)}$ in D_1 whose distance to base data is minimum. D_2 must have some other alternative $s_{(i,j')}$ chosen. Note that because of the uniformity condition of well-formed lineage, the distance of two alternatives of the same x-tuple, here $s_{(i,j)}$ and $s_{(i,j')}$, to base data is the same.

We show that well-behaved conditions then give us another pair of alternatives closer to base data than $s_{(i,j)}, s_{(i,j')}$, and one alternative present in D_1 (and not D_2) and the other in D_2 (and not D_1).

Since $s_{(i,j)}$ is not a base alternative, $\lambda(s_{(i,j)}) \neq \emptyset$. By the determinism property of well-formed lineage, $\lambda(s_{(i,j)}) \neq \lambda(s_{(i,j')})$. Consider some $s_{(k,l)} \in \lambda(s_{(i,j)})$ but not in $\lambda(s_{(i,j')})$, therefore $s_{(k,l)}$ is not from a base tuple. By uniformity, there exists some $s_{(k,l')} \in \lambda(s_{(i,j')})$. We now have $s_{(k,l)}$ and $s_{(k,l')}$ closer to base data, thus violating our assumption that $s_{(i,j)}$ and $s_{(i,j')}$ were the closest pair of alternatives to base data chosen in D_1 and D_2 respectively. □

Lemma A.1. *Given a* ULDB $D = (\bar{R}, S, \lambda)$ *with possible* LDBs $D_1, \ldots, D_n$*, and a maybe x-tuple* t *not in* D *with well-behaved non-empty lineage* $\lambda(t) \subseteq I(\bar{R})$ *the* ULDB $D' = D + (\{t\}, I(t), \lambda(t))$ *has possible instances* $D'_1, \ldots D'_n$ *such that* $\forall i$*,* $D_i \subseteq D'_i$*. We say that* D' preserves the possible instances *of* D*.*

Proof. Any possible LDB P'_i of D' makes the same alternative choices for all x-tuples of D as one and only one possible LDB P_j of D, or it wouldn't be consistent. Clearly, $P_j \subseteq P'_i$.

Conversely, for any possible LDB P_j of D, one and only one choice is possible for t: either the alternatives chosen by P_j for the x-tuples in the lineage of t (all the same, by uniformity) form the lineage of one of the alternatives of t, and this alternative (unique by determinism) must be chosen to get a consistent LDB, or they do not correspond to the lineage of any alternative, in which case the only choice for t is not picking an alternative, which is possible because t is a maybe x-tuple and none of the alternative's lineage is satisfied. Acyclicity ensures that the choice made for t does not affect the rest of the LDB. The obtained LDB is the unique possible LDB P'_i of D' s.t. $P_j \subseteq P'_i$. □

Proof of Theorem 4.4: The algorithm returns $D' = D + (R_q, I(R_q), \lambda_{R_q})$ which adds to D the x-relation R_q containing exclusively maybe x-tuples with well-behaved lineage. Therefore, by the above lemma, D' preserves the possible instances of D. We now show that $\forall D_i$ possible LDB of D, the corresponding LDB D'_i of D' is precisely $Q(D_i)$.

As $D_i \subseteq \bar{D}$, by monotonicity $Q(D_i) \subseteq Q(\bar{D})$. Every tuple in $Q(D_i)$ becomes an alternative in R_q with lineage pointing to alternatives of D that are picked by D_i. Since $D_i \subseteq D'_i$, D'_i can (and must) pick those alternatives. Hence, $Q(D_i) \subseteq D'_i$.

To show that $D'_i \subseteq Q(D_i)$, suppose D'_i picks some $s_{(i,j)} \notin Q(D_i)$. Clearly, $\lambda(s_{(i,j)}) \subseteq D_i$, and by definition of lineage for positive queries $Q(D_i|_{\lambda}(s_{(i,j)}))$ produces $s_{(i,j)}$, which implies by monotonicity that $s_{(i,j)} \in Q(D_i)$, a contradiction. □

Proof of Theorem 4.7: Clearly if $s_{(i,j_1)}, s_{(i,j_2)} \in \overline{\lambda}(s_{(k,l)})$, the alternative is extraneous.

It now suffices to show that if (k,l) is extraneous, $\exists s_{(i,j_1)}, s_{(i,j_2)} \in \overline{\lambda}(s_{(k,l)})$. We prove its contrapositive in two steps:

1. We show that the tuple t, corresponding to this alternative, appears in some possible instance if the base tuples in $\overline{\lambda}(s_{(k,l)})$ are picked. That is, $t \in Q(D)$ where D is the LDB constituting the base tuples $\overline{\lambda}(s_{(k,l)})$, and Q is the query performed to obtain the x-relation of $s_{(k,l)}$ from the base relations.

2. There exists a possible instance LDB of the base relations D' satisfies $D \subseteq D'$. Now $t \in Q(D)$ and $D \subseteq D'$ and so by Definition 4.2 $t \in Q(D')$. Therefore, t appears in a possible instance with D' and is not extraneous. □

Proof of Theorem 4.8: First note that if even one of the tuples in $h(t)$ has a ?, the ? in t is not extraneous because choosing the ? in the base tuple results in t not having any tuple. Now, we claim that even if one of the combinations of alternatives in $h(t)$ does not give an alternative in t, the ? in it is not extraneous. This follows from an argument similar to the proof of Theorem 4.7, i.e., choosing a possible instance containing the combination in $h(t)$ not having an alternative results in a possible instance of the database with no alternative being picked from t. Finally, the only way ? can be picked for t is if no alternative of t is satisfied; and, this can only happen if some combination of alternatives in $h(t)$ does not result in an alternative in t. □

Proof of Theorem 4.10: Let the x-relation added by Q be R_q. We first show that the newly generated lineage is minimal, i.e., removing any symbol from $\lambda(x)$ where x is a symbol of R generates a possible instance not in $Q(D)$. Next, if $Q(D)$ is not L-minimal and some $\lambda(y)$ can be made smaller without changing the possible instances where y is in some relation of D, then $\lambda(y)$ can be reduced in the same way in D too. □

Proof of Theorem 4.12: As defined in Section 4.3.1, a D-minimal ULDB does not have extraneous tuples or '?'s, i.e., all alternatives do appear in some possible instance. The tuple-membership problem for t and R thus returns "yes" if t is an alternative of the x-relation for R, and otherwise returns "no". Similarly, the tuple-certainty problem returns "yes" if there is an x-tuple with a single alternative t and no '?'. Finally, note that the D-minimization algorithm takes polynomial time in the size of D. Therefore, these problems are also answered in polynomial time using the procedure above after D-minimizing D.

The proof for instance-membership is by a reduction from the NP-complete graph 3-colorability problem, and for instance-certainty by a reduction from the Co-NP-complete graph non 3-colorability problem. The hardness of both these problems for c-tables was first shown in [2]. The colorability problems were reduced to instance-membership (or certainty) of the result of positive existential queries on base c-tables. These base c-tables, which represented the graph structure and possible colorings, can be represented as well-behaved base x-relations. Therefore, the instance membership and certainty problems are hard for ULDBs also. □

Proof of Theorem 4.14: The correctness of the algorithm follows from the fact that all lineage that constrains the possible instances for x-tuples in $\bar{X}$ is retained in D'. Since the algorithm needs only one traversal of the lineage of all x-tuples in $\bar{X}$, the running time is polynomial in the size of D. Finally, if D is D-minimal, there are no extraneous alternatives or '?'s. Now if we restrict the possible instances to $\bar{X}$, all alternatives in $\bar{X}$ still appear in some possible instance. Similarly, all x-tuples with '?' continue to give the empty instance in some possible instance in D'. Therefore, D' is D-minimal. □