Organization

Program Committee
Foto Afrati, NTUA, Greece
Patrick Bosc, IRISA/ENSSAT, France
Anish Das Sarma, Stanford University, USA
Curtis Dyreson, Utah State University, USA
Sander Evers, University of Twente, The Netherlands
Michael Fink, Vienna University of Technology, Austria
Manolis Gergatsoulis, Ionian University, Greece
Maurizio Lenzerini, University of Rome, La Sapienza, Italy
Olivier Pivert, IRISA/ENSSAT, France
Dan Suciu, University of Washington and Microsoft, USA

Organizing Committee
Ander de Keijzer, University of Twente, The Netherlands
Maurice van Keulen, University of Twente, The Netherlands

Preface

This is the third edition of the international workshop on Management of Uncertain Data. The previous editions took place in Vienna, Austria and Auckland, New Zealand. The edition in Auckland was a combined event with the workshop on Quality in Databases.

Research on uncertain data has grown over the past few years. Since the prequal to this workshop, the Twente Data Management workshop on Uncertain Data, the number of submissions about management of uncertain data to large conferences has grown rapidly. Also, more workshops on the topic are being organized, e.g. MOUND at ICDE and U'09 at KDD. This is a clear indication that there is much interest in the topic, not only from the database community, but also datamining, AI and IR.

This edition, we have five research talks addressing different topics in uncertain data. In addition, we have two tutorials. The first tutorial is about *probabilistic databases* and is given by Christoph Koch. The second tutorial is about *possibilistic databases* and is given by Olivier Pivert. The second tutorial is followed by a discussion on the differences and commonalities between probabilistic and possibilistic databases.

We would like to thank the reviewers for their time and effort. We would also like to thank the Centre Telematics and Information Technology for sponsoring the proceedings of the workshop.

Ander de Keijzer
Maurice van Keulen

Table of Contents

Functional Dependencies Over Possibilistic Databases: An Interpretation Based on the Possible Worlds Semantics 1
Patrick Bosc and Olivier Pivert

Continuous Uncertainty in Trio 17
Parag Agrawal, Jennifer Widom

Outerjoins in Uncertain Databases 33
Robert Ikeda and Jennifer Widom

Query Containment of Tier-2 Queries over a Probabilistic Database 47
Katherine F. Moore, Vibhor Rastogi, Christopher R'e, Dan Suciu

Embracing Uncertainty in Large-Scale Computational Astrophysics 63
Dan Suciu, Andrew Connolly, and Bill Howe

Functional Dependencies Over Possibilistic Databases: An Interpretation Based on the Possible Worlds Semantics

Patrick Bosc and Olivier Pivert

Irisa – Enssat, University of Rennes 1
Technopole Anticipa 22305 Lannion Cedex France
bosc@enssat.fr, pivert@enssat.fr

Abstract. In this paper, we introduce a definition of the concept of a functional dependency (FD) in the context of databases containing ill-known attributes values represented by possibility distributions. Contrary to previous proposals, this definition is based on the possible worlds model and consists in viewing the satisfaction of an FD by a relation as an uncertain event whose possibility and necessity can be quantified. We give the principle of a method for incrementally computing the related possibility and necessity degrees and tackle the issue of tuple refinement in the presence of an FD.

1 Introduction

Thirty years ago, two papers authored respectively by E.F. Codd [10] and W. Lipski [21] were published almost simultaneously on the issue of extending the relational database model so as to represent unknown (null) values. Since then, many authors have proposed techniques to model and handle databases involving uncertain or incomplete data. In particular, the last two decades have witnessed an explosion of research on this topic, e.g., [1, 4, 5, 7, 8, 12, 13, 15, 16, 18]. However, most of these works deal with *querying* uncertain databases while relatively few tackle issues related to the existence of *functional dependencies* in such a context.

For defining the semantics of uncertain databases, the possible worlds model is now widely accepted. It is founded on the fact that uncertainty in data makes it impossible to define what precisely the real world is. One can only describe the set of possible worlds which are consistent with the available information. For instance, in a scenario where one wishes to represent a mobile object, there can be some uncertainty about the precise position of the object at a given moment. Each possible position corresponds to a possible world, or more than one world in general if other attributes are imprecise or if several objects are represented. A world is constructed by choosing a precise candidate for each uncertain attribute value from the imprecise database.

Our starting point is to consider that an FD constrains the more or less possible worlds attached to an uncertain database and we consider the case where ill-known attribute values are represented by means of weighted sets of

candidate values (which could be either probability or possibility distributions depending on the uncertainty model used). In this paper, we focus on the possibilistic database framework, but the problem —if not the solutions— can be stated in a very similar way in a probabilistic database context. The basic idea underlying this work is that one can take advantage of the representation of an ill-known value as a weighted set of candidates for quantifying the level of consistency of a relation w.r.t. a functional dependency. In the possibilistic framework, this is done through two degrees: a possibility degree and a necessity degree of consistency. The main issue dealt with in this paper is integrity checking, i.e., how to enforce a certain level of consistency on a relation over which an FD is defined. A secondary issue concerns the way some ill-known attribute values may be refined (i.e., made more precise) when some new tuples are inserted into a relation constrained by an FD.

The remainder of the paper is structured as follows. Section 2 is devoted to a presentation of some basic elements of possibility theory and introduces a simple possibilistic database model. Section 3 gives a brief overview of some related work, concerning in particular functional dependencies in the presence of null values as well as so-called fuzzy functional dependencies in a possibilistic database context. In Section 4, the definition of a functional dependency based on the possible worlds model is given, and the notion of consistency level of a relation w.r.t. to an FD is defined. Section 5 presents a method for incrementally computing the possibility and necessity degree of consistency of a relation w.r.t. an FD. In Section 6, we tackle the issue of tuple refinement and show how an FD can be used to improve the level of informativeness of a possibilistic database in certain cases. Section 7 concludes the paper, recalls the main contributions and outlines some perspectives for future research.

2 A Reminder on Possibilistic Databases

2.1 Basic Notions About Possibility Theory

Possibility theory [27] provides an ordinal model for uncertainty where imprecision is represented by means of a preference relation coded by a total order over the possible situations. This framework is closely related to fuzzy set theory since the idea is to constrain the values taken by a variable thanks to a normalized fuzzy set (where at least one element has the degree 1), called a possibility distribution π. $\pi(a)$ expresses the degree to which a is a possible value for the considered variable. The normalization condition imposes that at least one of the values of the domain (a_0) is completely possible, i.e., $\pi(a_0) = 1$. This setting is particularly suited to take into account uncertainty represented by linguistic terms such as "tall", "medium" or "recent". In the discrete case assumed later on, a possibility distribution is written $\{\pi_1/a_1, ..., \pi_n/a_n\}$ where a_i is a candidate value and π_i its degree of possibility. Any event E is characterized by two measures: its possibility Π (expressing the fact that E may more or less occur) and its necessity N (expressing that E will occur more or less for sure). The necessity N of E is defined as: $N(E) = 1 - \Pi(\overline{E})$ where $\overline{E}$ is the event opposite

to E. The following results, where E, E_1 and E_2 denote events, are of interest further:

- $\Pi(E_1 \cup E_2) = max(\Pi(E_1),\ \Pi(E_2))$
- $\Pi(E_1 \cap E_2) = min(\Pi(E_1),\ \Pi(E_2))$ if E_1 and E_2 are logically independent
- $N(E1 \cap E2) = min(N(E1),\ N(E2))$
- $N(E1 \cup E2) = max(N(E1),\ N(E2))$ if E_1 and E_2 are logically independent
- $\Pi(E) < 1 \Rightarrow N(E) = 0$.

The two measures Π and N provide a total order over the set of regular (non fuzzy) events. The events can be rank-ordered according to Π for those which are not at all certain and according to N for those which are completely possible.

2.2 Possibilistic Databases

In contrast to a regular database, a possibilistic relational database D may have some attributes which take imprecise values. In such a case, a possibility distribution is used to represent all the more or less acceptable candidate values for the attribute. The first version of a possibilistic database model was introduced in [23]. From a semantic point of view, a possibilistic database D can be interpreted as a set of usual databases (also called worlds or interpretations) W_1, ..., W_p, denoted by $rep(D)$, each of which being more or less possible. This view establishes a straightforward semantic connection between possibilistic and regular databases. This relationship is particularly interesting since it offers a canonical approach to the definition of queries addressed to possibilistic databases [7] (but this aspect is not the topic of the present work). Any world W_i is obtained by choosing a candidate value in each possibility distribution appearing in D. One of these (regular) databases, let us say W_k, is supposed to correspond to the actual state of the universe modeled. Any world W_i corresponds to a conjunction of independent choices and according to previous formulas, the degree assigned to it is the minimum of the degrees tied to each of the chosen candidate values in the original possibilistic database D. Therefore, at least one of the worlds is completely possible, i.e., is assigned the degree of possibility $\Pi = 1$.

Example 1. Let us consider the possibilistic database D involving two relations: *im* and *pl* whose respective schemas are $IM(\#i,\ ac,\ date,\ loc)$ and $PL(ac,\ lg,\ msp)$. Relation *im* describes satellite images of airplanes and each image, identified by a number ($\#i$), taken on a certain location (loc) a given day ($date$) is supposed to include a single plane (ac). Relation *pl* gives the length (lg) and maximal speed (msp) of each type of aircraft. With the extensions of *im* and *pl* given in Figure 1, four worlds can be drawn, W_1, W_2, W_3 and W_4, since there are two candidates for *date* in the first tuple of *im* and two candidates for *ac* in the second one. Each of these worlds involves relation *pl* which contains only precise values (and then has a single interpretation) and one of the four regular relations im_1 to im_4 given in Figure 2, issued from the possibilistic relation *im*. The value Π specified is that of each world W_i made of the pair of relations (im_i, *pl*) and as it is expected, one of them is completely possible.$\diamond$

$\#i$	ac	$date$	loc
i_1	a_1	$\{1/d_1, 0.7/d_3\}$	c_1
i_3	$\{1/a_3, 0.3/a_4\}$	d_1	c_2

ac	lg	msp
a_1	20	1000
a_2	25	1200
a_3	18	800
a_4	20	1200

Fig. 1. Extensions of possibilistic relations im (left) and pl (right)

$\#i$	ac	$date$	loc
i_1	a_1	d_1	c_1
i_3	a_3	d_1	c_2

$\Pi = 1$

$\#i$	ac	$date$	loc
i_1	a_1	d_3	c_1
i_3	a_3	d_1	c_2

$\Pi = 0.7$

$\#i$	ac	$date$	loc
i_1	a_1	d_1	c_1
i_3	a_4	d_1	c_2

$\Pi = 0.3$

$\#i$	ac	$date$	loc
i_1	a_1	d_3	c_1
i_3	a_4	d_1	c_2

$\Pi = 0.3$

Fig. 2. The four interpretations of relation im

3 Related Work

Historically, the first type of imprecise value is known as *nulls* or null values [10, 21]. They are used to deal with situations where the attribute makes sense (applies), but the actual value is unknown. More precisely, it is assumed that any value of the domain can be the actual value. In the pioneering work [26], the author first establishes the fact that the satisfaction of an FD by a relation can be either $true$, $false$, or $unknown$ when that relation includes *nulls*, which leads to the notions of strong satisfiability (the FD is true) and weak satisfiability (the FD is not false). The characterization of the cases where it is $true$ (resp. $false$) is given and it is proven that Armstrong's inference rules are sound and complete for FDs defined on relations with nulls and the requirement of strong satisfiability. Further, Lien [20] and Atzeni and Morfuni [3] defined a sound and complete axiom system for FDs holding in incomplete relations by dropping the transitivity rule and adding the union and decomposition rules.

In [19], a sound and complete axiom system for both strong functional dependencies (SFDs, i.e., FDs whose satisfiability is strong) and weak ones (WFDs, i.e., FDs whose satisfiability is weak) is exhibited, which takes into account the interaction between SFDs and WFDs. The authors also show that the implication problem (i.e., finding the computational complexity of determining whether $G \models X \rightarrow A$, given a set of FDs G) for SFDs and WFDs can be solved in time polynomial in the size of the input set of FDs.

In [2], the authors consider a database model with disjunctive tuples and propose a cleaning technique which removes inconsistent worlds with respect to a set of functional dependencies. This objective is somewhat similar to that considered in Section 6 where we present a tuple refinement technique, but the

database model used is quite different and this has an important impact on the types of inconsistencies which can be removed, and the way to remove them.

On the other hand, several proposals for defining FDs in the context of values represented by possibility distributions have been made in the last two decades. Such extended FDs are often called fuzzy functional dependencies (FFDs) and can be divided into two categories depending on the nature of the comparison between two ill-known values. In [6, 22, 24, 25], this comparison is made in terms of *representations*, i.e., the result is a degree establishing the extent to which the two underlying possibility distributions are equal (at a syntactic level). For instance, in [24], the definition of a so-called FFD $X \rightsquigarrow Y$ on a relation r is:

$$\forall t_1,\ t_2 \in r,\ EQ(t_1.X,\ t_2.X) \rightarrow_{RG} EQ(t_1.Y, t_2.Y)$$

where $\rightarrow_{RG}$ denotes Rescher-Gaines implication ($a \rightarrow_{RG} b = 1$ if $a \leq b$, 0 otherwise) and $EQ(a,\ b)$ stands for the result of the comparison of values a and b (represented by the possibility distributions π_a and π_b) according to:

$$EQ(a,\ b) = min_{u \in U} 1 - |\pi_a(u) - \pi_b(u)|$$

where U denotes the referential considered.

The other type of approach, advocated by Chen [9] and Cubero *et al.* [11], suggests a comparison in the possibilistic framework, i.e., its result is the degree of possibility of the equality between two ill-known values. This view does not really concern the representations but the *values* involved (and the associated degrees). Using Zadeh's extension principle, the degree of possibility that a and b are equal (knowing that they are represented by the possibility distributions π_a and π_b) is given by:

$$\mu_{EQ}(a,\ b) = \Pi(a = b) = sup_{u \in U} min(\pi_a(u),\ \pi_b(v)).$$

In the probabilistic database framework, let us mention the work by Dey and Sarkar [14] who generalize the concept of a functional dependency into that of a *stochastic dependency* and extend the scope of normal forms for the extended relational structure. The basic idea is to represent how one attribute influences the probability distribution of another. This work is very much in the same spirit as the aforementioned approaches about fuzzy functional dependencies and, as them, is not founded on the possible worlds semantics.

With respect to the works about FDs in the presence of null values, the main contribution of the present paper is about the impact of using a more informative type of ill-known values, namely weighted sets of explicit candidate values. In such a framework, we show that the notions of SFD and WFD can be generalized into that of more or less possible and certain FDs. Contrary to the approaches about FFDs and stochastic FDs, which do not use the possible worlds model as a semantic basis but rather define "soft integrity constraints", we do not aim at extending the notion of an FD in the presence of ill-known values. Keeping its usual meaning, we want to take into account the fact that the satisfaction of a regular FD by a relation becomes an uncertain event when some data are

ill-known (in the same spirit of the aforementioned works about FDs and null values, the main difference being that the uncertainty model that we use allows us to *quantify* the uncertainty of such an event).

4 Consistency Level of a Database Relation

In the following, we consider that database design is performed as usual, ignoring the fact that some values to be stored may be ill-known (in other words, normalization is done using the FDs on the *worlds*). On the other hand, when data is to be inserted in the relations, it may be useful to check whether a certain level of consistency of the database is preserved w.r.t. the FDs. In the following, a simple possibilistic database model such as that illustrated in Example 1 will be used, since it is sufficient for the purpose considered here.

Example 2. Consider the relation schema $SP(supplier, product, warehouse)$ whose key is $(supplier, product)$. The FD $(supplier, product) \rightarrow warehouse$ is thus supposed valid. It is assumed that each attribute may take imprecise values. Let us suppose that the data is inserted by a human agent and comes from handwritten documents whose deciphering induces uncertainty in the relation content. The extension given in Table 1 is potentially inconsistent w.r.t. the FD

Table 1. An extension of relation *sp*

supplier	*product*	*warehouse*
{Dupont, Dupond}	valve	{Nantes, Mantes}
Dupont	valve	{Nantes, Mantes, Vanves}

$(supplier, product) \rightarrow warehouse$. Indeed, one can build from it the world represented in Table 2 which violates the FD.$\diamond$

Table 2. An extension of relation *sp*

supplier	*product*	*warehouse*
Dupont	valve	Nantes
Dupont	valve	Mantes

If possibility degrees are attached to the more or less possible candidate values, it becomes possible to quantify the consistency level. Let us introduce the notions of possibility and necessity of consistency of a relation w.r.t. an FD.

Definition 1. The possibility degree of consistency of a relation r w.r.t. a functional dependency F is defined as:

$\Pi(r, F)$ = the possibility degree attached to the most possible world of r among those which satisfy F.

The necessity degree of consistency of a relation r w.r.t. a functional dependency F is defined as:

$N(r, F) = 1 -$ the possibility degree attached to the most possible world of r among those which violate F.■

In the following, we will use the notations $\Pi(r)$ and $N(r)$ since one FD F only will be considered. Let us consider for example the relation given in Table 3. The most possible world among those which satisfy the functional dependency

Table 3. An extension of relation *sp*

supplier	*product*	*warehouse*
{1/Dupont, 0.7/Dupond}	valve	{1/Nantes, 0.8/Mantes}
Dupont	valve	{1/Nantes, 0.7/Mantes, 0.4/Vanves}

(*supplier*, *product*) → *warehouse* has the possibility degree 1 (it is {(Dupont, valve, Nantes)}). The most possible world among those which violate the FD has the possibility degree 0.8 (it is the world {(Dupont, valve, Mantes), (Dupont, valve, Nantes)}). Therefore, the database is completely possibly consistent ($\Pi(r) = 1$) and 0.2 necessarily consistent w.r.t. the FD ($N(r) = 0.2$). If one wants to enforce a minimal degree of necessity of consistency equal to 0.5, the insertion of the second tuple must be rejected (assuming that the first one is already present). However, in certain cases, one can consider that the tuple to be inserted is the one that must be kept, which leads to modifying some tuples already present. For instance, if the extension of the relation is that represented in Table 4 and

Table 4. An extension of relation *sp*

supplier	*product*	*warehouse*
{1/Dupont, 0.7/Dupond}	valve	{1/Nantes, 0.8/Mantes}

one wants to insert the (precise) tuple (Dupont, valve, Mantes), it seems possible to draw the conclusion that the tuple already in r was about Dupond (and not Dupont). Thus, one can transform it into ({0.7/Dupond, valve, {1/Nantes, 0.8/Mantes}) while inserting the new one. The fact that the possibility distribution from the first tuple may not be normalized anymore will be discussed further (in Section 6).

Computing the consistency level (Π, N) of a relation w.r.t. an FD by building all the possible worlds of that relation is intractable in general due to the huge

number of worlds — this number equals k^n assuming that every tuple involves k candidate values and n is the cardinality of the relation. However, checking the consistency level when an insertion is performed might be more feasible. This point is investigated in the next section.

5 Consistency Checking

Let us consider a relation r whose schema contains attributes A, B, and C. Let us assume that the FD $AB \rightarrow C$ is defined over that relation. Let us denote by $\Pi(r)$ (resp. $N(r)$) the current possibility (resp. necessity) degree of consistency of r w.r.t. the considered FD. Let us assume that the tuple t is to be inserted into relation r. Let us denote by r' the relation after insertion. First, let us characterize the tuples of r which are potentially inconsistent with t. Informally, it is any tuple t' such that $t'[AB]$ has at least one interpretation in common with $t[AB]$ and such that $t'[C]$ is not necessarily equal to $t[C]$.

Definition 2. Let us denote by $inc(t,\ r,\ F)$ the set of tuples which are potentially inconsistent with t according to F, as well as t itself. The tuples which are potentially inconsistent with t are those tuples t' such that the functional dependency F on r induces a *direct* dependency between the choices of candidate values from any such t' and the choices of candidate values from t when building a world of r constrained by F. Formally:

$$inc(t,\ r,\ F) = \{t\} \cup \{t' \in r - \{t\} \mid (t'.A \cap t.A) \neq \emptyset \wedge (t'.B \cap t.B) \neq \emptyset \wedge |t'.C \cup t.C| > 1\}. \blacksquare$$

In the following, $inc(t,\ r,\ F)$ will be denoted by $inc(t)$ since there is no ambiguity about the FD and the relation used.

Definition 3. Let us denote by $inc(t)^+$ the transitive closure of $inc(t)$. The set $inc(t)^+$ gathers tuple t itself and those tuples t' such that the functional dependency F on r induces a *transitive* dependency between the choices of candidate values from any such t' and the choices of candidate values from t when building a world of r constrained by F. Formally:

$$inc(t)^+ = inc(t)\ \cup_{t_i \in inc(t)} inc(t_i)\ \cup_{t_i \in \cup_{t_j \in inc(t)} inc(t_j)} inc(t_i) \cup \text{ etc.} \blacksquare$$

Let us denote by W_{inc} the most possible world of $inc(t)^+$ among those which satisfy the F. Let us denote by V_{inc} the most possible world of $inc(t)^+$) among those which violate F. Let us denote by $ind(t)$ the subrelation $r - inc(t)^+$.

Example 3. Let us consider a relation $r(A,\ B,\ C)$ containing the tuples:
$t_1 = (a_1,\ \{1/b_1,\ 0.8/b_2,\ 0.4/b_3\},\ \{0.7/c_1,\ 1/c_2,\ 0.3/c_3\})$,
$t_2 = (\{1/a_1,\ 1/a_3\},\ b_1,\ c_3)$,
$t_3 = (a_1,\ \{1/b_5,\ 0.7/b_6,\ 0.4/b_7\},\ \{0.7/c_1,\ 1/c_2\})$, and

$t_4 = (\{1/a_1,\ 1/a_3\},\ \{1/b_7,\ 0.6/b_6\},\ c_3)$
and let us assume that the FD $AB \rightarrow C$ is defined over r. Let us consider a new tuple $t_5 = (a_3,\ \{1/b_1,\ 0.4/b_2\},\ \{1/c_1,\ 0.8/c_2\})$. One has: $inc(t_5) = \{t_5,\ t_2\}$ and: $inc(t_5)^+ = \{t_5,\ t_1,\ t_2\}\diamond$

The following proposition states that the tuples from $ind(t)$ do not have to be taken into account for computing the values of $\Pi(r')$ and $N(r')$.

Proposition 1. The following formulas can be used to update the consistency degrees of relation r w.r.t. the considered functional dependency:

$$\Pi(r') = min(\Pi(r),\ \Pi(W_{inc})) \tag{1a}$$
$$N(r') = min(N(r), 1 - \Pi(V_{inc})). \tag{1b}$$

Proof. First let us deal with the possibility degree of consistency. Let us denote by W (one of) the most possible world(s) of r among those which satisfy the FD. W can be partitioned into two subworlds: W_1 which gathers representatives of the tuples from $ind(t)$, and W_2 which gathers representatives of the tuples from $inc(t)^+ - \{t\}$ (tuple t has not been inserted yet). According to the axioms of possibility theory, one has $\Pi(r) = min(\Pi(W_1),\ \Pi(W_2))$. By definition of $inc(t)^+$, the candidate choices related to any t' from $ind(t)$ are (directly and transitively) independent from those related to t. After insertion of t, (one of) the most possible world(s) of r' among those which satisfy the FD is W' $= W_1 \cup W_{inc}$ and the value of $\Pi(r')$ is thus equal to $min(\Pi(W_1),\ \Pi(W_{inc}))$. One has: $\Pi(W_{inc}) \leq \Pi(W_2)$ since W_{inc} is more constrained than W_2 due to the additional presence of a representative of t. Thus, $min(\Pi(W_1),\ \Pi(W_{inc})) = min(min(\Pi(W_1),\ \Pi(W_2)),\ \Pi(W_{inc}))$. Finally: $\Pi(r') = min(\Pi(r),\ \Pi(W_{inc}))$ and formula 1a holds.

Let us now consider the computation of $N(r')$. Let us denote by V (one of) the most possible world(s) of r among those which violate the FD. V can be partitioned into two subworlds: V_1 which gathers representatives of the tuples from $ind(t)$, and V_2 which gathers representatives of the tuples from $inc(t)^+ - \{t\}$. One has: $N(r) = 1 - \Pi(V)$ and according to the axioms of possibility theory: $\Pi(V) = min(\Pi(V_1),\ \Pi(V_2))$. First, let us notice that one has: $\Pi(V_1) = 1$ or $\Pi(V_2) = 1$ since the worst violation (i.e., the most possible pair of tuple representatives which violates the DF) is either in $ind(t)$ or in $inc(t)^+ - \{t\}$. If it is in $ind(t)$, V_2 is made of completely possible representatives of the tuples from $inc(t)^+ - \{t\}$ (again, let us recall that the distributions are normalized), and reciprocally. After insertion, the most possible world of r' which violates the FD can be partitioned into V_1' (representatives of the tuples from $ind(t)$) and V_2' (representatives of the tuples from $inc(t)^+$). Several cases may occur:

- the worst violation was in V_1 ($N(r) = 1 - \Pi(V_1)$):
  - if $\Pi(V_{inc}) \leq \Pi(V_1)$, then the worst violation is still in $V_1 = V_1'$ and V_2' can be built by adding a completely possible representative of t to V_2.

Then $\Pi(V_2') = 1$ and $N(r') = 1 - min(\Pi(V_1),\ \Pi(V_2')) = 1 - \Pi(V_1) = min(1 - \Pi(V_1),\ 1 - \Pi(V_{inc}) = min(N(r), 1 - \Pi(V_{inc}))$ and formula 1b holds, or

- the worst violation is now in V_{inc} ($\Pi(V_{inc}) \geq \Pi(V_1)$) and one can build V_1' by choosing completely possible representatives of tuples from $ind(t)$, which leads to $\Pi(V_1') = 1$. Hence $N(r') = 1 - min(\Pi(V_1'),\ \Pi(V_{inc})) = 1 - \Pi(V_{inc}) = min(1 - \Pi(V_1),\ 1 - \Pi(V_{inc})) = min(N(r), 1 - \Pi(V_{inc}))$ and formula 1b holds.

– the worst violation was in V_2 ($N(r) = 1 - \Pi(V_2)$) then we had $\Pi(V_1) = 1$. The worst violation is now necessarily in $V_2' = V_{inc}$ while $V_1' = V_1$. Notice that $\Pi(V_{inc}) \geq \Pi(V_2)$ since the addition of a new tuple can only increase the violation level. One has: $N(r') = 1 - min(\Pi(V_1'),\ \Pi(V_{inc}) = 1 - \Pi(V_{inc}) = min(1 - \Pi(V_2), 1 - \Pi(V_{inc}) = min(N(r), 1 - \Pi(V_{inc}))$ and formula 1b holds.■

Remark. Computing the worlds of $inc(t)^+$ can be expensive, but is in general much cheaper than computing the worlds of r since in general $|\, inc(t)^+ \,| << |\, r \,|$.

Example 4. Let us consider relation r containing the tuples $t_1 = (a_1,\ b_1,\ c_1)$, $t_2 = (a_1,\ b_1,\ \{1/c_1,\ 0.7/c_2\}$, $t_3 = (a_2,\ b_2,\ c_2)$, $t_4 = (a_2,\ b_2,\ \{1/c_2,\ 0.6/c_4\})$ and assume one wants to insert tuple $t_5 = (a_2,\ b_2,\ \{0.2/c_2,\ 1/c_3\})$.
One has: $inc(t_5)^+ = \{t_5,\ t_3,\ t_4\}$ and $ind(t_5) = \{t_1,\ t_2\}$.
Before insertion:
$W = \{(a_1,\ b_1,\ c_1),\ (a_2,\ b_2,\ c_2)\}$, $\Pi(W) = 1$.
$W_1 = \{(a_1,\ b_1,\ c_1)\}$, $\Pi(W_1) = 1$, $W_2 = \{(a_2,\ b_2,\ c_2)\}$, $\Pi(W_2) = 1$.
$\Pi(r) = \Pi(W) = min(\Pi(W_1),\ \Pi(W_2)) = 1$.
$V = \{(a_1,\ b_1,\ c_1),\ (a_1,\ b_1,\ c_2),\ (a_2,\ b_2,\ c_2)\}$, $\Pi(V) = 0.7$.
$V_1 = \{(a_1,\ b_1,\ c_1),\ (a_1,\ b_1,\ c_2)\}$, $\Pi(V_1) = 0.7$, $V_2 = \{(a_2,\ b_2,\ c_2)\}$, $\Pi(V_2) = 1$.
$N(r) = 1 - \Pi(V) = 1 - min(\Pi(V_1),\ \Pi(V_2)) = 0.3$.
After insertion:
$W' = W = \{(a_1,\ b_1,\ c_1),\ (a_2,\ b_2,\ c_2)\}$, $\Pi(W') = 0.2$.
$W_1' = W_1$, $\Pi(W_1') = 1$, $W_2' = W_{inc} = \{(a_2,\ b_2,\ c_2)\}$, $\Pi(W_2') = \Pi(W_{inc}) = 0.2$.
$\Pi(r') = \Pi(W') = min(\Pi(W_1'),\ \Pi(W_2')) = 0.2 = min(\Pi(r),\ \Pi(W_{inc})$.
$V = \{(a_1,\ b_1,\ c_1),\ (a_2,\ b_2,\ c_2),\ (a_2,\ b_2,\ c_3)\}$, $\Pi(V) = 1$.
$V_1' = \{(a_1,\ b_1,\ c_1)\}$, $\Pi(V_1') = 1$,
$V_2' = V_{inc} = \{(a_2,\ b_2,\ c_2),\ (a_2,\ b_2,\ c_3)\}$, $\Pi(V_2') = \Pi(V_{inc}) = 1$.
$N(r') = 1 - \Pi(V) = 1 - min(\Pi(V_1'),\ \Pi(V_2')) = 0 = min(N(r),\ 1 - \Pi(V_{inc}))$.◇

The following example shows that formulas 1a and 1b would not be correct if $inc(t)$ were used instead of $inc(t)^+$.

Example 5. Let us suppose that relation $r(A,\ B,\ C)$ contains the tuples:
$t_1 = (a_1,\ \{1/b_1,\ 0.8/b_2,\ 0.4/b_3\},\ \{0.7/c_1,\ 1/c_2,\ 0.3/c_3\})$ and
$t_2 = (\{1/a_1,\ 1/a_3\},\ b_1,\ c_3)$.
One has:
$\Pi(r) = 1$ (the world $\{(a_1,\ b_1,\ c_2),\ (a_3,\ b_1,\ c_3)\}$ satisfies the FD), and
$N(r) = 0$ (the world $\{(a_1,\ b_1,\ c_2),\ (a_1,\ b_1,\ c_3)\}$ of possibility 1 violates the FD).

Let us now try to insert tuple $t_3 = (a_3,\ \{1/b_1,\ 0.4/b_2\},\ \{1/c_1,\ 0.8/c_2\})$ into r. One has $inc(t_3) = \{t_3,\ t_2\}$ and:
$\Pi(W_{inc}) = 1$ (corresponding to the world of $inc(t_3)$: $\{(a_1,\ b_1,\ c_3), (a_3,\ b_1,\ c_1)\}$),
$\Pi(V_{inc}) = 1$ (associated with the world of $inc(t_3)$: $\{(a_3,\ b_1,\ c_3), (a_3,\ b_1,\ c_1)\}$).
According to formula 1a, $\Pi(r') = min(\Pi(r),\ \Pi(W_{inc})) = 1$. But in fact, the correct degree is 0.8 because the choices of candidates corresponding to the most possible world of r before insertion are not compatible with the choices corresponding to the most possible interpretation of $inc(t_3)$ satisfying the FD.
Using $inc(t)^+$ instead of $inc(t)$ in formulas 1a and 1b, one gets:
$inc(t_3)^+ = \{t_3,\ t_1,\ t_2\}$,
$\Pi(W_{inc}) = 1$ ($W_{inc} = \{(a_1,\ b_1,\ c_2),\ (a_1,\ b_1,\ c_3), (a_3,\ b_1,\ c_1)\}$),
$\Pi(V_{inc}) = 1$ ($V_{inc} = \{(a_1,\ b_1,\ c_2),\ (a_3,\ b_1,\ c_3), (a_3,\ b_1,\ c_1)\}$).
and the correct degrees:
$\Pi(r') = min(\Pi(r),\ \Pi(W_{inc})) = min(1,\ 0.8) = 0.8$
$N(r') = min(N(r), 1 - \Pi(V_{inc})) = min(0, 1 - 1) = 0$. $\diamond$

As to the constraint enforced by means of that consistency checking process, a simple solution is to refuse the insertion when $N(r)$ gets below a certain predefined threshold or, with a more laxist view, when $\Pi(r)$ becomes strictly less than 1. This way of doing corresponds to a policy of the type "first arrived, first served" as used in the classical database framework. However, such a policy may be debatable since it may be the case that the tuple to be inserted is correct whereas some already present tuples involve wrong candidates. In such a case, a solution based on tuple refinement may be of interest.

6 Tuple Refinement

Let us first characterize the cases where tuple refinement can occur in the framework of the model considered. A reasonable assumption is to consider that when the tuple to be inserted is precise and is not incompatible with another precise tuple already present in r, then one can consider it reliable and use it to refine some imprecise tuples already present.

Definition 4. Let us denote by $t = (a,\ b,\ c)$ the precise tuple to be inserted. Let us denote by $ref(t)$ the set of imprecise tuples concerned by the refinement process related to the insertion of t. $ref(t)$ contains those tuples t' from r which have $(a,\ b,\ c)$ among their interpretations and such that interpretations of the type $(a,\ b,\ x)$ with $x \neq c$ can be removed from them. This latter condition implies that $t'.A$ and $t'.B$ cannot be both imprecise. In other words:

$$ref(t) = \{t' \in r \mid \text{either } ((t'.A = t.A) \text{ and } (b \in t'.B) \text{ and } (c \in t'.C)) \\ \text{or } ((a \in t'.A) \text{ and } (t'.B = t.B) \text{ and } (c \in t'.C))\}. \blacksquare$$

In the first case, one must replace $t'.B$ by $t'.B - \{b\}$; in the second case, one replaces $t'.A$ by $t'.A - \{a\}$. Then, $\Pi(r)$ and $N(r)$ have of course to be updated.

Example 6. Let us consider relation r from Table 5. The degrees of consis-

Table 5. An extension of relation r

	A	B	C
t_1	a_1	$\{1/b_1,\ 0.8/b_2,\ 0.4/b_3\}$	$\{1/c_1,\ 0.7/c_2,\ 0.3/c_3\}$
t_2	$\{1/a_1,\ 1/a_2,\ 0.6/a_3\}$	b_1	$\{0.6/c_2,\ 1/c_1\}$
t_3	$\{1/a_1,\ 0.7/a_4\}$	$\{1/b_2,\ 0.7/b_3\}$	$\{0.6/c_3,\ 1/c_4\}$

tency of this relation w.r.t. the FD $AB \rightarrow C$ are:

$\Pi(r) = 1$ (valid world $\{(a_1,\ b_1,\ c_1),\ (a_1,\ b_2,\ c_4)\}$)
$N(r) = 1 - 0.8 = 0.2$ (invalid world $\{(a_1,\ b_2,\ c_1),\ (a_1,\ b_1,\ c_1),\ (a_1,\ b_2,\ c_4)\}$).

Consider now the insertion of tuple $t_4 = (a_1, b_1, c_2)$. We have: $ref(t_4) = \{t_1, t_2\}$. The relation r' resulting from the refinement is represented in Table 6.$\diamond$

Table 6. Relation r' (after insertion and refinement)

A	B	C
a_1	$\{0.8/b_2,\ 0.4/b_3\}$	$\{1/c_1, 0.7/c_2, 0.3/c_3\}$
$\{1/a_2,\ 0.6/a_3\}$	b_1	$\{0.6/c_2,\ 1/c_1\}$
$\{1/a_1,\ 0.7/a_4\}$	$\{1/b_2,\ 0.7/b_3\}$	$\{0.6/c_3,\ 1/c_4\}$
a_1	b_1	c_2

As this example illustrates, it may happen that some possibility distributions are not normalized in the resulting relation (this situation occurs when the candidate removed from a tuple t' was the only one which was (incorrectly) considered completely possible). The presence of such unnormalized distributions can lead to have $\Pi(r') < N(r')$ which contradicts one of the theorems of possibility theory. To solve this problem, a solution is to renormalize the modified distributions, which makes sense because we know for sure that it is either of the remaining candidates which is "the right one". A simple solution is to set to 1 the possibility degree of the most possible remaining candidates. Applying this method on the previous example, one gets the relation represented in Table 7.
Notice that the refinement process is in general *recursive* since some of the refined tuples may be precise and thus lead to refine some other tuples.

Example 7. Let us consider the relation from Table 8. The insertion of tuple $t_5 = (a_1,\ b_1,\ c_2)$ leads to refine t_2 into $t'_2 = (a_2,\ b_1,\ c_1)$. Tuple t_4, which was not concerned by the insertion of t_5, must now be refined into $t'_4 = (a_3,\ b_1,\ c_5)$.$\diamond$

Table 7. Relation r' (after insertion, refinement and renormalization)

A	B	C
a_1	$\{1/b_2,\ 0.4/b_3\}$	$\{1/c_1, 0.7/c_2, 0.3/c_3\}$
$\{1/a_2,\ 0.6/a_3\}$	b_1	$\{0.6/c_2,\ 1/c_1\}$
$\{1/a_1,\ 0.7/a_4\}$	$\{1/b_2,\ 0.7/b_3\}$	$\{0.6/c_3,\ 1/c_4\}$
a_1	b_1	c_2

Table 8. An extension of relation r

	A	B	C
t_1	a_1	$\{1/b_1,\ 0.8/b_2,\ 0.4/b_3\}$	$\{1/c_1,\ 0.7/c_2,\ 0.3/c_3\}$
t_2	$\{1/a_1,\ 1/a_2\}$	b_1	c_1
t_3	$\{1/a_1,\ 0.7/a_4\}$	$\{1/b_2,\ 0.7/b_3\}$	$\{0.6/c_3,\ 1/c_4\}$
t_4	$\{1/a_3,\ 0.8/a_2\}$	b_1	c_5

Let us now deal with the computation of the degrees of consistency of the resulting relation. Let us denote by t the tuple which has been inserted and by $mod(t_i)$ the result of the refinement (and possible renormalization) of tuple t_i. Let us denote:

$M(t) = \{mod(t_i) \mid t_i \in ref(t)^+\}$ where $ref(t)^+ = ref(t) \cup_{t' \in ref(t)} ref(t') \cup ...$

Set $M(t)$ contains the tuples which have been (directly or recursively) refined as a consequence of the insertion of t. To update the consistency degrees attached to r, one must first compute the set $incref(t) = [\cup_{t' \in M(t)} \Gamma(t')] \cup \Gamma(t)$ where:

$\Gamma(u) = \emptyset$ if $inc(u) = \{u\}$,
$\quad\quad = inc(u)^+$ otherwise.

Set $incref(t)$ is made of the tuples which are potentially inconsistent with t or with any of the tuples refined. Then, one may use $incref(t)$ instead of $inc(t)^+$ to compute W_{inc} and V_{inc} appearing in expressions 1a and 1b, and these formulas return the updated consistency degrees. Coming back to Example 6, we have:

$M(t_4) = \{mod(t_1),\ mod(t_2)\}$,
$\Gamma(t_4) = \emptyset$, $\Gamma(mod(t_1)) = \{mod(t_1),\ t_3\}$, $\Gamma(mod(t_2)) = \emptyset$.
Thus, $incref(t_4) = \{mod(t_1),\ t_3\}$.
The most possible world of $incref(t_4)$ among those satisfying (resp. violating) F is $\{(a_1,\ b_2,\ c_1),\ (a_1,\ b_3,\ c_4)\}$ (resp. $\{(a_1,\ b_2,\ c_1),\ (a_1,\ b_2,\ c_4)\}$) of possibility degree 0.7 (resp. 1).
Finally, using formulas 1a and 1b, we get the updated consistency degrees:
$\Pi(r) = min(1,\ 0.7) = 0.7$ and $N(r) = min(0.2, 1-1) = 0$.

It can be observed that refining tuples does not necessarily result into an increase of the consistency level (but this was not the objective, which was rather to "clean" the relation).

7 Conclusion

In this paper, we introduce a definition of the concept of a functional dependency (FD) in the context of databases containing ill-known attributes values represented by possibility distributions. The main contributions are the following:

- we defined two indices for quantifying the consistency level of a such relation w.r.t. an FD, taking into account the fact that the satisfaction of an FD by a possibilistic relation is a more or less possible (resp. certain) event,
- we showed how to update these indices in an incremental way,
- we pointed out how the presence of an FD makes it possible in certain cases to refine imprecise tuples.

From a practical viewpoint, the interest of this work is manifold. The indices defined can be used to inform the user about the "reliability" of a relation, as well as to enforce a minimal consistency level when updates are performed. They also enable to detect updates which induce a brutal decrease of the consistency level (one can then exhibit the conflictual tuples and possibly discover errors). As to the refinement mechanism, it makes it possible to eliminate some inconsistent interpretations, thus to keep the precision of a relation as high as possible.

This work is still preliminary and it opens several perspectives. First, it would be worthy to study how the incremental calculus of Π and N can be optimized, i.e., how $inc(t)^+$ can be determined as efficiently as possible (e.g., by means of appropriate index structures). Another aspect is related to the fact that in the approach presented here, every FD is dealt with individually. A question which arises concerns the definition of a sound and complete axiomatic system in the spirit of Armstrong's axioms and the work described in [19]. Still another perspective is about possibilistic database repair. For instance, it could be of interest to devise a method for computing the minimal repair of an inconsistent database yielding a possibility degree of consistency equal to 1. The approach proposed by Koch and Olteanu [17] in a probabilistic database framework could provide some interesting leads for this purpose. Finally, it would be interesting to study if FD's could be discovered through a tractable data mining process. In this case, the degrees Π and N could be used as scores to filter out the poor FDs.

References

1. L. Antova, T. Jansen, C. Koch, and D. Olteanu. Fast and simple processing of uncertain data. In *Proc. of ICDE'08*, pages 983–992, 2008.
2. L. Antova, C. Koch, and D. Olteanu. 10^{10^6} worlds and beyond: Efficient representation and processing of incomplete information. Technical Report INFOSYS-TR-2005-4, Saarland University Database Group, November 2005.
3. P. Atzeni and N. Morfuni. Functional dependencies and constraints on null values in database relations. *Information and Control*, 70(1):1–31, 1986.
4. D. Barbara, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *IEEE Trans. on Knowledge and Data Engineering*, 4(5):487–502, 1992.

5. O. Benjelloun, A. D. Sarma, A. Halevy, and J. Widom. Uldbs: Databases with uncertainty and lineage. In *Proc. of VLDB'06*, pages 953–964, 2006.
6. B. Bhuniya and P. Niyogi. Lossless join property in fuzzy relational databases. *Data and Knowledge Engineering*, 11(2):109–124, 1993.
7. P. Bosc and O. Pivert. About projection-selection-join queries addressed to possibilistic relational databases. *IEEE Trans. on Fuzzy Systems*, 13(1):124–139, 2005.
8. R. Cavallo and M. Pitarelli. The theory of probabilistic databases. In *Proc. of VLDB'87*, pages 71–81, 1987.
9. G. Chen. Fuzzy functional dependencies and a series of design issues of fuzzy relational databases. In *Fuzziness in Database Management Systems*, pages 166–185. Physica Verlag, 1995.
10. E. Codd. Extending the relational database model to capture more meaning. *ACM Transactions on Database Systems*, 4(4):397–434, 1979.
11. J. Cubero and M. Vila. , a new definition of fuzzy functional dependency in fuzzy relational databases. *Journal of Intelligent Systems*, 9(5):441–448, 1994.
12. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *Proc. of VLDB'04*, pages 864–875, 2004.
13. D. Dey and S. Sarkar. A probabilistic model and algebra. *ACM Trans. on Database Syst.*, 21(3):339–369, 1996.
14. D. Dey and S. Sarkar. Generalized normal forms for probabilistic relational data. *IEEE Transactions on Knowledge and Data Engineering*, 14(3):485–497, 2002.
15. T. Eiter, T. Lukasiewicz, and M. Walter. Extension of the relational algebra to probabilistic complex values. In *Proc. of FoIKS'00*, pages 94–115, 2000.
16. N. Fuhr and T. Rölleke. A probabilistic nf2 relational algebra for integrated information retrieval and database systems. In *Proc. of the 2nd World Conf. on Integrated Design and Process Technology*, pages 17–30, 1996.
17. C. Koch and D. Olteanu. Conditioning probabilistic databases. *PVLDB*, 1(1):313–325, 2008.
18. L. Lakshmanan, N. Leone, R. Ross, and V. Subrahmanian. Probview: A flexible probabilistic system. *ACM Trans. on Database Syst.*, 22(3):419–469, 1997.
19. M. Levene and G. Loizou. Axiomatization of functional dependencies in incomplete relations. *Theoretical Computer Science*, 206(1-2):283–300, 1998.
20. Y. Lien. On the equivalence of database models. *J. of the ACM*, 29(2):333–362, 1982.
21. W. Lipski. Semantic issues connected with incomplete information databases. *ACM Transactions on Database Systems*, 4(3):262–296, 1979.
22. W. Liu. The fuzzy functional dependency on the basis of the semantic distance. *Fuzzy Sets and Systems*, 59(2):173–179, 1993.
23. H. Prade. Lipski's approach to incomplete information databases restated and generalized in the setting of Zadeh's possibility theory. *Information Systems*, 9(1):27–42, 1984.
24. K. Raju and A. Majumdar. Fuzzy functional dependencies and lossless join decomposition of fuzzy relational database systems. *ACM Transactions on Database Systems*, 13(2):129–166, 1988.
25. P. Saxena and B. Tyagi. Fuzzy functional dependencies and independencies in extended fuzzy relational database models. *Fuzzy Sets and Systems*, 69(1):65–89, 1995.
26. Y. Vassiliou. Functional dependencies and incomplete information. In *Proc. of VLDB'80*, pages 260–269, October 1980.
27. L. Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1:3–28, 1978.

Continuous Uncertainty in Trio

Parag Agrawal, Jennifer Widom
{paraga,widom}@cs.stanford.edu

Stanford University

Abstract. We present extensions to Trio for incorporating *continuous uncertainty* into the system. Data items with uncertain possible values drawn from a continuous domain are represented through a generic set of functions. Our approach enables precise and efficient representation of arbitrary probability distribution functions, along with standard distributions such as Gaussians. We also describe how queries are processed efficiently over this representation, without knowledge of specific distributions. For queries that cannot be answered exactly, we can provide approximate answers using sampling or histogram approximations, offering the user a cost-precision trade-off. Our approach exploits Trio's *lineage* and *confidence* features, with smooth integration into the overall data model and system.

1 Introduction

There has been a great deal of interest recently in developing database management systems to handle data that is *uncertain* [1, 3, 10, 13]. Many of these efforts, including our own work in *Trio* [13], use a data model based on a set of *alternative values* for tuples, and/or *probabilities* (or *confidences*) assigned to a tuple's existence. However, some applications must manage data that cannot be represented with such forms of *discrete* uncertainty. For example, sensor measurements may be described by a Gaussian distribution around the reported reading (based on sensor calibration) [5]; astronomical data may contain locations described by two-dimensional Gaussians; a database that records time-of-day may use intervals bounding the possible time; predictive models stored in a database may include continuous probability distributions.

There have been a few proposals for managing continuous uncertainty in a DBMS. Reference [5] suggests using an abstract data type for continuous uncertain values, focusing specifically on Gaussian distributions. The *Orion* system [11] generalizes to other distributions using discrete histogram approximations, and defines a semantics for interpreting SQL over this data. In this paper we describe how to incorporate continuous uncertainty into Trio. Our proposal is more general than the previous work in both data representation and querying, but admittedly it is not implemented yet. More detailed discussion of related work is provided in Section 6.

Next we describe in brief the key components of our proposal: semantics of the data model and query language, data representation in the system, query

processing over this representation, interfaces to the uncertain data, and integration into other aspects of Trio, including Trio's model for discrete uncertainty and its important *lineage* feature.

Data Model Semantics: Models for uncertain data usually are based on *possible worlds*: a set of possible instances for the database. With discrete uncertainty, an uncertain database always represents a finite set of possible-instances; with continuous uncertainty, we may represent an infinite set. Specifically, the value of an uncertain attribute may be an arbitrary *probability distribution function* (*pdf*) over a continuous domain, describing the possible values for the attribute. For example, an uncertain temperature may be described as a Gaussian distribution around a mean temperature of 50 with variance 12; a time may be described as a uniform distribution between 9:20 and 9:25. Gaussian and uniform distributions are special cases of pdfs, as are the discrete alternatives on which Trio was based originally. In our model, a pdf may also be over a high-dimensional domain, representing multiple correlated uncertain attributes. For example, predicted object locations may be comprised of uncertain correlated *latitude-longitude* pairs, i.e., two-dimensional pdfs.

Query Semantics: With possible-world semantics, the semantics of a standard query is defined naturally: The result of a query Q over an uncertain database U must represent the result of applying Q to each possible-instance of U. There is no problem with these semantics even when the set of possible-instances is infinite, although some query results may be surprising. For example, a query filtering by equality on values that are continuous pdfs will always return an empty result: the probability of a value from a pdf being equal to any specific value is zero.

Representation: Previous work [5, 11] has used symbolic representations for a set of known distribution types, and histogram approximations for the rest. For example, Gaussian distributions can be represented by mean and variance; uniform distributions by their endpoints. We have taken a different approach to handling a wide class of distributions: we represent a pdf by a set of functions. For example, a pdf ρ can be represented by a pair of functions:

- The function *weight*(*low*,*high*) returns the total probability of a value in ρ lying between *low* and *high*. For example, *weight*(3,4) for the uniform distribution over $[2, 6]$ returns 0.25.
- The function *sample*(*low*,*high*) returns a random value between *low* and *high* according to ρ. For example, *sample*(3,4) for a Gaussian distribution with mean 4 is more likely to return a value close to 4 than close to 3.

We also generalize both functions to multidimensional pdfs. This approach subsumes the symbolic representation for specific distributions, while at the same time enabling higher efficiency and accuracy than histogram approximations for the rest.

Query Processing: In Trio, query results include *lineage* identifying the data from which each result value was derived [13]. Lineage is needed to properly represent uncertainty, and to compute result confidence values lazily. (Orion [11] uses *history* in a similar way.) In the presence of pdfs, we make all processing over pdf values lazy, deferring potentially expensive computations until they are needed. At query time, we simply generate lineage, and for those results involving pdfs, we extend the the lineage to include relevant predicates and mappings. The information contained in the lineage is sufficient to be able to compute the functions that encode the result lazily. For example, a query asking for temperatures less than 60 would generate lineage annotated with the "< 60" predicate. Now if we have a result pdf ρ' whose lineage points to a pdf ρ, the function $weight_{\rho'}(low,high)$ on ρ' is translated to $weight_{\rho}(low,\max(high,60))/weight_{\rho}(-\infty,60)$ on ρ.

The "translation" approach motivated above can be used to answer many queries efficiently. However, for expensive queries we support approximate answers, either using the *sample* function, or a histogram based on the *weight* function; both options offer the user a cost-precision trade-off. Query processing is modular and generic: our approach of mapping functions on result data to functions on data in the result's lineage is independent of specific distributions—all functions can be treated as "black boxes." However, for specific distributions we do introduce specialized processing that is more efficient.

Interfaces: It is difficult to know how to present a pdf in a database system API or user interface. Our approach is to offer an extensible suite of relevant properties of a pdf; examples are *mean*, *variance*, *weight in a range*, or *histogram approximation*. All of these properties are computed easily using calls to the functions that represent the pdf (like *weight* and *sample*), again possibly using an approximation. Users may add additional properties by providing translations to the functions.

Integration into Trio: In addition to lineage and discrete uncertainty, Trio's data model includes *confidence* values associated with tuples, denoting the probability of the tuple existing. In our new model and query language for continuous uncertainty, operations tend to modify the probability of a tuple's existence. Thus, the confidence feature is particular helpful for integrating pdfs into Trio. Consider for example the predicate "≤ 4" applied to a uniform distribution over $[3, 7]$. The result can be represented by a uniform distribution over $[3, 4]$ that exists with probability 0.25, or more generally one fourth of the original probability.

Not only do Trio's previous features (lineage and confidences) help integrate continuous uncertainty into the system, but the ability to represent and query pdfs enhances some pre-existing Trio functionality. For example, aggregate query results [8] may naturally be represented as continuous pdfs, and pdfs over discrete domains can sometimes yield more efficient representations than tuple alternatives, for certain types of data.

2 Data Model

2.1 Original ULDB Data Model

We present a brief overview of the the basic ULDB model; for details see [2]. Our extensions for continuous uncertainty will be presented in Section 2.2. We use a contrived example for illustrative purposes. Table `Reading(sensor-id,temp)` maintains temperature observations from sensors, and the table `Location(sensor-id,zone)` maintains locations of sensors.

Alternatives: ULDB relations are comprised of *x-tuples*. Each x-tuple consists of one or more *alternatives*, where each alternative is a regular tuple over the schema of the relation. For example, if sensor 1 is located in zone A or B, then in table `Location` we have:

Location(sensor-id,zone)
`(1,A) || (1,B)`

This x-tuple yields two *possible instances* or *possible worlds* for table `Location`. In general, the possible instances of a database D correspond to all combinations of alternatives for x-tuples in D. In cases like the example tuple above in which only some attributes are uncertain, attribute-level uncertainty may be used:

sensor-id	**zone**
1	`A || B`

Confidences: Numerical *confidence* values may be attached to alternatives of an x-tuple. Suppose sensor 1 is in zone A with confidence 0.4 and B with confidence 0.6, while sensor 2 is operating at all with confidence 0.7. Then we have:

Location(sensor-id,zone)
`(1,A):0.4 || (1,B):0.6`
`(2,C):0.7`

The sum σ of confidence values of all alternatives in an x-tuple must be at most 1. If σ for an x-tuple is less than 1, with probability $1-\sigma$ the tuple is not present. X-tuples are assumed to be independent

Lineage: Lineage in ULDBs is recorded at the granularity of tuple alternatives: lineage connects an x-tuple alternative to the other x-tuple alternatives from which it was derived. Consider joining tables `Location` and `Reading` to create a table `Temp(zone, temp)`. Let column ID serve as a unique identifier for each x-tuple, and let (i, j) denote the jth alternative of the x-tuple with ID i. See the join result below. For example, tuple $(B, 77)$ in `Temp` (identified by $(31, 2)$) is derived from $(1, B)$ in `Location` and $(1, 77)$ in `Reading`.

ID	**Location(sensor-id,zone)**
11	`(1,A):0.4 || (1,B):0.6`
12	`(2,C):0.7`

ID	Reading(sensor-id,temp)
21	(1,77)
22	(2,81)

ID	Temp(zone,temp)
31	(A,77):0.4 || (B,77):0.6
32	(C,81):0.7

```
lineage(31,1) = {(11,1), (21,1)}
lineage(31,2) = {(11,2), (21,1)}
lineage(32,1) = {(12,1), (22,1)}
```

Possible Worlds Semantics: Query results in Trio follow possible worlds semantics. Consider a ULDB D whose possible instances are $D_1, \ldots, D_n$. The result of a query Q on D must have as its possible instances $Q(D_1), \ldots, Q(D_n)$. For example, the join of `Location` and `Reading` above should logically perform the join in all four possible instances of the ULDB. Of course a query processing algorithm does not expand possible instances; it computes the ULDB representation of the query result directly from D.

Lineage imposes restrictions on the possible instances of a ULDB, effectively coordinating the uncertainty in derived data with the uncertainty in the data from which it was derived. Roughly speaking, a tuple alternative is present in a possible instance if and only if all the alternatives in its lineage are present, although the actual constraints are somewhat more complex [2]. For the example above, $(B, 77)$ is present in exactly those possible instances that also contain the tuples in its lineage, i.e., $(1, B)$ and $(1, 77)$ are present.

Lazy Confidence Evaluation: The confidence values associated with alternatives in a query result must be consistent with possible worlds semantics: the confidence of an alternative is always the sum of probabilities of all possible worlds that contain that alternative. It is $\#P$-hard to compute these confidence values in general. The use of lineage allows us to compute result confidences either during query processing or on-demand [13].

2.2 Continuous Uncertainty Extensions

The following extensions to the data and lineage components of the ULDB model enable it to manage continuous uncertainty. Note that here we are describing the model only. Specific representation and query processing details are discussed in Sections 3 and 4 respectively.

Pdf Attributes: We now allow *pdf attributes* in x-tuple alternatives. The value for a pdf attribute is given by a probability distribution function over a given domain. The domain may be discrete or continuous. For example, $P(A) = 0.4; P(B) = 0.6$ is a pdf over a discrete domain $\{A, B\}$; a pdf of a Gaussian distribution $G(77, 20)$ with mean 77 and variance 20 is a pdf over the continuous

domain of all real numbers $\mathcal{R}$. Suppose the sensors used to record temperature values in the `Reading` table have errors described by Gaussian distributions: say sensors 1 and 2 have variance of 20 and 30 respectively. The uncertainty in a sensor's temperature report can now be captured in the database by making `temp` a pdf attribute over $\mathcal{R}$. Using $G(\mu, \sigma^2)$ to represent a Gaussian pdf:

Reading(sensor-id,temp)
`(1, G(77,20))`
`(2, G(81,30))`

The `Reading` table now has an infinite number of possible worlds, since it contains a pdf attribute over a continuous domain. Any table with two tuples of the form $(1, x)$ and $(2, y)$ is a possible instance of `Reading`. As with other uncertain data in Trio, the values of pdf attributes are independent. Hence, the probability of a possible instance $(1, x), (2, y)$ is obtained by multiplying the value $G(77, 20)$ assumes at x with the value $G(81, 30)$ assumes at y. The probability of an individual possible world is infinitesimally small, but instances with values of x and y close to the respective means 77 and 81 are more likely.

Note that Trio's attribute-level uncertainty (Section 2.1) is captured by pdf attributes over finite discrete domains. Also note that the presence of alternatives along with pdf attributes allows us to represent certain cases that pdf attributes alone can't represent. For example, this x-tuple represents a temperature 78 with probability 0.3 or a Gaussian around 77 with variance 20:

Temperature(zone,temp)
`(A,78):0.4 || (A, G(77,20)):0.7`

Predicates applied to pdf attributes can affect confidence values of result alternatives. For example, consider a tuple that exists with confidence 0.8 and has just one pdf attribute whose value is uniformly distributed between 2 and 6 (denoted by $U(2, 6) : 0.8$). The result of applying a predicate $A \leq 5$ to this tuple is $U(2, 5) : 0.6$.

We also support correlated pdf attributes. Correlations may exist in the base data, for example uncertain location data may be represented by correlated x and y coordinates. A joint pdf (over possibly heterogeneous domains) is used to represent multiple correlated attributes. Correlated pdf attributes, and processing queries over them, are described in more detail in Section 5.1. Note that independent attributes in base data can yield correlated attributes in the result. For example, a selection query with predicate $A < B$ over pdfs A and B can make attributes corresponding to A and B be correlated in result. Lineage is used to correctly handle these correlations, as described in Section 4.2 .

Lineage Extensions: An important part of our ULDB extension is to include *predicates* and *mappings* as part of lineage, but at the table granularity. Predicates in the `where` clause of a query that reference at least one pdf attribute are

not evaluated at query time. Instead, they are stored as part of the new table-level *pdf-lineage*. In addition, references to pdf attributes in the `select` clause are stored in the pdf-lineage as mappings from result attributes to expressions over input attributes. Pdf-lineage allows us to evaluate the values for the pdf attributes in a result alternative lazily, along with evaluation of confidence values. The generation of pdf-lineage is described in Section 4.1, and its utilization for on-demand uncertainty processing is described in Section 4.2.

3 Representation and Interfaces

We now describe how pdf attributes can be represented in the system and exposed to the user or application. For now, we describe our techniques for pdfs over the simple domain of real numbers. We will discuss higher-dimensional domains for joint pdfs in Section 5.1.

A pdf is represented by an extensible set of functions. Consider a pdf value ρ with a density function f. This pdf can be represented by a function *weight*(l, h) that returns the weight of the pdf ρ between l and h.

$$weight(l, h) = \int_l^h f(x)dx$$

For example, for Gaussians with mean μ and variance σ^2:

$$weight(l, h) = erf(\frac{h-\mu}{\sigma\sqrt{2}}) - erf(\frac{l-\mu}{\sigma\sqrt{2}})$$

where *erf* is the error function. The *weight* function can completely describe any pdf, but we may often use additional functions in the representation for increased efficiency and convenience. One such function is *sample*(l,h), which returns values between any given l and h drawn at random according to the pdf.

Notice that a call to *sample* can be computed by using calls to *weight* and a uniformly-distributed random number generator. Conversely, an approximation to *weight* can be computed using calls to *sample* with Monte-Carlo simulations. In addition to *weight* and *sample*, we might use *inverse-cdf*, *fourier*, or *describe* for a pdf ρ representing a random variable X:

- The function *inverse-cdf*(v) returns a value r such that $v = weight(-\infty,r)$. Intuitively, *inverse-cdf*(v) returns the $100v$ percentile value for the pdf. For example, the *inverse-cdf* of a Gaussian with mean μ and variance σ^2 is:

  $$inverse\text{-}cdf(v) = \mu + \sqrt{2}\sigma erf^{-1}(2p-1)$$

  If the *inverse-cdf* function is not instantiated, it can be computed approximately using calls to *weight* and binary search, or using *sample*. Functions *weight* and *sample* can also be computed using *inverse-cdf*.

- The function *fourier*(x) returns the value of the Fourier transform of the pdf of the random variable X at x. The Fourier transform may be known as an expression for certain distributions. Efficient numerical computation of the Fourier transform and inverse Fourier transform has been studied extensively. Function *fourier* may be computed using *weight* and vice-versa.
- The function *describe*() returns a parametrized form of the pdf. For example, *describe* might return [`Type:"Gaussian",Parameters:`μ,σ^2]. For a known type of pdf, the system can compute *describe* using *weight*, *sample*, or *inverse-cdf*, and vice-versa.

Since the functions above can be computed using calls to other functions, not all of them need to be instantiated. In fact, all the above functions can be computed (approximately) if any one of them were instantiated. Of course when more functions are instantiated, it may be possible to obtain more precise answers more efficiently. The functions above are our starting point. We will provide hooks to the user to create additional functions.

Pdf data represented as functions is presented to the application or user through *interfaces*. Interfaces, like functions, return properties of the pdf, but they are not used to represent the pdf and hence cannot be used to compute functions or other interfaces. Here are two examples of interfaces:

- A *discrete histogram approximation* might be used by a human to visualize a pdf. This interface is computed easily using *weight*: the height of a histogram bar of width $2 \cdot \epsilon$ around a is the result of *weight*($a - \epsilon$, $a + \epsilon$).
- Some applications may ask for the *median* of a pdf attribute. The property of the median m is that *weight*($-\infty$,m) = *weight*(m,∞) = 0.5. Median m can be found approximately by calling *weight* repeatedly with binary search. The precision of the result improves as more calls are made.

Although *weight* is suitable for computing the discrete histogram interface, it isn't ideal for median, and possibly other interfaces. On the other hand, median can be computed easily using *inverse-cdf* by calling it with argument 0.5. If *inverse-cdf* is instantiated, it should be used for a median interface. If *inverse-cdf* is not instantiated, calling it from the median interface will degenerate to the same binary search over calls to *weight*. Some other interesting interfaces might be are mean, variance, percentile, inverse-percentile, and human-readable form of *describe*. All can be implemented using the functions described earlier, some more efficiently than others.

Our extensible functional representation, along with the notion of interfaces, sets up a framework permitting easy incorporation of new techniques for pdf attributes. We will discuss how the set-of-functions approach enables efficient query processing, and introduces a new optimization problem over a space of "plans" that can be compared in terms of precision and efficiency.

4 Query Processing

We consider select-project-join queries over x-relations with pdf attributes. For this paper, we consider only conjunctive `where` clauses. Generalizing the query

language is a topic of future work. Users and applications may pose queries, use *interfaces* (as described in Section 3) to examine the result data, and invoke confidence computation in the usual Trio style [13].

In Trio, query results include lineage identifying the data from which each result value was derived, as described in Section 2.1. Along with tracing the origins of result data, lineage is also used to properly represent possible worlds semantics, and to compute result confidence values lazily. For pdf attributes we also use lineage to handle uncertainty in a lazy manner, deferring potentially expensive computations until they are needed. More precisely, we generate lineage at query-time, then use it to compute the functions corresponding to the pdf attributes in the result data.

4.1 Query-Time Processing

Query-time processing involves generating both data and lineage. We describe each of these in turn, focusing on new requirements for managing pdf attributes.

Data Processing: In Trio, data processing at query time uses a standard relational processor. We use the algorithm described in [13] with the following modifications:

- Any predicate in the `where` clause referencing one or more pdf attributes is not evaluated. The query is processed using only the remaining predicates in the `where` clause. (Recall we assume conjunctive `where` clauses).
- A *placeholder* is created for each element in the *select* clause that is of type pdf.

For example, consider a Trio query over relation $R(A, B, C)$, where A and B are pdf attributes:

```
select A, B+5, C, D
from R
where A ≤ 5 and B ≤ A and C = 11
```

We instead execute the Trio query:

```
select X1, X2, C, D
from R
where C = 11
```

where $X1$ and $X2$ are placeholders.

Lineage Generation: For each alternative in the result, lineage is generated as in [13]. In addition, we generate table-level *pdf-lineage* as introduced in Section 2.2. Specifically, pdf-lineage is composed of two parts: *predicates* P, recording all unevaluated predicates from the `where` clause, and *mappings* M, recording

mappings from `select` clause placeholders to expressions over input attributes. Intuitively, pdf-lineage captures portions of the query that we wish to evaluate later.

For the example query above, P is $A \leq 5 \wedge B \leq A$, and M is $X1 \rightarrow R.A, X2 \rightarrow R.B{+}5$. Pdf-lineage, along with the standard Trio lineage is sufficient for confidence computation and evaluation of functions for the pdf attributes in the result. We discuss how lineage is used for these two tasks in the next section.

4.2 On-Demand Processing

We now describe how we compute pdf attributes and confidence values in the result data. This processing occurs on-demand after the data processing and lineage generation is complete. Since pdf attributes are represented as functions, the requirement is that we are able to answer any calls to functions representing result pdfs. For any result alternative, we can trace its lineage all the way to the base alternatives from which it is derived, as in [2]. We can similarly build transitive versions of P and M, say P_t and M_t, using a recursive procedure. For example, consider a base tuple t_1 with pdfs A, B, and a result tuple t_2 with pdf-lineage $P = A \leq 5 \wedge B \leq A$ and $M = X1 \rightarrow A, X2 \rightarrow B + 5$, as above. Suppose another query is executed on this result, producing a tuple t_3 with pdf-lineage $P = X2 \geq 10$ and $M = X3 \rightarrow X1 + 5, X4 \rightarrow X2$. The transitive pdf-lineage for the tuple t_3 is $P_t = A \leq 5 \wedge B \leq A \wedge B + 5 \geq 10$ and $M_t = X3 \rightarrow A + 5, X4 \rightarrow B + 5$.

The transitive lineage of a result tuple only points to base alternatives, and all predicates and mappings in P_t and M_t only refer to attributes in the base alternatives along with result placeholders. As in regular Trio, unfolding to the base data is required since the input relations being processed in any query may not be independent, while all base data is independent.

We now discuss four complimentary techniques for on-demand processing: *translation*, *discrete approximation*, *fourier*, and *sampling*.

Translation: Some simple queries permit a very efficient technique that gives precise answers. Consider for example an alternative with $P_t = A \leq 5 \wedge B \geq 10$, and $M_t = X1 \rightarrow A, X2 \rightarrow B$, where A and B are base pdfs and $X1$ and $X2$ are pdfs in the result alternative. The confidence of the result alternative is first computed as described in [2], but may need to be further scaled down because of unevaluated predicates over pdf attributes. For this example, the scaling factor s is computed as:

$$s = weight_A(-\infty, 5) * weight_B(10, \infty)$$

The result alternative exists only in possible worlds with values for A and B that satisfy the predicate, and the product is taken because base pdfs are assumed to be independent. Similarly, the *weight* functions for $X1$ and $X2$ are:

$$weight_{X1}(l, h) = \frac{weight_A(\min(l, 5), \min(h, 5))}{weight_A(-\infty, 5)}$$

$$weight_{X2}(l,h) = \frac{weight_B(\max(l,10), \max(h,10)))}{weight_B(10,\infty)}$$

This simple technique yields precise and efficient answers. Unfortunately, for somewhat more involved queries, this approach does not work. For example, if P_t has any predicate that involves more than one pdf, such as $A \leq B$, where A and B, we cannot compute confidence values and weight functions by simple translation.

Discrete Approximation: Discrete approximations to pdfs can be used to get approximate answers using a "numerical integration" approach. This technique, though less precise and efficient than the translation technique above, is applicable for a larger class of P_t and M_t. Consider for example an alternative with $P_t = A \leq B$ and $M_t = X1 \rightarrow A$. Let us assume that $weight_A(0,100)$ is almost 1. A can be approximated using 100 intervals between points $0,\ldots,100$ for this computation. This approximation can be represented using a vector $v = [weight(0,1),\ldots,weight(99,100)]$. The confidence scaling factor s can be approximated using:

$$s = \sum_{0 \leq i < 100} weight_A(i,i+1) \cdot weight_B(i,\infty)$$

An approximation to the weight function for the pdf $X1$ in the result can be computed approximately as:

$$weight_{X1}(j,k) = \frac{1}{s} \cdot \sum_{j \leq i \leq k} weight_A(i,i+1) \cdot weight_B(i,\infty)$$

This example uses a discrete approximation for the *weight* function. It is easy to see that *inverse-cdf* can also be used. There is a trade-off between cost and precision when this method is used: more points in the approximation increases both precision and cost. The predicate $A \leq B$ in the example above is equivalent to the comparison of an arithmetic expression with a constant: $A - B \leq 0$. The discrete approximation technique scales poorly as the number of pdf attributes in the arithmetic expression increases, in terms of both cost and precision. For example, an expression like $\sum_i A_i \leq 0$ with n pdfs cab be very inefficient: $O(d^{n-1})$ where d points are used to approximate each pdf. However, it should be noted that the discrete approximation technique is applicable for all queries.

Fourier: Addition and subtraction over pdf attributes is computed using a convolution operation on pdfs. A convolution is a kind of integration problem that can be solved using Fourier transforms, since the convolution operation translates to multiplication in Fourier space. Consider for example a result tuple with $M_t = X1 \rightarrow A+2B-C$ and an empty P_t, where A,B,C are pdf attributes.

$$fourier_{X1}(x) = fourier_A(x) \cdot 2fourier_B(x) \cdot -fourier_C(x)$$

For base data, these functions may be symbolic making them efficient and precise; or they may use the *weight* function to numerically approximate the Fourier transform yielding a less precise, more expensive, execution. Also, the *weight* function for the result can be computed numerically from the *fourier* function. This technique can be more efficient and precise than discrete approximations, particularly for expressions involving a large number of base pdfs. The limitation of the technique is that it is useful only for addition and subtraction operations over pdf attributes. It needs to be combined with other approaches when the query involves predicates.

Sampling: Random sampling over possible worlds can be used to compute confidences and to answer calls to functions representing the result. We use the *sample* function for this technique. For example, consider a result tuple with pdf-lineage: $P_t = A + B \leq 4$ and $M_t = X1 \rightarrow A + B$. The confidence scaling factor s can be approximated as the fraction of times the following expression is true using a large number of calls:

$$sample_A(-\infty, \infty) + sample_B(-\infty, \infty) \leq 4$$

A call to *sample* on $X1$ can be answered by repeatedly calling *sample* on A and B until the above expression is satisfied, then returning the value of the left-hand side. Sampling, like discrete approximations, is a very general technique that presents a cost precision trade-off: more precise answers are obtained by using more samples. In general, sampling scales better than discrete approximations as the number pdf attributes in expressions increase, in terms of both efficiency and precision.

Specialized Processing: Our approach for pdf attributes makes it easy to use specialized processing for known distributions. For example, consider a result tuple with $M_t = X1 \rightarrow A + B$ and an empty P_t. If both A and B are Gaussians, *describe* on $X1$ can be computed as [`Type:"Gaussian",Parameters:`$\mu_A + \mu_B, \sigma_A^2 + \sigma_B^2$] where $\mu_A, \mu_B, \sigma_A^2, \sigma_B^2$ are obtained using *describe* calls on A, B. Other functions including *weight*, *sample*, *inverse-cdf*, and even *fourier* can be computed efficiently from *describe* for known distributions. The framework allows for adding knowledge about specific distributions to the system in computing the *describe* function. This processing may be limited in scope with regards to complicated queries, but can yield precise answers efficiently when applicable.

Combining: For any interface, function, or confidence computation call, the answer may be produced by combining more than one of the techniques presented above. Combining the techniques results in multiple "query plans". Choosing a good plan is important, since each evaluation is potentially expensive, and wish to optimize for a combination of precision and efficiency.

5 Discussion

5.1 Correlated Attributes

Correlated attributes in base data are represented through shared functions. If X and Y are correlated, then they share all of their functions. A *weight* call now specifies a l and h on each attribute, i.e., *weight*(l_X, h_X, l_Y, h_Y). Similarly a call to *sample* requires four arguments, and returns a pair x, y. This representation using shared functions is equivalent to allowing higher dimensional domains for a single attribute, while making all dimensions accessible in queries. Recall we assume that all functions on base data are independent, thus dependent attributes need to share functions.

Correlation across attributes in query results may occur as a side-effect of a query (or a series of queries) as discussed earlier in Section 2.2. The query processor handles these correlations by tracing through to the base data for all aspects of the query that process uncertainty as described in Section 4.

5.2 Interfaces and Functions in Queries

Interfaces and functions may be called directly, and they may be used in queries. For example, a query might refer to *median(A)* for a pdf attribute A in the `where` or the `select` clause. This call triggers the computation of the interface (or function) which then produces a certain value used by the remainder of query processing.

6 Related Work

A number of systems have been developed recently for managing uncertain data, e.g., [1, 3, 4, 10, 13]. Many of these systems, including Trio, were designed to support discrete uncertainty. The Orion system [11] incorporates continuous uncertainty by using discrete approximations to process queries over a symbolic representation for known distributions. Reference [6] describes a system that uses sampling as the fundamental technique for managing uncertain data, including continuous uncertainty. A probabilistic XML system that manages continuous uncertainty for known distributions is described in [9].

The systems described above are DBMSs targeted specifically for managing uncertain data, usually with possible worlds semantics. There has been another line of work aimed at incorporating continuous variables (sometimes pdfs) into conventional database management. For example, techniques for querying and indexing pdf attributes are discussed in [5, 7], while *FunctionDB* [12] uses a symbolic representation and algebraic query processing for continuous data that can be represented as polynomials.

7 Future Work

This paper proposes a framework for incorporating continuous uncertainty into the Trio system. In doing so, it exposes numerous challenges for managing continuous uncertainty, and uncertain data in general. We identify the following as some interesting directions for future work:

- The cost-precision trade-offs of the proposed techniques need to be investigated both analytically and experimentally.
- As discussed in Section 4.2, combining our proposed techniques yields a space of "plans", whose selection presents new optimization challenges.
- Functions may be invoked in a query. Thus, It will be interesting to investigate if there are benefits associated with batch-processing these functions.
- The current proposal restricts the query language to have only a conjunctive `where` clause. We would like to allow a more general query language, along with some built-in language and data-type extensions specifically for querying pdfs.
- When pdf attributes are over discrete domains, they capture Trio's precious model. We can explore how the ideas in this paper may benefit Trio with discrete uncertain data.
- Last and most important, implementation in Trio is the next critical step to determine the viability of our approach.

References

1. L. Antova, C. Koch, and D. Olteanu. MayBMS: Managing Incomplete Information with Probabilistic World-Set Decompositions. In *Proc. of ICDE*, 2007.
2. O. Benjelloun, A. D. Sarma, A. Halevy, M. Theobald, and J. Widom. Databases with uncertainty and lineage. *VLDB Journal*, 17(2), 2008.
3. 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.
4. A. Deshpande and S. Madden. MauveDB: Supporting Model-based User Views in Database Systems. In *Proc. of ACM SIGMOD*, 2006.
5. A. Faradjian, J. Gehrke, and P. Bonnet. GADT: A Probability Space ADT for Representing and Querying the Physical World. In *Proc. of ICDE*, 2002.
6. R. Jampani, F. Xu, M. Wu, L. L. Perez, C. Jermaine, and P. J. Haas. MCDB: A Monte Carlo approach to managing uncertain data. In *Proc. of ACM SIGMOD*, 2008.
7. V. Ljosa and A. Singh. APLA: Indexing arbitrary probability distributions. In *Proc. of ICDE*, 2007.
8. R. Murthy and J. Widom. Making aggregation work in uncertain and probabilistic databases. In *Proc. of Workshop on Management of Uncertain Data*, 2007.
9. T. Scholte. Managing continuous uncertain data by a probabilistic xml database management system. Technical report, University of Twente, 2008.
10. S. Singh, C. Mayfield, S. Mittal, S. Prabhakar, S. Hambrusch, and R. Shah. Orion 2.0: Native support for uncertain data. In *Proc. of ACM SIGMOD*, 2009.

11. S. Singh, C. Mayfield, R. Shah, S. Prabhakar, S. Hambrusch, J. Neville, and R. Cheng. Database support for probabilistic attributes and tuples. In *Proc. of ICDE*, 2008.
12. A. Thiagarajan and S. Madden. Querying continuous functions in a database system. In *Proc. of ACM SIGMOD*, 2008.
13. J. Widom. Trio: A system for data, uncertainty, and lineage. *Managing and Mining Uncertain Data*, 2009.

Outerjoins in Uncertain Databases

Robert Ikeda and Jennifer Widom

Stanford University

Abstract. We consider the problem of incorporating outerjoins into uncertain databases. We motivate why outerjoins are useful, but tricky, in uncertain databases, arguing that standard possible-worlds semantics may be inappropriate for outerjoins. We explore a variety of alternative semantics through a running example, and we briefly discuss implementation considerations.

1 Introduction

Uncertain data appears in a number of applications including data integration, information extraction, and entity resolution. There has been a large amount of work recently on *uncertain database systems* for managing this data, e.g., [2–7]. Relational operations such as selection, projection, join, and aggregation have been defined and implemented for these systems. In traditional (certain) databases, there is a variation of the join operation, called *outerjoin*, that includes in its result "dangling" tuples, or those tuples that do not match any tuple of the other relation. We are unaware of any work that considers outerjoins in uncertain databases. Here, we motivate why outerjoins are useful, but tricky, in uncertain databases, and we explore various possible semantics for them.

Outerjoin is a widely used operation in traditional relational databases, because its result does not lose data when the join is "incomplete." With uncertain data, it seems even more likely that non-matching tuples will occur in a join, especially if the data is very noisy, or if the join condition is too strict. Thus, we expect outerjoins to be very useful in uncertain databases.

Unfortunately, it is not clear what the semantics of the outerjoin operator in an uncertain database should be. One obvious idea is to follow the *possible-worlds* semantics that is common for uncertain databases [3], but we will see that this semantics may be inappropriate for outerjoins. Through a running example we explore a variety of other possible semantics, illustrating some advantages and disadvantages of each.

1.1 Running Example

Consider an uncertain database containing information about which companies certain executives (CEOs and CFOs) work for. Suppose the information is extracted from unstructured sources, so it is uncertain; specifically, although executive names are consistent, they may be outdated, and company names may be

inconsistent. (Although this running example is contrived and simplistic for illustrative purposes, it is meant to be representative of information extraction and entity-resolution scenarios.) We have two tables, **CEO(ceo_name,company)** and **CFO(cfo_name,company)**. Sample contents of the two tables are as follows, where ‖ denotes *alternative values* for tuples [3], to be formalized in Section 2.

CEO (ceo_name, company)
Chambers, Cisco ‖ Chambers, Cisco Sys.
Schmidt, Google
Ellison, Oracle ‖ Ellison, Oracle Corp.
Ballmer, Microsoft ‖ Ballmer, Microsoft Corp.
Gates, Microsoft ‖ Gates, MSFT

CFO (cfo_name, company)
Calderoni, Cisco ‖ Calderoni, Cisco Sys.
Catz, Oracle
Jorgensen, Yahoo ‖ Jorgensen, Yahoo!
Liddell, Microsoft ‖ Liddell, Microsoft Corp.

Suppose we are interested in a natural join of the two tables on attribute **company**, but we don't want to lose CEO information for a company because we don't have CFO information, or vice-versa. Thus, we wish to compute a natural outerjoin of tables **CEO** and **CFO**.

First consider *possible-worlds* semantics. Here, the uncertain result must include every possible result obtained by choosing any combination of alternatives from the two input relations. In outerjoin, dangling (unmatched) tuples are "padded" with NULL values. Thus, using possible-worlds semantics, the natural outerjoin of **CEO** and **CFO** gives us:[1]

[1] As we will see in Section 2, we actually require *lineage* in addition to the data shown for correctness.

(company, ceo_name, cfo_name)
Cisco, Chambers, Calderoni ‖ Cisco Sys., Chambers, Calderoni
Oracle, Ellison, Catz
Microsoft, Ballmer, Liddell ‖ Microsoft Corp., Ballmer, Liddell
Microsoft, Gates, Liddell
Cisco, Chambers, NULL ‖ Cisco Sys., Chambers, NULL
Google, Schmidt, NULL
Oracle, Ellison, NULL ‖ Oracle Corp., Ellison, NULL
Microsoft, Ballmer, NULL ‖ Microsoft Corp., Ballmer, NULL
Microsoft, Gates, NULL ‖ MSFT, Gates, NULL
Cisco, NULL, Calderoni ‖ Cisco Sys., NULL, Calderoni
Oracle, NULL, Catz
Yahoo, NULL, Jorgensen ‖ Yahoo!, NULL, Jorgensen
Microsoft, NULL, Liddell ‖ Microsoft Corp., NULL, Liddell

Our feeling is that this result is cumbersome and nonintuitive. In general, outerjoin results using possible-worlds semantics are cluttered with NULLs, since every possible scenario of non-matching join tuples must be represented. A more intuitive result in this case is obtained using what we will call "inner-outerjoin semantics" (Section 5):

(company, ceo_name, cfo_name)
Cisco, Chambers, Calderoni ‖ Cisco Sys., Chambers, Calderoni
Oracle, Ellison, Catz
Microsoft, Ballmer, Liddell ‖ Microsoft Corp., Ballmer, Liddell
Microsoft, Gates, Liddell
Google, Schmidt, NULL
Yahoo, NULL, Jorgensen ‖ Yahoo!, NULL, Jorgensen

Although this result seems superior to the previous one, it may not always be ideal. We will explore two other alternatives to possible-worlds as well.

1.2 Remainder of Paper

In the remainder of this short paper, we first review the *ULDB* data model used by the Stanford *Trio* system [7] and as the basis for this work. We then formally apply Trio's possible-worlds semantics for outerjoin, motivating why other semantics for outerjoin may be more useful. Next, we define three possible alternative semantics, illustrating each one on our running example. We summarize and contrast the four semantics, and finally briefly discuss implementation considerations.

2 ULDB Data Model

We review Trio's *Uncertainty-Lineage Database* (*ULDB*) data model [7]. Each tuple in a ULDB relation consists of a set of mutually-exclusive *alternatives*.

Intuitively, the tuple takes the value of one of its alternatives. Consider again table **CEO** listing CEOs and the companies that they lead:

ID	**CEO (ceo_name, company)**
11	Chambers, Cisco ‖ Chambers, Cisco Sys.
12	Schmidt, Google
13	Ellison, Oracle ‖ Ellison, Oracle Corp.
14	Ballmer, Microsoft ‖ Ballmer, Microsoft Corp.
15	Gates, Microsoft ‖ Gates, MSFT

The ID column is added for identification purposes. We shall identify an alternative of a tuple by a pair (i, j), where i is the tuple ID, and j is the index of the alternative in tuple i. For instance, (Chambers, Cisco Sys.) is identified by (11,2). These identifiers are needed by lineage, as we shall see shortly.

ULDB relations follow the standard possible-worlds semantics: The possible-worlds of a relation (without lineage) are constructed by choosing exactly one alternative from each tuple. Each possible-world is a conventional relation. Note that the **CEO** table above has 16 possible-worlds.

In ULDBs, query result relations include *lineage* to the query's input relations. Lineage is critical for correctness, and is used in Trio for other features as well [7]. To show how lineage is generated by queries, let us add table **CFO**, now with IDs:

ID	**CFO (cfo_name, company)**
21	Calderoni, Cisco ‖ Calderoni, Cisco Sys.
22	Catz, Oracle
23	Jorgensen, Yahoo ‖ Jorgensen, Yahoo!
24	Liddell, Microsoft ‖ Liddell, Microsoft Corp.

Suppose we perform a standard natural (inner) join of tables **CEO** and **CFO** on attribute **company**. We obtain the following table:

ID	**(company, ceo_name, cfo_name)**
31	Cisco, Chambers, Calderoni ‖ Cisco Sys., Chambers, Calderoni
32	Oracle, Ellison, Catz
33	Microsoft, Ballmer, Liddell ‖ Microsoft Corp., Ballmer, Liddell
34	Microsoft, Gates, Liddell

$$\lambda(31,1) = (11,1) \wedge (21,1) \qquad \lambda(31,2) = (11,2) \wedge (21,2)$$
$$\lambda(32,1) = (13,1) \wedge (22,1)$$
$$\lambda(33,1) = (14,1) \wedge (24,1) \qquad \lambda(33,2) = (14,2) \wedge (24,2)$$
$$\lambda(34,1) = (15,1) \wedge (24,1)$$

The λ functions below the table capture *lineage*. For example, $\lambda(31,2) = (11,2) \wedge (21,2)$ says that the second alternative of result tuple 31 was derived from the

second alternative of **CEO** tuple 11 and the second alternative of **CFO** tuple 21. For complex queries, lineage formulas can be any boolean expression, and for derived relations the possible-worlds generated by the relation are dependent on lineage as well as on data. Full details are available in [3].

3 Semantics #1: Possible-Worlds

In Section 1.1 we already saw the ULDB relation produced by applying possible-worlds semantics to the natural outerjoin of **CEO** and **CFO**. Here is the table again, now with lineage:

ID	**(company, ceo_name, cfo_name)**
31	Cisco, Chambers, Calderoni ‖ Cisco Sys., Chambers, Calderoni
32	Oracle, Ellison, Catz
33	Microsoft, Ballmer, Liddell ‖ Microsoft Corp., Ballmer, Liddell
34	Microsoft, Gates, Liddell
35	Cisco, Chambers, NULL ‖ Cisco Sys., Chambers, NULL
36	Google, Schmidt, NULL
37	Oracle, Ellison, NULL ‖ Oracle Corp., Ellison, NULL
38	Microsoft, Ballmer, NULL ‖ Microsoft Corp., Ballmer, NULL
39	Microsoft, Gates, NULL ‖ MSFT, Gates, NULL
40	Cisco, NULL, Calderoni ‖ Cisco Sys., NULL, Calderoni
41	Oracle, NULL, Catz
42	Yahoo, NULL, Jorgensen ‖ Yahoo!, NULL, Jorgensen
43	Microsoft, NULL, Liddell ‖ Microsoft Corp., NULL, Liddell

$$\lambda(31,1) = (11,1) \wedge (21,1) \qquad \lambda(31,2) = (11,2) \wedge (21,2)$$
$$\lambda(32,1) = (13,1) \wedge (22,1)$$
$$\lambda(33,1) = (14,1) \wedge (24,1) \qquad \lambda(33,2) = (14,2) \wedge (24,2)$$
$$\lambda(34,1) = (15,1) \wedge (24,1)$$
$$\lambda(35,1) = (11,1) \wedge \neg(21,1) \qquad \lambda(35,2) = (11,2) \wedge \neg(21,2)$$
$$\lambda(36,1) = (12,1)$$
$$\lambda(37,1) = (13,1) \wedge \neg(22,1) \qquad \lambda(37,2) = (13,2)$$
$$\lambda(38,1) = (14,1) \wedge \neg(24,1) \qquad \lambda(33,2) = (14,2) \wedge \neg(24,2)$$
$$\lambda(39,1) = (15,1) \wedge \neg(24,1) \qquad \lambda(39,2) = (15,2)$$
$$\lambda(40,1) = (21,1) \wedge \neg(11,1) \qquad \lambda(40,2) = (21,2) \wedge \neg(11,2)$$
$$\lambda(41,1) = (22,1) \wedge \neg(13,1)$$
$$\lambda(42,1) = (23,1) \qquad \lambda(42,2) = (23,2)$$
$$\lambda(43,1) = (24,1) \wedge \neg(14,1) \wedge \neg(15,1) \qquad \lambda(43,2) = (24,2) \wedge \neg(14,2)$$

Note that some of the lineage formulas include negation. For example, $\lambda(35,1) = (11,1) \wedge \neg(21,1)$ says that the first alternative of result tuple 35 is present in exactly those possible-worlds in which the first alternative of **CEO** tuple 11 is present and the first alternative of **CFO** tuple 21 is not present.

Compute $R = S_1 \mathring{\bowtie}_P^{PW} S_2$

Step 1: Compute innerjoin. For each pair of tuples s_1 in S_1 and s_2 in S_2, if at least one pair of alternatives satisfies P:

- Create a tuple t in R containing the "horizontal innerjoin" of s_1 and s_2: t contains an alternative a for every pair of alternatives $a_1 \in s_1$ and $a_2 \in s_2$ that satisfies P, with $\lambda(a) = a_1 \wedge a_2$.

Step 2: Add the left relation right-padded with NULLs. For each tuple s_1 in S_1:

- Create a tuple t in R containing an alternative $a = a_1 \cdot \text{NULL}$ for each alternative a_1 in s_1, with $\lambda(a) = a_1 \wedge \neg a_{2_1} \wedge \neg a_{2_2} \wedge ... \wedge \neg a_{2_m}$, where the a_{2_i} are all alternatives in S_2 such that a_1 and a_{2_i} satisfy P.

Step 3: Add the right relation left-padded with NULLs. For each tuple s_2 in S_2:

- Create a tuple t in R containing an alternative $a = \text{NULL} \cdot a_2$ for each alternative a_2 in s_2, with $\lambda(a) = a_2 \wedge \neg a_{1_1} \wedge \neg a_{1_2} \wedge ... \wedge \neg a_{1_n}$, where the a_{1_i} are all alternatives in S_1 such that a_{1_i} and a_2 satisfy P.

Fig. 1. Algorithm for possible-worlds outerjoin.

Figure 1 specifies an algorithm that computes the possible-worlds outerjoin, which we denote by operator $\mathring{\bowtie}^{PW}$. (For simplicity and efficiency, like in Trio [7], our algorithm produces the correct result but does not automatically remove *extraneous data*, i.e., data that cannot appear in any possible-world.) In all of our algorithms, we use "$a \cdot$NULL" to create tuples from the left relation of the join right-padded with NULLs, and similarly "NULL$\cdot a$" for left-padded tuples from the right relation.

Note that although the result table in our example may appear to have $2^9 = 512$ possible-worlds, lineage constrains it to have only 128, as we would expect from the 16 possible-worlds of **CEO** combined with the 8 possible-worlds of **CFO**. Again, the reader is referred to [3] for details.

The result above is correct with respect to possible-worlds semantics, and therefore can be integrated seamlessly with other query operators. However, as discussed earlier, our feeling is that this result is unnecessarily cluttered with NULLs and therefore not very intuitive or useful. We will now explore more intuitive and compact results that still capture the "no lost data" spirit of outerjoin.

4 Semantics #2: Outer-Outerjoin

Our first alternative semantics is referred to as *outer-outerjoin* and denoted by operator $\mathring{\bowtie}^{O}$. Ideally, instead of including in the result a tuple t padded with NULLs whenever t may be dangling in *some* possible-world, we would like to only include t padded with NULLs whenever t is dangling in *every* possible-

world. Unfortunately, this property is very expensive to guarantee in general (essentially requiring an enumeration of possible-worlds), but it is approximated by the outer-outerjoin and does hold in many cases.

Here is the result of this semantics on our running example:

ID	**(company, ceo_name, cfo_name)**
31	Cisco, Chambers, Calderoni ‖ Cisco Sys., Chambers, Calderoni
32	Oracle, Ellison, Catz ‖ Oracle Corp., Ellison, NULL
33	Microsoft, Ballmer, Liddell ‖ Microsoft Corp., Ballmer, Liddell
34	Microsoft, Gates, Liddell ‖ MSFT, Gates, NULL ‖ Microsoft Corp., NULL, Liddell
35	Google, Schmidt, NULL
36	Yahoo, NULL, Jorgensen ‖ Yahoo!, NULL, Jorgensen

$\lambda(31,1) = (11,1) \wedge (21,1)$ $\quad$ $\lambda(31,2) = (11,2) \wedge (21,2)$
$\lambda(32,1) = (13,1) \wedge (22,1)$ $\quad$ $\lambda(32,2) = (13,2) \wedge (22, \text{NULL})$
$\lambda(33,1) = (14,1) \wedge (24,1)$ $\quad$ $\lambda(33,2) = (14,2) \wedge (24,2)$
$\lambda(34,1) = (15,1) \wedge (24,1)$ $\quad$ $\lambda(34,2) = (15,2) \wedge (24, \text{NULL})$
$\lambda(34,3) = (24,2) \wedge (15, \text{NULL})$
$\lambda(35,1) = (12,1)$
$\lambda(36,1) = (23,1)$ $\quad$ $\lambda(36,2) = (23,2)$

Refer to Figure 2 for an algorithm that computes the outer-outerjoin. Notice that NULL-padded alternatives in tuples that do join are assigned a special type of NULL lineage indicating a non-match in the horizontal "outerjoin" (Steps 1(ii) and 1(iii) in the algorithm)—see $\lambda(32,2)$ for example.

It should be obvious that, on our running example at least, this operator produces a result that is considerably more intuitive than the possible-worlds result, while still retaining the property that no tuples from either input relation are lost.

5 Semantics #3: Inner-Outerjoin

Although the outer-outerjoin operator is a significant improvement over the possible-worlds outerjoin, it still contains some NULL-padded results that may be considered unnecessary. Consider for example the second alternative of result tuple 32: (Oracle Corp., Ellison, NULL), derived from **CEO** tuple 13 and **CFO** tuple 22. Within the alternatives of these joining tuples, we have a match on "Oracle," so we might prefer to assume "Oracle" is the correct value, and omit the (Oracle Corp., Ellison, NULL) alternative. Similarly, consider the second and third alternatives of result tuple 34: (MSFT, Gates, NULL) and (Microsoft Corp., NULL, Liddell). This result tuple is derived from **CEO** tuple 15 and **CFO** tuple 24. Here, we have an exact match on "Microsoft," so again we may prefer to omit the alternatives without matches. The next semantics, which we refer to as *inner-outerjoin* and denote by operator $\overset{\circ}{\bowtie}^{I}$, is based on this intuition.

Compute $R = S_1 \mathring{\bowtie}_P^O S_2$

Step 1: For each pair of tuples s_1 in S_1 and s_2 in S_2, if at least one pair of alternatives satisfies P:

- Create a tuple t in R containing the "horizontal outerjoin" of s_1 and s_2:
  - (i) Add an alternative a to t for every pair of alternatives $a_1 \in s_1$ and $a_2 \in s_2$ that satisfies P, with $\lambda(a) = a_1 \wedge a_2$.
  - (ii) For every $a_1 \in s_1$ not used in step (i), add $a = a_1 \cdot \text{NULL}$ to t and set $\lambda(a) = a_1 \wedge (s_2.\text{ID}, \text{NULL})$.
  - (iii) For every $a_2 \in s_2$ not used in step (i), add $a = \text{NULL} \cdot a_2$ to t and set $\lambda(a) = a_2 \wedge (s_1.\text{ID}, \text{NULL})$.

Step 2: Add the left relation's dangling tuples right-padded with NULLs. For each tuple s_1 in S_1 not used in Step 1:

- Create a tuple t in R containing an alternative $a = a_1 \cdot \text{NULL}$ for each alternative a_1 in s_1, with $\lambda(a) = a_1$.

Step 3: Add the right relation's dangling tuples left-padded with NULLs. For each tuple s_2 in S_2 not used in Step 1:

- Create a tuple t in R containing an alternative $a = \text{NULL} \cdot a_2$ for each alternative a_2 in s_2, with $\lambda(a) = a_2$.

Fig. 2. Algorithm for outer-outerjoin.

Figure 3 contains an algorithm for computing the inner-outerjoin. As can be seen, it combines Step 1 from the possible-worlds outerjoin with Steps 2 and 3 from the outer-outerjoin. We saw the result of this semantics on our running example in Section 1.1; here it is again, now with lineage:

ID	**(company, ceo_name, cfo_name)**
31	Cisco, Chambers, Calderoni ‖ Cisco Sys., Chambers, Calderoni
32	Oracle, Ellison, Catz
33	Microsoft, Ballmer, Liddell ‖ Microsoft Corp., Ballmer, Liddell
34	Microsoft, Gates, Liddell
35	Google, Schmidt, NULL
36	Yahoo, NULL, Jorgensen ‖ Yahoo!, NULL, Jorgensen

$$\lambda(31,1) = (11,1) \wedge (21,1) \qquad \lambda(31,2) = (11,2) \wedge (21,2)$$
$$\lambda(32,1) = (13,1) \wedge (22,1)$$
$$\lambda(33,1) = (14,1) \wedge (24,1) \qquad \lambda(33,2) = (14,2) \wedge (24,2)$$
$$\lambda(34,1) = (15,1) \wedge (24,1)$$
$$\lambda(35,1) = (12,1)$$
$$\lambda(36,1) = (23,1) \qquad \lambda(36,2) = (23,2)$$

Compute $R = S_1 \overset{\circ}{\bowtie}{}^{I}_{P} S_2$

Step 1: Compute innerjoin. For each pair of tuples s_1 in S_1 and s_2 in S_2, if at least one pair of alternatives satisfies P:

- Create a tuple t in R containing the "horizontal innerjoin" of s_1 and s_2: t contains an alternative a for every pair of alternatives $a_1 \in s_1$ and $a_2 \in s_2$ that satisfies P, with $\lambda(a) = a_1 \wedge a_2$.

Step 2: Add the left relation's dangling tuples right-padded with NULLs. For each tuple s_1 in S_1 not used in Step 1:

- Create a tuple t in R containing an alternative $a = a_1 \cdot \text{NULL}$ for each alternative a_1 in s_1, with $\lambda(a) = a_1$.

Step 3: Add the right relation's dangling tuples left-padded with NULLs. For each tuple s_2 in S_2 not used in Step 1:

- Create a tuple t in R containing an alternative $a = \text{NULL} \cdot a_2$ for each alternative a_2 in s_2, with $\lambda(a) = a_2$.

Fig. 3. Algorithm for inner-outerjoin.

There is a tradeoff between inner-outerjoin and outer-outerjoin. Inner-outerjoin semantics effectively says that if a tuple t has at least one alternative satisfying the join condition, then alternatives in t that never satisfy the join condition can be dropped. While the result is more compact, it is our first semantics in which some values from the input relation do not appear in the result (although all tuples still appear in some form). In our running example, values "Oracle Corp." and "MSFT" from the input relations do not appear at all in the result.

6 Semantics #4: Cross-Outerjoin

Our last semantics is referred to as *cross-outerjoin* and denoted by operator $\overset{\circ}{\bowtie}{}^{X}$. Now, instead of using a "horizontal outerjoin" (Semantics #2) or "horizontal innerjoin" (Semantics #3) for matching tuples, this semantics uses a "horizontal cross-product."

Refer to Figure 4 for an algorithm that computes the cross-outerjoin. Note that since alternatives in the horizontal cross-product do not always agree on the join attribute **company**, we can no longer compute a natural join for our running example—we must retain the join attributes from the two input relations separately. Here is the result of this semantics on our running example:

Compute $R = S_1 \overset{\circ}{\bowtie}{}^{X}_{P} S_2$

Step 1: For each pair of tuples s_1 in S_1 and s_2 in S_2, if at least one pair of alternatives satisfies P:

- Create a tuple t in R with the "horizontal cross-product" of s_1 and s_2: t contains an alternative a for every pair of alternatives $a_1 \in s_1$ and $a_2 \in s_2$, with $\lambda(a) = a_1 \wedge a_2$.

Step 2: Add the left relation's dangling tuples right-padded with NULLs. For each tuple s_1 in S_1 not used in Step 1:

- Create a tuple t in R containing an alternative $a = a_1 \cdot \text{NULL}$ for each alternative a_1 in s_1, with $\lambda(a) = a_1$.

Step 3: Add the right relation's dangling tuples left-padded with NULLs. For each tuple s_2 in S_2 not used in Step 1:

- Create a tuple t in R containing an alternative $a = \text{NULL} \cdot a_2$ for each alternative a_2 in s_2, with $\lambda(a) = a_2$.

Fig. 4. Algorithm for cross-outerjoin.

ID	**(ceo_company, ceo_name, cfo_company, cfo_name)**
31	Cisco, Chambers, Cisco, Calderoni ‖ Cisco, Chambers, Cisco Sys., Calderoni ‖ Cisco Sys., Chambers, Cisco, Calderoni ‖ Cisco Sys., Chambers, Cisco Sys., Calderoni
32	Oracle, Ellison, Oracle, Catz ‖ Oracle Corp., Ellison, Oracle, Catz
33	Microsoft, Ballmer, Microsoft, Liddell ‖ Microsoft, Ballmer, Microsoft Corp., Liddell ‖ Microsoft Corp., Ballmer, Microsoft, Liddell ‖ Microsoft Corp., Ballmer, Microsoft Corp., Liddell
34	Microsoft, Gates, Microsoft, Liddell ‖ Microsoft, Gates, Microsoft Corp., Liddell ‖ MSFT, Gates, Microsoft, Liddell ‖ MSFT, Gates, Microsoft Corp., Liddell
35	Google, Schmidt, NULL, NULL
36	NULL, NULL, Yahoo, Jorgensen ‖ NULL, NULL, Yahoo!, Jorgensen

$$\begin{array}{ll}
\lambda(31,1) = (11,1) \wedge (21,1) & \lambda(31,2) = (11,1) \wedge (21,2) \\
\lambda(31,3) = (11,2) \wedge (21,1) & \lambda(31,4) = (11,2) \wedge (21,2) \\
\lambda(32,1) = (13,1) \wedge (22,1) & \lambda(32,2) = (13,2) \wedge (22,1) \\
\lambda(33,1) = (14,1) \wedge (24,1) & \lambda(33,2) = (14,1) \wedge (24,2) \\
\lambda(33,3) = (14,2) \wedge (24,1) & \lambda(33,4) = (14,2) \wedge (24,2) \\
\lambda(34,1) = (15,1) \wedge (24,1) & \lambda(34,2) = (15,1) \wedge (24,2) \\
\lambda(34,3) = (15,2) \wedge (24,1) & \lambda(34,4) = (15,2) \wedge (24,2) \\
\lambda(35,1) = (12,1) & \\
\lambda(36,1) = (23,1) & \lambda(36,2) = (23,2)
\end{array}$$

This result may appear cumbersome and not particularly useful. However, if we follow the cross-outerjoin with additional processing based on Trio's *horizontal subqueries* (intuitively, subqueries that treat the alternatives of a tuple as a relation [7]), we get a more intuitive result. Specifically, for each tuple t whose lineage includes both of the input relations (i.e., tuples produced by Step 1 of the algorithm), do the following: Treat t's alternatives as a relation, project once dropping the join attributes from the right input relation and once dropping the join attributes from the left input relation, and union the results without retaining duplicates. For the tuples whose lineage includes just one input relation (i.e., tuples produced by Steps 2 and 3 of the algorithm), simply project away the (NULL) join attributes from the other relation. Although it may sound complex, this processing is not difficult to compute, and together with the cross-outerjoin it yields the following result:

ID	**(company, ceo_name, cfo_name)**
31	Cisco, Chambers, Calderoni ‖ Cisco Sys., Chambers, Calderoni
32	Oracle, Ellison, Catz ‖ Oracle Corp., Ellison, Catz
33	Microsoft, Ballmer, Liddell ‖ Microsoft Corp., Ballmer, Liddell
34	Microsoft, Gates, Liddell ‖ Microsoft Corp., Gates, Liddell ‖ MSFT, Gates, Liddell
35	Google, Schmidt, NULL
36	Yahoo, NULL, Jorgensen ‖ Yahoo!, NULL, Jorgensen

$$\lambda(31,1) = ((11,1) \wedge (21,1)) \vee ((11,1) \wedge (21,2)) \vee ((11,2) \wedge (21,1))$$
$$\lambda(31,2) = ((11,1) \wedge (21,2)) \vee ((11,2) \wedge (21,1)) \vee ((11,2) \wedge (21,2))$$
$$\lambda(32,1) = ((13,1) \wedge (22,1)) \vee ((13,2) \wedge (22,1))$$
$$\lambda(32,2) = (13,2) \wedge (22,1)$$
$$\lambda(33,1) = ((14,1) \wedge (24,1)) \vee ((14,1) \wedge (24,2)) \vee ((14,2) \wedge (24,1))$$
$$\lambda(33,2) = ((14,1) \wedge (24,2)) \vee ((14,2) \wedge (24,1)) \vee ((14,2) \wedge (24,2))$$
$$\lambda(34,1) = ((15,1) \wedge (24,1)) \vee ((15,1) \wedge (24,2)) \vee ((15,2) \wedge (24,1))$$
$$\lambda(34,2) = ((15,1) \wedge (24,2)) \vee ((15,2) \wedge (24,2))$$
$$\lambda(34,3) = ((15,2) \wedge (24,1)) \vee ((15,2) \wedge (24,2))$$
$$\lambda(35,1) = (12,1)$$
$$\lambda(36,1) = (23,1)$$
$$\lambda(36,2) = (23,2)$$

Notice that we now have *disjunctive lineage*, which is generated by Trio when duplicate-elimination occurs [7].

This result may be our most intuitive of all, combining the best aspects of outer-outerjoin and inner-outerjoin. Specifically, it minimizes NULLs in the result, while not losing any values from the input data. In fact, we have recently been working on a new variant of Trio called *Trio-ER* [1], a workbench for entity-resolution. In Trio-ER, we have implemented operators whose behavior is very similar to the cross-outerjoin plus post-processing shown above.

7 Conclusions and Implementation

To summarize, we contrast the four semantics we have considered:

- The primary advantage of possible-worlds outerjoin is its formal foundations, and its adherence to the underlying semantics of Trio and other uncertain database systems. Its disadvantage, obviously, is the unwieldy nature of the results.
- Outer-outerjoin and inner-outerjoin vastly reduce the result size compared with possible-worlds, at the loss of a formal underlying semantics. The primary difference between the two is that outer-outerjoin guarantees to preserve all *values* (not just all *tuples*) present in the input relations, while inner-outerjoin does not. However, inner-outerjoin is more compact, and is appropriate in an entity-resolution scenario if values that agree are always thought to override those that don't.
- The final semantics we presented, cross-outerjoin, is not very intuitive as a stand-alone operator. However, when additional operations are applied as shown in Section 6, it may in fact be the most useful semantics in practice.

Let us briefly discuss implementation of the different semantics. As with outerjoin in conventional database systems, all of our semantics require a two-pass approach where dangling tuples are added after the innerjoin (or variant) is complete. Outer-outerjoin, inner-outerjoin, and cross-outerjoin differ only in how they combine matching tuples. Since we don't expect tuples to have large numbers of alternatives, these differences would probably amount to negligible in-memory processing. Step 1 in the possible-worlds semantics is similar, but Steps 2 and 3 require considerable additional processing or bookkeeping to generate correct lineage.

Considering the Trio system specifically, its implementation is layered over a conventional relational DBMS. Therefore, Trio queries are not executed by a special-purpose processor. Instead, they are translated to conventional SQL queries and operations over encoded ULDB relations [7]. All four of our semantics can be translated in the Trio system, with some requiring more complex SQL constructs than others. As an example, the following single query computes all three steps of the inner-outerjoin on our running example. (For Trio data encoding details such as the `xid` attribute, see [7]. Also, lineage must be computed separately, as in all Trio queries [7].) Note that the underlying SQL query processor would still typically require two passes for the SQL outerjoin included in this query, as discussed above.

```
Select E.company, E.ceo_name, F.cfo_name
From CEO_trans as E Full Outer Join CFO_trans as F
     On Exists (Select * From CEO_trans as E2, CFO_trans as F2
                Where E2.xid = E.xid and F2.xid = F.xid
                And E2.company = F2.company)
Where E.company = F.company
And E.company Is Not Null And F.company Is Not Null
```

The translation for cross-outerjoin simply omits the `Where` clause in the query above, while outer-outerjoin and possible-worlds outerjoin are more complex.

Acknowledgements. We would like to thank Hyunjung Park for several helpful discussions.

References

1. P. Agrawal, R. Ikeda, H. Park, and J. Widom. Trio-ER: The Trio system as a workbench for entity-resolution. Technical report, Stanford University InfoLab, March 2009.
2. L. Antova, T. Jansen, C. Koch, and D. Olteanu. Fast and simple relational processing of uncertain data. In *Proc. of Intl. Conference on Data Engineering (ICDE)*, Cancun, Mexico, Apr. 2008.
3. O. Benjelloun, A. Das Sarma, A. Y. Halevy, M. Theobald, and J. Widom. Databases with uncertainty and lineage. *The VLDB Journal*, 17(2):243–264, 2008.
4. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *The VLDB Journal*, 16(4):523–544, 2007.
5. P. Sen and A. Deshpande. Representing and querying correlated tuples in probabilistic databases. In *Proc. of Intl. Conference on Data Engineering (ICDE)*, Istanbul, Turkey, Apr. 2007.
6. S. Singh, C. Mayfield, S. Mittal, S. Prabhakar, S. Hambrusch, and R. Shah. Orion 2.0: native support for uncertain data. In *Proc. of ACM SIGMOD Intl. Conference on Management of Data*, Vancouver, Canada, June 2008.
7. J. Widom. Trio: A system for data, uncertainty, and lineage. In C. C. Aggarwal, editor, *Managing and Mining Uncertain Data*. Springer, 2008.

Query Containment of Tier-2 Queries over a Probabilistic Database *

Katherine F. Moore, Vibhor Rastogi, Christopher Ré, Dan Suciu
kfm, vibhor, chrisre, suciu@cs.washington.edu

Department of Computer Science & Engineering
The University of Washington, Seattle

Abstract. We study the containment problem for a query language over probabilistic relational databases that allows queries like *"is the probability that q_1 holds greater than* 0.2 *and the probability that q_2 holds greater than* 0.6*?"* where q_1 and q_2 are Boolean conjunctive queries. In addition to being a fundamental problem in its own right, the containment problem is the key problem that an optimizer must solve for many standard optimizations (such as picking up an index or using a materialized view). Our main technical result is that the containment problem is decidable, and we give an EXPSPACE-algorithm based on linear programming for it. We believe that we are the first to study the containment problem for any such probabilistic languages.

1 Introduction

An increasing number of data-centric applications are forced to cope with imprecision: in RFID applications, it is difficult to precisely determine the exact location of people or objects [5, 14], in data integration, researchers have allowed schema mappings to be imprecise to lower the overall cost of integration [6, 8], in information extraction, the state-of-the-art techniques produce extractions automatically, but with a loss of precision [12, 20]. Motivated by these applications, researchers have created systems that model the imprecision in the data using probability theory [2, 13, 16, 17].
The first generation of these systems (such as Trio [19], Mystiq [4]) allow a user to pose a standard query (say in SQL). It is then the job of the database system to combine the probabilities in the data to produce a set of result tuples, where each tuple is annotated with a probability score that reflects the extent to which that tuple is an answer to the posed query.

Example 1. Consider an application that monitors persons and objects as they move through a building using RFID [18]. A user Kate, may want to know the probability that each of her possessions is in her office; she writes a query as in Fig. 1(a). In response to this query, a first-generation system would return a list of her possessions together with a probability that each object is present in her Office (shown in Fig. 1(b)).

```
SELECT DISTINCT O.name
FROM Objects O, Locations L
WHERE L.id = O.id
  AND L.loc = 'Office 380'
  AND O.owner = "Kate"
```

(a)

Name	P
Book	0.9
Laptop	0.3
Mug	0.1

(b)

```
SELECT DISTINCT O.name
FROM Objects O, Locations L
WHERE L.id = O.id
  AND L.loc = 'Office 380'
  AND O.owner = "Kate"
HAVING CONFIDENCE > 0.8
```

(c)

Fig. 1: (a) A first-generation probabilistic query and (b) its output on such a system. Query (c) shows the type of query we study in this paper.

As the research in the area of probabilistic data management has matured, researchers have begun to advocate increasingly sophisticated query languages that allow more complex, direct manipulations of the imprecision in the data [3,9]. A notable feature of these languages is that they allow probability values to be manipulated as first-class citizens. An example of such a query is shown in Fig. 1(c) which asks *"which objects that belong to Kate are in 'Office 380' with probability greater than* 0.8*?"*. These queries are particularly important in applications like RFID event detection, where we want to trigger a real-world action in response to an event (such as sending an email alert when a laptop leaves a room). In addition, the query language that we study captures the essential features of other (more expressive) languages such as recently proposed by Koch [11] or discussed by Fagin et al [7].
In this work, we study the containment problem for conjunctions of queries which are a significant generalization of queries as in Fig. 1(c) (defined formally in § 2.3). Containment is a property of a pair of queries over uncountably many databases, and so it is not immediately clear that the problem is decidable. The main technical contribution of this paper is first showing that containment of such queries is decidable, and further that we can decide this property in EXPSPACE via an algorithm based on Linear Programming.

Outline In §2, we give background on probabilistic databases, provide syntax and semantics for our query language, and formally state our problem. In §3, we show how to construct a linear program that captures the set of databases that will model a particular query. Our main result, the decidability of query containment, is given in §4. We discuss related work (§5) and conclude (§6).

2 Preliminaries

We first give a brief background on probabilistic databases. Then we define our query language and illustrate its use with some examples.

⋆ This work was partially funded by NSF IIS-0713576

2.1 Probabilistic Databases and Orders

A probabilistic database D is a pair $D = (\mathcal{W}, P)$ where $\mathcal{W} = \{W_1, W_2, \ldots, W_n\}$ is a finite set of standard, finite relational databases all conforming to the same schema over an infinite domain $\mathbb{D}$ and $P : \mathcal{W} \rightarrow [0, 1]$ is a distribution function, that is:

$$\sum_{W \in \mathcal{W}} P(W) = 1$$

We refer to $W \in \mathcal{W}$ such that $P(W) > 0$ is a *possible world*.
This definition is very general and contains all (discrete) relational representations in the literature including tuple independent databases [4], x-tables [3], bid tables [15], factor graphs [17], and world-sets [1].
For a given tuple t in the database, the (marginal) probability of a tuple t is denoted $P(t)$ and is defined by

$$P(t) = \sum_{W : W \ni t} P(W)$$

	Name	Office	CurrentLocation
(t_1)	Kate	380	380
(t_2)	Kate	380	Coffee Room
(t_3)	Julie	380	380

Fig. 2: A Standard Relation R. A probabilistic database is a subset of tuples together with a probability function.

Example 2. Fig. 2 shows a standard relational database with a single relation that contains three tuples, $R = \{t_1, t_2, t_3\}$, this database tracks the offices and current locations of various people.
One possible database D is where each tuple is independent with probability p. Our work does not consider tuple independent databases, but we give a brief explanation of them here for comparison. In a tuple independent database, the set of possible worlds is the set of all subsets of tuples in R, that is, $\mathcal{W} = \mathcal{P}(R)$. The probability $P(W)$ of a world W with k tuples (for $k = 1, 2, 3$) is $P(W) = p^k(1-p)^{3-k}$.

In this work, we view worlds as sets of tuples and then order these worlds by set inclusion (denoted with $\subseteq$). This order is helpful to understand the semantics of the queries that we study in this paper. We do not assume that the tuples in the database are independent, unless explicitly stated. If we arbitrarily assign probabilities to the worlds in this database, we could get many different probability distributions. One possible lattice is depicted in Figure 3.
The result of executing the query, Q(name), on the database shown in Figure 3 is $\{Kate, Julie\}$. However, if the confidence value was 0.8 instead, then the result would be $\{Julie\}$.

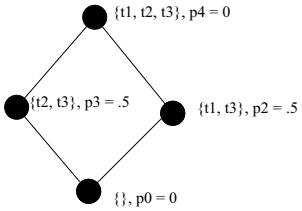


Fig. 3: Lattice Created by Probabilistic Data

```
Q(name) ::=
SELECT DISTINCT R.name
FROM R
WHERE R.currentLocation = '380'
HAVING CONFIDENCE >= .4
```

2.2 Syntax and Semantics of Tier-2 Queries

We consider queries with two tiers: A first tier query is a standard conjunctive Boolean query and is denoted by a lower case q below. The second tier of queries is a set of conjuncts each of which makes a probability statement about first tier queries:

$$q ::= \exists x_1, \exists x_2, \ldots R_1(\bar{x}_1), \ldots R_m(\bar{x}_m)$$
$$Q ::= \exists y_1, \exists y_2, \ldots \exists y_k . P(q_1) \star p_1, P(q_2) \star p_2, \ldots, P(q_n) \star p_n$$

Where $\star$ represents either the $>$ or $\geq$ operator, and $p_i \in (0, 1]$.
Our Tier-2 queries only allow the comparison between the query probability and the values using $>$ or $\geq$. This is essential for the Tier-2 queries to be "monotonic", in a sense that we will make precise below.
As an example, a query that checks for the existence of Kate's book in her office with high probability (similar to the pseudo-SQL query in Fig. 1(c)) is expressed as:

$$\exists i . P\left(\texttt{Object}(i, \text{'Book'}, \text{'Kate'}), \texttt{Location}(i, \text{'Office 380'})\right) > 0.8$$

The semantics of Tier-2 queries requires that we define the semantics of Tier-1 queries. For a Tier-1 query, q, we evaluate q in a given world W as if W is a standard deterministic database. We write $W \models q$ whenever q is true on W.

Definition 1 ($P(q)$). *Given a Tier-1 query q and a probabilistic database $D = (\mathcal{W}, P)$, we define $P(q)$ as*

$$P(q) = \sum_{W \in \mathcal{W} : W \models q} P(W)$$

Definition 2 (Tier-2 Semantics). *Given a Tier-2 query Q and a probabilistic database $D = (\mathcal{W}, P)$, we say that Q is true on D, and denote $D \models Q$, if there exists a valuation $\theta : \{y_1, \ldots, y_k\} \to \mathbb{D}$ such that for each $i = 1, \ldots, n$ then $P(q_i[\theta(\overline{y})/\overline{y}]) \geq p_i$ where $q_i[\theta(\overline{y})/\overline{y}]$ denotes the query that results when each y_i is substituted with $\theta(y_i)$.*

Thus, Tier-2 queries are for probabilistic databases what standard conjunctive queries (called here Tier-1 queries) are for standard databases. Tier-2 queries are also "monotone". To make this statement precise, we first need to define an order relation on probabilistic databases.

Definition 3 ($F(W, D)$). *Given a probabilistic database $D = (\mathcal{W}, P)$, and a world $W \in \mathcal{W}$, we define the filter of W, written $F(W, D)$ as:*

$$F(W, D) = \{W' \in D | W \subseteq W'\}$$

Definition 4 ($D \leq D'$). *Given two probabilistic databases D and D' over the same set of possible worlds $\mathcal{W}$, we define the order $D \leq D'$ if for all $W \in \mathcal{W}$, $P(F(W, D)) \leq P(F(W, D'))$.*

The next Proposition shows that Tier-2 queries are monotone. The proofs of the Proposition and the Lemma are straightforward and omitted.

Lemma 1. *If $D \leq D'$ and q is a Tier-1 query, then $P(q) \leq P'(q)$.*

Proposition 1. *If $D \leq D'$ and Q is a Tier-2 query, then $D \models Q \implies D \models Q'$*

2.3 Problem Statement and Main Result

We now have all the necessary notation to define our problem formally:

Definition 5 (Tier-2 Query Containment Problem). *Let Q and Q' be Tier-2 Queries, we say that Q is contained in Q' if for* any *probabilistic database D, the following holds:*

$$D \models Q \implies D \models Q'$$

The Tier-2 Query containment Problem is to decide, given as input two Tier-2 queries Q and Q', is Q contained in Q'?

Since containment is a property of an infinite set of databases, it is not immediately clear that containment is decidable. The main result of this work is that it is decidable (in EXPSPACE) using the algorithm of §4.

2.4 Examples

To assist with the reader's understanding of this problem, we introduce two small examples and answer the question: Is Q contained in Q'?

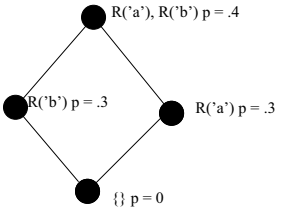


Fig. 4: $D \models Q$

Example 3.

$$Q = P(R('a')) \geq .7, P(R('b')) \geq .7$$
$$Q' = P(R('a'), R('b')) \geq .4$$

To answer the containment question, we first construct a database, D, that satisfies Q. A picture of such a database is shown in Figure 4.
Now, it is also clear that $D \models Q'$, but are there any other databases, D' such that $D' \models Q$ but $D' \not\models Q'$? No. To explain this, we present the following argument. Q has two parts; one of which requires a world where $R('a')$ is true, and one where $R('b')$ is true. If we allow p_1 to represent the probability of all worlds that contain 'a' but not 'b', p_2 to represent the probability of all worlds that contain 'b' but not 'a', and p_3 to represent the probability of all worlds that contain both 'a' and 'b', then the following restrictions hold:

$$p_1 + p_3 \geq 0.7$$
$$p_2 + p_3 \geq 0.7$$

If $p_3 < 0.4$ then it must also be the case that $p_1 > .3$. If this were true, we would have that $p_1 + p_2 + p_3 > 0.3 + 0.7 = 1$. However, this is an invalid database, and thus we reach a contradiction.

Example 4.

$$Q = P(\exists x.R(x)) \geq 1$$
$$Q' = \exists y.P(R(y)) \geq .5$$

Figure 5 shows two different databases, where the one on the left is such that $D \models Q$ and $D \models Q'$. However, the database on the right serves as a counterexample to the containment because for this one, $D' \models Q$, but $D' \not\models Q'$.

3 Tier-2 Queries, Q, as Linear Programs

In this section, we start from a 2nd tier query Q, and then we define a *set of canonical linear programs* for that query, called $SLP(Q)$. We

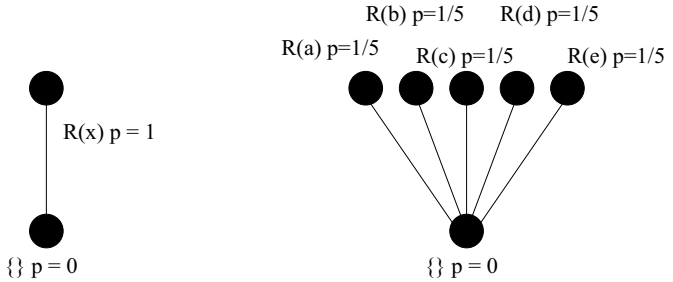


Fig. 5: Two related databases with different results.

prove that $SLP(Q)$ captures the semantics of Q. Because the semantics of Q is defined over probabilistic databases with arbitrarily large sets of worlds, the number of probability values $p_1, p_2, p_3, \ldots$ in the probabilistic database can be arbitrarily large: on the other hand, $SLP(Q)$ has a *fixed* set of variables that only depends on the query, and not on a database. This allows us to use $SLP(Q)$ to concisely capture the semantics of Q, because it makes a statement about arbitrarily many probabilistic values.

3.1 Queries without 2nd-Tier Quantifiers

When the query has no second tier quantifiers, then $SLP(Q)$ is a single linear program, which we denote $LP(Q)$. Let

$$Q = P(q_1) \geq p_1, \ldots P(q_n) \geq p_n$$

$LP(Q)$ has variables $\boldsymbol{v}$ that correspond to equivalence classes of conjuncts of q_i. Fig. 6 shows the algorithm that constructs $LP(Q)$. In this program, a variable v_X represents the probability mass assigned to the worlds W such that the sum of the mass of all worlds W where $W \models q_i$, $i \in X \geq p_i$.

Definition 6. *Let Q be a tier-2 query and $LP(Q)$ its associated linear program (Fig. 6), then for any world W let $\Sigma(W) = \{i | W \models q_i\}$ (note that $\Sigma(W)$ is in $\mathfrak{C}$). For any probabilistic database $D = (\mathcal{W}, P)$ the solution associated with D is denoted $\boldsymbol{v}(D)$ and is an assignment to the variables of $LP(Q)$ defied by*

$$v_X = \sum_{W:\Sigma(W)=X} P(W)$$

Theorem 1. *Let $Q = P(q_1) \geq p_1, \ldots, P(q_n) \geq p_n$, $LP(Q)$ be as defined in Fig. 6, and $D = (\mathcal{W}, P)$. Then, the following two statements are equivalent:*
(1) $D \models Q$
(2) $\boldsymbol{v}(D)$, as defined in Definition 6, is a feasible solution to $LP(Q)$.

The theorem follows from the following lemma:

Input: A query $Q = P(q_1) \geq p_1, \ldots P(q_n) \geq p_n$
Returns: A Linear Program $LP(Q)$

---

1. For every $X \subseteq \{1, \ldots, n\}$, define a conjunctive query:

$$q_X \stackrel{def}{=} \bigwedge_{i \in X} q_i$$

2. For $X \subseteq \{1, \ldots, n\}$, define its closure $X^* = \{i | q_X \subseteq q_i\}$. Note that $q_X \equiv q_{X^*}$. Let $\mathfrak{C}$ be the set of closed sets.
3. For each $X \in \mathfrak{C}$, we create a variable v_X.
4. We then add the following linear constraints over v_X.
   - For every $X \in \mathfrak{C}$, add a constraint $v_X \geq 0$.
   - Add the constraint $\sum_{X \in \mathfrak{C}} v_X = 1$.
   - For every q_i in Q, add $\sum_{X \in \mathfrak{C}: X \ni i} v_X \geq p_i$

Fig. 6: The algorithm to construct $LP(Q)$

Lemma 2. *Let $D = (\mathcal{W}, P)$ be a probabilistic database and let $\boldsymbol{v}(D)$ be its associated solution then, for all i*

$$\sum_{X: X \ni i} v_X = \sum_{W: W \models q_i} P(W)$$

Proof. It suffices to observe that $W \models q_i$ if and only if $\Sigma(W) \ni i$.

The theorem follows since each database D such that $D \models Q$ is a feasible solution. On the other hand, given a feasible solution $\boldsymbol{v}$, we take $\mathcal{W} = \{I_X \mid q_X\}$ where I_X is any canonical instance for the conjunctive query q_X and $P(I_X) = v_X$.
An example of how to construct $LP(Q)$ is given in the Appendix.

3.2 Queries with 2nd-Tier Quantifiers

In the prior section, we handled only the case where Q did not have any existential quantifiers at the Tier-2 level. We address this limitation here and show how to construct $SLP(Q)$ whenever Q is of the form:

$$Q = \exists y_1, \exists y_2, \ldots, \exists y_k . P(q_1) \geq p_1, \ldots P(q_n) \geq p_n$$

We define some sets:

$$C = \{c | c \text{ is a constant that appears in one of the } q_i\}$$
$$C' = \{c_1 \ldots c_k | c_i \text{ is a fresh constant }\}$$
$$CONST = C \cup C'$$

Then for each mapping, $\theta : \{y_1, \ldots, y_k\} \rightarrow CONST$, θ provides a substitution for the y's in Q to produce $\hat{Q}_\theta$:

$$\hat{Q}_\theta = P(q_1[\theta(\overline{y})/\overline{y}]) \geq p_1, \ldots P(q_n[\theta(\overline{y})/\overline{y}]) \geq p_n$$

Define $\mathcal{L}_\theta$ as the the linear program that corresponds to $\hat{Q}_\theta$.

Theorem 2. *Let $D = (\mathcal{W}, P)$, then the following statements are equivalent:*
- $D \models Q$
- $\exists\theta$ *such that* $\boldsymbol{v}(D)$ *is a feasible solution for* $\mathcal{L}_\theta$

Proof (Theorem 2). The proof of this theorem follows directly from the proof of Theorem 1, and from the semantics of second tier queries given in Section 2.2.

All possible mappings and their corresponding LPs are added to $SLP(Q)$. An example of how to construct $SLP(Q)$ is given in the Appendix.

4 Query Containment

In this section, we show that Query Containment for conjunctive, tier-2 queries is decidable. There are two tier-2 queries as input:

$$Q = \exists y_1, \ldots, y_k.P(q_1) \geq p_1, \ldots P(q_n) \geq p_n$$
$$Q' = \exists y_1, \ldots, y_l.P(q'_1) \geq p'_1, \ldots P(q'_m) \geq p'_m$$

4.1 Preliminaries

The key technical hurdle is illustrated by trying to decide containment for the following pair of queries.

Example 5. Let $Q = P(\exists x.R(x)) \geq 1$ and $Q' = \exists y.P(R(y)) \geq N^{-1}$ for some N. Intuitively, Q states that the relation R is not empty, while the query Q' says that some particular value is present with probability at least N^{-1}. Example 4 is very similar to this one, but less general. There is no containment relationship between these two queries, but providing a counterexample is surprisingly subtle. In particular, we want to a provide a counterexample database D such that $D \models Q$, but $D \not\models Q'$. One such counterexample D is a database with $N+1$ worlds $I_i = \{R(c_i)\}$ for $i = 1, \ldots, N+1$ and $P(I_i) = (N+1)^{-1}$. This is somewhat jarring because we seem to need to make $N+1$ copies of the same database. A little thought shows that this construction is necessary, *any counter example must have at least* $N+1$ *worlds*. This is so that we can drive down the importance of any particular evaluation of the constants.

The proof of the main containment theorem expands on this idea. To do so, we need to be precise by what we mean about copying. Thus, we introduce a little bit of notation. A *canonical world* for a conjunctive query $q = g_1, \ldots, g_n$ is a world I such that there exists a bijective homomorphism $h : \mathbf{var}(q) \to \mathbb{D}$ such that $I = \bigcup_{i=1,\ldots,m} h(g_i)$. For example, $R(a,b)$ and $R(b,c)$ are both canonical worlds for $q = \exists x.R(x,y)$, but $R(a,a)$ is not.
Given a query $q'(\boldsymbol{y})$ where y denotes the head variables of q' (thus q' is not necessarily a boolean query), and a query q, let $\mathrm{Cert}(q', q) = \{\boldsymbol{t} \mid q \subseteq q'(\boldsymbol{t})\}$, i.e., those values of t that are certain answers of q' on the canonical world for q. Denote $\mathrm{Cert}(Q', q) = \bigcup_{q' \in Q'} \mathrm{Cert}(q', q)$.

Lemma 3. *Given a Boolean conjunctive query, q_i, with at least a single variable, there is an infinite set of canonical worlds for q_i, $\{I_1, \ldots, I_n, \ldots\}$ such that for* any *conjunctive query $q'_j(\boldsymbol{y})$ and any $\boldsymbol{t}$ of the same arity as the head of q'_j exactly one of the following two conditions holds:*
(1) $\boldsymbol{t} \in \mathrm{Cert}(q'_j, q_k)$ and so $I_i \models q'_j(\boldsymbol{t})$ for each i, or otherwise
(2) there is at most one I_i such that $I_i \models q'_j(\boldsymbol{t})$.

Proof. To construct the desired set of worlds, simply map each variable in q_i to a distinct, fresh variable in each world. This set of worlds has the desired property.

We call the set I_i for $i = 1, 2, \ldots$ the set of copies for q_i. Notice that its construction does *not* depend on q'_j.

4.2 Main Result

Consider the case when there are no Tier-2 quantifiers in Q (y variables). We will remove this restriction later in the section.

The Linear Program We construct a set of canonical linear programs $CLP(Q)$: there is one program for each $q'_j \in Q'$. Consider some fixed $q'_j \in Q'$ with probability p'_j. The program for q'_j has the same variables as $LP(Q)$, and all constraints of $LP(Q)$. In addition, we add a constraint for each $t_0 \in \mathrm{Cert}(Q', q)$ of the following form:

$$\sum_{X \in \mathfrak{C} : q_X \subseteq q'_j(t_0)} v_X < p'_j$$

An example of how to construct $CLP(Q)$ is given in the Appendix.
This program captures containment in the following sense:

Theorem 3. *There is a feasible solution to some program in $CLP(Q, Q')$ if and only if Q is not contained in Q'.*

Proof. Suppose there is a feasible solution, we show how to construct a probabilistic database $D = (\mathcal{W}, P)$ that is a counterexample to containment. Note that what is needed here is that such a counterexample exists; it is constructed only for the sake of the proof.
First, given a feasible solution $\boldsymbol{v}$, we define $\varepsilon(\boldsymbol{v})$ to be the smallest slack in the linear program. More precisely,

$$\varepsilon(\boldsymbol{v}) = \min_{t_0 \in \mathrm{Cert}(Q', q)} p'_j - \left(\sum_{X \in \mathfrak{C} : q_X \subseteq q'_j(t_0)} v_X \right)$$

There are finitely many constraints, and the slack is non-zero in each constraint, so we have that $\varepsilon(\boldsymbol{v}) > 0$. Choose N such that $N^{-1} 2^n = \varepsilon/2$ where n is the number of q_i queries in Q.
For each q_X, we generate N canonical worlds for q_X that we denote with $\mathcal{W}_X = \{I_{(X,1)}, \ldots, I_{(X,N)}\}$. Here, we choose any N worlds from the copy

worlds defined above. We let $\mathcal{W} = \bigcup_X \mathcal{W}_X$. We set the probability function P as $P(I_{X,i}) = v_X N^{-1}$, that is the mass for v_X is evenly distributed among all its copies.
This must be a counterexample because on the database D, $\exists y.P(q'_j(y)) < p'_j$ holds. Indeed, for a fixed t_0 we can write:

$$\begin{aligned} P(q'_j(t_0)) &\leq \sum_{X \in \mathfrak{C}: q_X \subseteq q'_j(t_0)} v_X + \sum_{Y \in \mathfrak{C}: q_Y \not\subseteq q'_j(t_0)} N^{-1} \\ &< p'_j - \varepsilon + 2^N 2^{-N} \varepsilon/2 < p'_j \end{aligned}$$

The first line follows by construction: each copy I corresponding to an X in the first sum is such that $I \models q'_j(t_0)$. Thus, for any choice of t_0 it must be that $q'_j(t_0)$ is satisfied in at most one world, and hence with probability $\leq N^{-1}$. This follows as a result of Lemma 3 and the fact that there are at most 2^N such sets, Y. Then, by our selection of N and the definition of $\varepsilon(\boldsymbol{n})$, we have the second line. Since our choice of t_0 was arbitrary, this holds for all t_0 simultaneously. Thus, we have that $q'_j(t_0) < p'_j$ for any t_0. Finally, we note that $\boldsymbol{v}$ is a feasible solution to $LP(Q)$ as well, and hence by Thm. 1, $D \models Q$. Thus, we have produced a counterexample to containment.
Now, we prove the other direction. Assume that we have any counterexample database D. Construct $\boldsymbol{v}(D)$, which is an assignment of probabilities to worlds in D. Observe:

$$P(q'_j(t_0)) \geq \sum_{X \in \mathfrak{C}: q_X \subseteq q'_j(t_0)} v_X$$

Thus, for each t_0 we have $P(q'_j(t_0)) < p'_j$. Since $D \models Q$, this solution also satisfies $LP(Q)$. Hence, a counterexample implies a feasible solution.

Q Contains Quantifiers We guess a mapping θ from (the Tier-2 quantifiers in Q) to constants so that $\theta : \{y_1, \ldots, y_k\} \rightarrow CONST$ and then apply the previous theorem. This gives a simple NEXPSPACE algorithm and then by Savitch's theorem:

Corollary 1. *We can decide containment in* EXPSPACE

5 Related Work

As we mentioned in the introduction, there is a large body of work in probabilistic databases [15,19] that deals with how to represent and store probabilistic data. The first generation of probabilistic databases mainly varied in their ability to represent distributions succinctly including tuple independent databases [4], x-tables [16], and more succinct representations such as factor graphs [17].
Recently, there has been a renewed interest in query languages for probabilistic data from the MayBMS group [1,9]; their query language is fully compositional. As a result, their language can be viewed as allowing an

arbitrary number of tiers as opposed to only two. They do not study the containment problem.

There has also been renewed interest in containment. Notably the breakthrough results of Green [10], who showed containment results for queries over relations annotated with elements from a semiring. The language study in his work is a classical language similar to a first-generation probabilistic database: it does not manipulate the annotations within the query language. It is an interesting extension of our work to study containment for a rich query language that manipulates (more general) semiring annotations.

Fagin et. al. [7] study a language for reasoning about probability that is similar to the one we present here. They also demonstrate that systems of linear inequalities can be used to capture the set of probability spaces that will satisfy a particular statement in their language. Additionally, their result states that when the queries are restricted to propositional logic the construction is in NP. However, this work does not consider query containment.

6 Conclusions and Future Work

We presented a decidable characterization of containment for a rich query language over probabilistic databases, which is the first such algorithm to address query containment of a probabilistic language that allows direct manipulation of uncertainty. The future work for this project is in three directions: (1) more expressive annotations and query languages, such as arbitrary levels of nesting, (2) we plan to study query containment over restricted, but practically important, classes of probabilistic databases, such as tuple independent probabilistic databases, and (3) we plan to find tractable subclasses of queries for the containment problem.

References

1. Lyublena Antova, Christoph Koch, and Dan Olteanu. 10^{10^6} worlds and beyond: Efficient representation and processing of incomplete information. In *ICDE*, pages 606–615, 2007.
2. Lyublena Antova, Christoph Koch, and Dan Olteanu. Maybms: Managing incomplete information with probabilistic world-set decompositions. In *ICDE*, pages 1479–1480, 2007.
3. Omar Benjelloun, Anish Das Sarma, Alon Halevy, Martin Theobald, and Jennifer Widom. Databases with uncertainty and lineage. *The VLDB Journal*, 17(2):243–264, 2008.
4. Nilesh N. Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875, 2004.
5. Yanlei Diao, Boduo Li, Anna Liu, Liping Peng, Charles Sutton, Thanh Tran 0002, and Michael Zink. Capturing data uncertainty in high-volume stream processing. In *CIDR*, 2009.
6. Xin Luna Dong, Alon Y. Halevy, and Cong Yu. Data integration with uncertainty. In *VLDB*, pages 687–698, 2007.

7. Ronald Fagin, Joseph Y. Halpern, and Nimrod Megiddo. A logic for reasoning about probabilities. *Inf. Comput.*, 87(1/2):78–128, 1990.
8. Avigdor Gal, Maria Vanina Martinez, Gerardo I. Simari, and V. S. Subrahmanian. Aggregate query answering under uncertain schema mappings. In *ICDE*, pages 940–951, 2009.
9. Michaela Götz and Christoph Koch. A compositional framework for complex queries over uncertain data. In *ICDT*, pages 149–161, 2009.
10. Todd J. Green. Containment of conjunctive queries on annotated relations. In *ICDT*, pages 296–309, 2009.
11. Christoph Koch. On query algebras for probabilistic databases. *SIGMOD Record*, 37(4):78–85, 2008.
12. Imran R. Mansuri and Sunita Sarawagi. Integrating unstructured data into relational databases. In *ICDE*, page 29, 2006.
13. Christopher Re, Nilesh N. Dalvi, and Dan Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, pages 886–895, 2007.
14. Christopher Ré, Julie Letchner, Magdalena Balazinska, and Dan Suciu. Event queries on correlated probabilistic streams. In *SIGMOD Conference*, pages 715–728, 2008.
15. Christopher Re and Dan Suciu. Materialized views in probabilistic databases for information exchange and query optimization. In *VLDB*, pages 51–62, 2007.
16. Anish Das Sarma, Martin Theobald, and Jennifer Widom. Exploiting lineage for confidence computation in uncertain and probabilistic databases. In *ICDE*, pages 1023–1032, 2008.
17. Prithviraj Sen and Amol Deshpande. Representing and querying correlated tuples in probabilistic databases. In *ICDE*, pages 596–605, 2007.
18. University of Washington. RFID Ecosystem. http://rfid.cs.washington.edu/.
19. Jennifer Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, pages 262–276, 2005.
20. Yahoo! Research. The Purple Sox System. http://research.yahoo.com/node/498.

A Appendix: Examples

To provide the reader with more intuition regarding how to construct $LP(Q)$, $SLP(Q)$, and $CLP(Q)$, we present a complete example here.

A.1 Constructing $LP(Q)$

Let

$$Q = P(R('a')) \geq .2, P(R('b')) \geq .8$$

Because Q has only constants we can construct a single linear program, $LP(Q)$. First, we observe that

$$q_1 = R('a') \text{ and } q_2 = R('b')$$

In this case there are no shared tuples between q_1 and q_2, so the linear program we construct will have four variables, $v_{\{\}}, v_{\{1\}}, v_{\{2\}}$, and $v_{\{1,2\}}$. Where $v_{\{\}}$ represents the probability mass assigned to the canonical world for an "empty query", $v_{\{1\}}$ represents the probability mass assigned to the canonical world for q_1, $v_{\{2\}}$ represents the probability mass assigned to the canonical world for q_2, and $v_{\{1,2\}}$ represents the probability mass assigned to the canonical world for $q_1 \wedge q_2$.
From this, we get the following LP:

Non-Negativity Constraints	$v_{\{\}}, v_{\{1\}}, v_{\{2\}}, v_{\{1,2\}} \geq 0$
Validity Constraint	$v_{\{\}} + v_{\{1\}} + v_{\{2\}} + v_{\{1,2\}} = 1$
Constraint for q_1	$v_{\{1\}} + v_{\{1,2\}} \geq .2$
Constraint for q_2	$v_{\{2\}} + v_{\{1,2\}} \geq .8$

A.2 Constructing $SLP(Q)$

Now, let

$$Q = \exists y_1, y_2 . P(R(y_1)) \geq .2, P(R(y_2)) \geq .8$$

In this case, there are two mappings we need to consider. The first mapping (θ_1) gives different constants for y_1 and y_2, whereas the second mapping (θ_2) yields the same constant for y_1 and y_2.

θ_1 mapping Here, we assume that $\theta_1(y_1) \neq \theta_1(y_2)$. Since the actual values of the constants can be chosen arbitrarily, assume that

$$\theta_1(y_1) =' a'$$

$$\theta_1(y_2) =' b'$$

In this case,

$$\hat{Q}_{\theta_1} = P(R('a')) \geq .2, P(R('b')) \geq .8$$

As with the prior example, we have four "canonical worlds", and thus $\mathcal{L}_{\theta_1}$ is:

Non-Negativity Constraints	$v_{\{\}}, v_{\{1\}}, v_{\{2\}}, v_{\{1,2\}} \geq 0$
Validity Constraint	$v_{\{\}} + v_{\{1\}} + v_{\{2\}} + v_{\{1,2\}} = 1$
Constraint for q_1	$v_{\{1\}} + v_{\{1,2\}} \geq .2$
Constraint for q_2	$v_{\{2\}} + v_{\{1,2\}} \geq .8$

θ_2 mapping This time, we assume that $\theta_2(y_1) = \theta_2(y_2)$. Since the actual values of the constants can be chosen arbitrarily, we simply choose

$$\theta_2(y_1) = \theta_2(y_2) =' a'$$

This mapping provides a different query:

$$\hat{Q}_{\theta_2} = P(R('a')) \geq .2, P(R('a')) \geq .8$$

Here, $n = 2$ and of the four subsets of $\{1, 2\}$ only two are closed: $\{\}$ and $\{1, 2\}$. Thus, this query yields only 2 canonical worlds (the one that contains $R('a')$ and the empty world), and only two variables are

needed for $\mathcal{L}_{\theta_2}$. We construct a new linear program:

Non-Negativity Constraint	$v_{\{\}}, v_{\{1,2\}} \geq 0$
Validity Constraint	$v_{\{\}} + v_{\{1,2\}} = 1$
Constraint for q_1	$v_{\{1,2\}} \geq .2$
Constraint for q_2	$v_{\{1,2\}} \geq .8$

$SLP(Q) = \{\mathcal{L}_{\theta_1}, \mathcal{L}_{\theta_2}\}$.

A.3 Constructing $CLP(Q)$

For this step, we need two queries, Q and Q'. Let

$$Q = P(R('a')) \geq .2, P(R('b')) \geq .8$$

and

$$Q' = \exists y_1, y_2.P(R(y_1), R(y_2)) \geq .3, P(R('b')) \geq .4$$

We also assume that we are given $LP(Q)$ (which was constructed in an earlier section of this example). In order to check containment, we want to know if there are any databases that will satisfy Q but not Q'. Since Q' is a conjunct of two sub-queries, we only need a solution that finds a counterexample for one of our two sub-queries.

Sub-query $q_1 = P(R(y_1), R(y_2)) \geq .3$ If we evaluate the arity-2 query $R(y_1), R(y_2)$ on the canonical worlds from the $LP(Q)$, we get only one world, the one that contains both $R('a')$ and $R('b')$. The linear program for Q uses variable $v_{\{1,2\}}$ to represent the probability mass assigned to this world, so we add a new constraint to $LP(Q)$ to get an LP for containment. The new program is:

$LP(Q)$	$\vdots$
$\neg q_1$ Constraint	$v_{\{1,2\}} < .3$

This program is added to $CLP(Q)$.

Sub-query $q_2 = P(R('b')) \geq .4$ If we evaluate the arity-1 query $R(y_1)$ on the canonical worlds from the $LP(Q)$, we get two worlds, the one that satisfies $R('b')$ only and the one that contains both $R('a')$ and $R('b')$. The linear program for Q uses variables $v_{\{2\}}$ and $v_{\{1,2\}}$ to represent the probability mass assigned to these worlds (respectively). We add a new constraint to $LP(Q)$ to get an LP for containment. This new program is:

$LP(Q)$	$\vdots$
$\neg q_2$ Constraint	$v_{\{2\}} + v_{\{1,2\}} < .4$

This program is also added to $CLP(Q)$, which (now complete) contains two linear programs, one for each sub-query of Q'.
The first LP has a feasible solution, and thus provides a counter-example to containment, whereas the second LP has no feasible solution.

Embracing Uncertainty in Large-Scale Computational Astrophysics

Dan Suciu[1], Andrew Connolly[2], and Bill Howe[1]

[1] Department of Computer Science and Engineering
{suicu,billhowe}@cs.washington.edu
[2] Department of Astronomy
ajc@astro.washington.edu
University of Washington

Abstract. A revolution is underway in astronomy resulting from massive astrophysical surveys providing a panchromatic view of the night sky. The next generation of surveys and the simulations used to calibrate them can produce in two nights what the previous generation produced over many years. This enormous image acquisition capability allows the telescope to revisit areas of the sky with sufficient frequency to expose dynamic features and transient events; e.g., asteroids whose trajectories may intersect Earth. At least three such surveys are planned; their collective output must be integrated and calibrated against computational simulations, prior surveys, and each other.

Relational databases have been shown to be effective for astronomy at yesterday's scale, but new access to the temporal dimension and increased intercomparison of multiple sources generate new sources of uncertainty that must be modeled explicitly in the database. Conventional relational database management systems are not cognizant of this uncertainty, requiring random variables to be prematurely and artificially collapsed prior to manpiulation. Previous results in probabilistic databases focus on discrete attribute values and are unproven at large scale.

In this paper, we present concrete examples of probabilistic query processing from computational astrophysics, and use them to motivate new directions of research: continuous-valued attributes and queries involving complex aggregates over such attributes.

1 Introduction

The last decade has witnessed a revolution in how we approach knowledge discovery in an astrophysical environment. The completion of several massive astrophysical surveys provides a panchromatic view of the night sky; spanning the γ and X-ray spectrum (Fermi and Chandra satellites) through the optical and ultraviolet (the SDSS, GALEX surveys) to the measurements of the cosmic microwave background in the submillimeter and radio (the WMAP and PLANCK satellites). In conjunction with this, simulations of the Universe are becoming larger and more complex—a single simulation today can use as many as a billion resolution elements (Figure 1). While each of these massive data sources, both observational and simulated, provide insights into the highest energy events in our universe as well as the nature of the dark matter and dark energy that drives our accelerating universe, it is only when they are combined, by collating data from several different surveys or matching simulations to observations, that their full scientific potential will finally be realized. The scientific returns from the total will far exceed those from any one individual component.

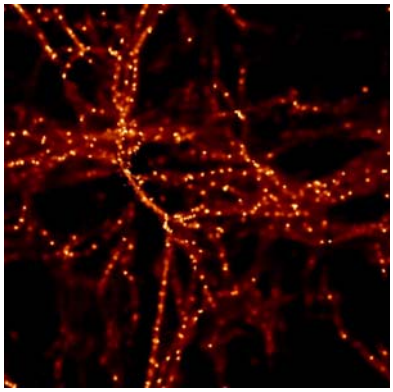


Fig. 1. The neutral hydrogen content of a simulated region of the Universe 25 million light-years wide as it existed 2 billion years after the Big Bang. The areas of highest density (yellow) are sites of galaxy formation.

The recognition of this need to federate massive astrophysical databases has led to initiatives designed to seamlessly interlace data distributed across the globe. Users will soon be able to return multi-frequency attributes of sources identified within regions of the sky without needing to worry about how different surveys or simulations interact or what are the underlying protocols for communicating between them. The virtual observatory (VO) is indicative of the growing awareness in the US and worldwide that efficient organization, distribution, and analysis of scientific data is essential to the continuing evolution of scientific inquiry.

As virtual observatories come online, the rate of data acquisition threatens to overwhelm our ability to access, analyze, and visualize these massive datasets. Further, observations provide only an imperfect, indirect representation of the observed phenomenon—we must build systems that not only tolerate uncertainty, but *embrace* uncertainty by providing probabilistic reasoning capabilities at the system level. Research in database and data management has developed techniques for scalable processing of large data sets, but do not attempt to capture uncertainty. All three modern commercial database systems (IBM's DB2, Oracle, and Microsoft's SQL Server) can optimize and execute SQL queries on parallel database systems, based on research that was done more than a decade ago [5, 14]. New trends in computing, such as the map-reduce programming model introduced by Google [12] and extensions [17, 21, 28] to process massive datasets, have been adopted and expanded as query processing paradigms in systems such as Pig Latin [28], Sawzall [29], Dryad [21], and SCOPE [7]. We are aggressively evaluating these frameworks for general utility in scientific data analysis, but these results are complementary to the development of a theory of scalable query processing over probabilistic science databases, for two reasons. First, the theoretical algorithmic complexity of query answering over probabilistic databases is an independent research question from the design of an effective parallel implementation, map-reduce or otherwise. Second, map-reduce and similar frameworks are increasingly supporting complete relational algebra expressions rather than just simple primitives citeolston:08,isard:07,abouzeid:09, so work on relational modeling and optimization does not preclude a map-reduce implementation.

In principle, relational databases offer the scalability and performance needed to process the data from astrophysical surveys; indeed, the Sloan Digital Sky Sur-

vey (SDSS; http://www.sdss.org), the largest current astronomical database, is powered by a relational database and supports SQL queries. The challenge posed by the new generation of cosmology surveys, however, stems from a significantly larger scale compounded by higher dimensionality (measurements now have a temporal extent) and new sources of uncertainty. The growth in size can be understood if we consider the volume of data generated by the previous generation survey of the SDSS. That ten year experiment surveyed 8,000 sq degrees of the sky and detected $\sim 10^8$ stars and galaxies, forming a 40 TB data set. In contrast, the next decade will see the Dark Energy Survey (http://www.darkenergysurvey.org), PanSTARRS (http://pan-starrs.ifa.hawaii.edu/public/home.html) and the Large Synoptic Survey Telescope (LSST; http://www.lsst.org) that will produce **nightly** data rates of $\sim$0.5 TB, 5 TB and 20 TB respectively. The first of these surveys, DES will begin operations in 2011 and cover 5,000 sq degrees over a five year period, PanSTARRS and LSST will cover 20,000 sq degrees **every three nights** and are expected to begin operation in 2014. Beyond their sheer scale, all of these surveys will open the temporal domain through repeated observations of the sky many times over the length of the surveys (up to a thousand times in the case of the LSST). This will enable the extraction of temporal patterns for 10^8 sources with tremendous scientific potential, ranging from detection of moving objects, classification of astrophysical sources, and monitoring of anomalous behaviour. Individual images can no longer be analyzed independently—objects too dim to be recognized in any single image are inferred probabilistically by studying multiple images at different timesteps. However, this inference requires one to reason about and manage multiple possible probabilistic interpretations of the data simulatneously—a capability missing in the formalisms underpinning existing data management software.

In order to extract interesting temporal patterns from the data one needs to characterize the nature of sources from data sets with inherently different noise properties—data may be missing due to incomplete observations and individual sources may drop below the detection threshold of the image. Therefore, we are working to extend and apply recently developed techniques for *probabilistic databases* in order to extract patterns from temporal astrophysical surveys. We model the probabilistic inference associated with the pattern extraction task as a SQL query, then apply techniques such as *safe plans* [10] to execute them in a relational database engine, which will enable the engine to (among other things) evaluate the query in parallel on a cluster.

2 Background and Running Example

A probabilistic database is a relational database where the rows in the database are random variables. In the simplest case, only the value of an attribute is a random variable. Consider the two probabilistic tables in Figure 2. The first table, `Objects` stores the type of each object. The attribute `Type` is a discrete random variable, and its distribution is given explicitly for each object id: thus, for object id `x2234`, `Type` is `Quasar`, `Main Sequence Star`, or `White Dwarf`, with probabilities 0.1, 0.6, and 0.3 respectively, and this is represented by storing three distinct rows in the table, all with the same object id and with the three different values together with their probabilities. The second table in the figure, `Properties`, stores the properties measured periodically (e.g. daily): thus, for each object there are several rows in `Properties`. All these measurements are noisy, and are normally given by continuous random variables (most of them are

`Objects:`

	OID	Type	P
$t_{1,1}$	x2234	Quasar	$p_{1,1} = 0.1$
$t_{1,2}$	x2234	Main Sequence Star	$p_{1,2} = 0.6$
$t_{1,3}$	x2234	White Dwarf	$p_{1,3} = 0.3$
$t_{2,1}$	x5542	Quasar	$p_{2,1} = 1.0$
$t_{3,1}$	xg413	Main Sequence Star	$p_{3,1} = 0.7$
$t_{3,2}$	xg413	Quasar	$p_{3,2} = 0.3$
$t_{4,1}$	y5553	White Dwarf	$p_{4,1} = 0.1$
$t_{4,2}$	y5553	Binary Star	$p_{4,2} = 0.9$

(a)

`Properties:`

	OID	Brightness	Color	P
s_1	x2234	19.7	0.31	$q_1 = 0.2$
s_2	x2234	19.7	0.12	$q_2 = 0.8$
s_3	xg413	21.2	0.32	$q_3 = 0.7$
s_4	xg413	19.7	0.24	$q_4 = 0.7$
s_5	x5542	21.2	0.13	$q_5 = 0.5$

(b)

Fig. 2. Example of a probabilistic database. This is a *block-independent-disjoint* database: the 8 tuples (rows) in `Objects` are grouped in four groups. The random variables corresponding to tuples in a group are disjoint, e.g., t_1^1, t_1^2, t_1^3 are disjoint, meaning that at most one can be true; so are t_4^1, t_4^2. Tuples from different blocks are independent, e.g., t_1^2, t_2^2, t_4^1 are independent; the five tuples in `Properties` are independent probabilistic events.

Normal distributions). In the figure, we have represented each row as a discrete event, whose probability indicates a confidence that the row is in the database. However, the natural representation of this data is to use a continuous probability distribution.

Query evaluation on probabilistic databases involves probabilistic inference. Consider, for example, the SQL query in Figure 4 (a), asking for all object types that had a measured brightness < 20. The query joins `Objects` and `Properties` on the object id, and returns the type of the object, provided the brightness is < 20. A probabilistic database needs to examine the *lineage* of each answer, and compute a confidence score for that answer. For example, consider the object type `Quasar`. It is in the answer because of contributions from the first and the third object, and because of three rows in `Properties`, thus, its probability is:

$$p(\texttt{Quasar}) = 1 - (1 - p_{1,1}(1 - (1 - q_1)(1 - q_2)))(1 - p_{3,2}q_4) \qquad \text{(safe result)}$$

The algebra plan in Figure 4(c) computes very efficiently the probabilities of all object types, by incorporating the operations of the formula above into standard relational algebra operations: a join computes the probability $p_1 p_2$ while a projection with duplicate elimination computes the probability $1 - (1 - p_1)(1 - p_2)(1 - p_3) \cdots$ It is possible to modify a relational database engine to compute these probabilities on-the-fly. Alternatively, it is possible to translate back this query plan into SQL (as shown in Figure 4 (d)) and have it evaluated in any standard relational database engine, without having to modify the engine: given the current performance of todays commercial database engines, such a query can be evaluated in a few seconds on a database of hundreds of GB. In our own implementation of a probabilistic database system `mystiq.cs.washington.edu` we took the latter approach, where we translated the relational plan back to SQL.

It is important to note that not every relational algebra plan computes the correct output probabilities. The algebra plan in Figure 4 (b) is equivalent (over standard databases) to that in (c), yet it computes the probabilities incorrectly. In our example it returns the following:

$$p(\texttt{Quasar}) = 1 - (1 - p_{1,1}q_1)(1 - p_{1,1}q_2)(1 - p_{3,2}q_4) \qquad \text{(unsafe result)}$$

The difference is that plan (b) first computes a join, thus making a copy of $p_{1,1}$, and later projects and eliminates duplicates, thus treating the two copies of $p_{1,1}$ as two independent probabilistic events, which is incorrect.

`Observations`

id	T	X	Y	sigmaX	sigmaY	sigmaXY
a1234	10	2.34	0.46	0.2	0.1	0.3
a1235	10	0.33	3.03	0.1	0.3	0.1
...	...	...				
a5466	11	2.36	0.44	0.2	0.2	0.2
a5467	11	0.33	3.03	0.1	0.3	0.1
...	...	...				

For each observation, the uncertain location of each object (`id`, `T`, `X`, `Y`, `sigmaX`, `sigmaY`, `sigmaXY`) is given by the two-dimensional Normal distribution $N(\mu, \Sigma)$, where:

$$\mu = \begin{pmatrix} \texttt{X} \\ \texttt{Y} \end{pmatrix} \quad \Sigma = \begin{pmatrix} \texttt{sigmaX} & \texttt{sigmaXY} \\ \texttt{sigmaXY} & \texttt{sigmaY} \end{pmatrix}$$

Fig. 3. The `Observations` table stores individual observations at each time stamp.

The complexity of query processing in probabilistic databases has been intensively studied [9, 10, 15, 22, 36]. It was proven that, in general, computing the exact output probabilities is a #P-hard problem in the size of the input database [9], due to the interaction of joins and duplicate eliminations. What this means in practical terms is that it is not possible to compute exactly the output probabilities for every SQL query. However, over databases with discrete random variables, certain queries *can* be computed efficiently, and their computation can be expressed as a relational algebra plan, which manipulates the probabilities explicitly: this is illustrated in Figure 4. Such queries are called *safe queries*. Interestingly, not every relational algebra plan computes the output probabilities correctly: plans that compute the probabilities correctly are called *safe plans*. The plan in Figure 4 (b) is unsafe, while the plan in (c) is safe. A survey of the state of the art of the theory of safe queries and safe plans can be found in [10].

3 Two Concrete Problems

We consider two problems in astrophysics to motivate requirements for probabilistic databases. the identification of moving objects, and probabilistic classification.

3.1 Tracking Moving Objects

As with the variable luminosities it is the dynamic range of motions of sources across the sky coupled with the confusion due to the many sources that are moving in our own Solar System that drives the complexity of the problem.

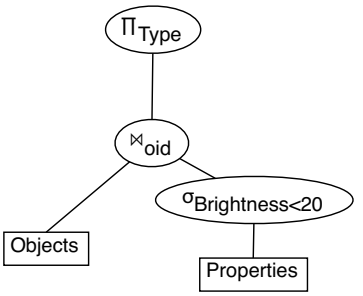
Within the Solar System there are approximately 10^7 sources that move relative to the Earth. The majority of these sources are the Main Belt Asteroids that reside between the orbit of Mars and Jupiter. These can be considered as the background noise that limits our ability to identify the more scientifically

```
SELECT x.Type, confidence( )
FROM Objects x, Properties y
WHERE x.OID = y.OID
 and  y.Brightness < 20
GROUP BY x.Type
```

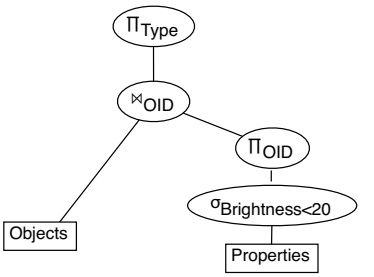
(a)

```
SELECT x.Type, 1-prod(1-x.P*y.P)
FROM Objects x, (SELECT OID, 1-(1-prod(P))
                 FROM Properties
                 WHERE Brightness < 20
                 GROUP BY Type) y
WHERE x.OID = y.OID
GROUP BY x.Type
```

(d)



(b)



(c)

Fig. 4. A SQL query on the data in Figure 2(a) returning the types of all objects with a brightness below 20. Here `confidence()` is an aggregate operator returning the output probability. The figure shows an unsafe plan in (b) and a safe plan in (c). The safe plan re-written in SQL is shown in (d): the aggregate function `prod` is not supported by most relational engines, and needs to be rewritten in terms of `sum`, logarithms, and exponentiation.

compelling asteroids such as the Kuiper Belt Objects (KBOs) and the potentially hazardous Earth-crossing Near Earth Objects (NEOs).

NEOs with sizes in excess of 1km that strike the Earth have the potential to cause extinction level events through their impact and resulting climate change (similar to the events that may have led to the extinction of the dinosaurs). Such impacts are expected to occur every 500,000 years. On smaller scales (>140m) asteroid impacts would cause substantial damage (equivalent to 100 megatons of TNT) if impacting a populated area (with a 5% chance of impact over the next century). Because of this NASA has a high priority goal of mapping asteroids with sizes > 300m within the next 10 years to assess the potential risk to the Earth from impacts.

The challenge in finding these asteroids comes from the fact that we have multiple observations of the sky every three nights (i.e. we cannot continuously view one region of the sky as the asteroids are distributed over several thousand square degrees). For NEOs we will likely detect 50,000 sources against a background of 10^6 Main Belt Asteroids. Each of these sources moves with an average velocity of one degree per day. Sampling the orbits every three days and accounting for uncertainties in the positional and velocity measurements the combinatorics in connecting subsequent observations are daunting.

More distant asteroids, in particular those beyond the orbit of Neptune, have the potential to explain the origins of our Solar System. Kuiper Belt Objects are in orbits at distances of > 30 Astronomical Units (AU) and have a composition that is identical to the planetesimals that coalesced to form the planets.

Mapping their positions, distances, dynamics and colors (from which their composition can be determined) will constrain rate of accretion, collisions and orbital perturbations that led to the formation of the inner planets as well as providing statistical evidence for the possibility of additional existing and/or vanished planets in the outer Solar System.

There are currently ∼1000 known KBOs which compares to the expected 10-100,000 KBOs from surveys such as the LSST. Moving at 1 arcsecond per hour KBOs will move 2 degrees per year. Simple algorithms that search all of the available phase space for the orbital parameters would be prohibitive in computational time. The joint analysis of one year of data would increase the population of known KBOs by a factor of 50 and our sensitivity to asteroids a factor of 100 smaller in mass. This will enable studies of KBOs at distance in excess of 50 AU where we find a dramatic (and unexplained) drop in the asteroid population.

3.2 Working with Probabilistic Classifications

As described in Section 2, the next generation of astrophysics surveys will open the temporal domain probing a broad range of classes of sources, from the most energetic events in the universe to new classes of physics. Classifications will be derived based on repeated measurement of the same attributes for sources or by "coaddition" of the input images to provide a higher signal-to-noise measures. In each of these cases, the measurements and classifications will be inherently probabilistic. In the simplest case, these classifications will be uni-modal and can be approximated by a Gaussian (e.g. the likelihood of a source being a star or a galaxy). In more complex examples, such as an estimate of the redshift of a galaxy from its colors [8, 20, 26], the probability functions are non-Gaussian and often multimodal.

Understanding how the properties of galaxies depend on their class enables a better understanding of the physical processes that govern the evolution of the universe. Designing new ways of querying databases with probabilistic classifications and uncertain measurements is, therefore, a critical component of any future astrophysical survey. We provide two examples that will guide our development of probabilistic databases. In the initial example we will address the question of how galaxies are related to the dark matter halos in which they reside. Do the properties of galaxies depend simply on the mass of the dark matter halo or are galaxy properties influenced by larger scale structures? Locally we can address these questions using large spectroscopic surveys. We find, for example, that environment plays an important role in determining the properties of galaxies (e.g. their star formation and morphology; [19]. At higher redshifts, we do not have complete spectroscopic samples and, therefore, estimates of redshift and type must be undertaken probabilistically.

How do we use probabilistic classifications to determine and understand the relation between the properties of galaxies and their mass or environment? The classical approach is to subdivide a parent galaxy catalog into a series of subsamples (e.g. assuming categorical classifications based on luminosity or type of a galaxy) and to consider the clustering of these subsamples in isolation (e.g., [18, 27, 40]). This has been successful in characterizing simple relationships such as the morphology-density relation [30] but there are many reasons why this is not an optimal use of the data. The properties of galaxies (luminosity, morphology, star formation) are usually continuous in nature and how we choose to discretize a sample into subgroups is often arbitrary. Treating all galaxies within a subgroup as equal ignores the properties of the galaxies within

that group; we are discarding information. Finally, all of the classifications are inherently noisy so fixed classification thresholds will bias the derived relations.

To address these issues new statistics have been developed, marked correlation functions (MCFs), that do not require that we subdivide a sample of galaxies [37]. In their simplest form, the marked correlation functions, $M(r)$ are essentially, weighted correlation functions such that,

$$M(r) = \frac{\sum_i \sum_j w_i w_j \, \mathcal{I}(r_{ij} = r)}{\langle w \rangle^2 \sum_i \sum_j \mathcal{I}(r_{ij} = r)} = \frac{1 + W(r)}{1 + \xi(r)}, \tag{1}$$

where $\mathcal{I} = 1$ if the separation r_{ij} between galaxy i and galaxy j is r, and $\mathcal{I} = 0$ otherwise, so that the sum over pairs (i, j) includes only those pairs with separation $r_{ij} = r$. Here w_i is the weight or the mark (e.g. the luminosity or morphology) of galaxy i, $\langle w \rangle = \sum_i w_i / N_{gal}$ is the mean mark, and so $W(r)$ and $\xi(r)$ are the weighted and unweighted two-point correlation functions. Weighting galaxies by different marks yields datasets which are each biased differently relative to each other, and to the underlying dark matter distribution. In principle, we can determine which properties of galaxies result in a weighting of the galaxy distribution which minimizes the bias relative to the dark matter and over what redshift ranges this holds. In the context of a halo model that describes the mass and clustering of dark matter halos, marks provide a probe of the role of mass and environment in determining galaxy properties [38].

Extending these analyses to the clustering of the dark matter we can consider gravitational lensing signatures due to the growth of structure as a function of the age of the universe [3]. Foreground structures induce a lensing signal in background sources. By averaging the ellipticities of galaxies inside circular apertures, the coherent induced shear can be measured and can be used to estimate galaxy and cluster masses, the cosmological mass power spectrum, and higher order statistics. The size of the lensing distortions depends upon both the distances traveled, and upon the growth function which determines the amplitude of the deflecting mass concentrations. Weak lensing is an attractive cosmological probe because the physical effect, gravitational deflection of light, is simple and well understood. Furthermore, the deflecting masses are dominated by dark matter, the evolution of which is purely gravitational and hence calculable. Lensing is currently regarded as one of the most promising probes of the dark energy.

The uncertainties in this case comes from the use of colors to estimate the distances to the lens and background galaxies (i.e. photometric redshifts). As in all inversion problems: the data are both noisy and incomplete. A consequence of this is that photometric redshifts have broad error distributions as well as the presence of multiple minima. The scatter within the relation, its dependence on redshift and galaxy type, and the number of catastrophic outliers will all determine our ability to constrain the equation of state for dark energy. Given the prominent role that photometric redshifts play in current and planned cosmological surveys, it is both timely and necessary that we address how we account for these uncertainties when analyzing massive data sets and how can we, in the context of a database design, minimize the impact of the photometric redshift uncertainties to maximize our ability to constrain dark energy and dark matter [25].

4 Research Challenges

Our first aim is to store the temporal astrophysics data in a cluster of relational databases. Next, we will explore a theory of petascale relational query processing

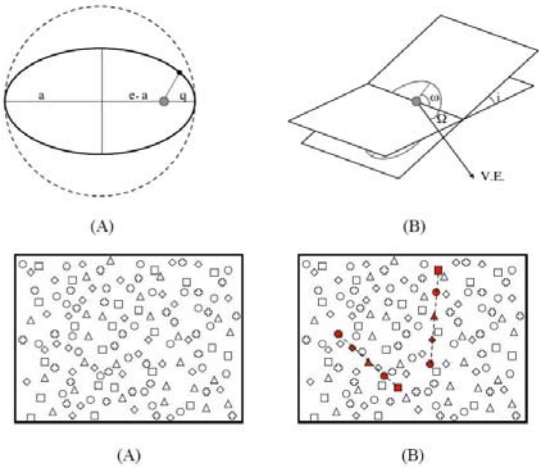


Fig. 5. To determine the orbital parameters of a moving source shown in the top panel requires six positional measures (i.e. three observations each with a Right Ascension and declination). To link three subsequent observations we must match moving sources over a period of three to 90 days. The lower panel shows a sequence of 4 sets of observations superimposed where the tracks of two moving objects have been highlighted in the right panel. the combinatorics associated with a naive linkage model that does a simple linear forward prediction results in a large number of false positives [24].

by addressing two specific challenges in computational, data-intensive astrophyics: Trajectory-Fitting Queries over Uncertain Paths and Scalable Analyses of Probabilistic Classifications.

In this section, we describe two challenges derived from the concrete scenarios described in Section 3.

4.1 Safe Queries over Continuous Attributes

The theory of safe queries was developed only for probabilistic databases represented by discrete random variables. In contrast, astrophysical data fundamentally requires continuous random variables expressed as probability density functions (pdf). Our second aim is to develop new, fundamental techniques for processing SQL queries over probabilistic databases that have both discrete and continuous random values. A naive solution is to simply replace the the continuous distribution with a discrete one by sampling. However, manipulating closed-form expressions is simpler, more accurate, and far more efficient than manipulating a set of possible worlds, and is therefore preferred when possible. In particular, we will identify and characterize the class of SQL queries that can be evaluated efficiently over probabilistic databases with continuous random variables. In the case of discrete attribute values, there exists a clear separation between *safe queries*, which can be computed efficiently using a safe plan and *unsafe queries*, which are provably #P-hard: we will study whether a similar dichotomy result holds for queries over continuous attribute values. A particularly hard challenge are fuzzy joins, where the join predicate is given by a user defined function that returns a confidence score. An index structure for fuzzy joins have been described for the special case when the function is the Jacquard similarity between two strings [2], with applications to data cleaning; we plan to explore extensions of that technique to user-defined similarity functions needed in the

continuous domain. Another challenge comes from the fact that continuous random variables may or may not be closed under certain algebraic operations. For example, the sum of two Normal distributions is always another Normal distribution, but the sum of a Normal distribution and a uniform distribution is not expressible by a standard pdf. In contrast, the sum of any two discrete numerical random variables is always a discrete random variable.

We approach this challenge by focusing on the first problem mentioned in Section 3, detecting moving objects. The challenge here is to transform a set of uncertain point measurements into a set of object trajectories by fitting curves subject to physical constraints (e.g. that each track must be able to be described by a set of orbital parameters). Each point is generally modeled as a two dimensional Gaussian distribution. Points at two different time stamps can either be the same fixed object, in which case their coordinates should be the same during all time stamps, or can be the same moving object, in which case their coordinates should evolve along a well defined trajectory, or are unrelated objects. By aggregating across many time stamps we expect to identify moving objects with high confidence. We describe this in some detail in the following example.

Example 1. Assuming we have 100 observations (over a period of, say, five months), each with 10^7 moving objects in `Observations(id, T, X, Y, ...)`. Starting with three time slices `T1,T2,T3`, at the beginning, the middle, and the end of the observation period, we will compute triples of observations `(id1,id2,id3)` that are close in space in the three time slices. Over this time period, the known distribution of orbital parameters constrains how far an object may move in a given time period, which will allow us to aggressively prune the set of candidate triples that need to be considered. For example, over an eight day period, an orbit can be approximated by a quadratic in x and y and asteroids are known to rarely move more than 1 degree per day. Moreover, given the endpoints `id1` and `id3`, the trajectory between these endpoints constrains the position in the middle, further reducing the number of candidates `(id1,id2,id3)`. In total, we expect to generate about 10^8 candidate triples, about 10 times more than the total number of objects observed.

Next, for each candidate triple, we will compute an approximate candidate trajectory, which is defined by six parameters $a, b, \ldots, f$ such that the quadratic trajectory is:

$$x = at^2 + bt + c$$
$$y = dt^2 + et + f$$

Furthermore, the errors in the coordinates translate into errors of the parameters, leading to six more parameters. All this information can be computed using a SQL Stored Procedure, and stored in a new relation, `Trajectories(tid, a, b, ...)`: with 10^8 records of (conservatively) 500 bytes each, for a total of 50GB. The attribute `tid` is a unique key.

At this point we need to *validate* the trajectories, by using the other time slices in the `Observation` table. A correct trajectory should predict correctly the position of its object in all time slices $T = 1, 2, 3, \ldots, 100$. A few misses are tolerable, due to measurement errors, but most predictions are expected to be fairly accurate. To do this, we first predict, for each candidate trajectory and each timestamp t, the position of its object at time stamp t. This results in a new probabilistic table, where each predicted position is (approximated by) a Normal distribution, and which has a foreign key to `Trajectories`:

```
CREATE MATERIALIZED VIEW
```

```
      Predictions(new(PID), C.TID, T.t, X, Y, probabilities...) AS
SELECT C.a*T.t*T.t + C.b*T.t + C.c AS X,
       C.d*T.t*T.t + C.e*T.t + C.d AS Y, probability...
FROM Trajectories C, TimeStamps T
```

Here TimeStamps is the active domain of the timestamps: e.g. the set of the 20 timestamps. PID is a unique identifier created for each prediction point.

To validate the trajectories, we compute for each predicted point, the probability that it is actually observed in `Observations`. This is a spatial join between two probabilistic tables, `Predictions` and `Observations`:

```
CREATE MATERIALIZED VIEW PredictionsConfidence(PID, ....) AS
SELECT P.PID,  confidence()  /* here we aggregate probabilities */
FROM Predictions P, Observations O
WHERE P.T = Observations.T AND closeEnough(P.x,P.y,O.x,O.y)
GROUP BY P.PID
```

This is a query with a fuzzy join, defined by the predicate `closeEnough`: we assume that the confidence score computed for the prediction depends on the closeness of the predicted point to the real point. Finally, we join this back with the trajectories, to get a confidence score on the trajectories:

```
CREATE MATERIALIZED VIEW TrajectoreisCOnfidence(TID, ...) AS
SELECT C.TID, confidence()
FROM Trajectories T, PredictionsConfidence P
WHERE T.TID = P.TID
GROUP BY C.TID
```

Here the $\approx$ 100 confidence scores for one trajectory (one per timestamp) are aggregated into a global confidence score for that trajectory (hence the role of the `GROUP BY`): a few misses are tolerable, but many misses will resulg in a low confidence score for that trajectory. Finally, the trajectories are sorted in decreasing order of confidence score and filtered by some threshold.

4.2 Complex Aggregates on Probabilistic Data

A second challenge is to develop general query processing techniques for computing complex aggregates over probabilistic data with continuous values. In SQL, aggregates come in two forms: *value aggregates* that are returned to the user in the `SELECT` clause, like in the query *"`COUNT` the number of galaxies in each region"*; and *predicate aggregates* that appear in the `HAVING` clause, like in the query *"find all regions where the number of galaxies is greater than 20"*. In the case of probabilistic databases, value aggregates are interpreted as expected values. For example if the `type` of an object is a discrete probability distribution with possible outcomes `star`, `quasar`, `galaxy` etc., then counting the number of galaxies in a region results in the expected value of that number given the joint distributions of `type` attributes of all objects in the region. In the case of discrete random attributes, linear aggregate functions such as `COUNT` and `SUM` can be computed straightforwardly, by using the linearity of expectation, but other aggregates, such as `MIN`, `MAX`, `AVG`, are more difficult to compute, and their complexity depends on the structure of the SQL query (e.g. how many joins there are); the case of `AVG` is particularly difficult, even for queries without joins [23]. Predicate aggregates, on the other hand, are interpreted as a probability representing a confidence score: for each region the system computes the probability that the number of objects of type `galaxy` is greater than 20. To compute this

confidence score one generally has to compute the entire probability density function of the aggregate value. Some techniques for predicate aggregate have been developed for probabilistic databases with discrete random variables [33].

This challenge requires new techniques to extend value aggregates with `MIN`, `MAX` to queries with joins, and to extend both value and predicate aggregates to continuous attributes. To explore the challenge further, consider the following two examples in astrophysics: clustering with intrinsic uncertainty and gravitational lensing analysis.

Example 2. As described in section 3.2, galaxies are classified by clustering on more than 20 measurable attributes. Specifically, the `type` of an observed object is a function of fluxes, wavelength, morphology, moments, and more. These attributes are collected repeatedly by the sky surveys and stored in the `Observation` table (Figure 3). Each measurement is inherently uncertain and may be measured thousands of times over the course of the survey. Consequently, these values are represented as a normal distribution (i.e., the mean and variance). To determine object type, one may learn a partition function based on training set of manually labeled data, converting a set of continuous random variable measurements into a single discrete random variable.

The uncertain type of the objects can help answer a variety of astrophysical questions. For example, we can reason probabilistically about objects that change type from one time step to the next or disappear completely; previously these cases would be handled as anomalies. Returning to examples of complex aggregates, we can find regions in the sky with a concentration of galaxies above a specified threshold but with a bound on the minimum luminosity:

```
SELECT Region.id, COUNT(*), MIN(o.luminosity) FROM  Object o, Region r
WHERE Object.type = 'galaxy' and  inRegion(Object, Region)
GROUP BY Region HAVING COUNT(*) > $c AND MIN(o.luminosity) < $l
```

Here `Region` is a collection of regions in the sky (defined by the two diagonally opposed points) that form a partition of the sky. `inRegion` is a user defined function checking if an object is in the given region: it is a deterministic predicate, hence it defines a spatial join, but not a fuzzy join. Luminosity and type are both uncertain values, complicating the semantics and evaluation of this query.

Example 3. As a second application, we plan to use aggregate queries for scalable model-fitting to study the evolution of the growth of structure in the universe through gravitational lensing. The uncertainty in classification in this case comes in two ways. The distances to galaxies (lenses and the lenses themselves) are derived based on the colors of galaxies. These distance estimates (photometric redshifts) are inherent uncertain. While the uncertainties are often Gaussian they can also have multimodal probability density functions and complex forms. The second classification uncertainty is the label for the measured ellipticity (due to the gravitational shear). These measures, for inherently low signal-to-noise galaxies, which are barely resolved relative to the telescope point-spread-function, must be averaged over a large number of galaxies to provide a statistically significant measure of lensing (e.g. by aggregating regions on the sky). To accomplish this we will cluster groups of galaxies into shells at various distances, and then compute for each shell and each region in the sky the average shape distortion of the galaxies in that shell and that region. By comparing this average distortion to one predicted by a random model, we can perform the gravitational analysis (i.e. a shear correlation function):

```
SELECT Shell.id, Region.id, avg(Object.distortion)
FROM Object, Shell, Region
```

```
WHERE Object.type='galaxy'
AND  inRegion(Object, Region) AND inShell(Object, Shell)
GROUP BY Shell.id, Region.id
```

There are two forms of uncertainty that must be handled. First, the redshift shell to which a galaxy belongs will be a discrete random variable rather than a fixed shell. Second, the distortion will be given by a continuous, multimodal random variable. Thus, the average aggregate operator needs to handle both continuous and discrete random variables in its input.

This challenge requires an effective representation of the probability density function (pdf) of the aggregate value for various patterns of aggregate operators and query structures. Representations for some patterns are known: For the `COUNT` over safe queries the pdf can be computed by a safe plan [33], but for `SUM` queries this is not possible even for queries without joins. For `SUM`, one approach is to examine lossy representations of the pdf, in terms of moments, which can be computed effectively for `SUM`. Effective representations for other query patterns are considered open problems.

5 Related Work

Probabilistic databases have been studied intensively in recent years [1, 4, 10, 22, 36], motivated by a wide range of applications that need to manage large, imprecise data sets. The reasons for imprecision in data are as diverse as the applications themselves: in sensor and RFID data, imprecision is due to measurement errors [13, 35]; in information extraction, imprecision comes from the inherent ambiguity in natural-language text [16]; and in business intelligence, imprecision is tolerated because of the high cost of data cleaning [6]. In some applications, such as privacy, it is a requirement that the data be less precise. For example, imprecision is purposely inserted to hide sensitive attributes of individuals so that the data may be published [31]. Imprecise data has no place in traditional, precise database applications like payroll and inventory, and so, current database management systems are not prepared to deal with it. In contrast, in a *probabilistic database management system*, is a system that can store probabilistic data and supports SQL queries over this data. The major challenge studied in probabilistic databases is the integration of query processing with probabilistic inference. A number of techniques have been described recently: lineage-based representations [4], safe plans [11], algorithms for top-k queries [32, 39], and representations of views over probabilistic data [34].

6 Conclusions

We have described concrete problems for probabilistic databases arising from a new generation of massive sky surveys and massive astrophysical simulations. To address these problems, we recommend extensions to existing probablistic theories to accommodate 1) continuous random variable attributes in the context of *safe plans*, 2) scalable evaluation strategies for complex aggregates over continuous attributes, 3) scalable implementations over parallel databases clusters. In general, we advocate exploration of domain science as a driver for applications and requirements for probabilistic databases, and we offer this initial treatment in Astronomy as an exemplar.

Acknowledgements

This work was partially funded by NSF IIS-0713576 and the eScience Institute at the University of Washington.

References

[1] L. Antova, T. Jansen, C. Koch, and D. Olteanu. Fast and simple relational processing of uncertain data. In *ICDE*, 2008.

[2] A. Arasu, V. Ganti, and R. Kaushik. Efficient exact set-similarity joins. In *VLDB*, pages 918–929, 2006.

[3] M. Bartelmann, L. J. King, and P. Schneider. Weak-lensing halo numbers and dark-matter profiles. A&A, 378:361–369, Nov. 2001.

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

[5] D. Bitton, H. Boral, D. DeWitt, and K. Wilkinson. Parallel algorithms for the execution of relational database operations. *ACM Transactions on Database Systems*, 8(3):324–353, September 1983.

[6] D. Burdick, P. Deshpande, T. S. Jayram, R. Ramakrishnan, and S. Vaithyanathan. Efficient allocation algorithms for olap over imprecise data. In *VLDB*, pages 391–402, 2006.

[7] R. Chaiken, B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. Scope: easy and efficient parallel processing of massive data sets. In *Proc. of the 34th Int. Conf. on Very Large DataBases (VLDB)*, 2008.

[8] A. J. Connolly, I. Csabai, A. S. Szalay, D. C. Koo, R. G. Kron, and J. A. Munn. Slicing through multicolor space: Galaxy redshifts from broadband p hotometry. *Astronomical Journal*, 110:2655+, Dec. 1995.

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

[10] N. Dalvi and D. Suciu. Management of probabilistic data: Foundations and challenges. In *PODS*, pages 1–12, Beijing, China, 2007. (invited talk).

[11] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDB J.*, 16(4):523–544, 2007.

[12] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. In *Proc. of the 6th USENIX Symp. on Operating Systems Design & Implementation (OSDI)*, 2004.

[13] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, pages 588–599, 2004.

[14] D. DeWitt and J. Gray. Parallel database systems: the future of high performance database systems. *Communications of the ACM*, 35(6):85–98, 1992.

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

[16] R. Gupta and S. Sarawagi. Creating probabilistic databases from information extraction models. In *VLDB*, pages 965–976, 2006.

[17] Hadoop. `http://hadoop.apache.org/`.

[18] A. J. S. Hamilton. Evidence for biasing in the CfA survey. ApJ, 331:L59–L62, Aug. 1988.

[19] D. W. Hogg, M. R. Blanton, J. Brinchmann, D. J. Eisenstein, D. J. Schlegel, J. E. Gunn, T. A. McKay, H.-W. Rix, N. A. Bahcall, J. Brinkmann, and A. Meiksin. The Dependence on Environment of the Color-Magnitude Relation of Galaxies. ApJ, 601:L29–L32, Jan. 2004.

[20] O. Ilbert, S. Lauger, L. Tresse, V. Buat, S. Arnouts, O. Le Fèvre, D. Burgarella, E. Zucca, S. Bardelli, G. Zamorani, D. Bottini, B. Garilli, V. Le Brun, D. Maccagni, R. Picat, J.-P. an d Scaramella, M. Scodeggio, G. Vettolani, A. Zanichelli, C. Adami, M. Arnaboldi, M. . Bolzonella, A. Cappi, S. Charlot, T. Contini, S. Foucaud, P. Franzetti, I. Gavignaud, L. Guzzo, A. Iovino, H. J. McCracken, B. Marano, C. Marinoni, G. Mathez, A. Mazure, B. Meneux, R. Merighi, S. Paltani, R. Pello, A. Pollo, L. Pozzetti, M. Radovich, M. Bondi, A. Bongiorno, G. Busarello, Y. Ciliegi, P. an d Mellier, P. Merluzzi, V. Ripepi, and D. Rizzo. The VIMOS-VLT Deep Survey. Galaxy luminosity function per morpholo gical type up to z = 1.2. A&A, 453:809–815, July 2006.

[21] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *Proc. of the European Conference on Computer Systems (EuroSys)*, pages 59–72, 2007.

[22] R. Jampani, F. Xu, M. Wu, L. Perez, C. Jermaine, and P. Haas. MCDB: a Monte Carlo approach to managing uncertain data. In *SIGMOD*, pages 687–700, 2008.

[23] T. Jayram, S. Kale, and E. Vee. Efficient aggregation algorithms for probabilistic data. In *SODA*, 2007.

[24] J. Kubica, A. Moore, A. Connolly, and R. Jedicke. A multiple tree algorithm for the efficient association of asteroid observations. In *The Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2005.

[25] Z. Ma, W. Hu, and D. Huterer. Effects of Photometric Redshift Uncertainties on Weak-Lensing Tomo graphy. ApJ, 636:21–29, Jan. 2006.

[26] B. Mobasher, P. Capak, N. Z. Scoville, T. Dahlen, M. Salvato, H. Aussel, D. J. Thompson, R. Feldmann, L. Tasca, O. Lefevre, S. Lilly, C. M. Carollo, J. S. Kartaltepe, H. McCracken, J. Mould, A. Renzini, D. B. Sanders, P. L. Shopbell, Y. Taniguchi, M. Ajiki, Y. Shioya, T. Contini, M. Giavalisco, O. Ilbert, A. Iovino, V. Le Brun, V. Mainieri, M. Mignoli, and M. Scodeggio. Photometric Redshifts of Galaxies in COSMOS. ApJS, 172:117–131, Sept. 2007.

[27] P. Norberg, C. M. Baugh, E. Hawkins, S. Maddox, D. Madgwick, O. Lahav, S. Cole, C. S. Frenk, I. Baldry, J. Bland-Hawthorn, T. Bridges, R. Cannon, M. Colless, C. Collins, W. Couch, G. Dalton, R. De Propris, S. P. Driver, G. Efstathiou, R. S. Ellis, K. Glazebrook, C. Jackson, I. Lewis, S. Lumsden, J. A. Peacock, B. A. Peterson, W. Sutherland, and K. Taylor. The 2dF Galaxy Redshift Survey: the dependence of galaxy clustering on luminosity and spectral type. MNRAS, 332:827–838, June 2002.

[28] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *SIGMOD'08: Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 1099–1110, 2008.

[29] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan. Interpreting the data: Parallel analysis with Sawzall. *Scientific Programming*, 13(4), 2005.

[30] M. Postman and M. J. Geller. The morphology-density relation - The group connection. ApJ, 281:95–99, June 1984.

[31] V. Rastogi, D. Suciu, and S. Hong. The boundary between privacy and utility in data publishing. In *VLDB*, 2007.

[32] C. Re, N. Dalvi, and D. Suciu. Efficient Top-k query evaluation on probabilistic data. In *ICDE*, 2007.

[33] C. Re and D.Suciu. Efficient evaluation of having queries on a probabilistic database. In *Proceedings of DBPL*, 2007.

[34] C. Re and D.Suciu. Materialized views in probabilistic databases for information exchange and query optimization. In *Proceedings of VLDB*, 2007.

[35] C. Re, J. Letchner, M. Balazinska, and D. Suciu. Event queries on correlated probabilistic streams. In *SIGMOD*, Vancouver, Canada, 2008.

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

[37] R. K. Sheth, A. J. Connolly, and R. Skibba. Marked correlations in galaxy formation models. *ArXiv Astrophysics e-prints*, Nov. 2005.

[38] R. K. Sheth and G. Tormen. On the environmental dependence of halo formation. MNRAS, 350:1385–1390, June 2004.

[39] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang. Probabilistic top- and ranking-aggregate queries. *ACM Trans. Database Syst.*, 33(3), 2008.

[40] I. Zehavi, D. H. Weinberg, Z. Zheng, A. A. Berlind, J. A. Frieman, R. Scoccimarro, R. K. Sheth, M. R. Blanton, M. Tegmark, H. J. Mo, N. A. Bahcall, J. Brinkmann, S. Burles, I. Csabai, M. Fukugita, J. E. Gunn, D. Q. Lamb, J. Loveday, R. H. Lupton, A. Meiksin, J. A. Munn, R. C. Nichol, D. Schlegel, D. P. Schneider, M. SubbaRao, A. S. Szalay, A. Uomoto, and D. G. York. On Departures from a Power Law in the Galaxy Correlation Function. ApJ, 608:16–24, June 2004.