

The Fourth International VLDB Workshop
on
Management of Uncertain Data

edited by
Ander de Keijzer and Maurice van Keulen
University of Twente

CTIT Workshop Proceedings Series

Sponsor



Centre for Telematics and Information Technology (CTIT)

Publication Details

Proceedings of the Fourth International VLDB Workshop on Management of Uncertain Data

Edited by Ander de Keijzer and Maurice van Keulen

Published by the Centre for Telematics and Information Technology (CTIT),

University of Twente

CTIT Workshop Proceedings Series WP10-04

ISSN 0929-0672

Organizing Committee*Co-chairs*

Ander de Keijzer, University of Twente, The Netherlands

Maurice van Keulen, University of Twente, The Netherlands

Publicity chair

Ghita Berrada, University of Twente, The Netherlands

Program Committee

Patrick Bosc, IRISA/ENSSAT, France

Matthew Damigos, NTUA, Greece

Guy de Tré, University of Ghent, Belgium

Curtis Dyreson, Utah State University, USA

Michael Fink, Vienna University of Technology, Austria

Maarten Fokkinga, University of Twente, The Netherlands

Manolis Gergatsoulis, Ionian University, Greece

Nikos Kiourtis, NTUA, Greece

Christoph Koch, Cornell University, USA

Birgitta König-Ries, University of Jena, Germany

Maurizio Lenzerini, University of Rome La Sapienza, Italy

Dan Olteanu, Oxford University, UK

Olivier Pivert, IRISA/ENSSAT, France

Giuseppe Psaila, University of Bergamo, Italy

Christopher Ré, University of Wisconsin-Madison, USA

Anish Das Sarma, Yahoo!Research, USA

V.S. Subrahmanian, University of Maryland, USA

Dan Suciu, University of Washington, USA

Martin Theobald, Max Planck Institute, Germany

Vasilis Vassalos, AUEB-RC, Greece

Jef Wijsen, Université de Mons, Belgium

Vladimir Zadorozhny, University of Pittsburgh, USA

Workshop Program

Monday, September 13th, 2010
Grand Copthorne Waterfront Hotel, Singapore

- 09.00 **Opening**
Ander de Keijzer and Maurice van Keulen
- 09.05 **Invited Talk**
From MUD to MIRE: Managing Inherent Risk in the Enterprise
Peter J. Haas
- 10.00 **Session 1: Provenance and answer explanation**
WHY SO? or WHY NO? Functional Causality for Explaining Query Answers
Alexandra Meliou, Wolfgang Gatterbauer, Katherine F. Moore, and Dan Suciu
- 10.30 **Coffee break**
- 11.00 **Session 2: Non-relational UDBMSs**
Extending Magic Sets Technique to Deductive Databases with Uncertainty
Qiong Huang and Nematollaah Shiri
Storing and Querying Probabilistic XML Using a Probabilistic Relational DBMS
Emiel S. Hollander and Maurice van Keulen
Time-aware Reasoning in Uncertain Knowledge Bases
Yafang Wang, Mohamed Yahya, and Martin Theobald
- 12.30 **Lunch Break**
- 14.00 **Session 3: Query processing in UDBMSs**
Query Containment for Databases with Uncertainty and Lineage
Foto N. Afrati and Angelos Vasilakopoulos
Dissociation and Propagation for Efficient Query Evaluation over Probabilistic Databases
Wolfgang Gatterbauer, Abhay K. Jha, and Dan Suciu
Generalized Uncertain Databases: First Steps
Parag Agrawal and Jennifer Widom
- 15.30 **Coffee Break**
- 16.00 **Session 4: Applications of UDBMSs**
Tuple Merging in Probabilistic Databases
Fabian Panse and Norbert Ritter
Uncertain Databases in Collaborative Data Management
Reinhard Pichler, Vadim Savenkov, Sebastian Skritek and Hong-Linh Truong
Handling Uncertainty and Correlation in Decision Support
Katrin Eisenreich and Philipp Rösch
- 17.30 **Closing**

Preface

This is the fourth edition of the international VLDB workshop on Management of Uncertain Data. Previous editions of this workshop took place in New Zealand, Austria and France. Research on uncertain data has grown over the past few years. Besides workshops on the topic of uncertain data, also sessions at large conferences, such as VLDB, on the same topic are organized.

This edition, we have ten research talks, in four sessions, addressing different topics in uncertain data. In addition, we start the workshop with an invited talk by Peter Haas from IBM Research, entitled *From MUD to MIRE: Managing Inherent Risk in the Enterprise*.

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

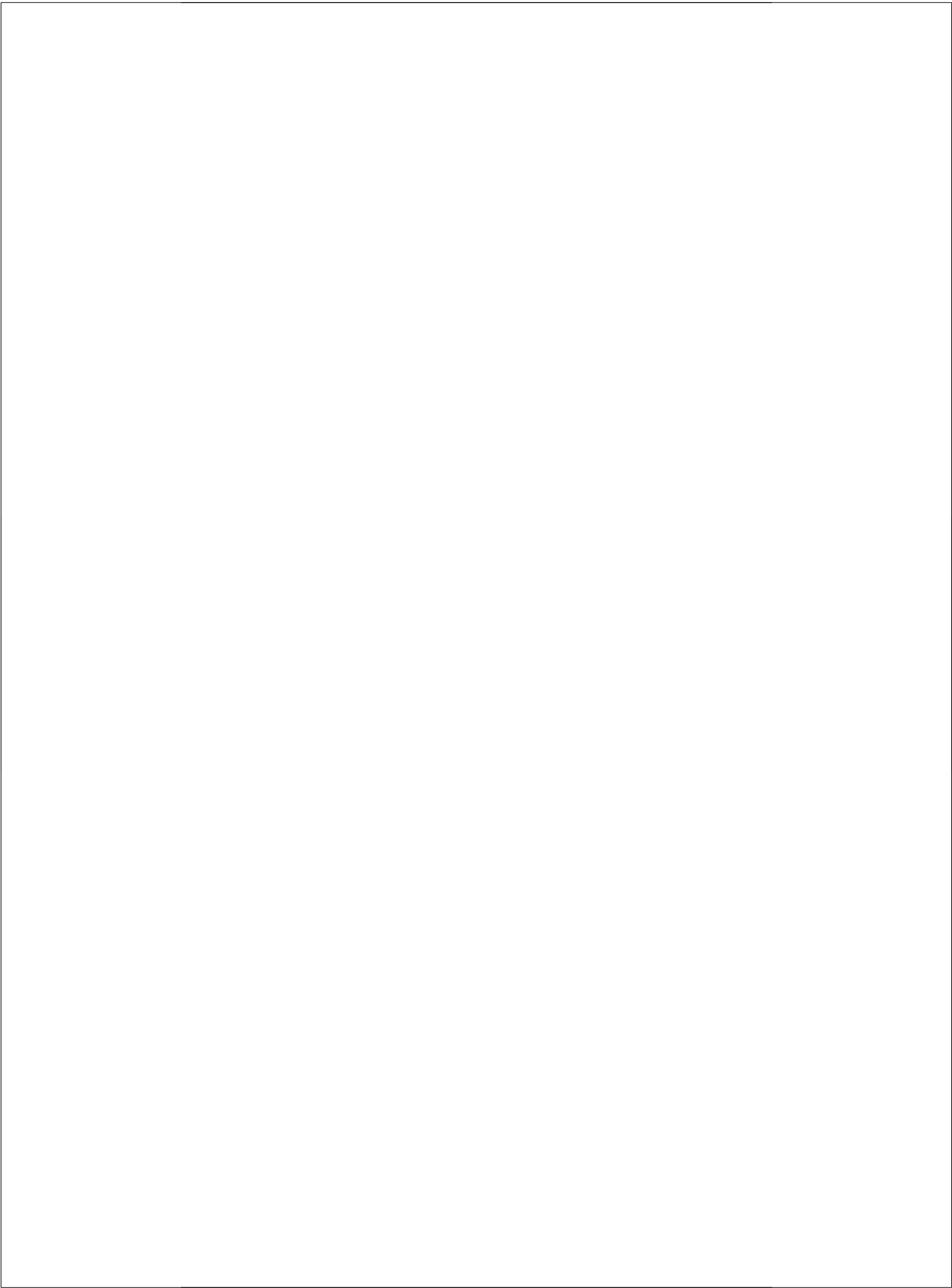
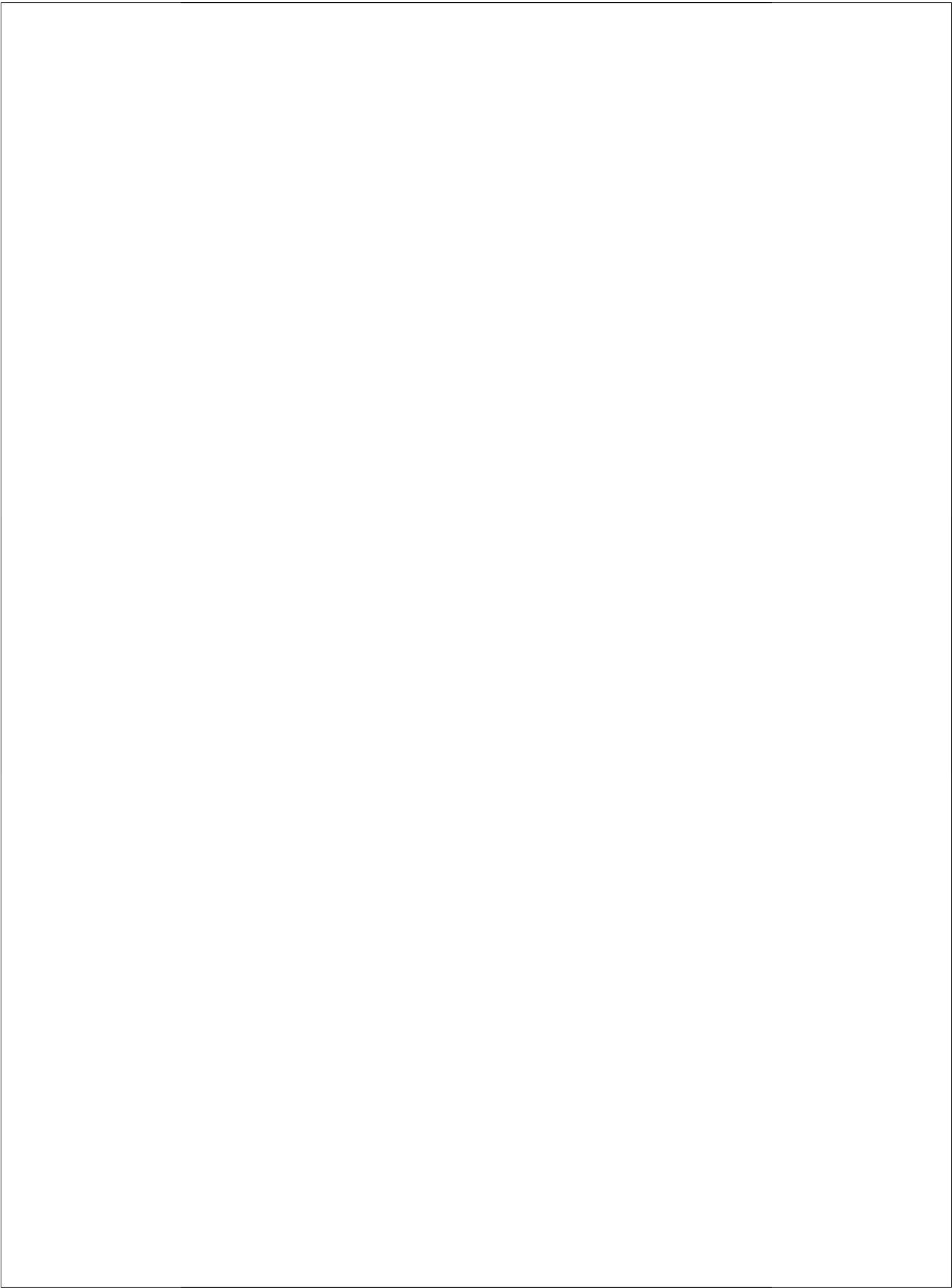


Table of Contents

From MUD to MIRE: Managing Inherent Risk in the Enterprise <i>Peter J. Haas</i>	1
WHY SO? or WHY NO? Functional Causality for Explaining Query Answers <i>Alexandra Meliou, Wolfgang Gatterbauer, Katherine F. Moore, and Dan Suciu</i>	3
Extending Magic Sets Technique to Deductive Databases with Uncertainty <i>Qiong Huang and Nematollaah Shiri</i>	19
Storing and Querying Probabilistic XML Using a Probabilistic Relational DBMS <i>Emiel S. Hollander and Maurice van Keulen</i>	35
Time-aware Reasoning in Uncertain Knowledge Bases <i>Yafang Wang, Mohamed Yahya, and Martin Theobald</i>	51
Query Containment for Databases with Uncertainty and Lineage <i>Foto N. Afrati and Angelos Vasilakopoulos</i>	67
Dissociation and Propagation for Efficient Query Evaluation over Probabilistic Databases <i>Wolfgang Gatterbauer, Abhay K. Jha, and Dan Suciu</i>	83
Generalized Uncertain Databases: First Steps <i>Parag Agrawal and Jennifer Widom</i>	99
Tuple Merging in Probabilistic Databases <i>Fabian Panse and Norbert Ritter</i>	113
Uncertain Databases in Collaborative Data Management <i>Reinhard Pichler, Vadim Savenkov, Sebastian Skritek and Hong-Linh Truong</i>	129
Handling Uncertainty and Correlation in Decision Support <i>Katrin Eisenreich and Philipp Rösch</i>	145



From MUD to MIRE: Managing Inherent Risk in the Enterprise

Peter J. Haas

peterh@almaden.ibm.com

IBM ALmaden Research Center
San Jose, California, USA

Two questions always seem to arise when talking with industrial colleagues about probabilistic databases (prDBs): "Where do the probabilities come from?" and "Who is going to use this stuff in the real world?" In this talk I will discuss my recent experiences in trying to deal with these questions. One compelling answer to the first question is that, with the recent spike in popularity of "business analytics," an increasingly important source of uncertainty arises from the use of complex stochastic models to predict future or hypothetical data values. As a result, I have been viewing my own work on the Monte Carlo Database System (MCDB) as being less about prDBs per se, and more about stochastic predictive analytics over big data. Much work remains to be done in this space. For the second question, I would argue that an increasingly important driver of probDB is risk management. Most people have very poor intuition about the nature of probability and risk, and succumb to the "flaw of averages" in its many insidious forms. There has been some exciting recent work on developing interactive tools that managers, executives, and other decision-makers can use to better understand the risks and rewards associated with investment and policy decisions. These tools are part of an emerging "probability management" infrastructure for coherent risk assessment within and across enterprises. These ideas are beginning to take hold in companies such as Royal Dutch Shell, Merck Pharmaceutical, Oracle, Wells Fargo Bank, and IBM. Exploring the role of risk in our work opens up new areas of research and also gives our community the opportunity to have enormous real-world impact by playing a key role in the probability-management ecosystem of the future. Some recent work by myself and others illustrates some of the possibilities.

WHY SO? or WHY NO? Functional Causality for Explaining Query Answers

Alexandra Meliou, Wolfgang Gatterbauer, Katherine F. Moore, and Dan Suciu

University of Washington
{ameli, gatter, kfm, suciu}@cs.washington.edu

Abstract. In this paper, we propose *causality* as a unified framework to *explain query answers* and non-answers, thus generalizing and extending several previously proposed definitions of provenance and missing query result explanations. Starting from the established definition of *actual causes* by Halpern and Pearl [12], we propose *functional causes* as a refined definition of causality with several desirable properties. These properties allow us to apply our notion of causality in a database context and apply it uniformly to define the causes of query results and their individual contributions in several ways: (i) we can model both *provenance* as well as *non-answers*, (ii) we can define explanations as either *data* in the input relations or relational *operations* in a query plan, and (iii) we can give graded degrees of responsibility to individual causes, thus allowing us to *rank causes*. In particular, our approach allows us to explain contributions to relational *aggregate functions* and to rank causes according to their respective responsibilities, aiding users in identifying errors in uncertain or untrusted data. Throughout the paper, we illustrate the applicability of our framework with several examples. This is the first work that treats “positive” and “negative” provenance under the same framework, and establishes the theoretical foundations of causality theory in a database context.

1 Introduction

When analyzing uncertain data sets, users are often interested in *explanations* for their observations. Explaining the causes of surprising query results allows users to better understand their data, and identify possible errors in data or queries. In a database context, explanations concern results returned by explicit or implicit queries. For example, “Why does my personalized newscast have more than 20 items today?” Or, “Why does my favorite undergrad student not appear on the Dean’s list this year?” Database research that addresses these or similar questions is mainly work on *lineage of query results*, such as why [8] or where provenance [3], and very recently, explanations for non-answers [15,4]. While these approaches differ over what the response to questions should be, all of them seem to be linked through a common underlying theme: understanding *causal relationships* in databases.

Humans usually have an intuition about what constitutes a cause of a given effect. In this paper, we define the foundational notion of *functional causality* that can model this intuition in an exact mathematical framework, and show how it can be applied to encode and solve various causality related problems. In particular, it allows us to uniformly model the questions of WHY SO? and WHY NO? with regards to query answers. It also allows us to represent different previous approaches, thus illustrating causality to be a critical element unifying prior work in this field.

N(ewsFeeds)			R(outing)	
<i>nid</i>	<i>story</i>	<i>tag</i>	<i>tag</i>	
1	... share lead in Singapore championship ...	Golf	Obama	
2	... economic downturn affected sensitive ...	Business	DB_conf	
3	... with sequences shot in Singapore ...	Movies	Golf	
4	... when President Obama meets former ally ...	Obama	Technology	
5	... Singapore slow down hiring ...	Business	Health	
6	... Oscars 2010: Academy's 'best' choice ...	Movies		
7	... HP launches cloud lab in Singapore ...	Technology		
8	... struggles to corral votes for health bill ...	Health		
9	... VLDB conference this year in Singapore ...	DB_conf		
10	... at the Indianapolis Motor Speedway ...	Indy 500		
11	... Indianapolis host to SIGMOD 2010 ...	DB_conf		
12	... VLDB in Singapore promises to be ...	DB_conf		
13	... more people in Indianapolis this year ...	Indy_500		
14	... Gatorade drops Tiger woods ...	Golf		

Query answer:	
P(ersonalized alerts)	
<i>cities</i>	
	Paris
	Singapore
	Athens
	Vancouver

Fig. 1: Example of a personalized alert-feed (P) as a result of a query filtering all news (N) based on a carefully constructed routing table (R).

Example 1. A major travel agency monitors a large number of news feeds in order to identify trends, opportunities, or alerts about various cities. Central to this activity is a carefully personalized routing table and query, which filters what information to forward to each specialized travel agent by carefully chosen keywords. Fig. 1 shows the routing table for one user R , as well as a sample news feed. The query issuing alerts to this user is:

```

select    C.name
from      NewsFeeds N, Routing R, City C
where     C.name substring N.story and N.tag = R.tag
group by  C.name
having    count(*) > 20

```

The result is a list of cities that are drawn to the attention of this particular agent, shown in Fig. 1. As popular destinations, Paris and Athens are predictable answers, and so is Vancouver because of the recent Olympics. But this agent believes Singapore is an error, and wants to know what entries in the Routing table caused it to appear on her watch list. She wants to ask “Why am I being alerted about Singapore?”. The system should answer that the keywords DB_conf, technology, and golf are causes with various degrees of responsibility.

As illustrated in Example 1, we want to allow users to ask simple questions based on the results they receive, and hence, allow them to learn what may be the *cause* of any surprising or undesirable answer. Such questions can refer to either presence (WHY SO?) or absence (WHY NO?) of results. Furthermore, the user should be provided with a ranking of causes based on their individual contribution or *responsibility*. Unexpected results are often an indication of errors, and tracking their causes is a crucial step in repairing faulty data, or mistakes in queries. Our ultimate goal is to define a language that allows users to specify causal queries for given results. In this paper, we (i) lay the theoretical groundwork and define a formal model that allows us to capture such causality-related questions in a uniform framework, and (ii) illustrate the applicability of our scheme through various examples.

Summary and outline. We start by reviewing existing work on causality in AI in Section 2, and propose *functional causes* as a refined notion that mitigates problems of existing definitions (Sect. 2.1). In Sect. 3 we highlight several desirable properties of functional causes, which are important for their applicability in a database context. Section 4 gives several examples of applying our framework to give WHY SO? and WHY NO? explanations to *database queries*. We show that our unifying approach generalizes provenance

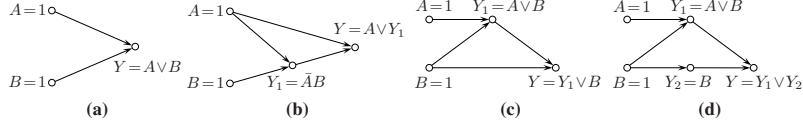


Fig. 2: (a) Alice (A) and Bob (B) each throw a rock at a bottle, which breaks if it gets hit by either rock ($Y = A \vee B$). (b) Alice's throw *preempts* Bob's ($A = 1 \Rightarrow Y_1 = 0$). (c,d) *Expansion* causes problems for the HP definition: Introducing node Y_2 , which merely repeats the value of B , does not change the function $Y(X)$, but makes A an actual cause.

as well as non-answers (Sect. 4.1), handles contributions to aggregate functions by ranking causes according to their responsibilities for the result (Sect. 4.2), and can also model causes other than tuples (Sect. 4.3).

2 Causality Definitions

Due to space limitations, we briefly overview the two most established definitions of causality from the AI and philosophy literature, and refer the reader to our technical report [20] for more details, discussion of issues and implications, examples, and proofs of all results.

Counterfactual Causes. With a long tradition in philosophy [16], the argument of counterfactual causality is that the relationship between cause and effect can be understood as a counterfactual statement, i.e. an event is considered a cause of an effect if the effect would not have happened in the absence of the event. We focus on the boolean case, and in our notion, the variable assignment (event) $X = x^0$ is a cause of expression ϕ , iff $X = x^0 \wedge \phi$ and $[X \leftarrow \neg x^0] \Rightarrow \neg \phi$. However, counterfactual causality cannot explain causality for slightly more complicated scenarios such as for *disjunctive causes*, i.e. when there are two potential causes of an event.

Actual Causes. The HP definition of causality [12] is based on counterfactuals, but can correctly model disjunction and many other complications. It is the most established definition in the field of structural causality, and relies on the use of a *causal network* (much like a Bayesian network), representing dependencies between variables (e.g. Fig. 2a). In a database context, the variables can be tuples, but they can represent in general any element that may be causally relevant. Every node in the causal network is governed by a *structural equation* that determines the node's assignment based on its input. A causal model is commonly denoted as $M = (N, \mathcal{F})$, where N the set of variables, and \mathcal{F} the set of structural equations. The idea is that X is a cause of Y if Y counterfactually depends on X under "some" *permissive contingency*, where "some" is elaborately defined.¹ The heart of the definition is condition AC2 in [12, Def. 3.1], which is effectively a generalization of counterfactual causes. The requirement is that there exists some assignment of the variables for which X is counterfactual, and that this assignment does not make any fundamental changes to the *causal path* of X (the descendants of X in the causal network). The use of the causal network makes the HP definition very flexible, allowing it to capture different scenarios of causal relationships. For example, it correctly handles disjunctive causes and preemption, i.e. when there are two potential causes of an event and one chronologically *preempts* the other (e.g. Fig. 2b).

The HP definition does however have some limitations which make its application to a database context problematic. In the well studied *Shock C* example (see [22]), actual cau-

¹ Contingencies relate to possible world semantics: "Is there a possible world that makes X counterfactual?"

sality produces unintuitive results; a variable is determined to be a cause of a tautology, which in a data context is semantically spurious. A less known but equally important issue of the definition is its lack of robustness to minor network variations. The addition of “dummy” nodes, which do not affect the function or assignments of other nodes, can change the causality of variables (Fig. 2c,2d). This is problematic in a database setting, where we care about query semantics rather than syntax. We revisit this issue in Sect. 3.1, and also refer the reader to our technical report [20] for an extensive discussion.

2.1 Functional Causes

A fundamental challenge in applying causality to queries is that causality is defined over an entire *network*: it is not enough to know the dependency of the effect on the input variables, we also need to reason about intermediate dependent nodes. This requirement is difficult to carry over to a database setting, where we care about the semantics of a query rather than a particular query plan. Our approach is to represent a causal network with two appropriate functions that *semantically capture* the causal dependencies of a network. The two key notions we need for that are *potential functions* and *dissociation expressions*.

Figure 3 represents a causal network in our framework. In contrast to the HP approach, only input variables from \mathbf{X} can be causes and part of permissive contingencies. As in the HP approach, every dependent node Y is described by a structural equation F_Y , which assigns a truth value to Y based on the values of its parents. The *Boolean formula* Φ_Y of Y defines its truth assignment based on the input variables \mathbf{X} , and is constructed by recursing through the structural equations of Y ’s ancestors. For example, in Fig. 2b, $\Phi_Y(\mathbf{X}) = A \vee (A \wedge B)$, where $\mathbf{X} = \{A, B\}$. We denote as $\Phi(\mathbf{X}) = \Phi_{Y_j}(\mathbf{X})$, where Y_j is the effect node, and we say that the causal network has formula Φ . The *potential function* P_Φ is then simply the unique multilinear polynomial representing Φ . It is equal to the probability that Φ is true given the probabilities of its input variables.

Definition 1 (Potential Function). The potential function $P_\Phi(\mathbf{x})$ of a Boolean formula $\Phi(\mathbf{X})$ with probabilities $\mathbf{x} = \{x_1, \dots, x_k\}$ of the input variables is defined as follows:

$$P_\Phi(\mathbf{x}) = \sum_{\varepsilon \rightarrow \{0,1\}^k} \left(\prod_{i=1}^k x_i^{\varepsilon_i} \right) \Phi(\varepsilon), \quad x_i^{\varepsilon_i} = \begin{cases} x_i & \text{if } \varepsilon_i = 1 \\ 1 - x_i & \text{if } \varepsilon_i = 0 \end{cases}$$

The potential function is a sum with one term for each truth assignment ε of variables \mathbf{X} . Each term is a product of factors of the form x_i or $1 - x_i$ and only occurs in the sum if the formula is true at the given assignment ($\Phi(\varepsilon) = 1$). For example, if $\Phi = X_1 \wedge (X_2 \vee X_3)$ then $P_\Phi = x_1 x_2 (1 - x_3) + x_1 (1 - x_2) x_3 + x_1 x_2 x_3$, which simplifies to $x_1 (x_2 + x_3 - x_2 x_3)$. We use delta notation to denote changes ΔP in the potential function due to changes in the inputs: Given an actual assignment \mathbf{x}^0 and a subset of variables \mathbf{S} , we define $\Delta P_\Phi(\mathbf{S}) := P_\Phi(\mathbf{x}^0) - P_\Phi(\mathbf{x}^0 \oplus \mathbf{S})$, where $\mathbf{x}^0 \oplus \mathbf{S}$ (denoting XOR) indicates the assignment obtained by starting from \mathbf{x}^0 and inverting all variables in \mathbf{S} .

To semantically capture differences in causality between networks with logically equivalent boolean formulas (e.g. Fig. 2a,2b), we use *Dissociation Expressions* (DEs):

Definition 2 (Dissociation Expression). A dissociation expression with respect to a variable X_0 is a Boolean expression defined by the grammar:

$$\Psi ::= X \in \mathbf{X}$$

$$\Psi ::= \sigma(\Psi_1, \Psi_2, \dots, \Psi_k), \quad X_0 \in V(\Psi_i) \cup V(\Psi_j) \Rightarrow V(\Psi_i) \cap V(\Psi_j) \subseteq \{X_0\}$$

where $V(\Psi_i)$ is the set of input variables of formula Ψ_i .

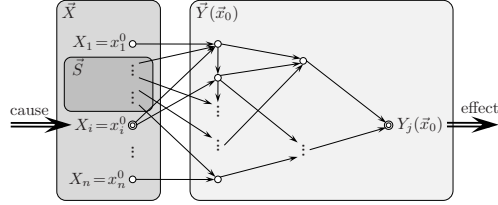


Fig. 3: FC framework: the causal network is partitioned into the input variables X with cause under consideration X_i , and dependent variables Y with effect variable Y_j . Support $S \subseteq X \setminus \{X_i\}$ corresponds to permissive contingency from the HP framework.

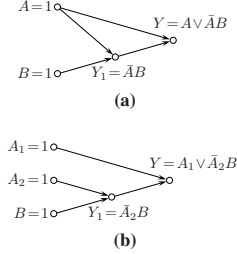


Fig. 4: A causal network CN (a) and its dissociation network DN (b) with respect to B .

Dissociation expressions allow us to semantically capture within a Boolean formula, the causal dependencies of a variable X_0 in a causal network. This is possible by recording the effect of X_0 along different network paths and disallowing any variable from being combined with X_0 in more than one subexpression. We illustrate with a detailed example.

Example 2. In the network of Fig. 4a, variable A contributes to the causal path of B at two locations. This “independent” influence can be represented by the dissociation expression $\Psi = A_1 \vee (\bar{A}_2 \wedge B)$, which essentially separates A into two variables A_1 and A_2 (see Fig. 4b). $\Psi' = A \vee (\bar{A} \wedge B)$ is not a valid DE with respect to B because, for its subexpressions $\Psi'_1 = A$ and $\Psi'_2 = \bar{A} \wedge B$, it is $B \in V(\Psi'_1) \cup V(\Psi'_2)$ but $V(\Psi'_1) \cap V(\Psi'_2) = \{A\} \not\subseteq \{B\}$. We demonstrate how Ψ captures semantically the network structure: The HP definition checks actual causality of B in the network of Fig. 4a by determining the value of Y for a setting $\{A=0, B=1\}$, while forcing Y_1 to its original value. The dissociation expression $\Psi(A_1, A_2, B) = A_1 \vee (\bar{A}_2 \wedge B)$, with potential function $P_\Psi(a_1, a_2, b) = a_1 + b - a_1b - a_2b + a_1a_2b$, allows us to perform the same check by simply computing $P_\Psi(0, 1, 1)$. In this case $P_\Psi(0, 1, 1) = 0 \neq P_\Psi(1, 1, 1)$, which was the original variable assignment, meaning that the change altered values on the causal path.

The grammar-based definition of dissociation expressions allows us to identify expressions that are valid DEs with respect to a variable. We will now define mappings, called *foldings*, from DEs to Boolean formulas, which are used to formally define correspondence between formulas. For instance, $\bar{A} \vee B$ is a valid dissociation expression with respect to B but does not correspond to formula $A \vee (\bar{A} \wedge B)$. A folding basically maps a set of input variables X' to another set X , transforming formula Ψ to Ψ' . If Ψ' is grammatically equivalent to Φ , then Ψ is a dissociation expression of Φ . For example, $f(\{A_1, A_2, B\}) = \{A, A, B\}$ defines a folding from $\Psi = A_1 \vee (\bar{A}_2 \wedge B)$ to the formula $\Phi = A \vee (\bar{A} \wedge B)$. In simple terms, a DE Ψ with a folding to Φ is a representation of Φ with a larger number of input variables.

Definition 3 (Expression Folding). Given $f : X' \rightarrow X$ mapping variables X' to X , the folding (\mathcal{F}, f) of a dissociation expression $\Psi(X')$ defines a formula $\Phi = F(\Psi)$, s.t:

$$\Psi ::= X' \Rightarrow \mathcal{F}(X) = f(X')$$

$$\Psi ::= \sigma(\Psi_1, \Psi_2, \dots, \Psi_k) \Rightarrow \mathcal{F}(\Psi) = \sigma(\mathcal{F}(\Psi_1), \mathcal{F}(\Psi_2), \dots, \mathcal{F}(\Psi_k))$$

The dissociation of input variables into several new input variables captures the distinct effect of variables on the causal path, thus providing the necessary network semantics. Using $|\Psi|$ to denote the cardinality of the input set of Ψ , then $|\Psi| \geq |\Phi|$, and if $|\Psi| = |\Phi|$ then $\Psi = \Phi$.

Theorem 1 (DE Minimality). *If \mathcal{D} the set of all DEs w.r.t. $X_0 \in \mathbf{X}$ with a folding to $\Phi(\mathbf{X})$, then \exists unique $\Psi_i \in \mathcal{D}$ of minimum size: $|\Psi_i| = \min_{\Psi \in \mathcal{D}} |\Psi|$ and $\forall j \neq i, |\Psi_j| = |\Psi_i| \Rightarrow \Psi_j = \Psi_i$.*

The DE of minimum size replicates those variables, and only those variables, that affect the causal path at more than one location. It is simply called the dissociation expression of Φ , with input nodes \mathbf{X}_t (Fig. 4b). A folding maps \mathbf{X}_t back to the original input variables: $\mathbf{X} = f(\mathbf{X}_t)$. The reverse mapping is denoted $\mathbf{X}_t = [\mathbf{X}]_t = \{X_i \mid f(X_i) \in \mathbf{X}\}$. We often refer to the *dissociation network* of Φ , meaning the causal network representing the DE of Φ (e.g. Fig. 4b).

Definition 4 (Functional Cause). *The event $X_i = x_i^0$ is a cause of ϕ in a causal model iff:*

FC1. *Both $X_i = x_i^0$ and ϕ hold under assignment \mathbf{x}^0*

FC2. *Let P_Φ and P_Ψ be the potential functions of Φ and its DE w.r.t. X_i , respectively.*

There exists a support $\mathbf{S} \subseteq \mathbf{X} \setminus \{X_i\}$, such that:

(a) $\Delta P_\Phi(\mathbf{S} \cup X_i) \neq 0$

(b) $\Delta P_\Psi(\mathbf{S}'_t) = 0$, for all subsets $\mathbf{S}'_t \subseteq [\mathbf{S}]_t$

Condition FC2(b) is analogous to AC2(b) of the HP definition, which requires checking that the effect does not change for all possible combinations of setting the dependent nodes to their original values. Similarly, FC ensures that no part of the changed nodes (the support \mathbf{S}) is counterfactual in the dissociation network.

Intuition. The definition of functional causes captures three main points: (i) a counterfactual cause is always a cause, (ii) if a variable is not counterfactual under any possible assignment of the other variables, then it cannot be a cause, and (iii) if $X = x^0$ is a counterfactual cause under some assignment that inverts a subset \mathbf{S} of the other variables, then no part of \mathbf{S} should be by itself counterfactual.

We use the rock thrower example from [12], depicted in Fig. 2a and 2b, to demonstrate how functional causes (like actual causes) can handle preemption.

Example 3. *The two different models of the problem, with and without preemption (Fig. 2b and 2a respectively) are characterized by logically equivalent Boolean expressions: $A \vee \bar{A}B = A \vee B$. However, B is not a cause (actual or functional) in Fig. 2b, because Bob's throw is preempted by Alice's. The minimal dissociation expression for $\Phi = A \vee (\bar{A} \wedge B)$ with respect to B is $\Psi = A_1 \vee (\bar{A}_2 \wedge B)$, and is depicted in Fig. 4b. Then:*

$$P_\Phi = a + b - ab \text{ and } P_\Psi = a_1 + b - a_1b - a_2b + a_1a_2b$$

For $\mathbf{S} = \{A\}$, $\Delta P_\Phi(B, \mathbf{S}) \neq 0$. If (F, f) the folding of Ψ into Φ , then $[\mathbf{S}]_t = \{A_1, A_2\}$, and $\Delta P_\Psi(A_1) \neq 0$, so B is not a functional cause.

Hence, the definition of functional causes effectively captures the difference between the two networks for the two thrower example (Fig. 2a, 2b) while *only focusing on the input nodes*, as opposed to the HP definition that requires the inspection of the values of all the dependent nodes under all assignments. In the case of the simple network, $P_\Phi = P_\Psi$ and for $\mathbf{S} = \{A\}$, B can be shown to be a cause. However, in the more complicated network, the potential function of the dissociation expression gives priority to A 's throw and determines that B is not a cause of the bottle breaking.

If the causal network is a tree, then the causal formula is itself a dissociation expression with potential P_Φ . Then, (FC2) simplifies to: (a) $\Delta P_\Phi(\mathbf{S}, X_i) \neq 0$ and (b) $\forall \mathbf{S}' \subseteq \mathbf{S} : \Delta P_\Phi(\mathbf{S}') = 0$. Causal networks which are trees form an important category of causality

problems as they model many practical cases of database queries, and they are characterized by desirable properties, as we show in Sect. 3.3.

Responsibility. Responsibility is a measure for degree of causality, first introduced by Chockler and Halpern [6]. We redefine it here for functional causes.

Definition 5 (Responsibility). *Responsibility ρ of a causal variable X_i is defined as $\frac{1}{|S|+1}$ where S the minimum support for which X_i is a functional cause of an effect under consideration. $\rho := 0$ if X_i is not a cause.*

Responsibility ranges between 0 and 1. Non-zero responsibility ($\rho > 0$) means that the variable is a functional cause, $\rho = 1$ means it is also a counterfactual cause.

3 Formal Properties

Functional causality encodes the *semantics of causal structures* with the help of potential functions which are dependent only on the input variables. Functional causes are a refined notion of actual causes. Even though the definition of AC does not exclude dependent variables, functional causality does not consider them as possible causes, as their value is fully determined from the input variables. The relationship of functional causality of input variables to actual and counterfactual causality is demonstrated in the following theorem.

Theorem 2. *Every $X = x^0$ that is a counterfactual cause is also a functional cause, and every $X = x^0$ that is a functional cause is also an actual cause.*

Actual causes are more permissive than functional causes, as indicated by the limitations mentioned in Sect. 2. The issue is analyzed extensively in [20]. In this section we demonstrate that *functional causality* provides a more powerful and robust way to reason about causes than *actual causality*. In addition, we give a transitivity result and use it to derive complexity results for certain types of causal network structures.

3.1 Causal Network Expansion

Functional, as well as actual causes, rely on the causal network to model a given problem. The two different models of the thrower example displayed in Fig. 2(a,b) demonstrate that changes in the network structure can help model priorities of events, which in turn can redefine causality of variables.

In Fig. 2b, B is removed as a cause by the addition of an intermediate node in the causal network structure that models the preemption of the effect by node A (Alice's rock is the one that breaks the bottle). This change is also visible in the causal Boolean formula, which is transformed from $\Phi = A \vee B$ to $\Phi_1 = A \vee (\bar{A} \wedge B)$. As we know from Boolean algebra, the two formulas are equivalent as they have the same truth tables. However, they are not *causally equivalent*, as they yield different causality results. Therefore, the grammatical form of the Boolean expression is important in determining causality, and the functional definition captures that through dissociation expressions. It is important to understand how changes in the causal network affect causality, and whether we can state meaningful properties for those changes.

We define *causal network expansion* in a standard way by the addition of nodes and/or edges to the causal structure. A network CN_e with formula Φ_e is a *node expansion* (respectively *edge expansion*) of CN with formula Φ if it can be created by the addition of a node (respectively edge) to CN , while $\Phi_e \equiv \Phi$. CN_e is a *single-step expansion* if it is either a node or an edge expansion of CN .

Definition 6 (Expansion). A causal network CN_e is an expansion of network CN iff \exists set $\{CN_1, CN_2, \dots, CN_k\}$ with $CN_1 = CN$ and $CN_k = CN_e$, such that CN_{i+1} is a single step expansion of CN_i , $\forall i \in [1, k]$.

Networks represented by the formulas $\Phi_1 = A \vee (\bar{A} \wedge B)$ and $\Phi_2 = (A \wedge \bar{B}) \vee B$ are both expansions of $\Phi = A \vee B$, but note that Φ_1 and Φ_2 are not expansions of one another. As shown by the thrower example, network expansion can remove causes. As the following theorem states, it can only remove, not add causes.

Theorem 3. If CN_e with formula Φ_e is an expansion of CN with formula Φ and $X_i = x_i^0$ is a cause in ϕ_e then $X_i = x_i^0$ is also a cause in ϕ .

Specifically in the case where no negation of literals is allowed, changes to the structure do not affect the causality result:

Theorem 4. If CN_e with formula Φ_e is an expansion of CN with formula Φ that does not contain negated variables then ϕ and ϕ_e have the same causes.

The properties of formula expansion are important, as they prevent unpredictability due to causal structure changes. Note that the Halpern and Pearl definition does not handle formula expansion as gracefully. Figure 2 demonstrates with an example that the HP definition allows introducing new causes with expansion. $A = 1$ is not a cause in the simple network of Fig. 2c but becomes causal after adding node Y_2 in Fig. 2d. Therefore, network expansion is *unpredictable for actual causes*, as there are examples where it can both remove (Fig. 2b) or introduce new causes (Fig. 2d). This is a strong point for our definition, as causality is tied to the network structure, and erratic behavior due to minor structure changes, as is the case in this example, is troubling.

3.2 Functional causes and transitivity

Functional causality only considers *input nodes* in the causal network as permissible causes for events. Under this premise, the notion of *transitivity of causality* is not well-defined, since dependent variables are never considered permissible causes of events in their descendants. In order to ask the question of transitivity, we allow a dependent variable Y_1 to become a possible cause in a *modified causal model* M' with Y_1 as additional input variable. We achieve this with the help of an external intervention $[Y_1 \leftarrow y_1^0]$, setting the variable to its actual value y_1^0 . The new model is then $M' = (N, \mathcal{F}')$ with modified structural equations $\mathcal{F}' = \mathcal{F} \setminus \{F_{Y_1}\} \cup \{F'_{Y_1}\}$, where $F'_{Y_1} = y_1^0$, and hence new input variables $\mathbf{X}' = (\mathbf{X}, Y_1)$ with original assignment $\mathbf{x}'^0 = (\mathbf{x}^0, y_1^0)$.

We can now ask the question of transitivity as follows: Assume that an assignment $X = x^0$ is a cause of $Y_1 = y_1^0$ in a causal model M . Further assume that $Y_1 = y_1^0$ is a cause of $Y_2 = y_2^0$ in the modified network $[Y_1 \leftarrow y_1^0]$. Is then $X = x^0$ a cause of $Y_2 = y_2^0$ in the original network M ? In agreement with recent prevalent (yet not undisputed) opinion in causality literature [14,22], functional causality is *not* transitive, in general.

Intransitivity of causality is not uncontroversial [17] and humans generally feel a strong intuition that causality *should be* transitive. It turns out that functional causality is actually transitive in an important type of network structure that relates to this intuition: Transitivity holds if there is no causal connection between the original cause (X) and the effect (Y_2) except through the intermediate node (Y_1). This property allows us to deduce a lower complexity for determining causality in restricted settings in Sect. 3.3.

Definition 7 (Markovian). A node N is Markovian in a causal network CN iff there is no path from any ancestor of N to any descendent of N that does not pass through N .

Proposition 5 (Markovian transitivity). Given a causal model M in which $X = x^0$ is a cause of $Y_1 = y_1^0$ with responsibility ρ_1 , and Y_1 is Markovian. Further assume that $Y_1 = y_1^0$ is a cause of $Y_2 = y_2^0$ with responsibility ρ_2 in the modified causal model $[Y_1 \leftarrow y_1^0]$. Then $X = x^0$ is a cause of $Y_2 = y_2^0$ in M with responsibility $\rho = (\rho_1^{-1} + \rho_2^{-1} - 1)^{-1}$.

3.3 Complexity

Analogous to Eiter and Lukasiewicz's result that determining actual causes for Boolean variables is NP-hard [9], determining functional causality is also NP-hard, in general.

Theorem 6 (Hardness). Given a Boolean formula Φ on causal network CN and assignment x^0 of the input variables, determining whether $X_i = x_i^0$ is a cause of $\phi = \Phi(x^0)$ is NP-hard.

Even though determining functional causality is hard, there are important cases that can be solved in polynomial time.

If the causal network is a tree, then the dissociation network is the same as the causal network and there is a single potential function. Determining causality on a tree can be simplified, as a result of the Markovian transitivity property (Proposition 5) and the fact that all nodes in a tree are Markovian.

Lemma 7 (Causality in Trees). If $X_i = x_i^0$ is a cause of the output node Y in a tree causal network, and $p = \{X, Y_1, Y_2, \dots, Y\}$ the unique path from X to Y , then every node in p is a functional cause of all of its descendants in p . Consequently, X is a cause of all $Y_i \in p$.

Following from Lemma 7, causality in cases of tree-shaped causal structures with bounded arity (number of parents per node) is decidable in polynomial time.

Theorem 8 (Trees with arity $\leq k$). Given a tree-shaped causal network with formula Φ and bounded arity and actual assignment x^0 of the input variables, determining whether $X_i = x_i^0$ is a cause of $\phi = \Phi(x^0)$ is in P.

An even better result is given by Theorem 9, that covers the case of causal structures where the function at every node is a primitive boolean operator (AND, OR, NOT), without any restrictions on the arity.

Theorem 9 (Trees with Primitive Operators). Given a tree causal network with formula Φ where the function of every node is a primitive boolean operator, i.e. AND, OR, NOT, and assignment x^0 of the input variables, determining whether $X_i = x_i^0$ is a cause of $\phi = \Phi(x^0)$ is in P.

As demonstrated by Olteanu and Huang in [25], the lineage expressions of safe queries do not have repeated tuples. Lineage expressions for conjunctive queries with no repeated tuples correspond to causal networks that are trees. Following directly from Theorem 9, we get complexity results for safe queries.

Corollary 10 (Causes of Safe Queries). Determining the functional causes of safe queries can be done in polynomial time.

N(ews feeds)			F(iltered feed)	
<i>nid</i>	<i>story</i>	<i>source</i>	<i>story</i>	
1	... doing utmost to prevent more floods ...	AsiaOne	... doing utmost to prevent more floods ...	
2	... economic downturn affected ...	NYTimes	... economic downturn affected ...	
3	... with sequences shot in Singapore ...	AsiaOne	... sequences shot in Singapore ...	
4	... BP's chief executive apologizes ...	NYTimes	... BP's chief executive apologizes ...	
5	... apology for oil disaster ...	AsiaOne	... VLDDB held in Singapore ...	
6	... VLDDB held in Singapore ...	NYTimes	... discussed in a recent talk the ...	
7	... discussed in a recent talk the ...	NYTimes	... European stimulus measures ...	
8	... European stimulus measures ...	NYTimes		
9	... Singapore welcomes VLDDB ...	AsiaOne		

Fig. 5: News feed with aggregated data from different sources (left), and filtered feed (right).

In these tractable cases, due to the transitivity property, responsibility can also be computed in polynomial time, using the formula of Proposition 5.

Another important category of tractable networks are those that correspond to DNF and CNF formulas with no negated literals. This category covers important cases of join queries in a database context.

Theorem 11 (Positive DNF/CNF). *Given a positive DNF (or CNF) formula Φ and assignment \mathbf{x}^0 of the input variables, determining whether $X_i = x_i^0$ is a cause of $\phi = \Phi(\mathbf{x}^0)$ is in PTIME.*

4 Explaining Query Results

In this section, we show how causality can be applied to address examples from the database literature, like provenance and “Why Not?” queries, as well as examples showcasing causality of aggregates. We also demonstrate how our causality framework can model different types of elements that can be considered contributory to a query result, like *query operations* instead of tuples.

4.1 WHY SO? and WHY NO?

We revisit our motivating example (Example 1), but introduce a slight variation that aggregates data from different news sources to demonstrate how functional causality can be used to answer WHY SO? and WHY NO? questions.

Example 4 (News aggregator). *A user has access to the News feed relation N , depicted in Fig. 5. N contains news articles from two different sources, the NYTimes and the Singapore Press holdings portal, Asia One. The user, who resides in Singapore, likes to read more local news from Asia One, but she prefers the NY Times with regards to global interest news. Hence, she does not want to read on topics from Asia One that are also covered by the NY Times. Her filtered feed is constructed by the query:*

```

select  N.story
from    N
where   N.source='NYTimes' or
        not exists ( select  *
                      from    N as N1
                      where   topic(N1.story)=topic(N.story)
                      and N1.source='NYTimes' )

```


where $\text{topic}()$ is a topic extractor modeled as a user-defined function. The user's filtered feed will contain stories from NY Times, and only those stories from Asia One that NY Times does not cover. Simply, if S_{NY} is an article in NY Times covering a topic, and S_A an article in Asia One about the same topic, whether the user will see this topic in her feed or not follows a causal model similar to that of Fig. 4a, with boolean formula $\Phi = S_{NY} \vee (\bar{S}_{NY} \wedge S_A)$. The topic appears in F if it appears in either NY Times or Asia One, but the first gets priority.

When asking what is the cause of getting an article on the Orchard Rd floods, the user gets tuple 1 from relation N , as it is counterfactual. When asking what is the cause of seeing an article on VLDB, she gets the NY Times article (tuple 6), even though Asia One also had a story about it (tuple 9). The analysis is equivalent to the rock thrower example.

The framework can be used in a similar fashion to respond to "Why No?" questions. Assume tuple $t_{10} = (10, '... immigration officials arrest 300...', NYTimes)$, which was present in yesterday's news feed, but was since then removed. Tuple t_{10} is a functional cause to the WHY NO? question: "Why do I not see news on immigration", as it is counterfactual. Its removal from the feed caused the absence of immigration topics in the user's filtered view.

4.2 Aggregates

We next show how functional causality can be applied to determine causes and responsibility for aggregates. We focus here only on positive integers and give complexity results for WHY SO? and WHY NO? for WHY IS $\text{SUM} \geq c$? and WHY IS $\text{SUM} \not\geq c$?. In the following we denote with $\Omega \in \{\text{SUM}, \text{MAX}, \text{AVG}, \text{MIN}, \text{COUNT}\}$ an aggregate function evaluated over a multiset of values ($\Omega(V)$), \mathbf{X} is a vector of boolean values representing presence of absence of tuples, and op is an operator from the set $\{\geq, >, \leq, <, =, \neq\}$.

Definition 8 (Why so? and Why no?). Let $\omega^0 = \Omega(\mathbf{x}^0)$ be the value of an aggregate function for current assignment \mathbf{x}^0 . The question of WHY SO? (respectively, WHY NO?) for a condition $\omega^0 \text{ op } c$ that is true (respectively, false) under the current assignment corresponds to the question of which set of tuples $\{t_i\}$ from the tuple universe with original assignment $x_i^0 = 1$ (respectively, 0) is a cause of the event $\phi = (\omega^0 \text{ op } c = \text{true})$ (respectively, false) with responsibility ρ_i .

Example 5 (Sum example). Consider a tuple universe $T = [(10), (20), (30), (50), (100)]$ and a view $R(A)$ with the subset of tuples $\mathbf{R} = \{(20), (30), (100)\}$. Now consider the query `select SUM(R.A) from R` executed over the view R which returns 150. In our notation, this is represented with a vector $\mathbf{V} = [10, 20, 30, 50, 100]$, current assignment $\mathbf{x}^0 = [0, 1, 1, 0, 1]$, and $\text{SUM}(\mathbf{x}^0) = 150$ (see Fig. 6a).

WHY $\text{SUM} \geq c$?: t_3 is a cause of $\text{SUM}(\mathbf{x}^0) \geq 30$ with responsibility $\frac{1}{2}$. $\text{FC2}(a)$: $\text{SUM}(\mathbf{x}^1) \not\geq 30$ for $\mathbf{x}^1 = [0, 1, \underline{0}, 0, \underline{0}]$. $\text{FC2}(b)$: $\text{SUM}(\mathbf{x}^{1*}) \geq 30$ for every assignment \mathbf{x}^{1*} with $x_3^{1*} = 1$ and any subset of $\{x_5^1 = 0\}$ inverted to its original assignment. In contrast, t_2 is not a cause: While $\text{FC2}(a)$ holds for $\mathbf{x}^1 = [0, \underline{0}, \underline{0}, \underline{1}, \underline{0}]$ with $\text{SUM}(\mathbf{x}^1) \not\geq 30$ (and then t_2 would be counterfactual), $\text{FC2}(b)$ is not fulfilled for $\mathbf{x}^{1*} = [0, 1, \underline{0}, 0, \underline{0}]$.

WHY $\text{SUM} \not\geq c$?: t_4 is a cause of $(\text{SUM}(\mathbf{x}^0) \geq 180) = \text{false}$, as both x_4 and the condition are false under current assignment, but would hold for $\mathbf{x}^1 = [0, 1, 1, \underline{1}, 1]$.

			(b)										
			WHY SUM(x^0) \geq					WHY SUM(x^0) $\not\geq$					
			t_i	x_i^0	20	30	40	60	130	160	180	210	220
			t_1	0	—	—	—	—	—	1	—	$\frac{1}{2}$	—
			t_2	1	$\frac{1}{3}$	—	$\frac{1}{2}$	—	—	—	—	—	—
			t_3	1	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{1}{2}$	—	1	—	—	—	—
			t_4	0	—	—	—	—	—	1	1	$\frac{1}{2}$	—
			t_5	1	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{1}{2}$	1	1	—	—	—	—

Fig. 6: Sum example. (a): Relation R with tuples from tuple domain T . (b): Responsibility ρ_i of t_i for WHY SO? ($\text{SUM}(x^0) \geq c$) and WHY NO? ($\text{SUM}(x^0) \not\geq c$).

Figure 6b shows responsibility for different values of constant c in Example 5 and illustrates that responsibility for SUM is not monotone. In order to compute responsibility for a tuple t_i , one must find the smallest set of tuples that, when inverted (i.e. either inserted or deleted) make tuple t_i counterfactual for the condition. Determining the causes of an aggregate is in general NP-complete. We refer the reader to our technical report [20] for further theoretical analysis of aggregate causality and more examples.

4.3 Causes beyond tuples

Provenance and non-answers commonly focus on tuples as discrete units that have contribution to a query result. Our causality framework is not restricted to tuples, but can model any element that could be considered contributory to a result. To showcase this flexibility, we pick an example from Chapman and Jagadish [4] that models *operations* in workflows as possible answers to “Why not?” questions.

Example 6 (Book Shopper [4], Ex. 1). *A shopper knows that all “window display books” at Ye Olde Booke Shoppe are around \$20, and wishes to make a cheap purchase. She issues the query: Show me all window books. Suppose the result from this query is (Euripides, “Medea”). Why is (Hrotsvit, “Basilius”) not in the result set? Is it not a book in the book store? Does it cost more than \$20? Is there a bug in the query-database interface such that the query was not correctly translated?*

Chapman and Jagadish consider a discrete component of a workflow, called *manipulation*, as an explanation of a “Why not?” query. The workflow describing the query of the example is shown in Fig. 7b. Roughly, a manipulation is considered *picky* for a non-result if it prunes the tuple. For example, manipulation 1 of Fig. 7b is picky for “Odyssey”, as it costs more than \$20. Equivalently, a manipulation is *frontier picky* for a set of non-results, if it is the last in the workflow to reject tuples from the set. In this framework, the cause of a non-answer will be a frontier picky manipulation.

In Example 6, tuple $t = (\text{Hrotsvit}, \text{“Basilius”})$ passes the price test, but is cut by manipulation 2 as it doesn’t satisfy the seasonal criteria. The causal network representing this example is presented in Fig. 7c. Input nodes model the events: M_1 : manipulation 1 is not potentially picky with respect to t , and M_2 : manipulation 2 is not potentially picky with respect to t . At the end, the tuple appears only if neither manipulation is picky: $M_1 \wedge M_2$. Intermediate node Y_1 encodes the precedence of the manipulations in the workflow. A tuple will be stopped at point Y_1 of the workflow if M_2 is picky but M_1 was not: $M_1 \wedge \bar{M}_2$. It will pass this point if the opposite holds, so $Y_1 = \bar{M}_1 \wedge \bar{M}_2 = \bar{M}_1 \vee M_2$, and $Y = M_1 \wedge Y_1$.

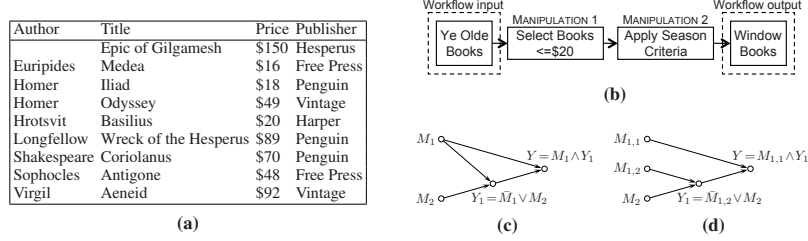


Fig. 7: (a) Books in “Ye Olde Booke Shoppe” [4]. (b) Variation of the query workflow from [4]. The causal network of Example 6 (c), and its DN with respect to M_2 (d).

Applying the FC framework for $M_1 = 1$ (M_1 is not picky), and $M_2 = 0$ (M_2 is picky), correctly yields that M_2 is the only cause: $S = \emptyset$, $\Delta I_\Phi(M_2) \neq 0$. If both manipulations were potentially picky ($M_1 = 0$ and $M_2 = 0$), the FC definition again correctly picks M_1 as the only cause with support $S = \{M_2\}$ (even though M_2 is potentially picky, the tuple never gets to it), which agrees with the WHY NOT? framework that selects as explanation the last manipulation that rejected the tuple.

5 Related Work

Our work is mainly related and unifies ideas from three main areas: research on *causality*, *provenance*, and *missing query result explanations*.

Causality. Causality is an active research area mainly in logic and philosophy with their own dedicated workshops (see e.g. [1]). The most prevalent definitions of causality are based on the idea of *counterfactual causes*, i.e. causes are explained in terms of counterfactual conditionals of the form *If X had not occurred, Y would not have occurred*. This idea of counterfactual causality can be traced back to Hume [22]. The best known counterfactual analysis of causation in modern times is due to Lewis [16]. In a databases setting, Miklau and Suciu [23] define *critical tuples* as those which can become counterfactual under some value assignment of variables. Halpern and Pearl [12] (HP in short) define a variation they call *actual causality*. Roughly speaking, the idea is that X is a cause of Y if Y counterfactually depends on X under “some” *permissive contingency*, where “some” is elaborately defined. Later, Chockler and Halpern [6] define the degree of *responsibility* as a gradual way to assign causality. Eiter and Lukasiewicz [9] show that the problem of detecting whether $X = x^0$ is an actual cause of an event is Σ_2^P -complete for general acyclic models and NP-complete for binary acyclic models. They also give an alleged proof showing that actual causality is always reducible to primitive events. However, Halpern [11] later gives an example for non-primitive actual causes, showing this proof to ignore some cases under the original definition. Chockler et al. [7] later apply causality and responsibility to binary Boolean networks, giving a modified definition of cause.

A general overview of various applications of causality in a database context is given in [19]. The complexity of computing causality and responsibility is studied in [21] for the case of conjunctive queries, leading to a strong dichotomy result.

Provenance. Approaches for defining data provenance can be mainly divided into three categories: *how*, *why*, and *where* provenance ([3,5,8,10]). In particular for the “why so” case, we observe a close connection between provenance and causality, where it is often the case that tuples in the provenance for the result of a positive query result are causes.

While none of the work on provenance mentions or makes direct connections to causality, those connections can be found. The work by Buneman et al. [3] makes a distinction between *why* and *where* provenance that can be connected to causality as follows: *why provenance* returns all tuples that can be considered causes for a particular result, and *where provenance* returns attributes along a particular causal path. Green et al. [10] present a generalization for all types of provenance as semirings; finding functional causes in a Boolean tree, if taken in a provenance context, yields degree-one polynomials for provenance semirings. View data lineage, as presented by Cui et al. [8] also addresses aggregates but lacks a notion of graded contribution.

In contrast, our approach can rank tuples according to their *responsibility*, hence our approach allows to determine a gradual contribution with counterfactual tuples ranked first. Also, in contrast to our paper, most of the work on provenance has little or no connection to the philosophical groundwork on causality. We take this work and significantly adapt it so that it can be applied to databases.

Missing query results. Very recent work has focused on the question “why no”, i.e. why is a certain tuple *not* in the result set? The work by Huang et al. [15] presents provenance for potential answers and never answers. In the case that no insertions or modifications can yield the desired result - usually for privacy or security reasons - the system declares that particular tuple a never answer. Both Huang’s work and Artemis [13] handle potential answers by providing tuple insertions or modifications that would yield the missing tuples. Alternatively, Chapman and Jagadish [4] focus on which *manipulation* in the query plan eliminated a specific tuple, while Tran and Chan [26] show how the query can be modified in order to include missing results in the answer. Lim et al. [18] adopt a third, explanation-based, approach. This approach aims to answer questions such as *why*, *why not*, *how to*, and *what if* for context-aware applications, but does not address a database setting.

Our work, unifies the above approaches in the sense that we model both, *tuples* or *manipulations*, as possible causes for missing query answers. Also, our approach unifies the problem of explaining missing query answers (*why* is a tuple not in the query result) with work on provenance (*why* is a tuple in the query result).

Other. Minsky and Papert initiated the study of the computational properties of Boolean functions using their representation by polynomials and call this the *arithmetic* instead of the logical form [24, p.27]. This method was later successfully used in complexity theory and became known as *arithmetization* [2].

6 Conclusions and Future Work

In this paper, we defined *functional causes*, a rigorous and extensible definition of causality encoding the semantics of *causal structures* with the help of powerful *potential functions*. Through theoretical analysis of its properties, we demonstrated that our definition provides a more powerful and robust way to reason about causes than other established notions of causality. Albeit NP-hard in the general case, common categories of causal networks that correspond to interesting database examples (e.g. safe queries) prove to be tractable. We presented several database examples that portrayed the applicability of our framework in the context of provenance, explanation of non-answers, as well as aggregates. We demonstrated how to determine causes of query results for SUM and COUNT aggregates, and how these can be ranked according to the causality metric of *responsibility*.

Providing support for causal queries allows users to better understand the reasons behind their observations, and is an important tool for identifying potential errors in uncertain or untrusted data. Overall, with this work we establish the theoretical foundations of causality theory in the database context, which we view as a unified framework that deals with query result explanations.

Acknowledgements. This work was partially supported by NSF grants IIS-0911036, IIS-0915054, and IIS-0713576. We would like to thank Christoph Koch for valuable insights, and Chris Ré for helpful discussions in early stages of this project.

References

1. International multidisciplinary workshop on causality. IRIT, Toulouse, June 2009.
2. L. Babai and L. Fortnow. Arithmetization: A new method in structural complexity theory. *Computational Complexity*, 1:41–66, 1991.
3. P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *ICDT*, 2001.
4. A. Chapman and H. V. Jagadish. Why not? In *SIGMOD*, 2009.
5. J. Cheney, L. Chiticariu, and W. C. Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4):379–474, 2009.
6. H. Chockler and J. Y. Halpern. Responsibility and blame: A structural-model approach. *J. Artif. Intell. Res. (JAIR)*, 22:93–115, 2004.
7. H. Chockler, J. Y. Halpern, and O. Kupferman. What causes a system to satisfy a specification? *ACM Trans. Comput. Log.*, 9(3), 2008.
8. Y. Cui, J. Widom, and J. L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Trans. Database Syst.*, 25(2):179–227, 2000.
9. T. Eiter and T. Lukasiewicz. Complexity results for structure-based causality. *Artif. Intell.*, 142(1):53–89, 2002. (Conference version in *IJCAI*, 2002).
10. T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, 2007.
11. J. Y. Halpern. Defaults and normality in causal structures. In *KR*, 2008.
12. J. Y. Halpern and J. Pearl. Causes and explanations: A structural-model approach. Part I: Causes. *Brit. J. Phil. Sci.*, 56:843–887, 2005. (Conference version in *UAI*, 2001).
13. M. Herschel, M. A. Hernández, and W. C. Tan. Artemis: A system for analyzing missing answers. *PVLDB*, 2(2):1550–1553, 2009.
14. C. Hitchcock. The intransitivity of causation revealed in equations and graphs. *The Journal of Philosophy*, 98(6):273–299, 2001.
15. J. Huang, T. Chen, A. Doan, and J. F. Naughton. On the provenance of non-answers to queries over extracted data. *PVLDB*, 1(1):736–747, 2008.
16. D. Lewis. Causation. *The Journal of Philosophy*, 70(17):556–567, 1973.
17. D. Lewis. Causation as influence. *The Journal of Philosophy*, 97(4):182–197, 2000.
18. B. Y. Lim, A. K. Dey, and D. Avrahami. Why and why not explanations improve the intelligibility of context-aware intelligent systems. In *CHI*, 2009.
19. A. Meliou, W. Gatterbauer, J. Halpern, C. Koch, K. F. Moore, and D. Suciu. Causality in databases. *IEEE Data Engineering Bulletin special issue on Provenance*, Sept. 2010. (to appear, see <http://db.cs.washington.edu/causality/>).
20. A. Meliou, W. Gatterbauer, K. F. Moore, and D. Suciu. Why so? or why no? functional causality for explaining query answers. *CoRR*, abs/0912.5340, 2009. (see <http://db.cs.washington.edu/causality/>).
21. A. Meliou, W. Gatterbauer, K. F. Moore, and D. Suciu. The causality and responsibility of query answers and non-answers. In *PVLDB*, 2011. (to appear, see <http://db.cs.washington.edu/causality/>).
22. P. Menzies. Counterfactual theories of causation. Stanford Encyclopedia of Philosophy, 2008.
23. G. Miklau and D. Suciu. A formal analysis of information disclosure in data exchange. In *SIGMOD*, 2004.
24. M. L. Minsky and S. Papert. *Perceptrons - expanded edition: An introduction to computational geometry*. MIT Press, 1987.
25. D. Olteanu and J. Huang. Secondary-storage confidence computation for conjunctive queries with inequalities. In *SIGMOD*, 2009.
26. Q. T. Tran and C.-Y. Chan. How to conquer why-not questions. In *SIGMOD*, 2010.

Extending Magic Sets Technique to Deductive Databases with Uncertainty

Qiong Huang and Nematollah Shiri

Department of Computer Science and Software Engineering
Concordia University, Montreal, Canada

Abstract. The magic sets (MS) rewriting technique was proposed to optimize the efficiency of bottom-up evaluation of datalog programs. This technique has been extended to logic programs with uncertainty, but its application is restricted to frameworks with set based semantics such as fuzzy logic. We show that for a more general case of multi-set semantics, a “straightforward” extension of MS technique could lead to incorrect computation. In this work, we propose an extension of the generalized magic sets technique to deductive databases with uncertainty which use multi-sets as the semantics structure, and establish its correctness. We have developed a testing platform and conducted numerous experiments to evaluate the performance of the proposed technique. The experimental results indicate that different programs enjoy different efficiency gain, depending on the *potential facts ratio*, which intuitively measures the capacity to improve efficiency. We observed that when this ratio ranges from 1% to 20%, the proposed optimization results in 1 to 550 times speed-up compared to evaluation of the original program. Our results also indicate that semi-naive combined with predicate partitioning technique yields the best performance.

1 Introduction

Uncertainty management has been a challenging issue in database and artificial intelligence research for a long time [1]. Standard logic programming and deductive databases, for their declarative and modularity advantages and their powerful top-down and bottom-up query processing techniques, have attracted the attention of many researchers for incorporating uncertainty. This has resulted in numerous frameworks for modeling and reasoning with uncertainty obtained by extending the standard case. On the basis in which uncertainty is associated with facts and rules, these frameworks are classified [6] into *Annotated Based* (AB) and *Implication Based* (IB). In the IB approach, the implication in each rule in the program is associated with a certainty value. The *parametric framework* (PF) proposed in [6] unifies and/or generalizes the class of IB frameworks.

As in the standard database, there are two sources of inefficiency in a bottom-up evaluation of logic programs with uncertainty: (1) repeated applications of rules which do not yield any fact with improved certainty; and (2) generation of atoms which are not related or contribute to the goal query. In the context

of PF, semi-naive (SN) [6] and SN with predicate partitioning methods (SNP) [8] have been developed to address the first problem. For the second problem, the magic sets (MS) techniques have been proposed for standard Datalog which takes into account the query structure and its bound arguments, if any, and rewrite the given program into a form which is more focused when computing the answers to the query [2]. It has been shown in the standard case that the rewritten programs when evaluated by SN method takes no more time than when the original program is evaluated in top-down [5].

Top-down query processing method for logic programs with uncertainty has been proposed in [13], which generates a large system of equations. While this is an interesting approach, a bottom-up method is more preferred for several reasons. For instance, termination could be a problem in top-down evaluation, which needs additional bookkeeping such as tabling or memoing done in XSB [12] to avoid useless calls in evaluating left-recursive rules. Also top-down requires unification, while bottom-up algorithms use term-matching for joins which is a one-way unification and hence easier. Existing optimization techniques such as indexing may be applied for joins of massive relations with ease.

Magic sets technique combines the advantages of top-down and bottom-up approaches. The basic idea of magic sets is that a bottom-up evaluation should be restricted to those facts that are “potentially relevant” to a given query. This is done by introduction of magic predicates and rules which ensure a rule is not fired unless magic predicates hold the necessary terms. There are extensions of magic sets technique to IB frameworks with uncertainty, but the works are restricted to either fuzzy logic or are set based, such as [10] in which the termination is guaranteed and hence the uncertainties of answers are not affected by evaluation re-ordering [11]. For multi-set based semantics, extending magic sets is more challenging.

Example 1. A p-program and its Magic Sets Rewritten Program

Original Program P :

$$\begin{aligned} p(X, Y) &\stackrel{0.5}{\leftarrow} a(X, Y); \langle ind, *, * \rangle. \\ p(X, Y) &\stackrel{0.5}{\leftarrow} p(Y, Z), p(Y, X); \langle ind, *, * \rangle. \\ D &= \{a(1, 2) : 0.5, a(1, 1) : 0.5, a(2, 1) : 0.5\}. \\ &?p(1, Y). \end{aligned}$$

Generalized Magic Sets Rewritten Program P^m :

$$\begin{aligned} p^{bf}(X, Y) &\stackrel{0.5}{\leftarrow} m\text{-}p^{bf}(X), a(X, Y); \langle ind, *, * \rangle. \\ m\text{-}p^{fb}(X) &\stackrel{1}{\leftarrow} m\text{-}p^{bf}(X); \langle max, max, max \rangle. \\ m\text{-}p^{bf}(Y) &\stackrel{1}{\leftarrow} m\text{-}p^{bf}(X), p^{fb}(Y, X); \langle max, max, max \rangle. \\ p^{bf}(X, Y) &\stackrel{0.5}{\leftarrow} m\text{-}p^{bf}(X), p^{fb}(Y, X), p^{bf}(Y, Z); \langle ind, *, * \rangle. \\ p^{fb}(X, Y) &\stackrel{0.5}{\leftarrow} m\text{-}p^{fb}(Y), a(X, Y); \langle ind, *, * \rangle. \\ m\text{-}p^{bf}(Y) &\stackrel{1}{\leftarrow} m\text{-}p^{fb}(Y); \langle max, max, max \rangle. \\ m\text{-}p^{bf}(Y) &\stackrel{1}{\leftarrow} m\text{-}p^{fb}(Y), p^{bf}(Y, Z); \langle max, max, max \rangle. \\ p^{fb}(X, Y) &\stackrel{0.5}{\leftarrow} m\text{-}p^{fb}(Y), p^{bf}(Y, Z), p^{bf}(Y, X); \langle ind, *, * \rangle. \\ D^m &= D \cup \{m\text{-}p^{bf}(1) : 1\}. \end{aligned}$$

Table 1. The Results of Evaluating the P-programs at Every Iteration i in Example 1

i	Original Program P	GMS Rewritten Program P^m
1	$p(2, 1) : 0.25, p(1, 2) : 0.25, p(1, 1) : 0.25$	$p^{bf}(1, 1) : 0.25, m_{\neg}p^{fb}(1) : 1, p^{bf}(1, 2) : 0.25$
2	$p(2, 1) : 0.29614258, p(1, 2) : 0.2734375$ $p(1, 1) : 0.29614258$	$p^{fb}(2, 1) : 0.29614258, p^{fb}(1, 1) : 0.29614258$
3	$p(2, 1) : 0.307269, p(1, 2) : 0.28288764$ $p(1, 1) : 0.31192228$	$p^{bf}(1, 1) : 0.304499, m_{\neg}p^{bf}(2) : 1$
4	$p(2, 1) : 0.31177515, p(1, 2) : 0.28540534$ $p(1, 1) : 0.31796566$	$p^{bf}(1, 1) : 0.31032723, p^{bf}(2, 1) : 0.25$ $p^{fb}(2, 1) : 0.30109218, p^{fb}(1, 1) : 0.31199324$ $m_{\neg}p^{fb}(2) : 1$
5	$p(2, 1) : 0.31319097, p(1, 2) : 0.2864514$ $p(1, 1) : 0.32022393$	$p^{bf}(1, 1) : 0.3141409, p^{bf}(1, 2) : 0.2782274$ $p^{fb}(2, 1) : 0.30162153, p^{fb}(1, 1) : 0.31380594$ $p^{fb}(1, 2) : 0.2734375$
...

Example 1 shows a p-program P in the PF (review of PF is provided in Section 2) and its magic sets rewritten program P^m . The term $\langle ind, *, * \rangle$ stands for disjunction function $ind(\alpha, \beta) = \alpha + \beta - \alpha \times \beta$, propagation function $*(\alpha, \beta) = \alpha \times \beta$, and conjunction function $*$. These functions are applied to compute the certainty values for the atoms at every iteration during the program evaluation. Table 1 shows the intermediate results of every atom whose associated certainty is improved at every iteration. As we see, there are certainty bias between P and P^m starting from the 3rd iteration which will affect more results of evaluating atoms as the evaluation continues. For instance, the first certainty improvement of $p(1, 1)$ in P is based on the derivations:

$$\begin{aligned}
p(1, 1) &\leftarrow a(1, 1); \\
p(1, 1) &\leftarrow p(1, 1), p(1, 1); \\
p(1, 1) &\leftarrow p(1, 2), p(1, 1);
\end{aligned}$$

However, the first improvement of $p^{bf}(1, 1)$ at 3rd iteration is based on:

$$\begin{aligned}
p^{bf}(1, 1) &\leftarrow m_{\neg}p^{bf}(1), a(1, 1); \\
p^{bf}(1, 1) &\leftarrow m_{\neg}p^{bf}(1), p^{fb}(1, 1), p^{bf}(1, 1); \\
p^{bf}(1, 1) &\leftarrow m_{\neg}p^{bf}(1), p^{fb}(1, 1), p^{bf}(1, 2);
\end{aligned}$$

where $p^{fb}(1, 1)$ has been improved at iteration 2 in which the certainties associated with $p^{fb}(1, 1)$ and $p^{bf}(1, 1)$ are different, noting that they represent the same atom $p(1, 1)$. The difference will affect more and more atoms during the evaluation process of P^m . This explains why evaluations of P and P^m may yield different results in general.

In this paper, we extend the generalized magic sets technique [4] to PF which is multi-set based. This results in challenges to adjust evaluation order of rules in the magic sets rewritten program when the evaluation process may not terminate in theory. The rest of this paper is organized as follows. Next we review the PF as a background, together with a review of fixpoint evaluations of programs

in PF. Section 3 introduces the generalized magic sets technique for programs with uncertainty and establishes its correctness. In section 4, we present the experiments to the proposed technique.

2 The Parametric Framework: A Review

Numerous frameworks have been proposed to manage uncertainty in deductive databases (DDBs). They differ in several ways, including (i) the mathematical foundation of uncertainty they represent, (ii) the way in which uncertainty is associated with facts and rules in a program, and (iii) the way in which they manipulate uncertainty. On the basis of (ii), these frameworks are classified into *annotated-based* (AB) and *implication-based* (IB) [6]. While we focus on parametric framework which, we strongly believe, could also benefit from the AB approach, as it has been shown that the two approaches are as expressive when extended with certainty constraints [7]. The parametric framework is a generic IB framework which simulates the computation of every IB framework through the use of specific parameters.

2.1 Syntax and Notations

Definition 1. (*P-program*): A parametric program P , *P-program* for short, is a 5-tuple $\langle T, R, D, P, C \rangle$, whose components are defined as follows.

- T is a certainty domain, which we assume to be a complete lattice with the meet and join operators denoted by \otimes and \oplus , respectively. It is convention to use \perp and \top to denote the least and greatest element in the lattice, respectively.
- R is a finite set of rules of the form

$$A \stackrel{\alpha}{\leftarrow} B_1, \dots, B_n; \langle f_d, f_p, f_c \rangle$$

where A, B_1, \dots, B_n are atoms, with each $\alpha \in T - \{\perp\}$.

- D is a mapping which associates with every rule in P a disjunction function $f_d \in F_d$, where F_d is the set of all disjunction functions.
- P is a mapping that associates each rule in P with a propagation function $f_p \in F_p$, where F_p is the set of all propagation functions.
- C is a mapping that associates each rule in P with a conjunction function $f_c \in F_c$, where F_c is the set of all conjunction functions.

For consistency, we require that rules with the same head predicate are associated with the same disjunction function. We refer to the collection $F_d \cup F_p \cup F_c$ as the combination functions. To differ from the set notation, we use $\{\!\{ \dots \}\!\}$ to denote a multi-set M . In our context, each element X in M is of the form $A : \alpha$, where A is an atom and $\alpha \in T$. We use \emptyset to denote the empty multi-set. A set is a special case of multi-set with 0 or 1 as the multiplicity of its elements.

2.2 Combination Functions

Combination functions allowed in PF should satisfy certain properties as postulate provided after the following properties.

1. Monotonicity: $f(\alpha_1, \alpha_2) \preceq f(\beta_1, \beta_2)$, if $\alpha_i \preceq \beta_i$, for $i \in \{1, 2\}$ and $\alpha_i, \beta_i \in T$.
2. Continuity: f is continuous w.r.t its arguments.
3. Bounded-Above: $f(\alpha_1, \alpha_2) \preceq \alpha_i$, for $i \in \{1, 2\}$.
4. Bounded-Below: $f(\alpha_1, \alpha_2) \succeq \alpha_i$, for $i \in \{1, 2\}$.
5. Commutativity: $f(\alpha, \beta) = f(\beta, \alpha)$, for $\forall \alpha, \beta \in T$.
6. Associativity: $f(\alpha, f(\beta, \gamma)) = f(f(\alpha, \beta), \gamma)$, $\forall \alpha, \beta, \gamma \in T$.
7. $f(\{\alpha\}) = \alpha$, $\forall \alpha \in T$.
8. $f(\emptyset) = \perp$.
9. $f(\emptyset) = \top$.

Postulate: Each type of the combination functions in PF should satisfy certain properties, postulated as follows.

- Every conjunction function $f_c \in F_c$ satisfies properties 1, 2, 3, 5, 6, 7, 9.
- Every disjunction function $f_d \in F_d$ satisfies properties 1, 2, 4, 5, 6, 7, 8.
- Every propagation function $f_p \in F_p$ satisfies properties 1, 2, 3, 5.

Definition 2. *There are three categories of disjunction functions $f_d \in F_d$:*

1. Type 1: $f_d = \oplus$, i.e., f_d coincides with the join in the certainty lattice.
2. Type 2: $\oplus(\alpha, \beta) \prec f_d(\alpha, \beta) \prec \top$, $\forall \alpha, \beta \in T - \{\perp, \top\}$.
3. Type 3: $\oplus(\alpha, \beta) \prec f_d(\alpha, \beta) \preceq \top$, $\forall \alpha, \beta \in T - \{\perp, \top\}$.

Note that unlike type 2 functions, a disjunction function of type 3 may return the top value \top when none of its arguments is \top . As examples of practical disjunction functions, we have $\max(\alpha, \beta)$ which is of type 1 used for instance in fuzzy logic, while the probability independence function $\text{ind}(\alpha, \beta) = \alpha + \beta - \alpha\beta$ is of type 2. An example of type 3 disjunction function is $\min(1, \alpha + \beta)$ defined over the unit interval $[0, 1]$. Presence of disjunction functions of types 2 and 3 in logic programs with uncertainty pose challenges for query processing and optimization techniques, including the magic sets technique studied in this paper.

2.3 Fixpoint Theory

Fixpoint theory in standard deductive database is concerned with computing the least model of the logic program in a bottom-up fashion, starting with the facts and applying the rules repeatedly until no new fact is derived. This has been extended in [6] to compute the fixpoint semantics of p-programs in PF.

Definition 3. [6] *Let P be any p-program, and P^* be the Herbrand instantiation of P . Also let Υ_P be the set of all valuations of P . The immediate consequence operator T_p is a mapping from Υ_P to Υ_P , such that for every valuation $\nu \in \Upsilon_P$ and every ground atom $A \in B_p$, $T_p(\nu)(A) = f_d(X)$, where B_p is the Herbrand base of P , f_d is the disjunction function associated with $\pi(A)$, the predicate symbol of A , and X is a multi-set of certainties associated with A :*

$$X = \{f_p(\alpha, f_c(\{\nu(B_1), \dots, \nu(B_n)\})) \mid (A \stackrel{\alpha}{\leftarrow} B_1, \dots, B_n; \langle f_d, f_p, f_c \rangle) \in P^*\}$$

The least fixpoint semantics of P , $lfp(P)$, is defined based on T_p as follows:

$$T_p^k = \begin{cases} \nu_{\perp} & \text{if } k = 0 \\ T_p(T_p^{k-1}) & \text{if } k \text{ is a successor ordinal} \\ \oplus \{T_p^l \mid l < k\} & \text{if } k \text{ is a limit ordinal} \end{cases}$$

Note that $\nu_{\perp}(A) = \perp$, for all $A \in B_p$. It has been shown that T_p is monotone and continuous. Initially, T_p assigns every EDB fact A the certainty α because $f_p(\alpha, f_c(\emptyset)) = \alpha$. For any ground atom $A \in B_p$, if A does not unify with the head of any rule in P , then $T_p(\nu)(A) = \perp$.

2.4 Bottom-up Fixpoint Evaluation Algorithms

The basic bottom-up least fixpoint evaluation of p-programs is the naive evaluation (N) extended from the standard naive method by considering the presence of certainties as follows. Given a p-program P , every atom is initially assigned the least certainty value \perp . At each iteration i , we apply every possible rule. For every ground atom A derived from a rule in P , we use the conjunction function f_c and the propagation function f_p to combine certainties of the ground atoms in the body into a certainty. A multi-set of certainties of A is then generated; the disjunction function f_d associated combines the multi-set of certainties derived. The evaluation terminates when the certainty of no atom is improved. Fig. 1 shows the multi-set based naive algorithm for p-programs.

```

1: procedure Naive( $P$ )
2: input : a p-program  $P$ ;
3: output: the least fixpoint of  $P$ ;
4: begin
5:   forall  $A \in B_p$ 
6:      $\nu_0(A) := \perp$ ;
7:   end forall
8:    $new_1 := \{A \mid (A : \alpha) \in D\}; i := 1$ ;
9:   repeat
10:     $i := i + 1$ ;
11:    forall  $(A \stackrel{\alpha_r}{\leftarrow} B_1, \dots, B_n; \langle f_d, f_p, f_c \rangle) \in P^*$ ;
12:       $M_i(A) := \{f_p(\alpha_r, f_c(\{\nu_{i-1}(B_1), \dots, \nu_{i-1}(B_n)\}))\}$ ;
13:    end forall;
14:     $\nu_i(A) := f_d(M_i(A))$ ;
15:     $new_i := \{A \mid A \in B_p, \nu_i(A) \succ \nu_{i-1}(A)\}$ ;
16:  until  $new_i = \emptyset$ 
17:  return  $\nu_i$ 
18: end procedure

```

Fig. 1. A multi-set based naive algorithm for p-programs [6]

The naive evaluation in PF suffers from the same redundant derivations as the naive method does in Datalog: (1) a tuple certainty pair $(A : \alpha)$ derived at iteration i continues to be derived at every future iteration, and (2) a goal structure is looked at only when the fixpoint is reached. Part of the redundant computation is avoided by using semi-naive (SN) algorithm [6] which at every iteration, considers only those rules in which the certainties of at least a subgoal in the body was improved in the previous iteration. Semi-naive evaluation with predicate partitioning (SNP) [8] is a refinement and extension of SN in which at each iteration, the tuples found for each relation derived are divided into two parts: new tuples and old tuples. For the next iteration, a rule is evaluated if at least one of its subgoals appears in the new part. To solve the second problem, we use the idea of magic sets proposed as follows.

3 Generalized Magic Sets (GMS) with Uncertainty

In this section, we introduce GMS rewriting technique for p-programs. Our extended GMS rewriting technique includes two steps: *straightforward* GMS and caching magic tuples. We first take a "straightforward" approach which might generate error in a certain circumstance. We use the term *straightforward* to show that the proposed procedure inherits major steps of GMS in Datalog [4, 3], while we extend each step to deal with combination functions. The challenge is the correctness of the rewriting under the multi-set semantics. In general, when *type 2* functions are applied in a non-linear p-program, the evaluation might yield incorrect results. In this case, we claim that the second stage that precomputes magic tuples is necessary. Otherwise, the GMS rewriting process can be done in the *straightforward* GMS, as in the standard case.

3.1 Straightforward GMS

The generalized magic sets rewriting (GMS) works on the idea of sideways information passing (SIP) strategy [4]. Intuitively, SIP induces an order among the rules and the subgoals in each rule when evaluating a logic program. Magic sets act like filters that hold certain values for bound variables. The difference between *straightforward* GMS in PF and in Datalog is that in PF we also take into account the presence of uncertainty and combination functions. The major steps of *straightforward* GMS are as follows:

- Generation of adorned rules
- Generation of rules with magic predicates
- Rewriting of the adorned rules

An adornment for an n -ary predicate p is a string of length n on the alphabet $\{b, f\}$, where b stands for bound argument and f for free. A predicate p adorned with a binding pattern a , denoted as p^a , indicates the bound and free arguments of p . For example, p^{bf} indicates that p is a binary predicate, the first argument of which is bound and the second is free.

Given a p-program P and a query Q , the GMS rewriting process starts from Q , and generates a set of adorned rules which have the same IDB predicate as Q . More bound IDB predicates are then discovered, so we use one of these bound predicates to generate more adorned rules until all discovered bound IDB predicates are considered. For each adorned rule, we generate a set of magic rules, by adding the proper magic predicates. Finally, we create a tuple for the magic predicate related to Q , called the *seed*. this *straightforward* GMS technique generate a rewriting of P , denoted as P^m , which includes all magic rules, the rewritten adorned rules and the *seed*.

Example 2. Straightforward GMS of the p-program in Example 1

Adorned Rules of P :

$$\begin{aligned} p^{bf}(X, Y) &\stackrel{0.5}{\leftarrow} a(X, Y); \langle ind, *, * \rangle. \\ p^{bf}(X, Y) &\stackrel{0.5}{\leftarrow} p^{fb}(Y, X), p^{bf}(Y, Z); \langle ind, *, * \rangle. \\ p^{fb}(X, Y) &\stackrel{0.5}{\leftarrow} a(X, Y); \langle ind, *, * \rangle. \\ p^{fb}(X, Y) &\stackrel{0.5}{\leftarrow} p^{bf}(Y, Z), p^{bf}(Y, X); \langle ind, *, * \rangle. \end{aligned}$$

Straightforward GMS rewritten program P^m :

$$\begin{aligned} p^{bf}(X, Y) &\stackrel{0.5}{\leftarrow} m_{\neg p^{bf}}(X), a(X, Y); \langle ind, *, * \rangle. \\ m_{\neg p^{fb}}(X) &\stackrel{1}{\leftarrow} m_{\neg p^{bf}}(X); \langle max, max, max \rangle. \\ m_{\neg p^{bf}}(Y) &\stackrel{1}{\leftarrow} m_{\neg p^{bf}}(X), p^{fb}(Y, X); \langle max, max, max \rangle. \\ p^{bf}(X, Y) &\stackrel{0.5}{\leftarrow} m_{\neg p^{bf}}(X), p^{fb}(Y, X), p^{bf}(Y, Z); \langle ind, *, * \rangle. \\ p^{fb}(X, Y) &\stackrel{0.5}{\leftarrow} m_{\neg p^{fb}}(Y), a(X, Y); \langle ind, *, * \rangle. \\ m_{\neg p^{bf}}(Y) &\stackrel{1}{\leftarrow} m_{\neg p^{fb}}(Y); \langle max, max, max \rangle. \\ m_{\neg p^{bf}}(Y) &\stackrel{1}{\leftarrow} m_{\neg p^{fb}}(Y), p^{bf}(Y, Z); \langle max, max, max \rangle. \\ p^{fb}(X, Y) &\stackrel{0.5}{\leftarrow} m_{\neg p^{fb}}(Y), p^{bf}(Y, Z), p^{bf}(Y, X); \langle ind, *, * \rangle. \\ D^m &= D \cup \{m_{\neg p^{bf}}(1) : 1\}. \end{aligned}$$

The process of generation of adorned rules creates a collection C for all adorned predicates. Initially, the binding pattern of Q is in C . The adorned predicates in C are processed one at a time and marked so that they are not processed again. Let p^a be an unmarked adorned predicate in C . For each rule r that has the predicate p in the head, we generate an adorned version of r , called adorned rule, denoted as r^{ad} . If r^{ad} includes more adorned predicates, they are added to C if it is not already there. The process terminates when there is no unmarked adorned predicate left in C . Note that termination is guaranteed since the number of adorned predicates for any program is finite.

Once the adorned rule r^{ad} is generated, we can generate generalized magic rules. A generalized magic rule is a rule whose head is a magic predicate. For each IDB subgoal p_i^{adi} in an adorned rule r^{ad} , where i indicates the position of p_i^{adi} in the body or r , we generate a magic rule r_m^{ad} with certainty \top , and define the predicate $m_{\neg p_i^{adi}}$ as the magic predicate of p_i . As the combination functions associated with r^{ad} , we use $\langle max, max, max \rangle$. This ensures that \top will be the certainty of all ground tuples in the rewritten program, which in turn ensures the rules extended with magic predicates do not yield the incorrect certainties

when evaluated since $f(\alpha, \top) = \alpha$, for every certainty $\alpha \in T$. The head of r_m^{ad} for the IDB subgoal $p_i^{ad_i}$ is the corresponding magic predicate of the head of r^{ad} . We then add $m_p_i^{ad_i}$ to the body of r_m^{ad} . For each subgoal at position $j \in [1, i]$ in the body of r^{ad} , we add $p_j^{ad_j}$ to the body of r_m^{ad} . Note that for an adorned rule, several magic rules defining $m_p_i^{ad_i}$ might be generated because an adorned predicate may have several occurrences in the same adorned rule. Finally, the remaining operations of a *straightforward* GMS include producing the adorned rules and seeding the facts for magic predicates. We add the magic predicate of h^a to the body of each adorned rule whose head is h and then get a collection of rewritten rules for the original rules, referred to as R^m . Then, we create a *seed* that contains the bound arguments in Q with predicate name $\pi(Q)$ and certainty \top . This completes the rewriting process and the rewritten program includes the magic rules, the rules collection R^m , the tuples from the original program, and the *seed*. Example 2 shows the adorned rules of p-program in Example 1 and its transformed program using the *straightforward* GMS.

3.2 Challenges of GMS in PF

An important difference between evaluating programs in PF and in Datalog is that while the fixpoint evaluation of Datalog program terminates in polynomial time (in the number of constants in the extensional database (EDB)), an evaluation of a p-program may terminate only at the limit ω . This may happen when a *type 2* combination function is associated with a recursive IDB predicate. With a T_p operator that is continuous, we may allow a fixpoint evaluation proceed until certainties derived are “close enough” to its value in the fixpoint but short to be equal. Given a p-program P and a query Q . The evaluation order of its magic sets rewritten program P^m will be changed when the expected magic atoms are not prepared [11]. Atoms evaluated at the same iteration in P might be evaluated at different iterations. This might lead to certainty bias between P and P^m . In general, for P^m let $T_p(T_p(X) \cup Y)$ be the evaluation results for a specific atom A , where T_p is the immediate consequence operator, X is the multi-sets of derivations related to magic atoms, and Y includes those potential derivations which magic atoms are to be prepared. Then $T_p(X \cup Y)$ represents all derivations of P at a next iteration. When a *type 2* disjunction function is involved, $T_p(T_p(X) \cup Y)$ and $T_p(X \cup Y)$ may be different, and this is the major problem of extending magic sets technique to logic programs with uncertainty. Example 1 is the simplest program, we found, that shows the exact situation we may confront in the context of multi-set semantics.

However, it is unnecessary in rewriting the given program with more care if any of the following conditions does not meet:

- The program is evaluated under the multi-set semantics
- *Type 2* disjunction function is associated to at least one rule
- The program must be non-linear.
- The given data set is cyclic

Without any of the above conditions, our magic sets extension of PF is similar to the work in [10]. In this case, when the binding information is passing through the body of the program during the evaluation process, the associated certainties will be also passed. Although computation bias might also occur if the evaluation order of the rewritten program is changed, the bias will be eliminated in a finite number of steps when all ground atoms are found. On the contrary, if all the conditions above are met, for GMS rewriting process we will add an extra rewriting step, as discussed in the following section, after which the written program will produce the same intermediate results at every iteration as the original program.

3.3 Caching Magic Tuples

We introduce an extra step “Caching Magic Tuples” after *straightforward* GMS such that the extended rewritten program may generate the same results w.r.t. the given query at every iteration. Given a *straightforward* rewritten program p^m , we standardize it by peeling off the certainty and the combination functions. Second, we evaluate the standardized program p^s to find all magic atoms, and add to p^m . The certainties of the cached magic atoms are \top . We then construct a *cached* GMS rewritten program $(p^m)'$ in which all magic rules are eliminated. Example 3 shows the intermediate results of a magic tuples caching for the program in Example 1. Table 2 shows the intermediate results of the original program P and the cached GMS program $(p^m)'$ in Example 3 in which with the given data set magic set rewriting does not contribute to efficiency improvement.

Example 3. Caching Magic Tuples for the p-program in Example 1

Standardized GMS program p^s :

$$\begin{aligned} p^{bf}(X, Y) &\leftarrow m_{\neg} p^{bf}(X), a(X, Y); \\ m_{\neg} p^{fb}(X) &\leftarrow m_{\neg} p^{bf}(X); \\ m_{\neg} p^{bf}(Y) &\leftarrow m_{\neg} p^{bf}(X), p^{fb}(Y, X); \\ p^{bf}(X, Y) &\leftarrow m_{\neg} p^{bf}(X), p^{fb}(Y, X), p^{bf}(Y, Z); \\ p^{fb}(X, Y) &\leftarrow m_{\neg} p^{fb}(Y), a(X, Y); \\ m_{\neg} p^{bf}(Y) &\leftarrow m_{\neg} p^{fb}(Y); \\ m_{\neg} p^{bf}(Y) &\leftarrow m_{\neg} p^{fb}(Y), p^{bf}(Y, Z); \\ p^{fb}(X, Y) &\leftarrow m_{\neg} p^{fb}(Y), p^{bf}(Y, Z), p^{bf}(Y, X); \\ D &= \{a(1, 2), a(1, 1), a(2, 1), m_{\neg} p^{bf}(1)\}. \end{aligned}$$

Cached GMS Program $(p^m)'$:

$$\begin{aligned} p^{bf}(X, Y) &\stackrel{0.5}{\leftarrow} m_{\neg} p^{bf}(X), a(X, Y); \langle ind, *, * \rangle. \\ p^{bf}(X, Y) &\stackrel{0.5}{\leftarrow} m_{\neg} p^{bf}(X), p^{fb}(Y, X), p^{bf}(Y, Z); \langle ind, *, * \rangle. \\ p^{fb}(X, Y) &\stackrel{0.5}{\leftarrow} m_{\neg} p^{fb}(Y), a(X, Y); \langle ind, *, * \rangle. \\ p^{fb}(X, Y) &\stackrel{0.5}{\leftarrow} m_{\neg} p^{fb}(Y), p^{bf}(Y, Z), p^{bf}(Y, X); \langle ind, *, * \rangle. \\ D^m &= \{a(1, 2) : 0.5, a(1, 1) : 0.5, a(2, 1) : 0.5, \\ &\quad m_{\neg} p^{bf}(1) : 1, m_{\neg} p^{bf}(2) : 1, m_{\neg} p^{fb}(1) : 1, m_{\neg} p^{fb}(2) : 1\}. \end{aligned}$$

Table 2. Evaluation Results in Every Iteration i of p-programs in Example 3

i	Original Program P	Cached GMS Program (P^m)
1	$p(2, 1) : 0.25$ $p(1, 2) : 0.25$ $p(1, 1) : 0.25$	$p^{bf}(2, 1) : 0.25, p^{fb}(2, 1) : 0.25$ $p^{bf}(1, 2) : 0.25, p^{fb}(1, 2) : 0.25$ $p^{bf}(1, 1) : 0.25, p^{fb}(1, 1) : 0.25$
2	$p(2, 1) : 0.29614258$ $p(1, 2) : 0.2734375$ $p(1, 1) : 0.29614258$	$p^{bf}(2, 1) : 0.29614258, p^{fb}(2, 1) : 0.29614258$ $p^{bf}(1, 2) : 0.2734375, p^{fb}(1, 2) : 0.2734375$ $p^{bf}(1, 1) : 0.29614258, p^{fb}(1, 1) : 0.29614258$
3	$p(2, 1) : 0.307269$ $p(1, 2) : 0.28288764$ $p(1, 1) : 0.31192228$	$p^{bf}(2, 1) : 0.307269, p^{fb}(2, 1) : 0.307269$ $p^{bf}(1, 2) : 0.28288764, p^{fb}(1, 2) : 0.28288764$ $p^{bf}(1, 1) : 0.31192228, p^{fb}(1, 1) : 0.31192228$
...

Finally, we have Theorem 1 which shows that the cached GMS rewritten program generates the same intermediate results at every iteration, and thus yield the same fixpoint as the original program w.r.t. the given query.

Theorem 1. GMS Correctness: Let P be a p-program, D be a collection of facts, and a query Q . Let P^m be the cached GMS rewritten program with facts collection $D^m = D \cup M$, where M is the set of all pre-computed magic tuples. Then, a fixpoint computations of P and P^m produce the same answers w.r.t Q .

Proof. Basis: On one hand, for any $h(\bar{A}) \in D_1$ in P , assume s rules are fired. They are of the form:

$$h(\bar{X}_0) \xleftarrow{\alpha} q_1(\bar{Y}_1), \dots, q_t(\bar{Y}_t), p_1(\bar{X}_1), \dots, p_n(\bar{X}_n); \langle f_d, f_p, f_c \rangle$$

where q_i is an EDB predicate, and p_j is an IDB predicate. Note that only those rules without IDB predicates in the body are fired in the first iteration because no IDB facts existed in the initialized database. Therefore, we have

$$\nu_1(h(\bar{A})) = T_p(\nu_0)(h(\bar{A})) = f_d(\{f_p(\alpha, f_c(\{\nu_0(q_1(\bar{E}_1)), \dots, \nu_0(q_t(\bar{E}_t))\}))\}).$$

On the other hand, for any adorned version $h^{ad}(\bar{A})$ in P^m , the same number s of rules are fired since all the magic facts are cached. They are of the form:

$$h^{ad}(\bar{X}_0) \xleftarrow{\alpha} magic_h^{ad}, q_1(\bar{Y}_1), \dots, q_t(\bar{Y}_t), magic_{p_1}^{ad_1}, p_1^{ad_1}(\bar{X}_1), \dots, \\ magic_{p_n}^{ad_n}, p_n^{ad_n}(\bar{X}_n); \langle f_d, f_p, f_c \rangle$$

Since $\nu_0(magic_h^{ad}) = \top$,

$$\begin{aligned} \nu_1(h^{ad}(\bar{A})) &= T_p(\nu_0)(h^{ad}(\bar{A})) \\ &= f_d(\{f_p(\alpha, f_c(\{\nu_0(magic_h^{ad}), \nu_0(q_1(\bar{E}_1)), \dots, \nu_0(q_t(\bar{E}_t))\}))\}) \\ &= f_d(\{f_p(\alpha, f_c(\{\nu_0(q_1(\bar{E}_1)), \dots, \nu_0(q_t(\bar{E}_t))\}))\}) \\ &= T_p(\nu_0)(h(\bar{A})) = \nu_1(h(\bar{A})) \end{aligned}$$

Induction: For any atom $h(\bar{A}) \in D_k$ in P , and $h^{ad}(\bar{A}) \in D_k^m$ in P^m , if $\nu_k(h(\bar{A})) = T_p(\nu_{k-1})(h(\bar{A})) = T_p(\nu_{k-1})(h^{ad}(\bar{A})) = \nu_k(h^{ad}(\bar{A}))$, then we have that $\nu_{k+1}(h(\bar{A})) = T_p(\nu_k)(h(\bar{A})) =$

$$\begin{aligned} &= f_d(\{f_p(\alpha, f_c(\{\nu_k(q_1(\bar{E}_1)), \dots, \nu_k(q_t(\bar{E}_t)), \nu_k(p_1(\bar{A}_1)), \dots, \nu_k(p_n(\bar{A}_n))\}))\}), \\ &= f_d(\{f_p(\alpha, f_c(\{\nu_k(magic_h^{ad}), \nu_k(q_1(\bar{E}_1)), \dots, \nu_k(q_t(\bar{E}_t)), \nu_k(magic_{p_1}^{ad1}), \\ &\quad \nu_k(p_1^{ad1}(\bar{A}_1)), \dots, \nu_k(magic_{p_1}^{adn}), \nu_k(p_n^{adn}(\bar{A}_n))\}))\}), \\ &= T_p(\nu_{k+1})(h^{ad}(\bar{A})) = \nu_{k+1}(h^{ad}(\bar{A})) \end{aligned}$$

Based on the discussion above, we may conclude that for every atom $h(\bar{A})$ in P and its adorned form $h^{ad}(\bar{A})$ in P^m , the certainty of $h(\bar{A})$ at every iteration is exactly same as $h^{ad}(\bar{A})$. Thus, their certainties in the limit are also the same.

4 Experiments and Results

Our experiments are tested on a typical desktop computer with a Pentium 4 CPU of 2.4GHz, 2GB RAM, 250GB hard disk. In standard Datalog, the same-generation cousin (SGC) program has been widely used as a test program and a number of data sets have been introduced to measure efficiency of query processing and optimization techniques. These data sets were extended with uncertainty in [9]. We used the data sets relevant to the non-linear variant of SGC, shown in Example 4, with test data size ranging from 5,000 to 100,000 tuples which could be handled within 2GB memory size.

Example 4. Non-linear variant of the SGC program

$$\begin{aligned} &sgc(X, Y) \stackrel{\alpha}{\leftarrow} flat(X, Y); \langle f_d, f_p, f_c \rangle. \\ &sgc(X, Y) \stackrel{\alpha}{\leftarrow} up(X, Z1), sgc(Z1, Z2), flat(Z2, Z3), sgc(Z3, Z4), \\ &down(Z4, Y); \langle f_d, f_p, f_c \rangle. \end{aligned}$$

4.1 Measurements

- **The evaluation time:** Given a program P , we use $\tau(P)$ to denote the time to compute the least fixpoint of P .
tuples in the magic predicates for the MS rewritten program.
- **The rewriting time:** Given a program P and a query Q , we use $\ell(P)$ to denote the time to rewrite P and the time for caching the tuples in the magic predicates.
- **Facts generated:** Given a program P , we use $\delta(P)$ to denote the set of IDB tuples in the fixpoint evaluation of P .
- **Potential Facts Ratio:** Given a program P and a query Q , we define this ratio as $\chi(P, Q) = |\{A : A \in \delta(P) \text{ and } A \text{ subsumes } Q\}| / |\delta(P)|$. This ratio indicates the portion of the derived facts that match with or subsumed by Q .
- **Speedup:** We define the speedup as the ratio $\lambda = \tau(P) / (\ell(P) + \tau(P^m))$, assuming that both P and P^m use the same evaluation algorithm.

4.2 Performance

In our experiments, we consider the certainty domain $[0, 1]$. The test cases were created using different certainty values, for examples 0.25, 0.5, 0.75, 0.9, and 1. However, we noted that when $\alpha = 0.5$, the fixpoint evaluation of the program in general requires more iterations to complete and hence show less improvement to make, to our disadvantage. This explains why we picked $\alpha = 0.5$ as the certainty associated with the rules and facts in the program and picked $\alpha = 1$ to simulate the standard case. As noted, magic sets technique may not result in increased efficiency when χ is large, i.e., approaches 100%. It happens when all the EDB facts are potentially relevant to a given query. We investigate the speedup when the *potential facts ratio* χ varies between 1% and 20%.

Table 3. Speedup λ of magic sets for different evaluation algorithms

	$\chi = 1\%$		$\chi = 5\%$		$\chi = 10\%$	
	$\alpha = 1$	$\alpha = 0.5$	$\alpha = 1$	$\alpha = 0.5$	$\alpha = 1$	$\alpha = 0.5$
$SN/(SN + GMS)$	5..700	2.9 .. 550	0.25..50	0.2..45.2	0.14..17.9	0.12..16
$SNP/(SNP + GMS)$	3.5..400	2 .. 280	0.5 ..38	0.4..33.9	0.27..11.1	0.3 .. 8

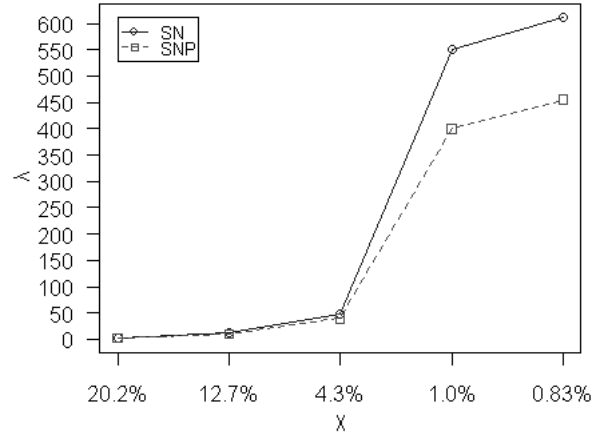


Fig. 2. The λ/χ graph for data set *Unm*

Table 3 shows how different magic sets technique affect different evaluation algorithms. Columns 2, 4, and 6 record the speedup range obtained for the standard case, where $\alpha = 1$, $\langle f_d, f_c, f_p \rangle = \langle max, min, min \rangle$, for query $Q = sgc(a, X)$.

Columns 3, 5, and 7 indicate this range for programs with uncertainty, where $\alpha = 0.5$ and $\langle f_d, f_c, f_p \rangle = \langle ind, *, * \rangle$ for the same query. As expected, we observed that different evaluation schemes yield different efficiency gains. Semi-naive evaluation benefits more from GMS rewriting than SNP does. When *Potential Facts Ratio* χ is large, the evaluation might not benefit from MS rewriting. For example, the speedup obtained by SN+GMS compared to that of SN for $\alpha = 0.5$ ranges from 0.12 to 16. Out of 8 data sets considered in our experiments, there is one type which does not benefit from the proposed MS rewriting technique when $\chi \geq 10\%$. This is because the *potential facts ratio* χ is a significant parameter affecting the speedup. The smaller the value of χ is, the larger speedup we observed. No matter the type of data set we have, we observed significant increased efficiency from GMS when χ is small. The degree of impact of GMS is determined by the complexity of the input data set and its structure. For instance, Fig. 2 shows the speedup (the vertical axis) for data set *Unm*, based on different χ values.

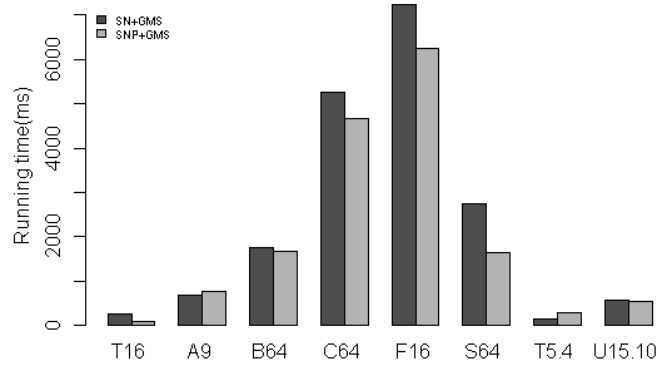


Fig. 3. Evaluation Time(τ) of SN+GMS vs. SNP+GMS

Although Table 3 shows that SN yields more speedups to itself than SNP, “SNP + GMS” still yields better performance than “SN + GMS” does, especially when the test data is more complex. Fig. 3 shows the evaluation time of “SNP + GMS” and “SN + GMS”. The test cases are selected from different types of data set and they represent the common situation for their types of data set.

5 Conclusion and Future Work

In this paper, we proposed an extension of the generalized magic sets technique for logic programs with uncertainty in the context of the parametric framework, and established its correctness. The results of our experiments show that GMS

yields different speedups for different evaluation schemes. This is determined by the notion of *potential facts ratio* χ . The smaller χ is, the more speedup we may obtain. In Datalog, generalized supplementary magic sets (GSMS) succeeds in avoiding repeatedly joins in programs and magic rules. We are investigating extension of GSMS for programs in PF.

Acknowledgments. This work was supported in part by Natural Sciences and Engineering Research Council (NSERC) of Canada and by Concordia University. We thank Jinzan Lai for his help in developing the prototype in this work.

References

- [1] S. Abiteboul et al.: The Lowell database research self-assessment. Commun. ACM, Volume 48, Issue 5, pp.111-118 (2005)
- [2] F. Bancihon, D. Maier, Y. Sagiv, and J. Ullman: Magic sets and other strange ways to implement logic programs. In Proc. 5th ACM SIGACT-SIGMOD symposium on Principles of database systems (PODS'86), pp. 1–15 (1986)
- [3] J. Ullman: Principles of Database and Knowledge-Base Systems, Volume II, Computer Science Press (1989)
- [4] C. Beeri, R. Ramakrishnan: On the power of magic. J. of Logic Programming, Volume 10, pp. 255–299 (1991)
- [5] J. Ullman: Bottom-up beats top-down for Datalog. In Proc. 18th Symposium on Principles of Database Systems (PODS), Philadelphia, Pennsylvania, United States, pp. 140–149 (1989)
- [6] L. Lakshmanan and N. Shiri: A Parametric Approach to Deductive Databases with Uncertainty. In Int'l Workshop on Logic in Databases (LID'96), D. Pedreschi and C. Zaniolo (Ed's), pp. 61–81, Springer, LNCS 1154, San Miniato, Italy, July 1–2 (1996)
- [7] N. Shiri: Expressive Power of Logic Frameworks with Certainty Constraints. FLAIRS Conf., pp. 759–765 (2005)
- [8] N. Shiri and Z. Zheng: Challenges in Fixpoint Computation with Multi-sets. Foundations of Info. and Know. Sys. (FoIKS), LNCS 2942, Vienna, Feb. 17–20 (2004)
- [9] N. Shiri and Z. Zheng: Optimizing Fixpoint Evaluation of Logic Programs with Uncertainty. Proc. 13 CSI Int'l Comp. Conf. (CSICC), Kish, Iran, March 9–11 (2008)
- [10] K. Jezek, M. Zima: Magic Sets Method with Fuzzy Logic. In Proc. 2nd Int'l Conf. on Advances in Information Systems (ADVIS), LNCS 2457, pp. 83–92, Turkey, January 01 (2002).
- [11] R. Ramakrishnan, D. Srivastava, and S. Sudarshan. Rule Ordering in Bottom-Up Fixpoint Evaluation of Logic Programs. IEEE Trans. on Knowl. and Data Eng., 6(4):501–517 (1994)
- [12] K. Sagonas, T. Swift, and D.S. Warren: XSB as an Efficient Deductive Database Engine. In Proc. ACM Conf. on Management of Data (SIGMOD), Minneapolis, Minnesota, pp. 442–453, May 24–27 (1994)
- [13] U. Straccia: Uncertainty Management in Logic Programming: Simple and Effective Top-Down Query Answering. KES Conf., pp. 753–760 (2005)
- [14] Vladimir A. Zorich: Mathematical Analysis I, Springer, pp. 84–85 (2004)

Storing and Querying Probabilistic XML Using a Probabilistic Relational DBMS

E.S. Hollander and M. van Keulen

Faculty of EEMCS, University of Twente, Enschede, The Netherlands
{e.s.hollander;m.vankeulen}@ewi.utwente.nl

Abstract. This work explores the feasibility of storing and querying probabilistic XML in a probabilistic relational database. Our approach is to adapt known techniques for mapping XML to relational data such that the possible worlds are preserved. We show that this approach can work for any XML-to-relational technique by adapting a representative schema-based (inlining) as well as a representative schemaless technique (XPath Accelerator). We investigate the maturity of probabilistic relational databases for this task with experiments with one of the state-of-the-art systems, called Trio.

1 Introduction

Data in a database is typically treated as being correct and indisputable. In many applications, this obviously is not really true. For example, data may be out of date, or some value may just be the most likely one and could very well be wrong. This is even more true for the results of automatic tasks like information extraction, natural-language processing, data integration, sensor data management, or data mining. To better support such applications, there is growing interest in the management of *uncertain data*, i.e., data for which we explicitly store the fact that it is uncertain together with information about its uncertainty.

In many of these applications, data is semi-structured, because a hierarchical representation is natural or when a source is already in this form [1]. Most research in the database community, however, is directed towards probabilistic relational databases. Several research prototype systems have been released into the open source community such as MayBMS [2, 3], Trio [4, 5], Mystiq [6], and Orion [7]. Although receiving less attention, uncertain semi-structured data, and in particular probabilistic XML, has also been used as a data model for uncertain data [8–10]. As far as we know, the work of Kimelfeld et al. is the only truly in depth work on querying probabilistic XML [11].

The contribution of this paper is twofold: (1) We present an approach for adapting existing XML-to-relational mapping techniques that preserves the possible worlds. We show how to concretely accomplish this by adapting a representative schema-based (inlining) [12] as well as a representative schemaless technique (XPath Accelerator) [13]. These lie, for example, at the heart of Oracle's object-relational storage schema [14] and MonetDB/XQuery [15], respectively.

Kind	Description
ind	<i>Independent choice</i> for each of its children.
mux	<i>Mutual exclusive choice</i> for one of its children.
det	<i>Deterministic choice</i> of all of its children. Often used in combination with ind or mux to choose multiple children in an all-or-nothing manner.
exp	<i>Explicit choice</i> of certain specific subsets of children.
cie	A choice based on a <i>conjunction of independent events</i> .

Table 1. Kinds of distributional nodes [1]

(2) Furthermore, we investigate the maturity of probabilistic relational databases for this application by experimenting with a few queries on mapped data of some XML documents on one of the state-of-the-art probabilistic database systems, namely Trio. Note that although we illustrate the adaptation of mapping techniques also with Trio, it is fairly straightforward to transfer the approach to the data models of other probabilistic databases.

2 Probabilistic Databases

2.1 Possible world theory

A probabilistic database PDB is a set of possible worlds $PDB = \{w_1, \dots, w_n\}$ each with its probability $\Pr(w_i)$ such that $\sum_{i=1..n} \Pr(w_i) = 1$. Each world is an ordinary database, so in case of probabilistic XML, a world is an ordinary XML tree and in case of probabilistic relations, a world is a set of ordinary relations.

The semantics of a query on a probabilistic database are defined as the set of answers of the query posed to each of the possible worlds individually. Consequently, the probability of a particular answer is the sum of the probabilities of all possible worlds for which the query produced that answer.

Since the number of possible worlds grows exponentially, implementations of probabilistic databases store all possible worlds in one compact representation. Algorithms for querying a probabilistic database directly work on the compact representation while strictly adhering to the semantics of querying as defined in terms of possible worlds.

2.2 Probabilistic XML

Probabilistic XML is an extension to XML for representing uncertainty in the data in a compact way. This is achieved by introducing *distributional nodes* to denote probabilistic distributions over subtrees (see Tab.1). There are several families of probabilistic XML with varying expressiveness depending on the kinds of distributional nodes allowed [1]. In this paper, we use the probabilistic XML model of [10, 16] which is equivalent with the $\text{PrXML}^{\{\text{mux}, \text{det}\}}$ family. In this model, mux nodes are called *probability nodes* (denoted with $\langle \text{prob} \rangle$ in XML and

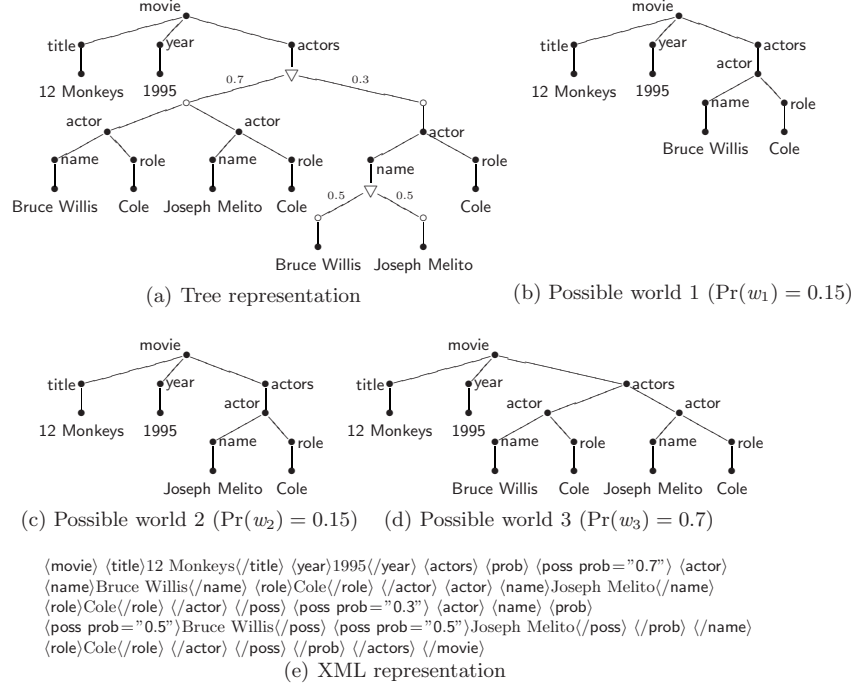


Fig. 1. Example of Probabilistic XML.

∇ in the tree representation) and *det* nodes are called *possibility nodes* (denoted with `<poss>` in XML and \circ in the tree representation).

Example 1. Figure 1 shows an example of a probabilistic XML instance. It is uncertain if there are two actors playing the role “Cole”, or that there is just one actor playing the role, but in this case it is uncertain which of the two names is the name of the actor. Figures 1(a), 1(b)–1(d), and 1(e) respectively illustrate the tree representation, the three possible worlds it encodes, and the XML representation. Note that this is a nested model, hence a possibility node may contain an entire subtree which may in turn contain distributional nodes.

2.3 Probabilistic Relations

In recent years, there has been much research into probabilistic relational databases culminating into several prototypes such as MayBMS [2] and Trio [4]. We have used Trio in this paper for our experiments.

Trio allows for multiple alternatives for a tuple. A tuple containing more than one alternative is called an x-tuple. Alternatives may or may not have associated probabilities (called confidence scores). If the probabilities do not add up to 1, the x-tuple is called a maybe x-tuple, because it is also possible that it does not exist at all. Figure 2 shows an example of an uncertain table in Trio with one x-tuple with two alternatives $t_{1,1}$ and $t_{1,2}$. Trio is also based on possible worlds theory, hence the example encodes two possible relations.

movie			
	id	title	year
$t_{1,1}$	1	Twelve Monkeys	1995
$t_{1,2}$	1	12 Monkeys	1995

Fig. 2. Example Trio table

In principle, x-tuples are independent, i.e., arbitrary combinations of alternatives from different x-tuples can exist and they do so with a probability that is the product of the probabilities of the original alternatives. To be able to express dependencies, Trio supports *lineage*. An alternative's existence in this way may depend on the existence of other alternatives. Lineage is expressed with a boolean formula such as $\lambda(t_{1,1}) = t_{2,1} \wedge t_{3,2}$. The set of possible worlds is restricted to those where the lineage formulas are true, in our example, to those where $t_{1,2}$, $t_{2,1}$, and $t_{3,2}$ co-exist. Lineage is typically introduced by queries, because the alternatives in the result depend on the alternatives in the base tables.

3 Storing and Querying XML in Relational Databases

In recent years, many approaches to storing and querying XML have been proposed. The ones mapping XML onto relational tables can be divided into two categories: schema-based and schemaless. The former constructs a relational schema based on the DTD or XML schema of the XML documents. The latter treats the XML documents as trees and stores each node of the tree as a tuple in one generic relation. We took two techniques representative for each category: Inlining and XPath Accelerator, respectively. We summarize both below. For details, we refer to [12] and [13, 15, 17], respectively.

movie: (title, year, actors)
actors: (actor*)
actor: (name, role)
title: (PCDATA)
year: (PCDATA)
name: (PCDATA)
role: (PCDATA)

Fig. 3. Example DTD

3.1 Schema-based: Inlining

The inlining technique by Shanmugasundaram et al. [12] was one of the first algorithms available that could store an XML-document in a relational database. It relies on DTDs to generate a relational schema. The technique first constructs a DTD graph in which each node represents an element type; the arrows are annotated with the multiplicities. In general, each element type generates one table; the graph is used, however, to *inline* the information of certain elements into the table of its parent in an attempt to reduce the number of tables.

Example 2. For example, suppose we have the DTD of Fig.3. The hybrid inlining technique recognizes that 'title', 'year', 'name', and 'role' can only occur once

in their respective parent elements. Therefore, they are inlined to produce the following relations:

- movie(id:int, title:string, year:int, actorsid:int)¹
- actor(id:int, parent:int, name:string, role:string)

These two relations suffice to store all data that is contained in XML documents conforming to this DTD.

3.2 Schemaless: XPath Accelerator

The XPath Accelerator [15] technique does not depend on the existence of a schema. Instead, it views the XML document as a tree and uses one table that stores both the information contained in the nodes as well as the structure of the tree. In this paper, we use a simplified version of the XPath Accelerator version described in [17]. The structure of the tree is encoded using three attributes:

- **pre**: the rank assigned to the node in a preorder traversal of the tree.²
- **size**: the size of the subtree below the node.
- **level**: the depth of the node in the tree, i.e., the length of the path from the node to the root.

pre	size	level	kind	prop
1	11	1	elem	movie
2	2	2	elem	title
3	1	3	text	Twelve Monkeys
4	2	2	elem	year
5	1	3	text	1995
6	6	2	elem	actors
7	5	3	elem	actor
8	2	4	elem	name
9	1	5	text	Bruce Willis
10	2	4	elem	role
11	1	5	text	Cole

Fig. 4. XPath Accelerator example

We use two additional attributes, kind and prop. The former contains the node kind. The latter contains for elements its tag and for text nodes its string content. Without loss of generality, we restrict ourselves to only element and text nodes. Storing, for example, the possible world of Fig.1(b) in this manner produces the table of Fig.4.

4 Mapping Probabilistic XML to Probabilistic Relations

Naively applying the techniques of Sec.3 for storing and querying XML using a relational database defies our purpose. If we would do that, we would end up with a certain database where all probability and possibility nodes are stored directly. Instead, we would like to leverage the functionality of the probabilistic RDBMS for storing and querying uncertain data. We therefore adapt the techniques in such a way that we *can* leverage this functionality.

¹ There are variants of the inlining technique that would also produce an ‘actors’ relation. Since it is superfluous here, we inline it for simplicity of the running example.

² “In a preorder traversal, a tree node v is visited and assigned its preorder rank $pre(v)$ before its children are recursively traversed from left to right.” [13]

4.1 General approach

Both probabilistic XML and probabilistic relations are based on possible worlds theory. Therefore, the semantics of a query are defined in the same way: as the set of the answers to the query for each possible world. To be able to leverage the functionality of probabilistic relational database, we need to make sure that the stored relational data encodes the same possible worlds as the original probabilistic XML.

The key to preserving the possible worlds lies in the observation that uncertain data is all about choices. In probabilistic XML we choose among subtrees, in probabilistic relations we choose among alternative tuples. In the sequel, we first view both models more formally in terms of choices and then show how to adapt the XML-to-relational techniques so that they preserve the possible worlds.

4.2 Viewing Probabilistic XML in terms of choices

Probability nodes (as all distributional nodes) can be seen as independent random variables. Their domains consist of (references to) their possibility node children. Let n_1 and n_2 be the higher and lower probability nodes in Fig. 1(a), respectively, and x_{n_i} its associated random variable. The assignment $x_{n_2} \leftarrow 1$ denotes the choice for the first (left) subtree of n_2 , i.e., the name “Bruce Willis”, and $x_{n_2} \leftarrow 2$ denotes the choice for the second (right) subtree, i.e., the name “Joseph Melito”. The probability of an assignment $\Pr(x_{n_2} \leftarrow j)$ is the probability associated with possibility node j below n_2 .

A complete choice θ is a set containing one assignment for each random variable. Each complete choice determines one particular possible world w_θ with probability $\prod_{(x \leftarrow j) \in \theta} \Pr(x \leftarrow j)$. Note that because probability nodes may be nested, it may happen that two different complete choices determine the same possible world (e.g., $\{x_{n_1} \leftarrow 1, x_{n_2} \leftarrow 1\}$ and $\{x_{n_1} \leftarrow 1, x_{n_2} \leftarrow 2\}$ denote the same possible world, namely Fig.1(d)).

Viewing it from an XML node’s perspective, the node only exists if it has been chosen, i.e., it only exists in those worlds determined by a complete choice that includes for each of its parent probability nodes n , the assignment $x_n \leftarrow j$ where j is a reference to the possibility node child of n that is a parent of the XML node.

4.3 Viewing probabilistic relations in terms of choices

In probabilistic relations, each x-tuple can be seen as a random variable x . Its domain consists of (references to) its alternatives. If it is possible that the tuple does not exist at all, there is a special value ‘ \perp ’ in the domain. The remaining probability mass is given to $x \leftarrow \perp$. Here too a complete choice determines a possible world, hence the probabilistic relation of Fig.2 encodes two possible worlds, i.e., two possible movie relations.

The lineage of Trio restricts the set of possible worlds to valid ones, i.e., to only those that respect the co-existence relationships between alternatives as defined by the lineage. In terms of random variables, only those complete choices are valid for which its assignments respect the lineage, i.e., if an assignment is associated with an alternative that needs to co-exist with other alternatives, then their associated assignments are also contained in the complete choice.

Other probabilistic databases have different data models and means to restrict the set of possible worlds. MayBMS, for example, associates a set of random variable assignments (called world set descriptor) with each tuple. Therefore, it is fairly straightforward to transfer our approach to other probabilistic databases.

5 Schema-based mapping: Adapted Inlining

Our adapted inlining technique has three phases.

- (1) We first construct an *event table*, i.e., an uncertain table with all random variables (attribute rvar) and their possible assignments (attribute ass).
- (2) We then map the data in the XML nodes to certain relational tables in the same way as the inlining technique prescribes except that we do not inline child element types that may contain uncertainty (see Sec.5.3). We furthermore mark the tuples with the ids of the event associated with its direct parent possibility node (it is not necessary to also mark them with the ids of the other ancestor possibility nodes as we will see later). XML nodes that do not have a parent possibility node (certain XML nodes) are marked with NULLs.
- (3) Finally, we execute queries that produce the same tables, but with the proper lineage attached expressing the dependence on the various random variable assignments. For ‘movie’ we execute the following query:

```
CREATE TABLE umovie AS
  SELECT movie.id, movie.title, movie.year, movie.actorsid
  FROM   movie, event
  WHERE (movie.rvar = event.rvar AND movie.ass = event.ass)
  OR    movie.rvar IS NULL;
```

5.1 Nesting

If the probabilistic XML document contains nested elements, step 3 above is performed from top to bottom. This happens in our example for ‘actor’ which is a descendant of ‘movie’. Since one movie can have multiple actors, the inlining technique creates another table ‘actor’ which contains an attribute with a reference to the parent. There may exist uncertainty surrounding the actors as well. In a probabilistic XML tree, a node can only exist when its parent node also exists. These dependencies need to be stored correctly.

This is where lineage fully comes into play. Trio ensures that the existence of a tuple depends on the existence of the tuples in its lineage (and the lineage thereof, and so on). Therefore, we create the uncertain tables for child element types

event			
	rvar	ass	
$t_{1,1}$	1	1	0.7
$t_{1,2}$	1	2	0.3
$t_{2,1}$	2	1	0.5
$t_{2,2}$	2	2	0.5

movie					
	rvar	ass	id	title	year
t_3	NULL	NULL	1	12 Monkeys	1995

umovie			
	id	title	year
t_4	1	12 Monkeys	1995

actor				
	rvar	ass	id	parent
t_5	1	1	1	1
t_6	1	1	2	1
t_7	1	2	3	1

uactor			
	id	parent	role
t_8	1	1	Cole
t_9	2	1	Cole
t_{10}	3	1	Cole

name				
	rvar	ass	id	parent
t_{11}	NULL	NULL	1	1
t_{12}	NULL	NULL	2	2
t_{13}	2	1	3	3
t_{14}	2	2	4	3

uname			
	id	parent	text
t_{15}	1	1	Bruce Willis
t_{16}	2	2	Joseph Melito
t_{17}	3	3	Bruce Willis
t_{18}	4	3	Joseph Melito

Fig. 5. Inlining-based mapping of example XML.

based on the resulting uncertain tables of their parent element types created previously. This ensures that the lineage expresses all dependencies in the tree.

We can create the table ‘uactor’ by issuing the following query (‘uname’ analogously).

```
CREATE TABLE uactor AS
SELECT actor.id, actor.parent, actor.role
FROM actor, event, umovie
WHERE ((actor.rvar = event.rvar AND actor.ass = event.ass)
OR      actor.event IS NULL)
AND      actor.parent = umovie.id;
```

5.2 Example

Figure 5 shows the result for the example tree of Fig.1(a). Note that we invented new identifiers for the tuples. Also note that ‘uactor’ and ‘uname’ contain only x-tuples with one alternative instead of more as you might expect. The lineage, however, expresses that these x-tuples depend on $t_{1,1}$, $t_{1,2}$, $t_{2,1}$, and $t_{2,2}$. Since the first two are mutually exclusive and last two as well, the lineage dependencies make some of the other x-tuples to be mutually exclusive as well.

Suppose we have a possible world in which $t_{1,1}$ exists. This corresponds to Fig. 1(d). In this possible world, t_{10} cannot exist, because it depends on $t_{1,2}$, which is mutually exclusive with $t_{1,1}$. Further down the tree, t_{17} cannot exist

either, because it depends on t_{10} . In this way, lineage preserves the dependencies that exist in the original tree.

We used a cartesian product of both tables, hence the event table must not be empty. This could happen if the probabilistic XML tree is certain. This problem can be avoided by using an outer join instead of the cartesian product. Unfortunately, this functionality was not available in Trio at the time of writing.

Since we refer to the parent table that already contains lineage, these will be taken into account when querying. In this way, we fully leverage Trio's functionality for calculating probabilities in the context of complex dependencies among x-tuples. Finally, observe that the resulting probabilistic relations encode the same possible worlds as the original probabilistic XML document.

5.3 Avoiding data duplication

Data duplication may occur if we would inline a value that is uncertain, because if an x-tuple contains an inlined uncertain attribute, it may result in several alternatives. In each of these alternatives, the other certain attributes are duplicated. In our example, this happens with 'name' and 'role': both 'name' and 'role' could be inlined according to the original inlining technique, but 'name' is uncertain, therefore if we would inline 'name' as well, the certain data (in our example the value "Cole") is duplicated in the alternatives for the name.

If more than one inlined attribute is uncertain, data duplication would grow exponentially. For example, if 'role' would be uncertain as well with two alternatives, then we end up with 4 alternatives for the one actor x-tuple.

The solution to this problem is to not inline element types which may be uncertain. As a consequence, the element type gets a relation of its own, with an accompanying reference to the parent tuple. Note that, in this way, values occur as many times as in the original document.

6 Schemaless mapping: Adapted XPath Accelerator

For XPath Accelerator, we calculate the pre, size and level values for every node in the probabilistic XML document, i.e., including the distributional nodes. We store the data for the XML nodes in the 'doc' table and the data for the possibility nodes in the event table where all possibility nodes of one probability node form one x-tuple. We also add a 'catch all' event t_0 (otherwise we would not select any certain nodes at the root of the document).

We combine the two tables in the same way as for the inlining technique except for the fact that we do not have ids for relating tuples in doc with the x-tuples in event. Instead, each node in the probabilistic XML document depends on all its ancestor possibility nodes. The ancestor-relationship can be expressed using the pre and size attributes. Hence we execute the following query:

event			
	pre	size	level
t_0	0	29	0
$t_{1,1}$	8	11	4
$t_{1,2}$	19	9	4
$t_{2,1}$	23	2	8
$t_{2,2}$	25	2	8

doc				
	pre	size	level	kind
t_3	1	28	1	elem
t_4	2	2	2	elem
t_5	3	1	3	text
t_6	4	2	2	elem
t_7	5	1	3	text
t_8	9	5	5	elem
t_9	10	2	6	elem
	⋮	⋮	⋮	⋮
t_{10}	20	9	5	elem
t_{11}	21	2	6	elem
t_{12}	24	1	9	text
t_{13}	26	1	9	text
t_{14}	27	2	6	elem
t_{15}	28	1	7	text

udoc				
	pre	size	level	kind
t_{16}	1	28	1	elem
t_{17}	2	2	2	elem
t_{18}	3	1	3	text
t_{19}	4	2	2	elem
t_{20}	5	1	3	text
t_{21}	9	5	5	elem
t_{22}	10	2	6	elem
	⋮	⋮	⋮	⋮
t_{23}	20	9	5	elem
t_{24}	21	2	6	elem
t_{25}	24	1	9	text
t_{26}	26	1	9	text
t_{27}	27	2	6	elem
t_{28}	28	1	7	text

	prop	
	movie	$\lambda(t_{16}) = t_0 \wedge t_3$
	title	$\lambda(t_{17}) = t_0 \wedge t_4$
	12 Monkeys	$\lambda(t_{18}) = t_0 \wedge t_5$
	year	$\lambda(t_{19}) = t_0 \wedge t_6$
	1995	$\lambda(t_{20}) = t_0 \wedge t_7$
	actor	$\lambda(t_{21}) = t_0 \wedge t_{1,1} \wedge t_8$
	name	$\lambda(t_{22}) = t_0 \wedge t_{1,1} \wedge t_9$
	⋮	⋮
	actor	$\lambda(t_{23}) = t_0 \wedge t_{1,2} \wedge t_{10}$
	name	$\lambda(t_{24}) = t_0 \wedge t_{1,2} \wedge t_{11}$
	Bruce Willis	$\lambda(t_{25}) = t_0 \wedge t_{1,2} \wedge t_{2,1} \wedge t_{12}$
	Joseph Melito	$\lambda(t_{26}) = t_0 \wedge t_{1,2} \wedge t_{2,2} \wedge t_{13}$
	role	$\lambda(t_{27}) = t_0 \wedge t_{1,2} \wedge t_{14}$
	Cole	$\lambda(t_{28}) = t_0 \wedge t_{1,2} \wedge t_{15}$

Fig. 6. XPath Accelerator-based mapping of example XML.

```

CREATE TABLE udoc AS
  SELECT DISTINCT doc.pre, doc.size, doc.level, doc.kind, doc.prop
  FROM   doc, event
  WHERE  doc.pre > event.pre AND doc.pre < (event.pre + event.size);

```

Unfortunately, this query produces the ‘udoc’ table of Fig.7 instead of the desired result of Fig.6. If an XML node depends on more than one ancestor possibility node, then the **DISTINCT** produces *or*-lineage as opposed to *and*-lineage. For example, the lineage of tuple t_8 does not contain the term $t_0 \wedge t_{1,1}$ but $t_0 \vee t_{1,1}$. At the time of writing, Trio does not have a keyword or some other construct that allows us to specify that tuples are dependent on all tuples that correspond with a certain predicate. Trio does support *and*-lineage, we only cannot construct it under the circumstances we have at hand here. To be able to reliably conduct our experiments, we have manually updated the underlying

udoc					
	pre	size	level	kind	prop
t_{24}	21	2	6	elem	name
t_{25}	24	1	9	text	Bruce Willis
t_{26}	26	1	9	text	Joseph Melito

$$\lambda(t_{24}) = (t_0 \vee t_{1,2}) \wedge t_{11}$$

$$\lambda(t_{25}) = (t_0 \vee t_{1,2} \vee t_{2,1}) \wedge t_{12}$$

$$\lambda(t_{26}) = (t_0 \vee t_{1,2} \vee t_{2,2}) \wedge t_{13}$$

Fig. 7. Excerpt of wrong or-lineage based udoc-result.

File	Size	#XML nodes	# ∇ nodes	# \circ nodes	Avg # \circ per ∇
1	10.4 kB	784	1	1	1
2	43.7 kB	2510	119	240	2.016807
3	54.9 kB	3193	129	265	2.054264
4	119.7 kB	7340	193	425	2.202073
5	186.7 kB	11490	255	570	2.235294
6	800.0 kB	52320	801	1872	2.337079

Fig. 8. Data set properties

PostgreSQL tables that encode the lineage for these tables such that the resulting table is associated with the exact lineage we need.

Many XML-to-relational mapping techniques are based on prefix-labelling schemes, such as ORDPATHs and DeweyIDs. These serve both as node ID as well as efficient ways of determining axis relationships. Since these are likely to produce similar query characteristics as the pre/size conditions of the XPath Accelerator, we have not investigated these approaches separately. Note that here too, values occur as many times as in the XML document, so space complexity is the same.

6.1 Query mapping

Our objective is to evaluate queries on the probabilistic XML data using the probabilistic relational backend. Having mapped the probabilistic XML data onto probabilistic relations according to the inlining or the XPath Accelerator technique in this way, mapping the queries is trivial: we can simply apply the same approach as in the original inlining and XPath Accelerator unaltered. Because the data represents exactly the same possible worlds, and each of the possible worlds conforms to the original techniques, the query answer conforms to the possible world semantics. See Sec. 7.2 for an example.

7 Experiments

7.1 Experimental setup

We experiment with real-life uncertain data obtained from a probabilistic data integration application [18]. The application integrates movie data from TV guide (www.tvguide.com) with the Internet Movie Database (www.imdb.com). For this

```

Inlining:
SELECT y.year
FROM umovie m, utitle t, uyear y
WHERE m.id = t.parentid
      AND t.title = 'District B13'
      AND m.id = y.parentid;

XPath Accelerator:
CREATE TABLE temp1 AS
SELECT DISTINCT v1.*
FROM udoc c, udoc v1
WHERE v1.pre > c.pre AND v1.pre < (c.pre + c.size)
      AND v1.kind = 'elem' AND v1.prop = 'movie';
CREATE TABLE temp2 AS
SELECT DISTINCT v1.*
FROM temp1 c, udoc v1
WHERE v1.pre > c.pre AND v1.pre < (c.pre + c.size)
      AND v1.kind = 'elem' AND v1.prop = 'title';

CREATE TABLE temp3 AS
SELECT DISTINCT v1.*
FROM temp2 c, udoc v1
WHERE v1.pre > c.pre AND v1.pre < (c.pre + c.size)
      AND v1.kind = 'text' AND v1.prop = 'District B13';
CREATE TABLE temp4 AS
SELECT DISTINCT v1.*
FROM temp3 c, udoc v1
WHERE v1.pre < c.pre
      AND (v1.pre + v1.size) >= (c.pre + c.size)
      AND v1.kind = 'elem' AND v1.prop = 'movie';
CREATE TABLE temp5 AS
SELECT DISTINCT v1.*
FROM temp4 c, udoc v1
WHERE v1.pre > c.pre AND v1.pre < (c.pre + c.size)
      AND v1.kind = 'elem' AND v1.prop = 'year';
SELECT DISTINCT v1.*
FROM temp5 c, udoc v1
WHERE v1.pre > c.pre AND v1.pre < (c.pre + c.size)
      AND v1.kind = 'text';

```

Fig. 9. Translations of the exact match query.

paper, it is not important to understand much about probabilistic data integration (the reader is referred to [16]), only that it semi-automatically fuses two XML documents producing a probabilistic XML document. By varying some thresholds in the probabilistic integration, we obtain probabilistic XML documents with varying amounts of uncertainty, hence of varying sizes (see Tab.8).

We experiment with 3 queries representative for 3 categories of querying:

1. [Exact match] //movie[title='District B13']/year/text()
"The year in which the movie 'District B13' has been released"
2. [Tree navigation] //movie[actors/actor/name='Brooke Smith']/title/text()
"The titles of all movies in which Brooke Smith is an actor"
3. [Join] //movie[year=//movie[actors/actor/name='David Belle']/year]/title/text()
"The titles of all movies of the same year as a movie with actor David Belle"

Note that movie titles, actor names and years are often ambiguous in the integration scenario, hence the chosen queries deliberately target highly uncertain sections of the resulting documents.

The experiments are performed on a PC with an AMD Athlon 64 X2 Dual Core 4000+ processor, 1 GB of internal memory and Windows XP Service Pack 3 installed. We have used PostgreSQL 8.2 as backend for Trio.

There are in total 72 runs in our experiments, namely one run for each combination of 6 data sets, 3 queries, 2 techniques, and with or without confidence calculation. For each run, we measured average query execution time for 5 executions on a hot database.

7.2 Query translation

Obviously, the abovementioned XPath queries need to be translated to SQL according to the technique involved: Inlining or XPath Accelerator. See Fig. 9 for

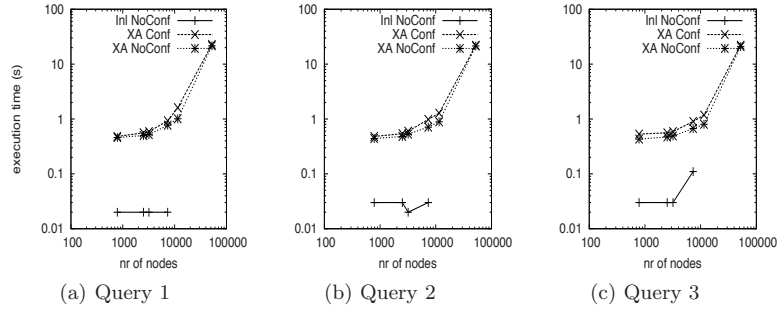


Fig. 10. Experimental results.

the translations of the exact match query. The translation for XPath Accelerator deviates from what is prescribed for this technique. XPath Accelerator requires self-joins for evaluating XPath steps. Unfortunately, Trio could not cope with the number of self-joins. Therefore, we split the query into one per step. We store the intermediate result in a table to be queried in the next step. This splitting also does not permit nested predicates, so we rewrote the query to the equivalent `//movie/title[.='District B13']/ancestor::movie/year/text()` before translating it.

7.3 Results

We were unable to obtain measurements for all 72 runs of the experiment due to practical problems with Trio:

- We have no measurements for Inlining queries that calculate confidences because Trio crashed.
- We have no measurements for Inlining queries on the largest two documents, because the data could not be imported into Trio.

We consulted one of the developers of Trio, but he also could not resolve these problems for us. We have verified that queries for both techniques return the same results under the same circumstances, so we are confident that we are measuring execution times for queries that do not return bogus results.

Results for the successful runs can be found in Fig. 10. ‘Inl’ stands for Inlining; ‘XA’ for XPath Accelerator; ‘Conf’ and ‘NoConf’ for with and without the calculation of confidence scores, respectively. The wobble in the ‘Inl NoConf’ line of Query 2 is caused by rounding an imprecise measurement (0.03 vs. 0.02).

The first thing that stands out is that it seems that the XPath Accelerator-based approach is much less efficient than the Inlining-based approach. We believe, however, that we cannot draw this conclusion from these results. The XPath Accelerator queries use several intermediate tables which causes much overhead. Furthermore, it also does not permit the query optimizer to globally

optimize the query. Since this query splitting is not inherent to the XPath Accelerator technique, but a measure taken because of practical problems with Trio, it would be unfair to draw this conclusion.

A second observation is that the query execution time does not significantly increase with increasing size except for the largest document for XPath Accelerator. The shape of the curve seems to indicate that evaluation of these queries scales exponentially. Unfortunately, we do not know how much time the Inlining queries would have taken on the largest document. Only for join queries, we see a rise at a size of around 7000 nodes.

A third observation is that the query execution times do not significantly differ for the three queries. Finally, calculation of confidences is typically an expensive operation. In this application context, however, we see that for XPath Accelerator the overhead for calculating the confidences is relatively negligible.

8 Conclusions

In this paper, we explored the feasibility of storing and querying probabilistic XML using an uncertain relational database. Our approach is to adapt existing techniques for mapping XML data to relational data. We showed how to do this for two representative techniques, namely a schema-based (inlining) and a schemaless one (XPath Accelerator). The key is to make sure that the result represents the same possible worlds as the original probabilistic XML document. In this way, no adaptation in the translation of XML queries is needed. The space complexity is the same as for the underlying mapping techniques.

The maturity of probabilistic relational databases also influences the feasibility. We investigated this by experimenting with a few queries on mapped data on one of the state-of-the-art probabilistic database systems, called Trio. Unfortunately, we encountered some problems with loading the mapped data and with calculating confidence scores for query results. Based on the experiments that did run smoothly or for which we could find a workaround, we observed, for example, exponential scaling for queries on data resulting from mapping XML with the adapted XPath Accelerator technique. On the other hand, confidence calculation proved relatively inexpensive here. Queries on mapped data from the adapted inlining technique appear to be more efficient, but loading mapped data from larger documents and confidence computation failed with this technique.

In this research, we only touched the surface by focussing on feasibility of the approach. For future work we, first of all, intend to expand our experiments to other probabilistic database systems to see if our conclusions hold in general. We also like to compare this approach to extending existing XML databases with support for probabilistic XML. It would be scientifically worthwhile to formally prove that the data and query mapping to the relational domain are indeed correct with respect to XPath semantics and possible world theory. It is also likely that such a formal investigation uncovers opportunities for query optimization. Finally, we intend to turn this work into a benchmark for probabilistic databases.

References

1. Abiteboul, S., Kimelfeld, B., Sagiv, Y., Senellart, P.: On the expressiveness of probabilistic XML models. *The VLDB Journal* **18**(5) (2009) 1041–1064
2. Huang, J., Antova, L., Koch, C., Olteanu, D.: MayBMS: a probabilistic database management system. In: *Proc. of SIGMOD*, Providence, Rhode Island, USA, June 29 - July 2. (2009) 1071–1074
3. Antova, L., Koch, C., Olteanu, D.: MayBMS: Managing incomplete information with probabilistic world-set decompositions. In: *Proc. of ICDE*, Istanbul, Turkey. (2007) 1479–1480
4. Mutsuzaki, M., Theobald, M., de Keijzer, A., Widom, J., Agrawal, P., Benjelloun, O., Sarma, A.D., Murthy, R., Sugihara, T.: Trio-One: Layering uncertainty and lineage on a conventional DBMS (demo). In: *On-Line Proc. of CIDR*, Asilomar, CA, USA, January 7–10, www.crdrrdb.org (2007) 269–274
5. Benjelloun, O., Sarma, A.D., Hayworth, C., Widom, J.: An introduction to ULDBs and the Trio system. *IEEE Data Engineering Bulletin* **29**(1) (2006) 5–16
6. Boulos, J., Dalvi, N., Mandhani, B., Mathur, S., Re, C., Suciu, D.: MYSTIQ: a system for finding more answers by using probabilities. In: *Proc. of SIGMOD*, Baltimore, Maryland, USA. (2005) 891–893
7. Cheng, R., Singh, S., Prabhakar, S.: U-DBMS: A database system for managing constantly-evolving data. In: *Proc. of VLDB*, Trondheim, Norway. (2005) 1271–1274
8. Hung, E., Getoor, L., Subrahmanian, V.: PXML: A probabilistic semistructured data model and algebra. In: *Proc. of ICDE*, Bangalore, India. (2003) 467
9. Abiteboul, S., Senellart, P.: Querying and updating probabilistic information in XML. In: *Proc. of EDBT*, Munich, Germany. (2006) 1059–1068 LNCS 3896.
10. van Keulen, M., de Keijzer, A., Alink, W.: A probabilistic XML approach to data integration. In: *Proc. of ICDE*, Tokyo, Japan. (2005) 459–470
11. Kimelfeld, B., Kosharovskiy, Y., Sagiv, Y.: Query evaluation over probabilistic XML. *The VLDB Journal* **18**(5) (2009) 1117–1140
12. Shanmugasundaram, J., Tufte, K., Zhang, C., He, G., DeWitt, D.J., Naughton, J.F.: Relational databases for querying XML documents: Limitations and opportunities. In: *Proc. of VLDB*, Edinburgh, Scotland, UK. (1999) 302–314
13. Grust, T.: Accelerating XPath location steps. In: *Proc. of SIGMOD*, Madison, Wisconsin. (2002) 109–120
14. Murthy, R., Banerjee, S.: XML Schemas in Oracle XML DB. In: *Proc. of VLDB*, Berlin, Germany. (2003) 1009–1018
15. Boncz, P., Grust, T., van Keulen, M., Manegold, S., Rittinger, J., Teubner, J.: MonetDB/XQuery: a fast XQuery processor powered by a relational engine. In: *Proc. of SIGMOD*, Chicago, IL. (2006) 479–490
16. van Keulen, M., de Keijzer, A.: Qualitative effects of knowledge rules and user feedback in probabilistic data integration. *The VLDB Journal* **18**(5) (2009) 1191–1217
17. Grust, T., Rittinger, J., Teubner, J.: Pathfinder: Xquery off the relational shelf. *IEEE Data Engineering Bulletin* **31**(4) (2008) 7–14
18. van Keulen, M., de Keijzer, A.: Qualitative effects of knowledge rules in probabilistic data integration. Technical Report TR-CTIT-08-42, CTIT, Univ. of Twente, Enschede, The Netherlands (2008) ISSN 1381-3625.

Time-aware Reasoning in Uncertain Knowledge Bases

Yafang Wang, Mohamed Yahya, and Martin Theobald
`{ywang,myahya,mtb}@mpi-inf.mpg.de`

Max-Planck Institute for Informatics
Saarbruecken, Germany

Abstract. Time information is ubiquitous on the Web, and considering temporal constraints among facts extracted from the Web is key for high-precision query answering over time-variant factual data. In this paper, we present a simple and efficient representation model for time-dependent uncertainty in combination with first-order inference rules and recursive queries over RDF-like knowledge bases. In the spirit of *data lineage*, the intensional (i.e., rule-based) structure of query answers is reflected by Boolean formulas that capture the logical dependencies of each derived answer fact back to its extensional roots (i.e., base facts). Our approach incorporates simple *weight aggregations* for *begin*, *end* and *during* evidences for base facts, but also generalizes the common *possible-worlds semantics* known from probabilistic databases to histogram-like confidence distributions for derived facts. In particular, we show that adding time to the latter probabilistic setting adds only a light overhead in comparison to a time-unaware probabilistic setting.

1 Introduction

Recent progress in information extraction has led to major breakthroughs in automatically building large ontological knowledge bases from high-quality Web sources, such as online news sites, or encyclopedias like Wikipedia. Projects such as DBpedia [1], KnowItAll [6] and its underlying extraction frameworks TextRunner [23] and Kylin/KOG [22], ReadTheWeb [4], as well as our own YAGO project [17], have successfully shown how to build structured knowledge representations from unstructured or weakly structured Web collections with high precision and recall. A major shortcoming that these knowledge bases still face is the lack of time information in the both the representation model and the types of queries they support. Thus, these information extraction techniques generally work well if we consider the quality of each of the extracted facts individually, but time constraints start playing a major role when the knowledge base is queried, i.e., when we would like to reason about multiple facts with respect to their temporal context. For example, a query looking for all teammates of David Beckham during his time at Real Madrid would only be meaningful if we have explicit information about *when* each of the team members of Real Madrid played for the club.

While extraction tools like TARSQI [20] generally perform well in detecting time adverbials in text, they also introduce a certain amount of errors. Even if

we would restrict ourselves to structured and mostly trustworthy sources such as Wikipedia infoboxes, achieving 100% precision in temporal fact extraction will likely remain an illusive goal. Key factors are the incorrect detection and resolution of temporal annotations caused by the high diversity of temporal expressions used in free text, as well as plain inconsistencies among different sources. As an illustration, one news article might report “*The hype and speculation have escalated ever since the January announcement that Beckham would join the Galaxy from Spanish giants Real Madrid on a deal that will earn him a reported \$250 million over five years.*”, while another article mentions “*Former England captain David Beckham left London Thursday to begin his new stint playing for the Los Angeles Galaxy*”¹. From these headlines, we could extract the fact (*David Beckham joins Los Angeles Galaxy*) with different time annotations. First, the granularity of these time annotations is different (“January” is a month, while “Thursday” is day), and second, the latter is only a relative time annotation that needs to be resolved and matched with the publication date of the news article. Furthermore, time information often is incomplete. In the first example, there is no information about the year when Beckham announced to *join* Los Angeles Galaxy, while in the second one, not even a month or week for when Beckham *left* Galaxy is stated. Thus, failures in recognizing and resolving temporal expressions are an important factor in introducing uncertainty and even inconsistency to temporal knowledge bases.

Moreover, for reasoning and query answering, new temporal facts need to be derived from existing temporal facts. Knowing, for example, the facts that a player *joined* and *left* a club, we could derive a time interval for when this player actually *played* for the club. Furthermore, teammates of the player and their corresponding time intervals could be derived as well, which calls for a principled approach to reasoning in temporal knowledge bases with uncertainty. For this purpose, we started building Timely-YAGO (T-YAGO for short) [21], which enriches our previously built knowledge base YAGO [17] by *validity intervals* for facts. Similar to work done on temporal databases [12], validity intervals provide simple, yet effective, support for query semantics built on interval intersections and unions in T-YAGO. Simple interval operations are however only of limited use for query processing (or reasoning) with *uncertainty*, i.e., with probabilistic models or otherwise statistically quantified degrees of uncertainty. In this paper, we adopt the common possible-worlds semantics known from probabilistic databases and extend it towards histogram-like confidence distributions that capture the validity of facts across time. Query processing is done via a Datalog-like, rule-based inference engine, which employs the *lineage* of derived facts for confidence computations to remain consistent with the possible-worlds model.

1.1 Example Setting

Consider the example knowledge base in Figure 1, which illustrates a number of facts about football players, coaches, and their teams. Initially, we are facing the situation where facts with temporal annotations have been extracted from different documents, which might yield different observations for when Beckham

¹ Citations taken from actual news articles

and Ronaldo have joined and left Real Madrid, respectively, each with a different frequency. Then the question arises how these different temporal annotations should be reconciled into a concise representation model. Suppose we are uncertain about the exact time point when Beckham *joined* and *left* Real Madrid. Then, what should be the time interval for when Beckham actually *played* for Real Madrid? Further, we observe that both Beckham and Ronaldo played for Real Madrid. Yet, what is their chance of being *teammates*, and when did they overlap? And what cup did Ronaldo *win* when he played for Real Madrid; or who is the *coach* of the England National Team, and when? Although there are various systems that manage uncertain data, none of them could readily solve the problems stated here. We aim to answer these questions in the following.

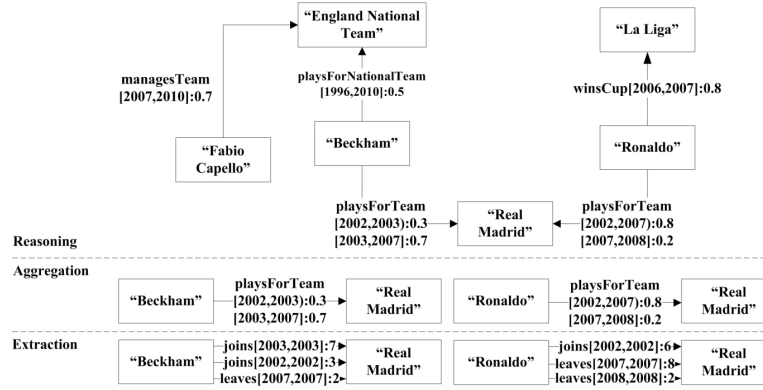


Fig. 1. Example for extracting facts with time annotations and a resulting temporal knowledge base.

1.2 Contributions and Outline

We propose an approach for representing and reconciling facts with temporal annotations for time-aware reasoning over uncertain and potentially inconsistent temporal knowledge bases. We briefly summarize the main contributions of this paper as follows:

- **Closed and Complete Representation Model for Temporal Knowledge Bases.** We develop a histogram-based data model for representing uncertainty about the validity of facts across time. In particular, we distinguish between *event* and *state* relations and show how to combine both into a unified framework for query processing (Section 2).
- **Temporal Fact Extraction with Histogram Aggregation.** We show how to reconcile multiple (potentially inconsistent) observations of facts with temporal annotations into a concise histogram at extraction time (Section 3).
- **Possible-Worlds based Reasoning over Temporal Knowledge Bases.** We employ data lineage in the form of Boolean formulas that capture the logical dependencies between base and derived facts, in a recursive, Datalog-like reasoning environment (Section 4).

- **System and Experiments.** We evaluate our system on a real-world temporal knowledge (Timely YAGO) with more than 270,000 (aggregated) temporal facts, using handcrafted rules for query processing and reasoning in the football domain (Section 5).

2 Data and Representation Model

Knowledge Base. We define a *knowledge base* $\mathcal{KB} = \langle \mathcal{F}, \mathcal{C} \rangle$ as a pair consisting of base facts \mathcal{F} and first-order inference rules \mathcal{C} . In Semantic Web applications, facts are often encoded in the Resource Description Framework (RDF) format, while the Web Ontology Language (OWL)—or more commonly one of its decidable subsets OWL-DL or OWL-lite—is used to express further constraints over the knowledge base. Just like RDF, our set of base facts \mathcal{F} constitutes a directed, labeled multi-graph, in which nodes are entities, and labeled edges represent relationships between the entities. For example, an RDF graph can have an edge between the entity *Beckham* and the entity *Real Madrid*. This edge would be labeled with the relation name *playsForTeam*. More formally, an RDF graph is defined as a set of entities Ent and a set of relations Rel , where every $R \in Rel$ is such that $R \subseteq Ent \times Ent$, and a set of triplets (or facts) $\mathcal{F} \subseteq (Rel \times Ent \times Ent)$. Unlike RDF, we also associate a *time histogram* H_f with each fact $f \in \mathcal{F}$. The time histogram H_f captures the (discrete) probability distribution of f being valid at a particular time point $t \in H_f$.

Inference Rules. We focus on a decidable subset of first-order logic for our inference rules \mathcal{C} . More specifically, we focus on Datalog-like Horn clauses, which can be employed for inferring new facts (i.e., reasoning) at query time. For example, a rule like

$$playsForTeam(x, y) \leftarrow joinsTeam(x, y) \wedge leavesTeam(x, y) \quad (1)$$

can be used to infer that an entity x has played for a particular team y . In the following, we will denote variables by lowercase identifiers and constants by uppercase names (with all variables implicitly being *universally quantified*).

Time Points, Intervals, and Histograms. A time point t denotes a smallest time unit of fixed granularity. We have a discrete series of ordered time points $0, \dots, N$ (with a special designator N which marks the end of the time range we consider). These time points could represent any desired—but fixed—granularity (e.g., years, days, or seconds, or even transaction-based counters).

A *validity interval* is represented as a left-closed, right-open or right-closed, interval, which is bounded by two time points (e.g., $[1990, 2010)$), thus denoting a discrete and finite set of time points. This way, we are able to support both *range-queries* (i.e., “*Is this fact valid in the range of [1999, 2006)?*”) and *snapshot queries* (i.e., “*Is this fact valid at time point 2006?*”). A snapshot query can then simply be seen as a special-case range query by using an interval consisting of just a single time point (e.g., $[2006, 2006]$). Every interval has a corresponding *confidence value* associated with it, which denotes the probability of the fact being valid for the given interval. Multiple, non-overlapping intervals can be concatenated to form a *time histogram*. Intervals in a time histogram do not necessarily have to be contiguous. A gap between two consecutive intervals is equivalent to an interval with a confidence value of 0.

Event and State Relations. In the following, we distinguish between *event* and *state* relations. In an event relation, a fact is valid at *exactly one* time point. By default, facts in an event relation are thus associated with a validity interval consisting of only one time point. For capturing uncertainty, however, validity intervals (and entire histograms) may cover more than one time point, as in the following example:

winsCup(Beckham, ChampionsLeague)[1999, 2001]:0.8

For simplicity, we assume a *uniform* distribution for the probability of a fact within the interval in this case. For example, for the interval [1999, 2001), which covers 2 time points with a confidence of 0.8, each time point in the interval would have a probability of 0.4. The confidences of all intervals (and implicitly also the confidences of the corresponding time points) must form a proper probability distribution, i.e., the sum of all intervals' confidences may be at most 1.

For a state relation, a fact is valid at *every* time point of an interval. Hence, all time points in the interval are (implicitly) associated with the probability of the interval, as in the following example:

playsForTeam(Beckham, United)[1992, 2003):0.3; [2003, 2007):0.4

Here, for the interval [1992, 2003), which covers 12 time points with a confidence of 0.3, the fact is valid at each time point with probability of 0.3; and for the interval [2003, 2007), the fact is valid at each time point with probability of 0.4. For facts in a state relation, the confidences of all intervals must form a proper probability distribution.

For both event and state relations, the sum p of confidences for the intervals in a histogram may be less than 1. In general, a fact is *invalid* for all time points outside the range of time points captured by the histogram with probability $1 - p$. Moreover, different operations for slicing and coalescing intervals apply, depending on whether a fact belongs to either an event or a state relation.

Slicing and Coalescing. In analogy to temporal databases [12], different operations for reorganizing time intervals (and thus histograms) apply. For an *event* relation, we can slice an interval into any set of disjoint and contiguous subintervals by applying our uniformity assumption of confidences. Further, we can coalesce any two contiguous intervals into a single interval, only if the individual time points in both intervals have the same probability. In this case, the confidence of the coalesced interval is the sum of the confidences of the two input intervals. For a *state* relation, however, slicing intervals into subintervals is generally not allowed. Further, we can coalesce any two contiguous subintervals into a single interval, only if they have the same confidence. In this case, the confidence of the coalesced interval in a state relation is the same as the confidence of the two input intervals.

Closed and Complete Representation Model. We remark that this model is a generalization of the possible-worlds data model used in various probabilistic database approaches (see, e.g., [2,5,11]), which now lets us express uncertainty about a fact's validity across time. In particular, this model allows for arbitrary Boolean combinations of both state and event facts for query processing, such that the distribution of confidences of any derived fact is guaranteed

to form a proper probability distribution again (*closedness*). Moreover, any discrete and finite distribution of confidences can be captured by this model, also for both base facts and derived facts (*completeness*). A detailed definition of these operations for query processing is provided in Section 4.

3 Temporal Fact Extraction and Histogram Aggregation

In our temporal model for extraction, each fact is associated with its possible earliest and latest time information. For example, from the sentence “*Beckham signed up for Real Madrid in 2003.*”, we infer that Beckham joined Real in the year 2003. Using *days* as our primary granularity for reasoning, we determine the possible earliest (begin) time point of starting his contract to be *2003-1-1* and the latest (end) time point as *2003-12-31* (using date-formatted time points for better readability). The begin and end time points then constitute an initial time interval *[2003-1-1, 2003-12-31]* for this occurrence (evidence) of the fact *joins(Beckham, Real)* in the document. But then the question arises, how we should reconcile multiple of these (potentially inconsistent) occurrences, which we are likely to observe in different documents during the extraction phase, and how to represent these in a concise histogram for query processing.

Fact	Time Expression	Begin Time	End Time	Frequency	Event Type
<i>joins</i> (<i>Beckham, Real</i>)	“July, 2003”	2003-7-1	2003-7-31	2	<i>begin</i>
	“Summer, 2003”	2003-6-1	2003-9-30	3	
<i>leaves</i> (<i>Beckham, Real</i>)	“June, 2007”	2007-6-1	2007-6-30	1	<i>end</i>
	“Early June, 2007”	2007-6-1	2007-6-10	2	
<i>hasContract</i> (<i>Beckham, Real</i>)	“Season 2003 to 2007”	2003-7-1	2007-6-30	2	<i>during</i>

Table 1. Examples of time expressions and their corresponding intervals.

The extraction stage produces facts which may be valid at both a single time point (e.g., a day or a year) and entire intervals (e.g., multiple days or years). Staying in our football example, we aim to *aggregate* multiple occurrences of such events into a single *state* fact *playsForTeam(Beckham, Real)/[2003-1-1, 2007-12-31]*. This state fact for *playsForTeam* can be inferred, for example, from two event facts *joins(Beckham, Real)/[2003-1-1, 2003-12-31]* and *leaves(Beckham, Real)/[2007-1-1, 2007-12-31]*. Besides events that indicate the *begin* and *end* of an interval, we can also directly extract events that happened *during* the period when Beckham played for Real, such as *hasContract(Beckham, Real)/[2003-7-1, 2007-6-30]*. Table 1 depicts a few examples of time expressions along with their corresponding intervals and possible observation frequencies as they occur at extraction time.

From these facts, we aim to derive the histogram for *playsForTeam(Beckham, Real)*. Notice, that even in case a player might have played for a team multiple times (which occurs frequently), our approach allows for aggregating multiple overlapping occurrences of *begin*, *end* and *during* events into a single histogram.

Merging Observations. Before presenting the forward and backward aggregation of event frequencies into a histogram, we first introduce the basic algorithm for reorganizing the bins of an output histogram, given two or more input histograms, as depicted in Algorithm 1. That is, at each time point where

the confidence of an input histogram changes (i.e., at every interval boundary of an input interval), the confidence in the output histogram may change as well, and a new bin in the output histogram is created. Initially, the input histograms correspond to the basic intervals we extracted for the *begin*, *end* and *during* events (see Table 1).

This (binary) reorganization operation of bins is associative and commutative, hence multiple input histograms can be reorganized into a single output histogram in arbitrary order. Runtime and the number of bins in the output histogram are *linear* in the number of bins in the input histograms. Notice that the smallest-possible width of a histogram bin is a single time point.

Algorithm 1 Reorganizing histograms.

Require: Two input histograms H_1, H_2

Let T be the disjoint union of begin and end time points from intervals in H_1 and H_2 , respectively (in ascending order)

Let H_3 be an empty output histogram

Set $t_b := -1$

For all $t_e \in T$ **do**

If $t_b > -1$

 Insert a new interval $[t_b, t_e)$ into H_3

 Set $t_b := t_e$

Return: H_3

Forward and Backward Aggregation of Frequencies. As we have finished the histogram reorganization from the basic *begin*, *end* and *during* events, we continue to aggregate and normalize the frequencies for our fact in the target relation *playsForTeam*. Intuitively, the confidence of the *playsForTeam* should increase while we aggregate frequencies of intervals that indicate a *begin* event; it should increase at the begin of a *during* interval but decrease at the end of a *during* interval; and it should decrease for intervals relating to *end* events. The amount of occurrences for *begin* and *end* events may however be imbalanced, such that we also need to normalize the frequencies of each of these two types individually, before combining them into a single histogram. To obtain an increasing confidence from *begin* events, we cumulate frequencies of each bin from the first bin to the last bin (forward aggregation). In contrast, to obtain a decreasing confidence from *end* events, we cumulate frequencies of each bin from the last bin to the first one (backward aggregation).

As shown in Figure 2, we first define the reorganized histograms H_1 and H_2 by aggregating the frequencies of all *begin* and *end* events of Table 1 according to their type. Forward aggregation then iterates over all bins of H_1 by cumulating the bins' weights as $H_1[i] = \sum_{0 \leq j \leq i} H_1[j]$, starting with the first bin $H_1[0]$. On the contrary, the backward aggregation iterates over all bins of H_2 by cumulating the bins' weights as $H_2[i] = \sum_{e \geq j \geq i} H_2[j]$, starting from the last bin $H_2[e]$. In the next step, both H_1 and H_2 are normalized to the weight of H_3 , i.e., the aggregated histogram of all *during* events, before all three histograms are again aggregated and normalized to form the final confidence distribution of the *playsForTeam* fact (step 3 in Figure 2). In case no *during* event could be extracted from the sources, an artificial *during* interval with the earliest and latest time points of *begin* and *end* events with weight 1 can be created as H_3 ,

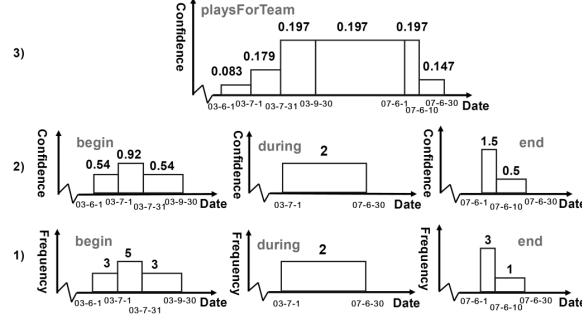


Fig. 2. Reorganizing and merging histograms based on the input facts from Table 1.

in order to normalize H_1 and H_2 . These various levels of aggregation are summarized in Algorithm 2. Figure 2 provides an illustration of these three iterative reorganization and aggregation steps based on the facts in Table 1.

We remark that this aggregation of observation frequencies is just one possible way of deriving an initial histogram at extraction time. In the following, we call facts like $playsForTeam(Beckham, Real)$, which are obtained from such a forward/backward aggregation step, the *base facts*. Confidences in a probabilistic sense are traced back to only those base facts at reasoning time. Further, we assume these base facts to be *independent*.

Algorithm 2 Merging histograms.

Require: Forward-cumulated *begin* histogram H_1 , backward-cumulated *end* histogram

H_2 , and aggregated *during* histogram H_3

Let H_4 be an empty output histogram

Reorganize H_1 , H_2 , H_3 , and H_4 using Algorithm 1

Normalize H_1 and H_2 such that $\sum_i H_1[i] = \sum_i H_3[i]$ and $\sum_i H_2[i] = \sum_i H_3[i]$

For all $i \in H_4$ **do**

Set $H_4[i] := H_1[i] + H_2[i] + H_3[i]$

Normalize H_4 such that $\sum_i H_4[i] = 1$

Return: H_4

4 Rule-based Reasoning, Lineage, and Possible Worlds

Our approach for reasoning in semantic knowledge bases is based on Datalog-like inference rules (Horn clauses), which can be employed to either enforce integrity constraints (Horn clauses with only negated literals) or provide means for actual inference and query answering (Horn clauses with exactly one positive literal). Recall that Horn clauses with exactly one positive literal can equivalently be rewritten as implications, where the positive literal becomes the head of the rule and the body is a conjunction of the remaining literals. Our key observation is that the logical dependencies of query answers (i.e., the *possible worlds* the entire knowledge base can take) are determined only by the way rules were processed in order to ground the query atoms (potentially recursively) down to the base facts.

In this paper, we focus on the case of Horn clauses with exactly one positive head literal, because it results in Boolean formulas with positive (i.e., conjunctive or disjunctive) lineage only.

Temporal Predicates. For reasoning about time intervals, we employ additional *temporal predicates* such as *overlaps*, *before*, *after*, etc. (see, e.g., Allen et al. [7] for an overview of temporal relations among intervals). These temporal predicates allow us to constrain the temporal relationships of time-annotated facts in the rules. Within the formulation of a rule, we also extend the given (binary) predicates by a third time variable t which is used as reference when reasoning with the temporal predicates (see Rules (2) and (3)). While this extension clearly remains in first-order logic, it—strictly speaking—no longer conforms with the core RDF data model.

Queries. Queries in Datalog can be expressed as Boolean combinations of literals (again, we do not allow negation). Hence, $teammates(Beckham, x)$ would retrieve all teammates of Beckham, while $teammates(x, y)$ would denote all pairs of teammates that could be inferred from the knowledge base. Literals in queries are grounded against the knowledge base. Semantically, a disjunction of two literals relates to a disjoint union of two sets of facts (obtained from grounding each literal), while a conjunction relates to a set intersection. Set operations in these reasoning settings are always duplicate eliminating.

Conjunctive vs. Disjunctive Lineage. When processing a query, predicates in the body of an inference rule are combined conjunctively, while multiple rules with the same head predicate create a disjunctive derivation of the query answer. In analogy to probabilistic databases, processing the body of a rule thus conforms to a join operation with conjunctive lineage, whereas grounding the same derived fact from multiple rules conforms to a duplicate-elimination step with disjunctive lineage [2, 16]. We thus adopt a similar notion of data lineage as in [2] to compute the individual confidences of bins in the time histogram of a derived fact. In a Datalog-like setting, however, rules are potentially recursive, such that the derivation of answers typically is less uniform than for a regular SQL query or materialized view. Lineage however remains acyclic also in our setting, because all rules are grounded against base facts to find valid answers.

ID	Fact	Histogram	Relation Type
F_1	$playsForTeam(Beckham, Real)$	$[2003, 2008]:0.8$	state
F_2	$playsForTeam(Ronaldo, Real)$	$[2002, 2008]:0.7$	state
F_3	$winsCupForTeam(Ronaldo, Real)$	$[2003, 2004]:0.6$	event

Table 2. Base facts with time histograms (intervals).

$$teammates(x, y) \leftarrow playsForTeam(x, z, t_1) \wedge playsForTeam(y, z, t_2) \wedge notEquals(x, y) \wedge overlaps(t_1, t_2) \quad (2)$$

$$teammates(x, y) \leftarrow playsForTeam(x, z, t_1) \wedge winsCupForTeam(y, z, t_2) \wedge notEquals(x, y) \wedge overlaps(t_1, t_2) \quad (3)$$

As an example, consider we want to retrieve the probability of Beckham and Ronaldo being teammates for Rules (2) and (3) and the base facts depicted in Table 2. We will next discuss how confidence computation works in this setting.

Confidence Computation. While grounding queries via rules yields exactly one Boolean lineage formula for a derived fact, the input confidences of base facts may vary across time. Hence our algorithm needs to ensure that the correct confidences are chosen as input when calculating the confidence of a result histogram. This is achieved via reorganizing the bins of the output histogram using Algorithm 1 and slicing and coalescing the input intervals of base facts belonging to an event relation accordingly. Notice that intervals from base facts belonging to a state relation do not have to be sliced, since a fact is defined to be valid at each time point of an interval with the same probability (see Section 2).

Thus, grounding the query $teammates(Beckham, x)$ over the above Rules (2) and (3) and base facts depicted in Table 2 results in the (single) grounded query answer $teammates(Beckham, Ronaldo)$ with lineage $(F_1 \wedge F_2) \vee (F_1 \wedge F_3)$. However, by simply multiplying the probability of each literal in the lineage of $teammates(Beckham, Ronaldo)$, we would get $0.8 \times 0.7 \times 0.8 \times 0.6 = 0.2688$. This is not correct, since the probability of $playsForTeam(Beckham, Real)$ is considered twice. Assuming independence among base facts, we can calculate the correct probability of $teammates(Beckham, Ronaldo)$ for the interval [2003, 2004] as $0.8 \times 0.7 \times 0.6 + 0.8 \times 0.7 \times (1 - 0.6) + 0.8 \times (1 - 0.7) \times 0.6 = 0.704$ (as can be verified by a truth table). For simplicity, we show the confidence computation only for a single interval. In general, one such computation can be triggered for each bin of a time histogram, again using Algorithm 1 for reorganizing the histogram, but with a possible-worlds-based confidence computation instead of the simple aggregation of Algorithm 2.

Our approach for confidence computations with time histograms can thus be summarized into the following two steps:

- 1) reorganizing bins of the output histogram using Algorithm 1, and
- 2) computing the confidence for a fact's validity at each bin of its histogram.

While step 1) is linear in the number of input bins, each confidence computation per output bin is #P-complete for general Boolean formulas [15]. We thus employ the Luby-Karp family of sampling algorithms for approximating the confidence computation. Different versions for Luby-Karp sampling [13] are available, depending on whether the formula is in CNF, DNF, or of generic Boolean shape, each with different approximation guarantees. Thus, as a simple optimization, our implementation is able to check for the structure of the formulas at query time, and it can select the most appropriate variant of Luby-Karp, or even an exact confidence computation if this is still feasible.

In our current implementation, lineage is transient, i.e., we keep lineage information only in memory at query processing time. For future work, we aim to investigate also making lineage persistent, thus being able to “learn” new facts from existing facts in the knowledge base and storing these derived facts along with their derivation in the knowledge base for further processing and faster subsequent inference.

5 System Setup and Experiments

Our system is implemented as an extension of URDF [18], which is a framework for efficient reasoning over uncertain RDF knowledge bases developed at the Max

Planck Institute for Informatics. URDF employs SLD resolution for grounding first-order formulas (Horn clauses) against an underlying knowledge base. Unlike most Datalog engines, URDF follows a top-down grounding approach, i.e., for an incoming query, it aims to resolve answers by processing rules recursively from the head predicate down to the body predicates, which are conjunctions of predicates found either in the knowledge base or which can in turn be processed via the head predicate of a rule. URDF is implemented in Java 1.6 with about 4,000 lines of code. All experiments were run on an Intel Xeon 2.40GHz server (in single-threaded mode) with 48GB RAM. We use Oracle 10g as backend for storing the T-YAGO knowledge base, which was installed on a second AMD Opteron 2.6 GHz server with 32GB RAM.

As competitors we employ the original URDF framework (without the temporal extension) and the IRIS [3] reasoner, a default reasoning engine used in many Semantic Web applications. In terms of reasoning, IRIS [3] is an open-source Datalog engine supporting built-in predicates. It is designed to be highly configurable, allowing for different Datalog evaluation strategies and the definition of custom data types and predicates.

5.1 Timely YAGO Knowledge Base

Our experiments are based on the semantic graph of T-YAGO [21]. For T-YAGO, we extracted more than 270K temporal facts from Wikipedia and freetext, with 16 distinct relationship types. Currently it covers the football domain, including relationships such as *playsForSeniorClub*, *participatedIn* and *winsCup*, but also raw facts for the *begin*, *end*, and *during* events of these relations, such as *joinsSeniorClub* or *leavesSeniorClub*. These raw facts can be integrated with the existing facts of the corresponding relations (e.g., *playsForSeniorClub*), in order to reconcile time histograms using the aggregation rules depicted in Table 4.

The facts and time histograms are stored in two separate tables. The **facts** table contains three columns for RDF triplets (i.e., first argument, second argument, and relation name) and a column for the fact id. The **time** table is composed of two columns (i.e., start time point and end time point) corresponding to the begin and end time point of an interval, a foreign key connecting to the fact's id, and a column for the fact's confidence at the interval.

5.2 Rules and Queries

Table 4 depicts 4 aggregation rules for reasoning about the time interval of a player's or coach's career period, as well as 9 partly recursive, hand-crafted inference rules for reasoning about people's activities and relationships in the football domain. As URDF (without time) and IRIS do not support time-aware reasoning, we remove all temporal predicates in the inference rules, such as *overlaps* or *after*, when comparing their runtimes and results. Table 4 illustrates 8 queries including single-fact queries, chains, stars and cliques of interconnected facts used as our baseline for experiments.

5.3 Experimental Results

Our experiments focus on investigating the overhead of time-aware query processing, compared to a time-oblivious setting. We compare the running times and

result precision of URDF (with time) to IRIS and URDF (without time). The running time of URDF (with time) includes grounding time (using SLD resolution) and histogram creation (i.e., possible-worlds-based histogram calculation time).

Baseline Runs Without Time. Since IRIS and URDF (without time) do not support time-aware reasoning, we compare grounding time and result precision of IRIS to URDF (without time) in the first experiment. The grounding time in URDF (without time) denotes the time to ground the query atoms, using the inference rules and queries depicted in Table 4. The measured time in IRIS is the time required to ground the query using magic sets rewriting, which includes both the rule rewriting step followed by a bottom-up query evaluation over the rewritten rules. We can see that URDF already outperforms IRIS for the grounding time (both without using time-specific predicates).

Overhead of Confidence Computations with Histograms. In the second experiment, we compare the grounding time and result precision of URDF (without time) to URDF (with time). Besides the grounding time consumed by URDF (without time), URDF (with time) also includes the possible-worlds-based histogram computation time. A comparable confidence computation for facts with just a single confidence value but without a time histogram is also shown on the left-hand side for URDF (without time).

Interestingly, Table 3 shows that URDF (with time) even partly achieves better runtimes than URDF (without time) for complex queries, because URDF (with time) does not ground any answers that do not satisfy the temporal predicates. This is also the main reason for the lower precision of URDF (without time) compared to URDF (with time). The grounding time of URDF (with time) is better than URDF (without time) for Queries 4, 5, 7 and 8, even when taking also the time for building the final histogram into account. However, the time for building the histogram for Query 6 is much worse than the others, yielding 14 results with 6,552 literals in 504 disjunctions in their lineage. Also, only for Query 6 we needed to employ Luby-Karp-based sampling (using $\varepsilon = 0.05$ and $\delta = 0.05$), while all the other confidences could be computed exactly.

	Without time information				With time histograms			T-URDF/URDF precision
	IRIS ms	URDF ms	PWs-conf ms	# results	T-URDF ms	PWs-conf ms	# results	
Q_1	6893	35	2	8	45	<1	8	8/8
Q_2	821	11	<1	5	12	<1	5	8/8
Q_3	7127	1905	1191	766	2113	1	184	184/766
Q_4	6686	699	188	239	308	5	58	58/239
Q_5	7628	3099	314	190	1423	51	114	114/190
Q_6	4317	693	20345	14	1054	87600	14	8/8
Q_7	6909	6712	574	183	3277	5	17	17/183
Q_8	7125	6396	190	133	4441	1	25	25/133
Σ	47506	19550	<22805	1538	12673	<87665	425	avg=0.545

Table 3. Experimental results.

6 Related Work

Temporal reasoning has a fairly long history through works in logics and AI, most notably in the seminal work by Allen et al. [7]. To the best of our knowledge, our

approach is the first to integrate reasoning, temporal probabilistic RDF data, and lineage. Recently, there has been an effort to expand information extraction along the temporal dimension, coined T-YAGO [21]. T-YAGO extends the rule-based approach used for extracting YAGO [17] to temporal facts from infoboxes and category information in Wikipedia, resulting in fairly large collections of facts with temporal annotations presented using the RDF data model. In [14], a pre-trained probabilistic model is used to extract temporal information from natural language sentences and annotate facts with time intervals. However, this approach does not support reasoning about relations or query answering. Work on temporal databases dates back to the early 1980's [12]. Different semantics for associating time with facts have been defined. In the context of this paper, we use the valid-time semantics, where the time recorded for facts in a database captures the reality of the world being modeled by the database. Extensions for traditional data models have been explored to accommodate temporal data in an efficient manner, both in terms of space and query processing. There has also been an extensive effort to develop query languages for querying temporal data. Most of these efforts were attempts to modify SQL to reduce the complexity of temporal queries. There is a wealth of research on probabilistic databases and the management of uncertain data. [11] is a state-of-the-art probabilistic database management system achieving scalability. [2,16] present a framework for dealing with uncertain data and data lineage. This approach allows for the decoupling of data and confidence computations when processing queries over uncertain data [16], allowing for a wider range of query plans to be used while still maintaining the correctness of confidence computations. For dealing with probabilistic reasoning in the context of information retrieval, [8] presents a probabilistic version of Datalog, which is one of the first works to introduce a notion of intensional query semantics. [9,10,19] present a probabilistic extension to RDF and how SPARQL queries over such an extension can be supported. However, no notion of temporal reasoning has been considered in these contexts.

7 Conclusions

We believe that adding time to a knowledge base is a crucial component for high-precision query answering. Time-aware information extraction increases the demand for coping with imprecise or otherwise uncertain data and is an excellent showcase for uncertain data management. Moreover, in our approach, we show that adding time histograms involves only a light overhead over a comparable probabilistic setting that does not consider time. Time-aware reasoning may even spare unnecessary computations for false-positive answers at an early stage and thus reduce the overall runtime for query answering. Currently, the way we aggregate occurrence frequencies into our initial time histograms for the base facts still is fairly abrasive from a probabilistic point of view. Our long-term goal thus is to find appropriate generative models which allow for incorporating the actual occurrences of facts in the documents into the probabilistic interpretation. The temporal extension however is already fully integrated into our URDF reasoning framework, which provides a unified and versatile reasoning platform, including, for example, also spatial reasoning extensions.

APPENDIX: Rules and Queries

Aggregation Rules

- $A_1: \text{joinsYouthClub}(a, b) \wedge \text{duringYouthClub}(a, b) \wedge \text{leavesYouthClub}(a, b)$
 $\rightarrow \text{playsForYouthClub}(a, b)$
 $A_2: \text{joinsSeniorClub}(a, b) \wedge \text{duringSeniorClub}(a, b) \wedge \text{leavesSeniorClub}(a, b)$
 $\rightarrow \text{playsForSeniorClub}(a, b)$
 $A_3: \text{joinsNationalTeam}(a, b) \wedge \text{duringNationalTeam}(a, b) \wedge \text{leavesNationalTeam}(a, b)$
 $\rightarrow \text{playsForNationalTeam}(a, b)$
 $A_4: \text{beginManagesTeam}(a, b) \wedge \text{duringManagesTeam}(a, b) \wedge \text{endManagesTeam}(a, b)$
 $\rightarrow \text{managesTeam}(a, b)$

Inference Rules

Players playing for teams are summarized into *playsForTeam*.

- $C_1: \text{playsForYouthClub}(a, b) \rightarrow \text{playsForTeam}(a, b)$
 $\text{playsForSeniorClub}(a, b) \rightarrow \text{playsForTeam}(a, b)$
 $\text{playsForNationalTeam}(a, b) \rightarrow \text{playsForTeam}(a, b)$
 If two players play for the same team at the same time, they are teammates.
 $C_2: \text{playsForTeam}(a, b, t_1) \wedge \text{playsForTeam}(c, b, t_2) \wedge \text{overlaps}(t_1, t_2) \wedge \text{notEquals}(a, c)$
 $\rightarrow \text{teammates}(a, c)$
 If one player plays for the same team after another player, then the former is a successor of the latter.
 $C_3: \text{playsForTeam}(a, b, t_1) \wedge \text{playsForTeam}(c, b, t_2) \wedge \text{after}(t_1, t_2) \wedge \text{notEquals}(a, c)$
 $\rightarrow \text{successor}(a, c)$
 If one player plays for the same team before another player, then the former is an ancestor of the latter.
 $C_4: \text{playsForTeam}(a, b, t_1) \wedge \text{playsForTeam}(c, b, t_2) \wedge \text{before}(t_1, t_2) \wedge \text{notEquals}(a, c)$
 $\rightarrow \text{ancestor}(a, c)$
 Players who have played for more than 1460 days (more than 4 years) for a team.
 $C_5: \text{playsForTeam}(a, b, t_1) \wedge \text{durationMoreThan}(t_1, 1460)$
 $\rightarrow \text{playedMoreThan4YearsForTeam}(a, b)$
 If a coach manages the team when a player is playing for the team, the coach trained this player.
 $C_6: \text{managesTeam}(a, b, t_1) \wedge \text{playsForTeam}(c, b, t_2) \wedge \text{overlaps}(t_1, t_2)$
 $\rightarrow \text{isCoachOf}(a, c)$
 If a coach manages a team, and this is a national team, then he is a coach of a national team.
 $C_7: \text{managesTeam}(a, b, t_1) \wedge \text{playsForNationalTeam}(c, b, t_2) \wedge \text{overlaps}(t_1, t_2)$
 $\rightarrow \text{isCoachOfNationalTeam}(a, b)$

Queries

Single-fact queries:

For which teams (and when) did David Beckham play?

$Q_1: \text{playsForTeam}(\text{DavidBeckham}, x)$

Which teams (and when) has Alex Ferguson managed?

$Q_2: \text{managesTeam}(\text{AlexFerguson}, x)$

Who are the ancestors of David Beckham?

$Q_3: \text{ancestor}(x, \text{DavidBeckham})$

Chain queries:

Who are the coaches of David Beckham, and which teams did they previously play for?

$Q_4: \text{isCoachOf}(x, \text{DavidBeckham}) \wedge \text{playsForTeam}(x, y)$

Who are teammates of David Beckham, who participated in the same activity as Zinedine Zidane?

$Q_5: \text{teammates}(\text{DavidBeckham}, y) \wedge \text{participatedIn}(y, z) \wedge \text{participatedIn}(\text{ZinedineZidane}, z)$

Star queries:

Who are the coaches of the England National Football Team, what cups did they win, and which activities did they join?

$Q_6: \text{isCoachOfNationalTeam}(x, \text{EnglandNationalFootballTeam}) \wedge \text{winsCup}(x, y)$
 $\wedge \text{participatedIn}(x, z)$

Who played for Manchester United for more than 4 years and was a teammate of David Beckham?

$Q_7: \text{playedMoreThan4YearsForTeam}(x, \text{ManchesterUnited}) \wedge \text{teammates}(x, \text{DavidBeckham})$

Clique query:

Who are the successors of David Beckham who won the same cup as Beckham?

$Q_8: \text{successor}(x, \text{DavidBeckham}) \wedge \text{winsCup}(x, z) \wedge \text{winsCup}(\text{DavidBeckham}, z)$

Table 4: Aggregation rules, inference rules, and queries used for the experiments.

References

1. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: DBpedia: A nucleus for a web of open data. In: ISWC (2007)
2. Benjelloun, O., Sarma, A.D., Halevy, A.Y., Theobald, M., Widom, J.: Databases with uncertainty and lineage. VLDB J. 17(2) (2008)
3. Bishop, B., Fischer, F.: IRIS- Integrated rule inference system (2008)
4. Carlson, A., Betteridge, J., Wang, R.C., Jr., E.R.H., Mitchell, T.M.: Coupled semi-supervised learning for information extraction. In: WSDM (2010)
5. Dalvi, N.N., Suciu, D.: Efficient query evaluation on probabilistic databases. VLDB J. 16(4) (2007)
6. Etzioni, O., Cafarella, M., Downey, D., Kok, S., Popescu, A.M., Shaked, T., Soderland, S., Weld, D.S., Yates, A.: Web-scale information extraction in KnowItAll. In: WWW (2004)
7. Fisher, M., Gabbay, D.M., (Eds.), L.V.: Handbook of temporal reasoning in artificial intelligence. Elsevier (2005)
8. Fuhr, N.: Probabilistic datalog - a logic for powerful retrieval methods. In: SIGIR (1995)
9. Fukushima, Y.: Representing probabilistic relations in RDF. In: ISWC-URSW (2005)
10. Huang, H., Liu, C.: Query evaluation on probabilistic RDF databases. In: WISE (2009)
11. Huang, J., Antova, L., Koch, C., Olteanu, D.: MayBMS: a probabilistic database management system. In: SIGMOD Conference (2009)
12. Jensen, C.S., Snodgrass, R.T.: Temporal data management. IEEE Trans. on Knowl. and Data Eng. 11(1) (1999)
13. Karp, R.M., Luby, M.: Monte-carlo algorithms for enumeration and reliability problems. In: FOCS. pp. 56–64 (1983)
14. Ling, X., Weld, D.S.: Temporal information extraction. In: AAAI'10. AAAI Press (2010)
15. Re, C., Dalvi, N.N., Suciu, D.: Efficient top-k query evaluation on probabilistic data. In: ICDE (2007)
16. Sarma, A.D., Theobald, M., Widom, J.: Exploiting lineage for confidence computation in uncertain and probabilistic databases. In: ICDE (2008)
17. Suchanek, F.M., Kasneci, G., Weikum, G.: Yago
18. Theobald, M., Sozio, M., Suchanek, F., Nakashole, N.: URDF: Efficient reasoning in uncertain RDF knowledge bases with soft and hard rules. Tech. Rep. MPI-I-2010-5-002, Max Planck Institute Informatics (MPI-INF) (2010)
19. Udea, O., Subrahmanian, V.S., Majkic, Z.: Probabilistic RDF. In: IRI (2006)
20. Verhagen, M., Mani, I., Sauri, R., Knippen, R., Jang, S.B., Littman, J., Rumshisky, A., Phillips, J., Pustejovsky, J.: Automating temporal annotation with TARSQI. In: ACL (2005)
21. Wang, Y., Zhu, M., Qu, L., Spaniol, M., Weikum, G.: Timely YAGO: harvesting, querying, and visualizing temporal knowledge from wikipedia. In: EDBT (2010)
22. Wu, F., Weld, D.S.: Automatically refining the wikipedia infobox ontology. In: WWW (2008)
23. Yates, A., Banko, M., Broadhead, M., Cafarella, M.J., Etzioni, O., Soderland, S.: TextRunner: Open information extraction on the web. In: HLT-NAACL (2007)

Query Containment for Databases with Uncertainty and Lineage

Foto N. Afrati and Angelos Vasilakopoulos

National Technical University of Athens
afrati@softlab.ece.ntua.gr, avasila@central.ntua.gr

Abstract. We define and investigate the computational complexity of the query containment problem for data that support both uncertainty and lineage. Query containment depends on the definition of database containment which, for traditional databases, is defined as a simple *set containment* for each relation. As this is not the case in the presence of uncertainty and lineage, we revisit the notion of database containment and define various kinds of it that may be natural in different practical situations. We investigate conjunctive query containment for the various kinds of query containment that we introduce.

Key words: Uncertainty, Lineage, Query Containment

1 Introduction

Uncertain data appears in many modern applications including information extraction from the web, bio-informatics, scientific databases, entity resolution and sensors. Those and many other applications also require keeping track of the derivation of data, called *provenance* or *lineage*. There has been a lot of recent research that considers systems managing data with uncertainty [1, 8, 16, 19, 21], systems managing data with lineage tracking [10, 11, 14, 15, 18] and systems that combine data with uncertainty and lineage [9]. Semantics and algorithms for computing queries have been defined in those systems. To the best of our knowledge the problem of query containment has not been considered for a database system that handles uncertain data and also supports lineage tracking. We are going to investigate different kinds of query containment for the ULDB (Uncertainty Lineage DataBase) data model [9] that will be based on different semantics of database containment for this model.

The problem of query containment arises in many important database applications like query optimization [5], data integration, query answering using views [3, 4, 22], data exchange and data warehousing. One of the reasons that the ULDB model was introduced was because it would be important in data integration and data exchange settings. In addition query optimization is recognized as one of the important open problems of the Trio system. It is already known from ordinary databases that both these problems rely on query containment.

A database query Q defines a mapping from databases to databases. A query Q_1 is said to be contained in a query Q_2 if for every database D , database

$Q_1(D)$ is contained in database $Q_2(D)$. For ordinary databases, a database D_1 is contained in a database D_2 if the tuples of every relation in D_1 are contained in the corresponding relation of D_2 *as a set*. A relation of an uncertain database however semantically is not a set; it represents a set of *possible instances* - *PIs* (that have no uncertainty). The answer of a query over an uncertain database is a new uncertain database. Thus, when moving to ULDBs or even to uncertain databases without lineage, the set containment between answers of queries no longer applies.

On the ULDB model a database consists of uncertain tuples. An uncertain tuple called *x-tuple* consists of a bag of tuples called *alternatives*. At most one of those alternatives can exist in a possible instance. In particular, apart from their data, alternatives also consist of: i) a unique identifier and ii) a lineage function that connects them to other alternatives through the set of their unique identifiers. As a result, ULDB database containment should not only consider data containment and it should also be based on the possible instances of a ULDB. Query containment between queries Q_1 and Q_2 will be based on ULDB database containment between the ULDB relations that are the answers of the two queries.

In this paper we start our investigation with considering various kinds of ULDB database containment. For each type of ULDB database containment we define corresponding kinds of ULDB query containment, give conjunctive query containment test and study its complexity. We also investigate in which cases each kind of containment is more suitable. In Figure 1 we list the various kinds of containment and show the relation between them. We will refer in Figure 1 in more detail in section 7, where we will also discuss that conjunctive query (CQ) equivalence for ULDB databases can result as both ways containment for the three last semantics in the figure. Even though the database semantics for those kinds of containment are different it will turn out that some of them are equivalent for conjunctive query containment. In [7] another notion of containment has been defined for uncertain databases without lineage. It was defined in order to be suitable for data integration purposes. We prove which conditions should hold for query containment under this kind as well.

The need for defining database containment in the presence of uncertainty and/or lineage has been noticed in [7], [9]. For databases with lineage, containment has been defined in various works: In [9] for the model of *LDBs* (Lineage DataBases) and in [17] for various kinds of semirings. In [17] database containment for databases with lineage is also used to study the complexity of computing conjunctive queries. In [20] query equivalence and containment is investigated for probabilistic data.

One obvious kind of ULDB containment is based on the LDB containment defined on Trio which requires the containment of lineage in the transitive closure of lineage of the containing relation (Semantics #3 in Figure 1). We show why this kind of ULDB containment may be inappropriate for some cases. Hence we relax this definition and yield a new kind of LDB containment (Data Containment) that requires the set containment of data of every possible instance

of a ULDB (Semantics #1 in Figure 1). Further we define two kinds of database containment that take into account only *base lineage* (lineage extended and referring only to alternatives with empty lineage, called *base alternatives*). The first kind requires *containing* base lineage (Semantics #2) and the second (Semantics #4 - also discussed and used in [6] in the context of Data Exchange) requires *same* base lineage of the data that is contained between two databases. The fifth database containment semantics will require the containment of all data and lineage, not only base (Semantics #5).

Query containment for ordinary databases is known to be NP-complete for conjunctive queries [12]. A ULDB represents a set of possible instances that are LDBs and whose number can be exponential to its size. In addition lineage imposes complex logical formulas to alternatives that can be true on each possible instance. Furthermore a possible instance is an LDB and contains a bag of tuples (if they have different lineage). Query containment under bag semantics for CQs is known to be Π_2^P -hard [2, 13]. So we would expect ULDB query containment to be harder than ordinary set or bag query containment. In contrast we show that ULDB query containment is NP-complete, for all the different kinds of containment which we study.

#	Semantics	Features	Implies DB cont.	CQ Containment Test
1	Data	Set Contained Data	—	Containment Mapping
2	CBase-Lineage	Contained Base Lineage	1	Containment Mapping
3	TR-Lineage	Contained Transitive Closure of Lineage	1	Onto Mapping
4	SBase-Lineage	Same Base Lineage	1,2	Onto Mapping
5	Same-Lineage	Same Lineage	1,2,3,4	Onto Mapping

Fig. 1. Comparison of Different Semantics

2 Running Example and the ULDB Data Model

In this section we present a motivating example illustrating the need of defining new kinds of database containment, suitable for query containment. Through this example we will also quickly explain the Trio ULDB model.

Basic notions of the ULBD model illustrated through a ULDB instance: An uncertain database with lineage (ULDB) represents a set of *possible instances* which are databases with lineage (for more detailed definitions see [9]).

Consider an uncertain database with lineage (ULDB) U containing information about names of persons who drive cars, stored in relation **Drives**(**name**, **car**) and about names of witnesses that saw a car near a crime-scene, stored in relation **Saw**(**witness**, **car**)¹. Data in **Drives** contain uncertainty (e.g., due to unclear writing). Suppose we have uncertainty whether *John* drives a *Honda* or a *Mazda* car. In the ULDB model this kind of uncertainty is represented through x-tuples. In general an x-tuple is a bag of ordinary tuples which we call *alternatives* and we separate them with symbol ‘||’. The semantics of alternatives in x-tuples are that at most one of them can be true in a *possible instance* (PI). If we can have a possible instance that selects none of the alternatives of an x-tuple, then we annotate this x-tuple with ‘?’ symbol. Figure 2 depicts the two relations **Drives** and **Saw** in our running example. We have an x-tuple with two alternatives (*John*, *Honda*) and (*John*, *Mazda*), whereas we are sure that *Kate* drives a *Honda* and a *Toyota* car. Data in relation **Saw** have also uncertainty. But they have also lineage: Suppose that *Cathy* said that she is certain that she saw *John* driving the car. As a result if *John* drives a *Honda* then *Cathy* saw a *Honda*. We encode this relation through lineage: e.g., the lineage of alternative (*Cathy*, *Honda*) will contain alternative (*John*, *Honda*).

In order to succinctly represent lineage connections between alternatives we attach to each x-tuple a unique *identifier*. For example, in Figure 2, x-tuple (*John*, *Honda*)||(*John*, *Mazda*) in **Drives** has identifier 11. If the identifier of an x-tuple is i , then we refer to its j -th alternative with an *alternative identifier* which will be a pair (i, j) . We represent the lineage connection between alternatives (*Cathy*, *Honda*) and (*John*, *Honda*) with a lineage function λ that connects alternative identifiers to sets of alternative identifiers, e.g.,: $\lambda(21, 1) = \{(11, 1)\}$. *Base data* of a ULDB instance consists of all data that have empty lineage. If two alternatives point, maybe after many lineage steps, to the same set of base data we say that they have the same *base lineage*.

Possible Instances (PI) of a ULDB: We note that lineage does not just track where data comes from, but also poses logical restrictions to the possible LDB instances that a ULDB represents. An alternative of an x-tuple in the answer of a conjunctive query can be true in a possible instance if and only if all the alternatives in its lineage also appear in this possible instance. x-tuples with non-empty lineage do not alter the number of possible instances; this number is determined solely from base data. In our example U has two possible instances: one for each possible alternative selection of x-tuple 11.

Computing Queries over ULDBs: Until now lineage was used to logically connect alternatives in the answer of a query to alternatives that produced this

¹ The general setting of our example is similar to the running example found in [9].

ID	Drives(name,car)	ID	Saw(witness,car)
11	John,Honda John,Mazda	21	Cathy,Honda Cathy,Mazda
12	Kate,Honda	22	Amy,Mazda
13	Kate,Toyota		$\lambda(21,1) = \{(11,1)\}$ $\lambda(21,2) = \{(11,2)\}$

Fig. 2. Running Example: ULDB U

ID	R_{Q_1}	ID	R_{Q_2}
31	John John	41	John John
32	John '?'	42	Kate
33	Kate '?'	43	Kate

$\lambda(31,1) = \{(11,1), (21,1)\}$
 $\lambda(31,2) = \{(11,2), (21,2)\}$
 $\lambda(32,1) = \{(11,2), (22,1)\}$
 $\lambda(33,1) = \{(12,1), (21,1)\}$

Fig. 3. ULDB Relation R_{Q_1} for:
 $Q_1(x) : -Drives(x,y), Saw(z,y)$

$\lambda(41,1) = \{(11,1)\}$
 $\lambda(41,2) = \{(11,2)\}$
 $\lambda(42,1) = \{(12,1)\}$
 $\lambda(43,1) = \{(13,1)\}$

Fig. 4. ULDB Relation R_{Q_2} for:
 $Q_2(x) : -Drives(x,y)$

answer. In our example we extend lineage's use in expressing logical connections between data that is known to be related (for example, in Figure 2 the lineage of alternative (21,1) is (11,1)). Suppose we perform a natural join of **Drives** and **Saw** over common attribute *car* and a projection of attribute *name* on the result. Conjunctive query $Q_1(x) : -Drives(x,y), Saw(z,y)$ performs this operation. Intuitively Q_1 will return the names of suspects, i.e., persons who drive a car that was seen near the crime-scene. The result of computing Q_1 over ULDB U is a new ULDB $Q_1(U)$ that includes ULDB relations **Saw** and **Drives** of U and a new ULDB relation R_{Q_1} which is shown in Figure 3. The semantics of $Q_1(U)$ are that its possible instances should be the same with the possible instances we would have if we first considered the possible instances of U and computed Q_1 over each one of them.

ID	Drives(name,car)	ID	Saw(witness,car)
11,2	John,Mazda	21,2	Cathy,Mazda
12,1	Kate,Honda	22,1	Amy,Mazda
13,1	Kate,Toyota		$\lambda(21,2) = \{(11,2)\}$

Fig. 5. D_1 : One Possible Instance of ULDB U

One of the two possible instances of ULDB U is shown in Figure 5. The corresponding possible instance D_{11} of $Q_1(U)$ consists of this LDB possible instance of U along with LDB relation $D_{Q_{11}}$ shown in Figure 6, which is the corresponding possible instance of relation R_{Q_1} . Note that a possible instance of a ULDB is an LDB, so it does not have uncertainty (no alternatives separated with $||$), but only ordinary tuples with unique tuple identifiers and lineage. Since we have no alternatives, LDB tuples are always present and their unique identifiers can be single numbers and not pairs. For uniformity in the LDB possible instances of our ULDBs we use alternative identifier pairs to identify a tuple.

ID	R_{Q_1}
31,2	John
32,1	John

$$\lambda(31, 1) = \{(11, 2), (21, 2)\}$$

$$\lambda(32, 1) = \{(11, 2), (22, 1)\}$$

Fig. 6. $D_{Q_{11}}$: One Possible Instance of ULDB relation R_{Q_1} for:
 $Q_1(x) : -Drives(x, y), Saw(z, y)$

ID	R_{Q_2}
41,2	John
42,1	Kate
43,1	Kate

$$\lambda(41, 2) = \{(11, 2)\}$$

$$\lambda(42, 1) = \{(12, 1)\}$$

$$\lambda(43, 1) = \{(13, 1)\}$$

Fig. 7. $D_{Q_{21}}$: Corresponding Possible Instance of ULDB relation R_{Q_2} for: $Q_2(x) : -Drives(x, y)$

Non TR-lineage query containment: As we mentioned we define first variants for ULDB database containment and then we base our variants for query containment on these definitions. It turns out that, for conjunctive queries our variants fall in only two classes, where the semantics in each class are shown to be CQ-containment equivalent, in that if two CQs are contained in each other according to one semantics in the class, then they are also contained according to the other semantics in the same class. Hence, the question arises whether the two classes have a meaningful distinction wrto CQ query containment. In this subsection, we take the TR-Lineage semantics, the Data and CBase containment semantics and show that the last two may be more desirable in certain situations.

Consider a second query $Q_2(x) : -Drives(x, y)$. The possible instance $D_{Q_{21}}$ of relation R_{Q_2} (shown in Figure 4) that refers to the same possible instance with $D_{Q_{11}}$ is the LDB $D_{Q_{21}}$ shown in Figure 7. The answer of Q_2 will return the names of persons who drive a car (not only of suspects). On the other hand a suspect's name which will be in the answer of Q_1 should drive a car in order to appear in the answer of Q_1 . According to the semantics of computing queries over a ULDB a suspect's name appears as driving a car in relation **Drives** and hence in the answer of Q_2 in every possible instance where this name appears as a suspect in the answer of Q_1 .

Now we also consider another kind of containment that is based on base lineage. In our example the instances of relations R_{Q_1} and R_{Q_2} (shown in Figures

6,7), referring to the possible instance which selects (11,2) from the base data, both contain a tuple with data *John*. This is true because there exists tuples *John* in both instances that also have base lineage pointing to (11,2), even though the base lineage of the contained R_{Q_1} also includes (22,1).

Hence it is natural to define ULDB containment by requiring data containment in the possible instances, a notion which we adopt in Semantics #1, or by requiring set containment of the base lineage from the containing to the contained database, a notion which we adopt in Semantics #2. Observe that if we adopt the definition of Trio's LDB TR-containment (presented in Section 5) we will have that $Q_1 \subseteq_{TR} Q_2$ does not hold. The reason is that Trio LDB containment requires the containment of all lineage (not only base) of the contained relation through the transitive closure of lineage of the containing relation.

3 Preliminaries

Definition 1 (LDB tuple). A tuple t^{LDB} of an LDB $D = (\bar{R}, S, \lambda)$ consists of three things: i) its data t which belongs to the set of relations \bar{R} , ii) its identifier symbol denoted as $ID(t)$ and belonging to the set S of symbols and iii) its lineage $\lambda(ID(t))$ (belonging to λ) which associates it with the set of the identifiers of tuples from which it is derived. Thus t^{LDB} is a triple $(ID(t), t, \lambda(ID(t)))$.

Note that the lineage λ is a function from the set S of symbols/identifiers to the powerset of S . When a tuple's identifier is clear from the context, we may abuse the above notation and refer to an LDB tuple only with its identifier ID (i.e., denote its lineage as $\lambda(ID)$). Alternatively, we may say that we want to have an LDB tuple with data t and lineage $\lambda(t)$ present in our database, when confusion does not arise.

A ULDB is an LDB with uncertainty. So again it will contain a lineage function λ and identifier symbols belonging to a set S . The difference is that instead of ordinary tuples it will consist of x-tuples and its identifiers will also refer to alternatives.

A *conjunctive query* (CQ) over a schema \bar{R} is an expression of the form $Q(\bar{x}) :- \phi(\bar{x}, \bar{y})$, where $Q(\bar{x})$ is called the *head* and $\phi(\bar{x}, \bar{y})$ is called the *body* of the query. The body $\phi(\bar{x}, \bar{y})$ is a conjunction of atomic formulas that are also called subgoals of the query. Let Q, Q' be conjunctive queries, and let h be a mapping from variables and constants of Q' to variables and constants of Q such that h is the identity for constants. We say that h is a *containment mapping* from Q' to Q if $h(head(Q')) = head(Q)$ and every atom in the body of Q' is mapped to an atom of the body of Q with the same predicate. A containment mapping from Q' to Q is *subgoal-onto* if we additionally have that the set of images of all the subgoals of Q' contains *every* subgoal of the body of Q .

Algorithm 1 (Computing CQs over LDBs) [6,9]

Suppose that Q is a conjunctive query and D is an LDB. Then the following holds: The answer of query Q posed on D is the LDB $Q(D)$ that we get when

we take all the LDB relations of D and add to them an LDB relation R_Q which will have: 1) tuples with data that we get from the head of Q when we substitute constants for variables in the body of Q and require that all subgoals become a tuple of D , and 2) lineage the union of the identifiers of all tuples that are the images of the subgoals of the body of Q in step 1. We attach to each such tuple in R_Q a new unique identifier.

A ULDB represents a set of possible instances that are LDBs. Thus, our definition of when two ULDB databases are contained in each other depends on a variant (\subseteq_L) of LDB database containment. (We will define various variants of LDB containment in the subsequent sections.)

Definition 2 (ULDB Database Containment \subseteq_L). Let \subseteq_L denote a variant of LDB containment. Let U and U' be two ULDB's. We say that U is L -contained in U' (denoted with \subseteq_L) if: i) for every possible instance D_i of U there exists a possible instance D'_j of U' such that: $D_i \subseteq_L D'_j$ and ii) for every possible instance D'_j of U' there exists a possible instance D_i of U such that: $D_i \subseteq_L D'_j$.

Definition 3 (ULDB Query Containment). Let \subseteq_L denote a variant of LDB containment. A conjunctive query Q_1 is ULDB contained in a conjunctive query Q_2 if for every ULDB U we have that: $Q_1(U) \subseteq_L Q_2(U)$.

4 Semantics #1 (Data containment - \subseteq_{Data}) and Semantics #2 (Contained Base Lineage - \subseteq_{CBase})

As we saw on Definition 1 an LDB tuple t^{LDB} has a “data part” (a tuple t), a unique identifier and a lineage function that connects it to other LDB tuples through their identifiers. So it is natural to define LDB containment considering not only data, but lineage as well.

It is natural for ULDB query containment between two queries Q and Q' to require that for every ULDB U , we will have that if a tuple with data t exists in relation R_Q of a possible instance of $Q(U)$ then a tuple with same data t will also exist in relation $R_{Q'}$ in the corresponding possible instance of $Q'(U)$. Our first Semantics of Data containment will be based on this notion.

Suppose there is a relation in U such that none of its tuple identifiers appears in the lineage of some tuple of U . Let S' be the set of identifiers in all tuples in this relation. We denote by S_- those symbols of S that are in S but are not in S' ². If there is no such relation then $S_- = S$.

According to Definition 3 ULDB query containment relies on semantics for LDB database containment. So our second CBase Semantics for LDB containment will be based on the notion of contained base lineage. This kind of LDB containment is also natural to define as we discussed in our running example in section 2. The formal definitions for those first two kinds of LDB database containment are:

² This is a technicality to enable meaningful query containment definition.

Definition 4 (Data LDB Containment \subseteq_{Data}). Let $D = (\bar{R}, S, \lambda)$ and $D' = (\bar{R}', S', \lambda')$ be two LDBs, where \bar{R} and \bar{R}' have the same schemas. We say that D is Data LDB-contained in D' (denoted as $D \subseteq_{Data} D'$), if:

1. $S_- \subseteq S'_-$.
2. For every relation $R_i \in D$ and its corresponding $R'_i \in D'$ the following holds: if $t \in R_i$ then there exists a tuple with data t in R'_i .

The above definition concerns only data. The other four semantics of database containment will concern lineage restrictions as well, hence they will also include a lineage condition $COND_i$, $i = 2 \dots 5$. Thus, we will define each of these four semantics by the same definition as above with the only modification that we add $COND_i$ in the end of clause (2) of Definition 4. Note that for Semantics #1 (Definition 4) $COND_1$ is empty.

With λ_B we denote base lineage. For Contained Base Lineage Containment, this condition, $COND_2$, will be the following (and accordingly the definition):

Definition 5 (Contained Base Lineage (CBase-lineage) LDB Containment \subseteq_{CBase}). The additional condition is:

$$COND_2 : \lambda'_B(t) \subseteq \lambda_B(t).$$

I.e., for CBase-lineage LDB database containment, we require also that the base lineage of tuple t in R'_i is contained in the base lineage of tuple t in R_i .

It turns out that for ULDB conjunctive query computing purposes the query containment test for the above two notions is the same. This is reasonable if we remember that the possible instances of a uncertain database depend on the base lineage. Since all of our five definitions are based on Definition 3 which reasons on possible instances, it is expected that even in semantics #1 the lineage will affect database containment. This result also holds for LDBs, since an LDB can be thought as a ULDB with only one alternative on each x-tuple (and itself as its single possible instance). We have that Data and CBase-lineage ULDB database and query containment definitions follow from general Definitions 2 and 3 if we replace \subseteq_L with \subseteq_{Data} and \subseteq_{CBase} .

Theorem 1. Suppose that Q_1 and Q_2 are two conjunctive queries. Then the following are equivalent:

- i) $Q_1 \subseteq_{Data} Q_2$.
- ii) $Q_1 \subseteq_{CBase} Q_2$.
- iii) there exists a containment mapping $h: Q_2 \rightarrow Q_1$ (not necessarily onto).

In [12] it was shown that checking for the existence of a containment mapping between two conjunctive queries is NP-complete. Hence we have:

Corollary 1. Suppose that Q_1 and Q_2 are two conjunctive queries. Then the following holds: Checking whether $Q_1 \subseteq_{Data} Q_2$ or $Q_1 \subseteq_{CBase} Q_2$ is NP-complete.

Example 1. It is now easy to check that in our running Example of section 2 we have that Q_1 is CBase-lineage and Data Query contained in Q_2 as we intuitively expected.

5 Semantics #3: Trio/Transitive Closure of Lineage Containment (TR-lineage - \subseteq_{TR}).

Consider Q_1 and Q_2 from the running example of section 2. Then $Q_1(U)$ is the union of ULDB relations shown in Figures 2 and 3 and $Q_2(U)$ is the union of ULDB relations shown in Figures 2 and 4. Database $Q_1(U)$ is CBase contained in $Q_2(U)$. However $Q_2(U)$ contains less information because, e.g., alternative (41, 1) with data *John* is derived only from relation **Saw** while its corresponding (31, 1) is derived from both relations **Saw** and **Drives**. Let us suppose that we can additionally have that an answer is more reliable if it is based on a testimony from a witness. Then alternative (41, 1) will not be reliable (and the system might delete it) in contrast with (31, 1). If less reliable answers are deleted then even the set containment of data between corresponding possible instances will not anymore hold. We need to “retain” all lineage of the contained database through the transitive closure of lineage of the containing database. This kind of LDB database containment was defined in [9]. With $\lambda^*(t)$ we denote the transitive closure of lineage $\lambda(t)$.

Definition 6 (Trio-Transitive Closure-Lineage (TR-lineage) LDB Containment \subseteq_{TR}). [9]³ The additional condition is:

$$COND_3 : \lambda(t) \subseteq \lambda^*(t).$$

Intuitively all LDB tuples of D also appear in D' with same data and same tuple identifier. In addition all the lineage information between LDB tuples of D is also retained through the transitive closure of D' 's lineage.

In Figure 8 we see an example of TR-lineage LDB containment (for clarity, lineage is represented through arrows). It is easy to check that $D \subseteq_{TR} D'$. We note that \subseteq_{TR} is not a partial order on LDBs. For example consider Figure 9, where we have both $D \subseteq_{TR} D'$ and $D' \subseteq_{TR} D$, but the lineage functions of D and D' are not equal (for example we have $11 \in \lambda'(31)$ while $11 \notin \lambda(31)$). On the other hand we can have same base lineage but non TR-lineage LDB database containment, as we see in Figure 10. A notion of containment based on exactly the same base lineage will be considered in the following section.

Theorem 2. *Given two conjunctive queries Q_1 and Q_2 we have that $Q_1 \subseteq_{TR} Q_2$ iff there exists a subgoal-onto containment mapping $h: Q_2 \rightarrow Q_1$.*

In [13] it was shown that checking for a subgoal-onto containment mapping between two conjunctive queries is NP-complete. Hence from the above theorem we have:

Corollary 2. *Suppose that Q_1 and Q_2 are two conjunctive queries. Then the following holds: Checking whether $Q_1 \subseteq_{TR} Q_2$ is NP-complete.*

³ In Trio same tuple identifier was required but we relax this too restricted definition.

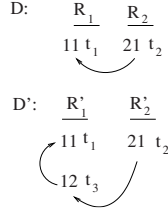


Fig. 8. Example of TR-lineage LDB containment $D \subseteq_{TR} D'$.

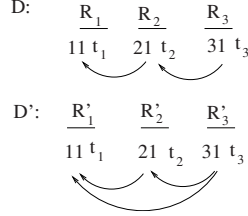


Fig. 9. Example of $D \subseteq_{TR} D'$ and $D' \subseteq_{TR} D$.

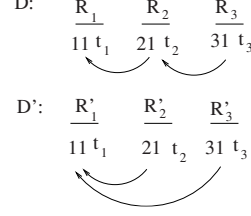


Fig. 10. Example of non TR-lineage LDB containment ($D \not\subseteq_{TR} D'$) but same base lineage.

6 Semantics #4 (Same Base Lineage - \subseteq_{SBase}) and Semantics #5 (Same Lineage Containment - \subseteq_{Same}).

Consider the example of Figure 10 of the previous section. Tuples 21 and 31 have the same base lineage (tuple with id 11) in both D and D' , and all data of D appears in corresponding relations in D' with same identifiers. But $D \not\subseteq_{TR} D'$. The reason is that the fact that 31 comes from 21 ($\lambda(31) = 21$) is not retained in D' , even if we compute the transitive closure of $\lambda'(31)$. Suppose that we consider reliable only the information resulting from tuple 11 and no other “source”, i.e., no other base tuple. Then the information about the connection between tuples 31 and 21 (which is non base) will not be important and we intuitively should have database containment between D and D' because tuples with data t_3 are considered reliable in both databases. On the other hand the fact that both t_3 tuples have 11 as base lineage should not be ignored.

In situations like this it is natural to require for the data of D to be included in the data D' with exactly the same base lineage. Such a condition is also important in ULDB data exchange purposes, for containment of certain answers [6]. The SBase-lineage semantics captures this situation by making sure that our tuples in the containing relation do not come from any unreliable “source”. We have the following definition for SBase-lineage LDB database containment:

Definition 7 (Same Base-Lineage (SBase-lineage) LDB Containment \subseteq_{SBase}). The additional condition is:

$$COND_4 : \lambda'_B(t) = \lambda_B(t).$$

The more strict kind of containment is requiring data containment with exactly the same lineage (not only base). It is important in cases where we want to retain all the lineage (not only the base one), like in TR-lineage containment, and also where we additionally do not want to have lineage connections to a tuple from a base tuple that exists only in the containing relation, like in SBase-lineage. We have the following definition:

Definition 8 (Same-Lineage LDB Containment \subseteq_{Same}). *The additional condition is:*

$$COND_5 : \lambda'(t) = \lambda(t).$$

It turns out that SBase-lineage conjunctive query containment holds between two CQs Q_1 and Q_2 if and only if there exists an onto containment mapping from Q_2 to Q_1 , like in TR-lineage CQ containment. The same holds also for Same-lineage CQ containment. It is formally stated in the following theorem:

Theorem 3. *Given two conjunctive queries Q_1 and Q_2 we have: 1. The following are equivalent: i) $Q_1 \subseteq_{SBase} Q_2$, ii) $Q_1 \subseteq_{Same} Q_2$ and iii) there exists an onto containment mapping $h: Q_2 \rightarrow Q_1$. 2. Checking whether $Q_1 \subseteq_{SBase} Q_2$ or $Q_1 \subseteq_{Same} Q_2$ is NP-Complete.*

SBase-lineage containment for LDBs is equivalent with the *Why*-semiring containment that is defined in [17] for databases with lineage, where it was shown that *Why*-semiring CQ containment between two CQs Q_1 and Q_2 holds iff there exists an onto containment from Q_2 to Q_1 .

7 Comparison between Different Semantics

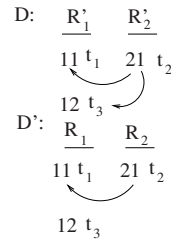


Fig. 11.
 $D \subseteq_{CBase} D'$
 $D \not\subseteq_{SBase} D'$

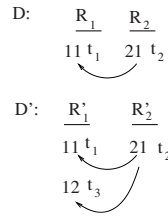


Fig. 12.
 $D \subseteq_{Data} D'$
 $D \subseteq_{TR} D'$
 $D \not\subseteq_{CBase} D'$
 $D \not\subseteq_{SBase} D'$
 $D \not\subseteq_{Same} D'$

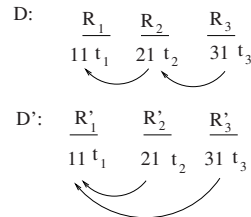


Fig. 13.
 $D \subseteq_{Data} D'$
 $D \subseteq_{CBase} D'$
 $D \subseteq_{SBase} D'$
 $D \not\subseteq_{TR} D'$
 $D \not\subseteq_{Same} D'$

In Figure 1 of the Introduction we have a comparison of the five notions of containment that we defined. Column *Implies DB cont.* shows implications between different kinds of database containment. For example if a ULDB or LDB database is contained in another under Semantics #4, then it will be contained under Semantics #1 and #2 as well. These implications between different database containment semantics are easy to check from their definitions. For

negative implication results we give counter-examples: In Figure 12 we have an example of Data containment and TR-lineage containment between databases (LDBs) D and D' , but with non CBase database containment (due to the fact that $\lambda_B(21) = 12$ of D' is not retained in D - note that for CBase the lineage containment goes from the containing to the contained database). For the same reason we do not have same base lineage nor same lineage between D (where $\lambda_B(11) = \{12\}$) and D' (where $\lambda_B(11) = \{11, 12\}$). In Figure 11 we have an example of CBase-lineage database LDB containment. Note that $D \subseteq_{CBase} D'$ in Figure 11 because the base lineage of 21 in D' is 11 which is a subset of the base lineage of 21 in D . But in the same figure we do not have SBase-lineage database containment between D and D' due to the fact that the connection: $12 \in \lambda_B(21)$ of D is not retained in D' . Lastly in Figure 13 we see an example of Data, CBase-lineage and SBase-lineage LDB database containment but with non TR-lineage nor Same-lineage containment, due to the fact that non-base lineage connection $\lambda(31) = \{21\}$ of D is not retained in D' . Note that these positive and negative results of database containment implications hold for both LDBs and ULDBs.

7.1 CQ Query Equivalence

It is easy to show that two conjunctive queries are equivalent in that they compute the same answer in all ULDB databases iff they are isomorphic. Since however there is not a unique representation for ULDBs, we need to clarify that by the “same” answer we mean two databases with the same possible instances (up to identifier renaming). Hence it is an interesting observation that the three last semantics in Figure 1 (for query containment) can be used to define CQ query equivalence as both ways containment.

8 Semantics #6: Uncertain Equality containment - \subseteq_E .

In [7] a new kind of containment for uncertain databases with no lineage was discussed that was suitable for uncertain data integration purposes. An uncertain database (with no lineage) is defined as a set of ordinary databases called *Possible Worlds* (PWs) (in convention we use the notion *Possible Worlds* whenever we have no lineage and we retain the notion of *Possible Instances* in the case with lineage). The definition of equality containment is the following:

Definition 9 (Equality Containment \subseteq_E). [7]

Consider two uncertain databases U_1 and U_2 . Let $T(U_1)$ denote the set of all tuples appearing in any possible world of U_1 and respectively we define $T(U_2)$. We say that U_1 is equality-contained in U_2 ($U_1 \subseteq_E U_2$), if and only if: $T(U_1) \subseteq T(U_2)$ and $PW(U_1) = \{W \cap T(U_1) \mid W \in PW(U_2)\}$.

Informally the above containment means that if we throw away from the possible worlds of U_2 all tuples that do not appear in any possible world of U_1 , then the resulting possible worlds are the worlds of U_1 . The notions of database

containment we defined in the previous sections were data-driven: all kinds of them implied set containment of data (Semantics # 1). In contrast the notion of equality containment focuses on retaining correlation and mutual exclusion between data: if two tuples in the contained database occur only in different worlds then no containing world can include both of them. In addition if two tuples occur only together in a possible world of the contained database then any containing world that has one of them must have the other as well.

Now consider an uncertain database U that is a set of n ordinary databases that are its possible worlds. Posing a query Q over it will give a new uncertain relation with n possible worlds, each containing the answer of Q over the possible worlds of U . We have that even an onto containment between two CQs, which yielded the stronger same-lineage CQ containment does not suffice for equality containment. This is not surprising since the notion of equality containment does not focus on data containment like the ones we presented. For example consider an uncertain database U containing the uncertain relation $R = \{\{(a, a)\}, \{(b, b)\}, \{(a, b), (b, a)\}\}$, where a and b are two different constants. Consider CQ $Q_1(x) :- R(x, x)$ and $Q_2(x) :- R(x, y)$. Then $Q_1(U) = \{\{a\}, \{b\}, \emptyset\}$ and $Q_2(U) = \{\{a\}, \{b\}, \{a, b\}\}$. If we restrict the third possible world of $Q_2(U)$ to the tuples $\{a, b\}$ that occur in $Q_1(U)$ then the resulting world which has both those tuples is not a possible world of $Q_1(U)$. Hence $Q_1(U)$ is not equality contained in $Q_2(U)$. On the other hand it is easy to check that there exists an onto containment mapping from Q_2 to Q_1 . We now state the following Theorem:

Theorem 4. *Given two conjunctive queries Q_1 and Q_2 we have that $Q_1 \subseteq_E Q_2$ iff there exists a containment mapping $h: Q_2 \rightarrow Q_1$ and a containment mapping $h': Q_1 \rightarrow Q_2$. In addition checking whether $Q_1 \subseteq_E Q_2$ is NP-complete.*

9 Conclusion

We have introduced several variants of ULDB database containment and corresponding variants of query containment. We have shown that the variants of ULDB database containment are all different. We studied the exact interrelationship among them as concerns implication (column *Implies DB cont.* in Figure 1). Specifically we showed for which pairs of variants the fact that database D_1 is contained in database D_2 in this variant implies also that database D_1 is contained in database D_2 in the other variant. For the pairs of variants that we have not shown implication we showed with examples that implication does not exist.

However, when we investigated conjunctive query containment we showed that the various variants can be partitioned in two classes that are CQ query containment equivalent. We finally observed that CQ query equivalence can be defined as both ways query containment for one of these classes.

On another perspective database containment was defined in [7] for uncertain databases without lineage. We also investigated CQ query containment under this definition. Various open problems remain, starting with finding the complexity of ULDB database containment for the various kinds of containment we

have defined here. Another question is how query optimization techniques can benefit from this investigation of query containment.

References

1. S. Abiteboul, P. C. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. *Theor. Comput. Sci.*, 78(1):158–187, 1991.
2. F. N. Afrati, M. Damigos, and M. Gergatsoulis. Query containment under bag and bag-set semantics. *Inf. Process. Lett.*, 110(10):360–369, 2010.
3. F. N. Afrati and N. Kiourtis. Computing certain answers in the presence of dependencies. *Inf. Syst.*, 35(2):149–169, 2010.
4. F. N. Afrati, C. Li, and P. Mitra. Rewriting queries using views in the presence of arithmetic comparisons. *Theor. Comput. Sci.*, 368(1-2):88–123, 2006.
5. F. N. Afrati, C. Li, and J. D. Ullman. Using views to generate efficient evaluation plans for queries. *J. Comput. Syst. Sci.*, 73(5):703–724, 2007.
6. F. N. Afrati and A. Vasilakopoulos. Managing lineage and uncertainty under a data exchange setting. In *Scalable Uncertainty Management 2010 (SUM 2010)*.
7. P. Agrawal, A. D. Sarma, J. Ullman, and J. Widom. Foundations of uncertain-data integration. In *VLDB 2010*.
8. D. Barbará, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *IEEE Trans. Knowl. Data Eng.*, 4(5):487–502, 1992.
9. O. Benjelloun, A. D. Sarma, A. Y. Halevy, M. Theobald, and J. Widom. Databases with uncertainty and lineage. *VLDB J.*, 17(2):243–264, 2008.
10. P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *ICDT*, pages 316–330, 2001.
11. P. Buneman and W. C. Tan. Provenance in databases. In *SIGMOD Conference*, pages 1171–1173, 2007.
12. A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, pages 77–90, 1977.
13. S. Chaudhuri and M. Y. Vardi. Optimization of *real* conjunctive queries. In *PODS*, pages 59–70, 1993.
14. Y. Cui and J. Widom. Practical lineage tracing in data warehouses. In *ICDE*, pages 367–378, 2000.
15. Y. Cui and J. Widom. Lineage tracing for general data warehouse transformations. *VLDB J.*, 12(1):41–58, 2003.
16. N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875, 2004.
17. T. J. Green. Containment of conjunctive queries on annotated relations. In *ICDT*, pages 296–309, 2009.
18. T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.
19. T. Imielinski and W. Lipski Jr. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984.
20. L. V. S. Lakshmanan, N. Leone, R. B. Ross, and V. S. Subrahmanian. Proview: A flexible probabilistic database system. *ACM Trans. Database Syst.*, 22(3):419–469, 1997.
21. A. D. Sarma, J. D. Ullman, and J. Widom. Schema design for uncertain databases. In *AMW*, 2009.
22. J. D. Ullman. Information integration using logical views. In *ICDT*, pages 19–40, 1997.

Dissociation and Propagation for Efficient Query Evaluation over Probabilistic Databases

Wolfgang Gatterbauer, Abhay K. Jha, Dan Suciu

University of Washington, Seattle WA
{gatter, abhaykj, sucio}@cs.washington.edu

Abstract. Queries over probabilistic databases are either *safe*, in which case they can be evaluated entirely in a relational database engine, or *unsafe*, in which case they need to be evaluated with a general-purpose inference engine at a high cost. We propose a new approach by which *every query* is evaluated inside the database engine, by using a new method called *dissociation*. A dissociated query is obtained by adding extraneous variables to some atoms until the query becomes safe. We show that the probability of the original query and that of the dissociated query correspond to two well-known scoring functions on graphs, namely *graph reliability* (which is #P-hard), and the *propagation score* (which is related to PageRank and is in PTIME): When restricted to graphs, standard *query probability* is graph reliability, while the *dissociated probability* is the propagation score. We define a *propagation score for self-join-free conjunctive queries* and prove that it is always an upper bound for query reliability, and that both scores coincide for all safe queries. Given the widespread and successful use of graph propagation methods in practice, we argue for the dissociation method as a highly efficient way to rank probabilistic query results, especially for those queries which are highly intractable for exact probabilistic inference.

1 Introduction

Evaluating queries over probabilistic databases (PDBs) is hard in general. Despite important recent advances [15,20], today's approaches to query evaluation are not practical. Existing techniques either split the queries into safe and unsafe and compute efficiently only the former [8,19], or work very well on certain combinations of queries and data instances but can not offer performance guarantees in general [15,18], or use general purpose approximation techniques and are thus generally slow [14,23]. In this paper, we propose a new approach to evaluate queries over tuple-independent probabilistic databases by which every query can be evaluated efficiently. We achieve this by replacing the standard semantics based on *reliability*, with a related but much more efficient semantics based on *propagation*.

The semantics of a query over a PDB is based on *reliability* [12], which has roots in *network reliability* [6]. It is defined as the probability that a source node remains connected to a target node in a directed graph if edges fail independently

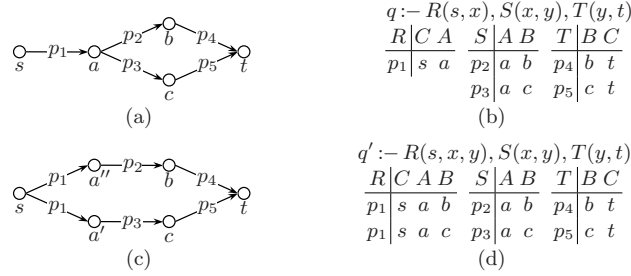


Fig. 1. The propagation score $\rho(t)$ in graph (a) corresponds to the reliability score $r(t)$ in graph (c) with node a dissociated into a' and a'' . (b,d): Corresponding chain queries q and q' with respective database instances.

with known probabilities. Computing network reliability is #P-hard. However, many successful practical applications use a semantics different from reliability, based on a *propagation scheme*. We illustrate with an example.

Example 1 (Propagation in k -partite digraphs). Consider the 4-partite graph in Fig. 1a in which each edge i is present with independent probability p_i . The reliability score $r(x)$ of a node x is the probability that the source node s is connected to the node x in a randomly chosen subgraph with edges directed left-to-right. The score of interest is the reliability of a target node t :

$$r(t) = p_1(1 - (1 - p_2p_4)(1 - p_3p_5))$$

While reliability can be computed efficiently for series-parallel graphs as in Fig. 1a, it is #P-hard in general, even on 4-partite graphs [6]. The probability of a query over a PDB corresponds precisely to network reliability. For example, in the case of a 4-partite graph, reliability is given by the probability of the chain query $q := R(s, x), S(x, y), T(y, t)$ over the PDB shown in Fig. 1b (we use interchangeably the terms query reliability and query probability in this paper).

In contrast, the propagation score of a node x is a value that depends on the scores of its parent nodes and the probabilities of the incoming edges: $\rho(x) = 1 - \prod_e (1 - \rho(y_e) \cdot p_e)$, where e is the edge (y_e, x) . By definition, $\rho(s) = 1$. In our example, the propagation score of the target node t is:

$$\rho(t) = 1 - (1 - p_1p_2p_4)(1 - p_1p_3p_5)$$

Unlike reliability, the propagation score can *always* be computed efficiently, even on very large graphs. The reason is that reliability has an *intensional* semantics, while propagation is *extensional*¹ [10,21]. Variants of propagation have thus

¹ *Extensional approaches* compute the probability of any formula as a function of the probabilities of its subformulas according to syntactic rules, regardless of how those were derived. *Intensional approaches* reason in terms of possible worlds and keep track of dependencies.

been successfully used in a range of applications where an exact probability computation is not necessary. Examples include similarity ranking of proteins [28], integrating and ranking uncertain scientific data [9], trust propagation in social networks [13], search in associative networks [7], models of human comprehension [22], keyword search in databases [2], and computing web page reputation with the renowned PageRank² algorithm [5].

With this paper, we introduce a propagation score for queries over PDBs, describe the connection to the reliability score, and give a method to compute the propagation score for every conjunctive query without self-joins efficiently with a *standard relational database engine*. We propose the propagation score as an alternative semantics to the reliability score. While the propagation score differs from the reliability score, we prove several properties showing that it is a reasonable substitute: (i) the propagation score is always greater than or equal to the reliability score, (ii) the two are guaranteed to coincide for all safe queries, and (iii) the propagation score is very close to the reliability score in our experimental validation.

To the best of our knowledge, no definition of a propagation score on hypergraphs exists, and it is not obvious how to define such a score for queries which are not represented by graphs but by hypergraphs. Also, when restricted to k -partite graphs, the propagation score depends on the directionality of the graph. In Fig. 1a the propagation score from s to t is different from that from t to s . In fact, the latter coincides with the reliability score. It is unclear what this directionality corresponds to for arbitrary queries.

Main contributions. Our first main contribution is defining the propagation score for any self-join-free conjunctive query in terms of a *dissociation*. A dissociation is a rewriting of both the data and the query. On the data, a dissociation is obtained by making multiple, independent copies of some of the tuples in the database. Technically, this is achieved by extending the relational schema with additional attributes. On a query, a dissociation extends atoms with additional variables. We prove that a dissociation can only increase the probability of a query, and define *the propagation score of a query* as the minimum reliability of all dissociated queries that are *safe*. This is justified by the fact that, in a k -partite graph, the propagation score is precisely the probability of one dissociated safe query. Thus, in our definition, choosing a direction for the network flow in order to define the propagation score corresponds to choosing a particular dissociation that makes the query safe. Safe queries [8] can be evaluated efficiently on any probabilistic database, and we show that every query (safe or not) admits at least one safe dissociation. Therefore, the propagation score can be computed efficiently.

Our second main contribution is establishing a one-to-one correspondence between safe dissociations and traditional query plans. This result leads to an efficient algorithm for computing the propagation score of a query: iterate over all query plans, retain only those that are minimal in the dissociation order, evaluate

² Note that the propagation score is *not* the same as PageRank. However, both share the common principle that the score of a node is defined only *in terms of the scores of its neighbors*, and not in terms of the entire topology of the graph.

those on the original data, and return the minimum probability. Importantly, there is *no need to dissociate the actual data*, which is an expensive step. We give a system R-style algorithm that enumerates all plans that correspond to minimal safe dissociated queries. A so far unknown corollary of our work is that every query plan gives an upper bound on query reliability.

Example 2 (Dissociation). *We have seen that the propagation score differs from the reliability score on the DAG in Fig. 1a. By inspecting the expressions of the two scores, one can see that they differ in how they treat p_1 : reliability treats it as a single event, while propagation treats it as two independent events. In fact, the propagation score is precisely the reliability score of the DAG in Fig. 1c, which has two copies of p_1 . We call this DAG the dissociation of the DAG in Fig. 1a. At the level of the database instance, dissociation can be obtained by adding a new attribute B to the first relation R (Fig. 1d). The dissociated query is $q^B :- R(s, x, y), S(x, y), T(y, t)$, and one can check that its probability is indeed the same as the propagation score for the data in Fig. 1a. The important observation here is that, while the evaluation problem for q is $\#P$ -hard because it is an unsafe query [8], the query q^B is safe and can therefore be computed efficiently.*

A query q usually has more than one dissociation: q has a second dissociation $q^A :- R(s, x), S(x, y), T(x, y, t)$ obtained by adding the attribute A to T (not shown in the figure). Its probability corresponds to the propagation score from t to s , i.e. from right to left. And there is a third dissociation, $q^{BA} :- R(s, x, y), S(x, y), T(x, y, t)$. We prove that each dissociation step can only increase the probability, hence $r(q) \leq r(q^B) \leq r(q^{BA})$ and $r(q) \leq r(q^A) \leq r(q^{BA})$. We define the propagation score of q as the smallest probability of all dissociations. The database system has to compute $r(q^B)$ and $r(q^A)$ and return the smallest score: on the graph in Fig. 1a this is $r(q^A)$, since $r(q) = r(q^A)$.

Outline. We review basic definitions (Sect. 2), then formally introduce query dissociation and the propagation score (Sect. 3). We prove its strong connection to query plans (Sect. 4), describe our experimental evaluation (Sect. 5), review related work (Sect. 6), before we conclude (Sect. 7). Details, all proofs, more experiments and several optimizations are covered in the technical report [11].

2 Preliminaries

We consider PDBs where each tuple t has a probability $p(t) \in [0, 1]$. We denote with D the database instance, i.e. the collection of tuples and their probabilities. A *possible world* is generated by independently including each tuple t in the world with probability $p(t)$. Thus, the database D is *tuple-independent*. Consider a Boolean conjunctive query q

$$q :- g_1, \dots, g_m$$

where g_1, \dots, g_m are relational atoms, sometimes called subgoals, over a vocabulary R_1, \dots, R_k . The focus of probabilistic query evaluation is to compute $\mathbf{P}[q]$,

which is the probability that the query is true in a randomly chosen world, and which we refer to as the *query reliability* $r(q)$ [12].

It is known that the data complexity of a conjunctive query q is either PTIME or #P-hard [8]. The class of PTIME queries, sometimes called *safe queries*, is best understood in the case of Boolean queries without self-joins, keys and deterministic relations. We will focus on this important case in this paper. With this restriction, the safe queries are precisely the *hierarchical queries*:

Definition 1 (Hierarchical queries [8]). *For every variable x in q , denote $sg(x)$ the set of subgoals that contain x . Then q is called hierarchical if for any two variables x, y , one of the following three conditions hold: $sg(x) \subseteq sg(y)$, $sg(x) \cap sg(y) = \emptyset$, or $sg(x) \supseteq sg(y)$.*

For example, the query $q:-R(x, y), S(y, z), T(y, z, u)$ is hierarchical, while $q:-R(x, y), S(y, z), T(z, u)$ is not. It is known that every hierarchical query can be computed in PTIME, and every non-hierarchical query is #P-hard.

We next introduce a query plan for a conjunctive query:

Definition 2 (Query plan P). *Let R_1, \dots, R_m be a relational vocabulary. A query plan, or simply plan, is given by the grammar*

$$P ::= R_i(\bar{x}) \mid \pi_{\bar{x}}P \mid \bowtie[P_1, \dots, P_k]$$

where $R_i(\bar{x})$ is a relational atom containing the variables \bar{x} and constants, $\pi_{\bar{x}}$ is the project operator with duplicate elimination, and $\bowtie[\dots]$ is the natural join, which we allow to be k -ary, for $k \geq 2$. We require that joins and projections alternate in a plan. We do not distinguish between join orders, i.e. $\bowtie[P_1, P_2]$ is the same as $\bowtie[P_2, P_1]$.

Denote q_P the query consisting of all atoms mentioned in (sub-)plan P . We define the *head variables* $\text{HVar}(P)$ inductively as

$$\begin{aligned} \text{HVar}(R_i(\bar{x})) &= \bar{x} \\ \text{HVar}(\pi_{\bar{x}}(P)) &= \bar{x} \\ \text{HVar}(\bowtie[P_1, \dots, P_k]) &= \bigcup_{i=1}^k \text{HVar}(P_i) \end{aligned}$$

A plan is called Boolean if $\text{HVar}(P) = \emptyset$. We assume the usual sanity conditions on plans to be satisfied: in a project operator $\pi_{\bar{x}}(P)$ we assume $\bar{x} \subseteq \text{HVar}(P)$, and each variable y is projected away in at most one project operator, i.e. there exists at most one operator $\pi_{\bar{x}}(P)$ s.t. $y \in \text{HVar}(P) - \bar{x}$.

A plan P is evaluated on a probabilistic database D using an *extensional semantics* [10], [21, p. 3]: Each subplan P returns an intermediate relation of arity $|\text{HVar}(P)| + 1$. The extra attribute stores the probability of each tuple. To compute the probability, each operator assumes the input tuples to be *independent*, i.e. the probabilistic join operator $\bowtie^p[\dots]$ multiplies the tuple probabilities $\Pi_i p_i$, and the probabilistic project operator with duplicate elimination π^p computes the probability as $1 - \Pi_i(1 - p_i)$. For a Boolean plan P , this results in a

single probability value, which we denote $score(P)$. In general, this is not the correct query reliability $r(q_P)$, which, recall, is defined in terms of possible worlds: $score(P) \neq r(q_P)$.

Definition 3 (Safe plan). A plan P is called safe if for any join operator $\bowtie^P[P_1, \dots, P_k]$ the following holds: $HVar(P_i) = HVar(P_j)$, for all $1 \leq i, j \leq k$.

The following are well known facts about safe queries and safe plans.

Proposition 1 (Safety). (1) Let P be a plan for the conjunctive query without self-joins q_P . Then P is safe iff for any probabilistic database, $score(P) = r(q_P)$. (2) Let q be a conjunctive query. Then the following are equivalent: q is safe; q is hierarchical; q admits a safe plan. Moreover, the safe plan is unique (up to reordering of the operands in join operators).

Example 3 (Safe plan). An example safe query and its unique safe plan:

$$q :- R(x, y), S(y, z), T(y, z, u) \\ P = \pi_{\emptyset}^P \bowtie^P [\pi_y^P R(x, y), \pi_y^P \bowtie^P [S(y, z), \pi_{y,z}^P T(y, z, u)]]$$

3 Dissociation and Propagation

In this section, we define the technique of *query dissociation* and the semantics of *propagation score*. We first define the approach formally, then describe in Sect. 4 an efficient method to evaluate propagation and illustrate with examples. All proofs and more details are provided in the appendix.

In the following, we write $Var(q)$ for set of variables in the body of a query q , $Var(g_i)$ for the variables in a subgoal g_i , and A for the active domain of a database D . We use the bar sign (e.g. \bar{x}) to denote both sets or tuples.

Definition 4 (Query dissociation). A dissociation of a conjunctive query $q :- g_1, \dots, g_m$ is a collection of sets of variables $\Delta = \{\bar{y}_1, \dots, \bar{y}_m\}$ with $\bar{y}_i \subseteq Var(q) - Var(g_i)$.

Definition 5 (Table dissociation). Given a query $q :- g_1, \dots, g_m$ with $g_i = R_i(\bar{x}_i)$, the active domain A , and a query dissociation $\Delta = \{\bar{y}_1, \dots, \bar{y}_m\}$. The dissociation of table R_i on $\bar{y}_i = (y_{i1}, \dots, y_{ik})$ is the relation given by query

$$R_i^{\bar{y}_i}(\bar{x}_i, \bar{y}_i) :- R_i(\bar{x}_i), A(y_{i1}), \dots, A(y_{ik})$$

Conceptually, a *dissociation of a table* is the multi-cross product with the active domain so that each tuple in the original table is copied to multiple tuples in the dissociated table. Recall that each tuple in the original table represents an independent probabilistic event. The dissociated table now contains multiple copies of each tuple, all with the same probability, yet considered to represent *independent* events. Thus, the dissociated table has a different probabilistic semantics than the original table.

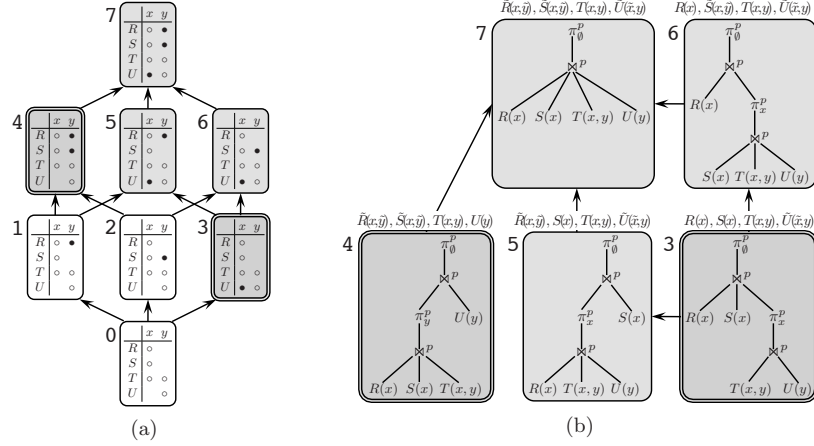


Fig. 2. (a): Partial dissociation order of $q := R(x), S(x), T(x, y), U(y)$. Safe dissociations are shaded (3, 4, 5, 6, 7), *minimal safe dissociations* are shaded in red and double-lined (3, 4). (b): All 5 possible query plans, their partial order and correspondence to safe dissociations in the partial dissociation order of q .

Definition 6 (Query dissociation semantics). Let $q := R_1(\bar{x}_1), \dots, R_m(\bar{x}_m)$ be a query and $\Delta = \{\bar{y}_1, \dots, \bar{y}_m\}$ a dissociation for q . The dissociated query is:

$$q^\Delta := R_1^{\bar{y}_1}(\bar{x}_1, \bar{y}_1), \dots, R_m^{\bar{y}_m}(\bar{x}_m, \bar{y}_m)$$

Thus, query dissociation works as follows: Add some variables to some atoms in the query. This results in the *dissociated query* over a new schema³. Transform the probabilistic database by replicating some of their tuples and by adding new attributes to match the new schema. This is the *dissociated database*. Finally, compute the probability of the dissociated query on the dissociated database⁴.

Our first major technical result in this paper shows that query dissociation can only increase the probability. We state it in a slightly more general form, by noting that the set of dissociations forms a partial order.

Definition 7 (Partial dissociation order). We define the partial order on the dissociations of a query as:

$$\Delta \succeq \Delta' \Leftrightarrow \forall i : \bar{y}_i \supseteq \bar{y}'_i$$

Theorem 2 (Partial dissociation order). For every two dissociations Δ and Δ' of a query q , the following holds over every database instance:

$$\Delta \succeq \Delta' \Leftrightarrow r(q^\Delta) \geq r(q^{\Delta'})$$

³ Note that several alternative dissociations are possible, in general.

⁴ Note that this is the *semantics* of a dissociated query, and not the way we actually evaluate queries. In Sect. 4 we give a method that evaluates the dissociated query without actually modifying the tables in the database.

Corollary 3 (Upper query bounds). *For every database and every dissociation Δ of a query q : $r(q^\Delta) \geq r(q)$.*

Corollary 3 immediately follows from Theorem 2 as every query is a dissociation of itself. The total number of dissociations corresponds to the cardinality of the power set of variables that can be added to tables. Hence, for every query with n non-head variables and m clauses, there are 2^K possible dissociations with $K = mn - k$ and $k = \sum_{i=1}^m |\text{Var}(g_i)|$ forming a partial order in the shape of a power set (see Fig. 2a).

We next define the *propagation score of a query* as the minimum probability of all those dissociations in the partial dissociation order that are *safe*.

Definition 8 (Safe dissociation). *A dissociation Δ_s of a query q is called safe if the dissociated query q^{Δ_s} is safe.*

Definition 9 (Propagation score). *The propagation score $\rho(q)$ for a query q is the minimum of the scores of all safe dissociations: $\rho(q) = \min_{\Delta_s} r(q^{\Delta_s})$.*

We propose to adopt the propagation score as an alternative semantics to reliability. While computing the reliability $r(q)$ is #P-hard in general, computing the propagation score $\rho(q)$ is always in PTIME in the size of the database. Further, $\rho(q) \geq r(q)$ and, if q is safe, then $\rho(q) = r(q)$ (both claims follow immediately from Corollary 3).

We now justify our definitions of dissociation and query propagation: When a graph is k -partite, then its reliability can be expressed by a conjunctive chain query q . Further, the propagation score over this graph corresponds to exactly one of several possible dissociations of this query q . *Query dissociation is thus a strict generalization of graph propagation on k -partite graphs.* And we define query propagation as corresponding to the dissociation with minimum reliability.

Proposition 4 (Connection propagation score). *Let $G = (V, E)$ be a $k+1$ -partite digraph with a source node s and a target node t , where each edge has a probability. The nodes are partitioned into $V = \{s\} \cup V_2 \cup \dots \cup V_k \cup \{t\}$, and the edges are $E = \bigcup_i R_i$, where R_i denotes the set of edges from V_i to V_{i+1} with $i \in \{1, \dots, k\}$. Then:*

(a) *The network reliability of the graph is $r(q)$, where:*

$$q := R_1(s, x_1), R_2(x_1, x_2), \dots, R_k(x_{k-1}, t)$$

(b) *Using $\bar{x}_{[i,j]}$ as short form for x_i, \dots, x_j , the propagation score (as defined in Example 1) is $r(q^\Delta)$, where:*

$$q^\Delta := R_1(s, \bar{x}_{[1,k-1]}), R_2(x_{[2,k-1]}), \dots, R_k(x_{k-1}, t)$$

4 Dissociations and Plans

Thus, in order to compute the propagation score of a query q , we need to compute several dissociated safe queries q^Δ . For that, we will not apply naively Def. 6,

because the table dissociation part (Def. 5) computes several cartesian products, and is very inefficient. Instead, we describe here an approach for computing $r(q^\Delta)$ without dissociating either the query or the tables. The second major technical result of our paper allows us to perform dissociation very efficiently.

Theorem 5 (Safe dissociation). *Let q be a conjunctive query without self-joins. There exists an isomorphism between safe dissociations Δ_s of q and query plans P for q . Moreover, the reliability of the dissociated query is equal to the score of the plan, $r(q^{\Delta_s}) = \text{score}(P)$.*

We describe this isomorphism briefly. Consider a safe dissociation q^Δ , and denote its unique safe plan P^Δ . This plan uses dissociated relations, hence each relation $R_i^{\tilde{y}_i}$ has some extraneous variables \tilde{y}_i . Drop all the variables \tilde{y}_i from the relations, and from all operators that use them: this transforms P^Δ into a regular (unsafe) plan P for q . Conversely, consider a plan P for q (P is not necessarily safe; in fact if q is unsafe then there is no safe plan at all). We define its corresponding safe dissociation Δ as follows. For each join operation $\bowtie^p[P_1, \dots, P_k]$, let the *join variables* be $\text{JVar} = \bigcup_j \text{HVar}[P_j]$: for every relation R_i occurring in P_j , add the variables $\text{JVar} - \text{HVar}[P_j]$ to \tilde{y}_i . For example, consider the lower join in Fig. 2b box (5): $\bowtie^p[R(x), T(x, y), U(y)]$. Here $\text{JVar} = \{x, y\}$ and the dissociation of this subplan is $\tilde{R}(x, \tilde{y}), T(x, y), \tilde{U}(\tilde{x}, y)$, where the tilde (\sim) indicates dissociated relations and variables. The complete safe query is shown at the top of the box (5). Note that while there is a one-to-one mapping between safe dissociations and query plans, unsafe dissociations do not correspond to plans (see Fig. 2a).

We have seen (right after Def. 2) that the extensional semantics of an unsafe plan P differs from the true reliability, $\text{score}(P) \neq r(q)$. Since we have shown that $\text{score}(P) = r(q^\Delta)$ for some dissociation Δ , we derive the following important corollary:

Corollary 6 (Query plans as bounds). *Let P be any plan for a query q . Then $\text{score}(P) \geq r(q)$. In other words, any query plan for evaluating a query inside a database gives an upper bound to the actual query reliability $r(q)$.*

To summarize, we have now a much more efficient way to compute $\min_\Delta(r(q^\Delta))$: iterate over all plans P , and compute $\min_P \text{score}(P)$. Thus, there no need to dissociate the input tables which is very inefficient: each plan P is evaluated directly on the original probabilistic database. However, this approach is inefficient in that it computes some redundant plans: for example, in Fig. 2 plans 5, 6, 7 are redundant, since, in the dissociation order, they are all greater than plan 3. It suffices to evaluate only the *minimal query plans*, i.e. those for which the corresponding dissociation is minimal among all safe dissociations: in our example, these are plans 3 and 4.

Example 4 (Safe dissociations). *Take the query $q := R(x), S(x), T(x, y), U(y)$. It is unsafe and $K = 4 \cdot 2 - 5 = 3$. Figure 2a shows its partial dissociation order*

2^3 , and Fig. 2b shows all 5 possible query plans. Two of those plans are minimal in the partial order of plans:

$$\begin{aligned} q^{(3)} &:- R(x), S(x), T(x, y), \tilde{U}(\tilde{x}, y) \\ q^{(4)} &:- \tilde{R}(x, \tilde{y}), \tilde{S}(x, \tilde{y}), T(x, y), U(y) \end{aligned}$$

The corresponding query plans over the original tables (and safe derivations over dissociated tables) are:

$$\begin{aligned} P^{(3)} &= \pi_{\emptyset}^P \bowtie^P [R(x), S(x), \pi_x^P \bowtie^P [T(x, y), U(y)]] \\ P^{(4)} &= \pi_{\emptyset}^P \bowtie^P [U(y), \pi_y^P \bowtie^P [R(x), S(x), T(x, y)]] \end{aligned}$$

The propagation score is the minimum of the scores of all minimal plans: $\rho(q) = \min_{i \in \{3,4\}} [\text{score}(P^{(i)})]$.

Note that “safetyzation by dissociation” is not monotone, and dissociation can also make a safe query unsafe. For example, the query $R(x), S(x, y), T(x, y, z)$ is safe, but its dissociation $\tilde{R}(x, \tilde{z}), S(x, y), T(x, y, z)$ is not.

We now describe an algorithm for enumerating all plans that correspond to minimal dissociations. Let $q :- g_1, \dots, g_m$. For each nonempty subset $\bar{s} \subseteq \{1, \dots, m\}$, denote $\text{HVar}[\bar{s}] \subseteq \text{Var}(q)$ to be the following set of variables:

$$\text{HVar}[\bar{s}] = \bigcup_{i \in \bar{s}} \text{Var}(g_i) \cap \bigcup_{j \notin \bar{s}} \text{Var}(g_j)$$

For any subplan P of some plan for q , denote $\bar{s}_P = \{i \mid g_i \in P\}$. One can easily check that, for any subplan P , $\text{HVar}[\bar{s}_P] \subseteq \text{HVar}(P)$. In other words, $\text{HVar}[\bar{s}]$ represents the minimal set of head variables of any subplan over the atoms in \bar{s} . In a minimal plan, we must have equality at each node that corresponds to a projection.

Proposition 7 (Minimal plans). *A plan corresponds to a minimal dissociation of a query iff:*

- (a) for every projection subplan P : $\text{HVar}(P) = \text{HVar}[\bar{s}_P]$
- (b) for every join subplan $P = \bowtie^P [P_1, \dots, P_k]$: if x is a variable s.t. $\exists i. x \in \text{HVar}(P_i) - \text{HVar}[\bar{s}_P]$ then $\forall j \in \{1, \dots, k\}, x \in \text{HVar}(P_j)$. In other words: $\bigcup_i \text{HVar}(P_i) - \bigcap_i \text{HVar}(P_i) = \text{HVar}[\bar{s}_P]$.

The second condition says this. Suppose we have a variable x that appears in some, but not all of the operands P_i of a join. Then we could have projected it earlier, resulting in a smaller dissociation. The only case when we cannot do it is when x is needed later, i.e. $x \in \text{HVar}[\bar{s}_P]$.

Example 5 (Minimal plans). We illustrate for the non-minimal plan $P^{(5)}$ from Fig. 2 and its subplan $P = \bowtie^P [R(x), T(x, y), U(y)]$. P includes subgoals R , T , and U . Here, $\text{HVar}(P) = \{x, y\}$, whereas $\text{HVar}[\{R, T, U\}] = \{x\}$. The difference is $\{y\}$, which should appear in any subplan of P according to Proposition 7(b). However, it does not appear in $R(x)$. Moving $R(x)$ to the later join with $S(x)$ makes the condition fulfilled and results in the minimal plan $P^{(3)}$.

Algorithm 1: (Minimal Plans) generates all plans that correspond to a minimal safe dissociation

Input: Query $q := g_1, \dots, g_m$

Output: All minimal plans $\mathcal{P}(\bar{s})$ for $\bar{s} = \{1, \dots, m\}$

```

foreach  $i \in \{1, \dots, m\}$  do
  if  $\text{HVar}[\{i\}] = \text{Var}(g_i)$  then  $\mathcal{P}[\{i\}] = \{g_i\}$ 
  else  $\mathcal{P}[\{i\}] = \{\pi_{\text{HVar}[\{i\}]}^p(g_i)\}$ 
foreach  $\bar{s} \subseteq \{1, \dots, m\}$  for increasing size of  $|\bar{s}|$  do
  Set  $\mathcal{P}[\bar{s}] = \emptyset$ 
  foreach  $k \geq 2$  and every partition of  $\bar{s} = \bar{s}_1 \cup \dots \cup \bar{s}_k$  do
    if  $\bigcup_i \text{HVar}(P_i) - \bigcap_i \text{HVar}(P_i) = \text{HVar}[\bar{s}]$  then
      forall  $P_1 \in \mathcal{P}[\bar{s}_1], \dots, P_k \in \mathcal{P}[\bar{s}_k]$  do
         $\mathcal{P}[\bar{s}] = \mathcal{P}[\bar{s}] \cup \{\pi_{\text{HVar}[\bar{s}]}^p[\bowtie^p[P_1, \dots, P_k]]\}$ 

```

This proposition gives us immediately Algorithm 1 for enumerating all minimal plans, bottom up. It computes, for each non-empty subset $\bar{s} \subseteq \{1, \dots, m\}$, the set $\mathcal{P}[\bar{s}]$ of all minimal plans over the atoms g_i , with $i \in \bar{s}$, and can use any Systems R style optimizer.

We end this section by commenting on the number of minimal safe dissociations. Not surprisingly, this number is exponential in the size of the query. To see this, consider the following query: $q := R_1(x_1), \dots, R_n(x_n), U(x_1, \dots, x_n)$. There are exactly $n!$ minimal safe dissociations: Take any consistent preorder \preceq on the variables. It must be a total preorder, i.e. for any i, j , either $x_i \preceq x_j$ or $x_j \preceq x_i$, because x_i, x_j occur together in U . Since it is minimal, \preceq must be an order, i.e. we can't have both $x_i \preceq x_j$ and $x_j \preceq x_i$ for $i \neq j$. Therefore, \preceq is a total order, and there are $n!$ such. Note that while the number of safe dissociations is exponential in the size of the query (we do not address query complexity in this paper), the number of query plans is independent of the size of the database, and hence our approach has PTIME data complexity for *all* queries [26].

In summary, our evaluation approach allows to rank answers to *every* query (safe or unsafe) in polynomial time in the size of the database, and is conservative w.r.t. the ranking according to the possible world semantics for both safe queries.

5 Experiments on Synthetic Data

We next illustrate both the (i) efficiency and (ii) quality of the dissociation method for *increasingly intractable* queries and datasets. For timing, we compare against MayBMS [1], an existing state-of-the-art PDB system, to calculate exact probabilities. For quality, we take the results of MayBMS as ground truth for query reliability and report the relative difference between the propagation and reliability scores.

Experimental setup. ► *Queries:* We consider two canonical unsafe ‘chain queries’ $q_1 := R(x), S(x, y), T(y)$ and $q_2 := R(x), S_1(x, y), S_2(y, z), T(z)$. As will

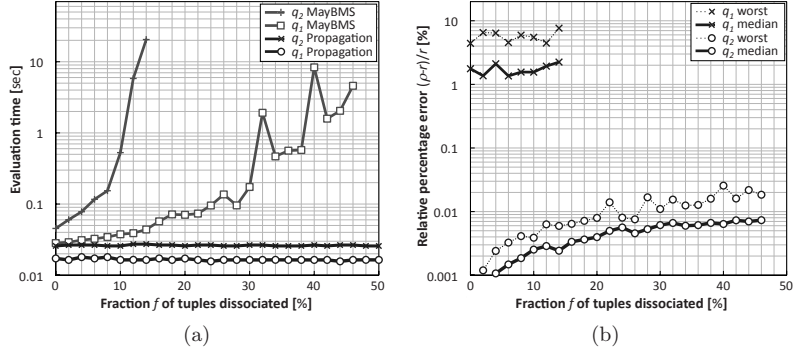


Fig. 3. Results for timing (a) and quality (b) experiments. See text for details.

become clear from the experiments, it is increasingly hard to get the true query reliability scores for more complex queries and datasets, and already with these two simple queries, we quickly enter the area of intractable queries and datasets.

► **Datasets:** Given a fraction $f \in [0, 1]$, we generate the dataset as follows: Both relations R and T contain 500 tuples each. For every tuple x in R , the functional dependency $x \rightarrow y$ on S holds with probability $1 - f$. For those x that satisfy the FD, we randomly choose a y from T and create one tuple (x, y) in S ; for those x that violate the FD, we randomly choose 2 or 3 y from T to create 2 or 3 tuples in S . Hence, the parameter f is the fraction of tuples that are dissociated, and also serves as *measure of the intractability* of the query for MayBMS. Relations S_1 and S_2 are generated analogously according to FDs $x \rightarrow y$ and $y \rightarrow z$. We limit the fanout for each x to be maximum 2 or 3, as otherwise, the *treewidth* increases rapidly and, even for small fraction f , the query quickly becomes intractable for MayBMS, thus limiting the range of f we can get the ground truth for. The probability of each tuple is sampled uniformly from an interval $[0, u]$, with u chosen to create a query reliability of around 0.5. ► **Equipment:** The experiments are run on a Windows Server 2008 machine with Quad Core 2 GHz and 8 GB RAM. Our implementation is done in Java, wherein the program sends an SQL query to a SQL Server 2008 database. The competitor, MayBMS, is implemented in C as an extension of the PostgreSQL 8.3 backend. ► **Evaluation:** For each parameter f in steps of 2%, we generate 20 datasets and evaluate both MayBMS and propagation on each. For timing, we take the sum over all 20 datasets. For quality, we report both the *median* and *worst percentage error* over the 20 datasets, where percentage error is $\frac{\rho-r}{r} \cdot 100\%$, with ρ and r being propagation and reliability scores, respectively. To compute the propagation score, we execute the query using a left-right plan. It is easy to see that, using this plan, the tuples dissociated are exactly those that violate the FDs discussed above. Hence f indeed controls the fraction of tuples that are dissociated to those that could have been dissociated.

Timing experiments. Figure 3a demonstrates that the two simple queries are indeed not trivial, and evaluation with an exact solver like MayBMS quickly becomes intractable. In contrast, propagation evaluates independent of f . The results also reaffirm why we need an approach like this: exact probabilistic inference is expensive, and to scale up we need to go from intensional approaches to *effective extensional approaches* like the one proposed in this paper.

Quality experiments. Figure 3b shows the percentage error for varying fractions of dissociations. We only report data points for which MayBMS finished within 1 min, and we could get the ground truth. The error is very low for q_1 and one could argue this is because each dissociation was small; since each x is connected to at most 3 y in S . But even then, this graph demonstrates that the error doesn't go up steeply for increasing fraction f . In fact, the slope seems to decrease and it looks like the error converges. For q_2 , we could not get enough data points to see the behavior as MayBMS did not finish in time. But one can see, there is an order of difference between the error rate. This isn't surprising, because here dissociation happens twice and the error is compounded. In summary, while we don't have a concrete bound of how much propagation deviates from reliability in the general case, our experiments suggest that if each tuple is dissociated only a few times, then the error rate does not increase steeply as the number of dissociations increases. The error rate may even converge. But this analysis of the exact relationship between query reliability and query propagation requires future theoretical and experimental work.

6 Related Work

Current approaches of probabilistic query evaluation either use exact methods or sampling approaches. *Exact approaches* [1,15,18,25] work well on queries with simple lineage expressions, but perform very poorly on database instances with high tree-width or long lineage expressions. *Sampling approaches* [14,16,23], on the other hand, may not be efficient even on simple queries. Some deterministic approximation algorithms [20,24] have been proposed that can approximate the query probability within any error bound, but they have no guarantee over running time. These approaches decompose a formulae recursively into independent/disjoint components via some heuristic until the approximation guarantee is within the error bound. In contrast, our approach is the only *fully extensional approach* to approximating query probabilities in probabilistic databases and therefore scales independent of the database size.

Our approach of focusing on the extensional evaluation of probabilistic databases seems vaguely related to work on *possibilistic databases* [3,4]. This body of work suggests, as foundation for quantifying uncertainty, a completely different theory, namely that of possibility theory instead of probability theory. However, we are not aware of any experimental evaluation of a possibilistic database, nor of a characterization of the exact data complexity of this model. Hence, it is difficult to compare in terms of theoretical or practical performance.

While our work is motivated by ranking query answers, it is conceptually very different from a recent number of proposals of alternative ranking semantics (see e.g. [17]). While those works are all based on the standard reliability semantics and vary in alternative ranking methods, our goal with this paper is to propose an efficient way to evaluate those scores that can be used for ranking.

Query dissociation is also related to recent work in graphical models that tries to give upper bounds on the partition function of a Markov random field. Wainwright et al. [27] develop a method to *obtain optimal upper bounds* by replacing the original distribution using a *collection of tractable distributions*, i.e. such for which the partition function can be calculated efficiently by a recursive algorithm. In our work, efficient approximations of distributions at the schema level are those that allow a *safe query plan*, and thus can be evaluated in a relational database engine. One up to now unknown corollary of our theory is that *every* query plan is an upper bound on query reliability. We further give an algorithm to find *the minimum of all instance-independent upper bounds*.

7 Conclusion

In this paper, we developed a new scoring function called *propagation* for ranking query results over probabilistic databases. Our semantics is based on a sound and principled theory of *query dissociation*, and can be evaluated efficiently in an off-the-shelf relational database engine for *any type of self-join-free conjunctive query*. We proved that query propagation is an upper bound to query reliability, that both scores coincide for safe queries, and that propagation naturally extends the case of safe queries to unsafe queries. Finally, our experiments validated that propagation is a viable alternative for evaluating intractable queries, i.e. cases of queries and database instances that cannot be handled by the current state-of-the-art in probabilistic query evaluation.

Acknowledgement. This work was supported in part by NSF grants IIS-0513877, IIS-0713576, and IIS-0915054. We thank the anonymous reviewers for helpful comments that improved the quality of presentation of this paper.

References

1. L. Antova, T. Jansen, C. Koch, and D. Olteanu. Fast and simple relational processing of uncertain data. In *ICDE*, 2008.
2. G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *ICDE*, 2002.
3. P. Bosc and O. Pivert. About projection-selection-join queries addressed to possibilistic relational databases. *IEEE T. Fuzzy Systems*, 13(1):124–139, 2005.
4. P. Bosc, O. Pivert, and H. Prade. A model based on possibilistic certainty levels for incomplete databases. In *SUM*, 2009.
5. S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.

6. C. J. Colbourn. *The combinatorics of network reliability*. Oxford University Press, New York, 1987.
7. F. Crestani. Application of spreading activation techniques in information retrieval. *Artif. Intell. Rev.*, 11(6):453–482, 1997.
8. N. N. Dalvi and D. Suciu. Management of probabilistic data: foundations and challenges. In *PODS*, 2007.
9. L. Detwiler, W. Gatterbauer, B. Louie, D. Suciu, and P. Tarczy-Hornoch. Integrating and ranking uncertain scientific data. In *ICDE*, 2009. (see <http://db.cs.washington.edu/propagation>).
10. N. Fuhr and T. Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.*, 15(1):32–66, 1997.
11. W. Gatterbauer, A. K. Jha, and D. Suciu. Dissociation and propagation for efficient query evaluation over probabilistic databases. Technical report, UW-CSE-10-04-01, 2010. (see <http://db.cs.washington.edu/propagation>).
12. E. Grädel, Y. Gurevich, and C. Hirsch. The complexity of query reliability. In *PODS*, 1998.
13. R. V. Guha, R. Kumar, P. Raghavan, and A. Tomkins. Propagation of trust and distrust. In *WWW*, 2004.
14. R. Jampani, F. Xu, M. Wu, L. L. Perez, C. M. Jermaine, and P. J. Haas. MCDB: a Monte Carlo approach to managing uncertain data. In *SIGMOD*, 2008.
15. A. Jha, D. Olteanu, and D. Suciu. Bridging the gap between intensional and extensional query evaluation in probabilistic databases. In *EDBT*, 2010.
16. S. Joshi and C. M. Jermaine. Sampling-based estimators for subset-based queries. *VLDB J.*, 18(1):181–202, 2009.
17. J. Li, B. Saha, and A. Deshpande. A unified approach to ranking in probabilistic databases. *PVLDB*, 2(1):502–513, 2009.
18. D. Olteanu and J. Huang. Using OBDDs for efficient query evaluation on probabilistic databases. In *SUM*, 2008.
19. D. Olteanu, J. Huang, and C. Koch. Sprout: Lazy vs. eager query plans for tuple-independent probabilistic databases. In *ICDE*, 2009.
20. D. Olteanu, J. Huang, and C. Koch. Approximate confidence computation in probabilistic databases. In *ICDE*, 2010.
21. J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers, San Mateo, Calif., 1988.
22. M. R. Quillian. Semantic memory. In *Semantic Information Processing*, pages 227–270. MIT Press, 1968.
23. C. Ré, N. N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, 2007.
24. C. Ré and D. Suciu. Approximate lineage for probabilistic databases. *PVLDB*, 1(1):797–808, 2008.
25. P. Sen and A. Deshpande. Representing and querying correlated tuples in probabilistic databases. In *ICDE*, 2007.
26. M. Y. Vardi. The complexity of relational query languages (extended abstract). In *STOC*, 1982.
27. M. J. Wainwright, T. Jaakkola, and A. S. Willsky. A new class of upper bounds on the log partition function. *IEEE Transactions on Information Theory*, 51(7):2313–2335, 2005.
28. J. Weston, A. Elisseeff, D. Zhou, C. S. Leslie, and W. S. Noble. Protein ranking: from local to global structure in the protein similarity network. *Proc Natl Acad Sci USA*, 101(17):6559–63, Apr 2004.

Generalized Uncertain Databases: First Steps*

Parag Agrawal, Jennifer Widom

Stanford University

Abstract. Existing uncertain databases have difficulty managing data when exact confidence values or probabilities are not available. Confidence values may be known imprecisely or coarsely, or even be missing altogether. We propose a *generalized uncertain database* that can manage data with such incomplete knowledge of uncertainty. We develop a semantics for generalized uncertain databases based on Dempster-Shafer theory. We propose a representation scheme for generalized uncertain databases that generalizes the *Trio* representation. Our approach builds upon Trio’s query processing techniques to extend them to operate on generalized uncertain databases.

1 Introduction

There has been a lot of recent research on the topic of *uncertain databases*. The intent is to enable databases to manage data that has incomplete or imprecise information. However, most uncertain databases require that *exact* confidence values (or probabilities) are attached to the data being managed. Effectively, only *aleatory uncertainty* [17] can be represented and managed. On the other hand, knowledge about uncertainty may be incomplete in the case of *epistemic uncertainty* [17]. In this paper, we take initial steps towards managing incomplete uncertainty, by proposing the notion of a *generalized uncertain database*: a database that manages incomplete information about uncertainty. In addition to aleatory uncertainty, generalized uncertain databases can also manage epistemic uncertainty.

There are many different ways in which uncertainty can be incomplete. We use uncertain data about movies and release years, obtained as a result of information extraction, say, to illustrate some different kinds of uncertainty. In the following, (1) is an example of “complete” uncertainty, while (2), (3) and (4) illustrate three kinds of incomplete uncertainty.

1. *Exact confidence values*: **The Godfather** was released in 1972 with confidence .8 and 1974 with confidence .2.
2. *Missing confidence values*: **Shawshank Redemption** was released in 1984 or 1994.
3. *Imprecise confidence values*: **Pulp Fiction** was released in 1994 with confidence at least .5. (Effectively, confidence is between .5 and 1.)

* This work was supported by the National Science Foundation under grants IIS-0414762 and IIS-0904497.

4. *Coarse confidence values:* **Die Hard** was released in 1988 or 1989 with confidence .8, or 1990 with confidence .2.

A naive way to managing incomplete uncertainty would be to assign exact confidence values when values are not available, thereby coercing data into an existing uncertain data model; for example *Trio* [19] assigns confidence values *uniformly* to all alternatives in the case of missing confidences. Such an approach may yield confidence values for query result tuples that are not implied by the data. Also, queries asking for all tuples that “could have confidence .5”, “have confidence at least .5”, or “have confidence at most .5” cannot be answered.

Incomplete uncertainty has been researched extensively from a mathematical perspective, and this research has resulted in the development of rich theories like Dempster-Shafer theory [17]. But the current breed of uncertain data models and systems are not designed to handle such incomplete uncertainty, since they are based on Bayesian probability. Previous work has recognized the need to manage specific kinds of incomplete uncertainty [3], but has not provided solutions.

Incomplete uncertainty also arises when we relax the strict independence assumption that is made by most models and systems for uncertain data. Consider the following two pieces of uncertain data with exact confidence values:

- **The Godfather** was released in 1972 with confidence .8 and 1974 with confidence .2.
- **The Godfather II** was released in 1972 with confidence .3 and 1974 with confidence .7.

Suppose we want to know whether **The Godfather** was released at least one year before **The Godfather II**. Assuming independence, the confidence would be .56. Without assuming independence, Fréchet-Hoeffding bounds indicate that the confidence is at least $\max(.8 + .7 - 1, 0) = .5$ and at most $\min(.8, .7)$, but we cannot be more precise than that. Hence, we might get imprecise confidence values having started with exact confidences, when no assumption can be made about independence.

A generalized uncertain database can manage a combination of aleatory and epistemic uncertainty, and hence is a strict generalization of uncertain databases: a generalized uncertain database degenerates to an uncertain database when exact confidence values are provided.

The key ingredients of our approach to managing incomplete uncertainty in databases are:

- *Data and query semantics:* We propose a semantics for generalized uncertain databases based on Dempster-Shafer theory, and the associated semantics for queries over such databases. We have chosen Dempster-Shafer theory because it is a mature and elegant generalization of Bayesian probability theory that incorporates epistemic uncertainty in addition to aleatory uncertainty. We demonstrate how the new semantics degenerate to the semantics for current uncertain databases. (Section 2)

- *Representation*: We propose a representation scheme for generalized uncertain databases. Our new scheme extends the representation used by Trio, but preserves the notion of *lineage*. We demonstrate how our motivating examples can be represented using this scheme. In fact, the use of lineage ensures that our scheme is *complete* for generalized uncertain databases, and hence can represent all generalized uncertain database instances. (Section 3)
- *Query processing*: We adapt Trio’s query processing techniques to operate over our representation for generalized uncertain databases. We reuse without modification the *eager data processing* aspects of Trio query processing, which also computes lineage. We generalize the *lazy confidence computation* problem to a *lazy uncertainty evaluation* problem to agree with the specified semantics. Like confidence computation, uncertainty evaluation is #P-hard in general, but PTIME when the lineage is conjunctive. (Section 4)

We emphasize that our new proposal requires only minor modifications to the semantics, representation, and query processing algorithms for uncertain databases, while still enabling the management of various kinds of incomplete uncertainty in a unified manner.

We discuss related work in Section 5 and list interesting directions for future research in Section 6.

2 Semantics

In this section, we define data-model semantics for generalized uncertain databases, and the associated query semantics.

2.1 Data Model

The standard semantics for uncertain databases uses the notion of *possible worlds*: the possible states for the database, with a probability distribution defined over the set of possible states. We relax the requirement of a probability distribution, and instead only require a Dempster-Shafer *mass distribution* [17].

Definition 1 (Generalized Uncertain Database) A generalized uncertain database U consists of a finite set of possible worlds PW and a mass function $m : 2^{PW} \rightarrow [0, 1]$ such that:

$$\sum_{S \in 2^{PW}} m(S) = 1$$

Notice that m assigns mass to every subset of possible worlds (although many may have mass zero), as opposed to the traditional Bayesian way of assigning a probability to each individual possible world. It is useful to interpret mass $m(S)$ assigned to a set of possible worlds S as probability mass constrained to stay within S , but free to be assigned anywhere within S . For example, when $S = \{W_1, W_2\}$, it is unspecified how $m(S)$ is to be split between W_1 and W_2 .

Since we use the Dempster-Shafer interpretation for the mass function, we also have the *belief* and *plausibility* functions, which can be interpreted as the lower and upper bounds for probabilities:

$$\begin{aligned} bel(S) &= \sum_{A \subseteq S} m(A) \\ pl(S) &= \sum_{A \cap S \neq \emptyset} m(A) \end{aligned}$$

This interpretation allows us to answer queries that ask for tuples which “could have probability more than .5” (mentioned in Section 1). We don’t further describe details or intuition for belief or plausibility functions in this paper, and instead refer the reader to [17]. We do, however, note the following relationships:

$$\begin{aligned} m(S) &= \sum_{A \subseteq S} -1^{|S-A|} bel(A) \\ pl(S) &= 1 - bel(\bar{S}) \end{aligned}$$

We now illustrate how generalized uncertain databases allow us to represent incomplete uncertainty. We consider single-tuple databases based on our examples in Section 1 and construct mass functions for each.

1. *Exact confidence values*: This is the standard probabilistic case, and hence all the mass is contained in sets with a single possible world. It is worth noting that the mass function is the same as the belief and the plausibility function, and hence mass can be interpreted as probability.

W_1	W_2
The Godfather, 1972	The Godfather, 1974

$$\begin{aligned} m(S) &= .8, & S &= \{W_1\} \\ &= .2, & S &= \{W_2\} \\ &= 0, & & otherwise \end{aligned}$$

2. *Missing confidence values*: Since no confidence values are known, all the mass is assigned to the set of all possible worlds. For all individual possible worlds, the belief and plausibility functions are 0 and 1 respectively.

W_1	W_2
Shawshank Redemption, 1984	Shawshank Redemption, 1994

$$\begin{aligned} m(S) &= 1, & S &= \{W_1, W_2\} \\ &= 0, & & otherwise \end{aligned}$$

3. *Imprecise confidence values*: Imprecision about confidence can be represented by assigning mass to the set with all possible worlds. For our example, we get belief and plausibility values of .5 and 1 for W_1 , and 0 and .5 for W_2 .

W_1	W_2
Pulp Fiction, 1994	\emptyset

$$\begin{aligned}
 m(S) &= .5, & S &= \{W_1\} \\
 &= .5, & S &= \{W_1, W_2\} \\
 &= 0, & & \text{otherwise}
 \end{aligned}$$

4. *Coarse confidence values*: In this case, confidence is specified over sets of possible worlds, but not exactly to each individual possible world.

W_1	W_2	W_3
Die Hard, 1988	Die Hard, 1989	Die Hard, 1990

$$\begin{aligned}
 m(S) &= .8, & S &= \{W_1, W_2\} \\
 &= .2, & S &= \{W_3\} \\
 &= 0, & & \text{otherwise}
 \end{aligned}$$

Since all the above kinds of uncertainty can be captured in generalized uncertain databases, we can provide a unified semantics for various kinds of incomplete uncertainty and exact probabilities. The Trio system allows relations with exact probabilities and missing confidences, but a query joining these relations coerces the relation with missing probabilities by assigning them uniformly, and hence the results are ad-hoc. In contrast, our new proposal is able to combine such information in a principled manner.

We now formalize the observation¹ that our new data-model semantics generalize semantics for probabilistic databases [19].

Observation 1 (Probabilistic Databases) *Probabilistic databases are those generalized uncertain databases where:*

$$\sum_{|S|=1} m(S) = 1$$

Probabilistic mass functions are those mass functions that satisfy the property above.

We also observe that our semantics generalizes uncertain databases without probabilities as described in [19].

Observation 2 (Uncertain Databases without Probabilities) *Uncertain databases without probabilities are those generalized uncertain databases where $m(PW) = 1$.*

¹ We omit all proofs in this presentation.

2.2 Query Semantics

We now describe semantics of queries over generalized uncertain databases.

Definition 2 (Queries over Generalized Uncertain Databases) *The result of a relational query Q over a generalized uncertain database U with possible worlds PW_U and mass function m_U is a generalized uncertain database R . The possible worlds PW_R and mass function $m_R : 2^{PW_R} \rightarrow [0, 1]$ for R are:*

$$PW_R = \{Q(W) | W \in PW_U\}$$

$$m_R(S) = \sum_{\substack{A \text{ such that} \\ S = \{Q(W) | W \in A\}}} m_U(A)$$

Notice that these semantics correspond to those of probabilistic databases in spirit: the set PW_R is the same in both cases, and the mass function over PW_R is constructed in a manner similar to the construction of probability values in probabilistic databases. In fact, the following observation formalizes that query semantics for generalized uncertain databases degenerate to those of probabilistic databases.

Observation 3 *When the mass function m for a generalized uncertain database is probabilistic: $\sum_{|S|=1} m(S) = 1$, the mass function for the generalized uncertain database resulting from the application of a relational query is also probabilistic, and is the same as that provided by semantics for probabilistic databases.*

For completeness, we also make the following observation:

Observation 4 *When the mass function for a generalized uncertain database corresponds to an uncertain database without confidences: $m(PW) = 1$, the mass function for the generalized uncertain database resulting from the application of a relational query also corresponds to an uncertain database without confidences.*

3 Representation

In this section, we describe our new representation scheme for generalized uncertain databases. This scheme is a modification of the representation scheme used by Trio [19] in the sense that it preserves the notions of *alternatives*, *x-tuples* and *lineage*. The significant change is that probabilities associated with alternatives are replaced by a mass function defined for each x-tuple.

We start by describing our scheme without lineage, and then incorporate lineage. A database instance is represented by a set of *x-tuples*, each with an associated *x-mass function*. We use a running example to describe these notions, and to show how a generalized uncertain database instance is interpreted from a representation instance.

- *Alternatives and x-tuples*: Databases 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, the following relation consists of two x-tuples from our running example.

t_1	Die Hard, 1988	Die Hard, 1989	Die Hard, 1990
t_2	Pulp Fiction, 1994		?

Possible worlds for the database are obtained by picking exactly one alternative from each x-tuple. Symbol “?” is a special alternative value; picking it denotes that none of the other alternatives are picked [19]. The first x-tuple has three possible states, while the second x-tuple has two possible states. Hence, the possible world set PW has six possible worlds as shown below.

W_1	W_2	W_3
Die Hard, 1988	Die Hard, 1989	Die Hard, 1990
Pulp Fiction, 1994	Pulp Fiction, 1994	Pulp Fiction, 1994
W_4	W_5	W_6
Die Hard, 1988	Die Hard, 1989	Die Hard, 1990

- *x-mass functions*: An x-tuple with the set of alternatives A has an associated x-mass function $m : 2^A \rightarrow [0, 1]$ such that $\sum_{S \subseteq A} m(S) = 1$. For the x-tuples above, the mass functions over single-tuple databases as shown in Section 2 are valid x-mass functions:

$$\begin{aligned}
 x_1(S) &= .8, & S &= \{(\text{Die Hard}, 1988), (\text{Die Hard}, 1989)\} \\
 &= .2, & S &= \{(\text{Die Hard}, 1990)\} \\
 &= 0, & & \text{otherwise} \\
 x_2(S) &= .5, & S &= \{(\text{Pulp Fiction}, 1994)\} \\
 &= .5, & S &= \{\} \\
 &= 0, & & \text{otherwise}
 \end{aligned}$$

The intuitions described for mass functions in Section 2 carry over to x-mass functions. We have an x-mass function for each x-tuple in the representation. From these functions, we interpret the mass function m over the possible worlds to get a generalized uncertain database from a representation instance. Conceptually, we first interpret each x-mass function as a *basic mass function* over the set of possible worlds, and then “combine” these basic mass functions to obtain the mass function for the generalized uncertain database.

- *Basic mass functions*: In general, the basic mass function $m_i : 2^{PW} \rightarrow [0, 1]$ for an x-mass function $x_i : 2^A \rightarrow [0, 1]$ is constructed as follows:

$$\begin{aligned}
 m_i(S) &= x_i(B), & S &= \{W \in PW : \exists a \in B \subseteq A, a \in W\} \\
 &= 0, & & \text{otherwise}
 \end{aligned}$$

Consider the x-mass functions x_1, x_2 for the tuples above. We create the corresponding basic mass functions m_1, m_2 as follows:

$$\begin{aligned}
 m_1(S) &= .8, & S &= \{W_1, W_2, W_4, W_5\} \\
 &= .2, & S &= \{W_3, W_6\} \\
 &= 0, & & \text{otherwise} \\
 m_2(S) &= .5, & S &= \{W_1, W_2, W_3\} \\
 &= .5, & S &= \{W_1, W_2, W_3, W_4, W_5, W_6\} \\
 &= 0, & & \text{otherwise}
 \end{aligned}$$

- *Combination*: The basic mass functions m_i for all x-tuples are combined to obtain the mass function m for the generalized uncertain database using Dempster's combination rule:

$$m = \bigoplus_i m_i$$

where the operator \oplus is defined² as follows [17]:

$$\begin{aligned}
 (m_1 \oplus m_2)(S) &=_{\text{defn}} \sum_{A \cap B = S} m_1(A) \cdot m_2(B), & S &\neq \emptyset \\
 &=_{\text{defn}} 0, & S &= \emptyset
 \end{aligned}$$

m is uniquely defined by the above expression, since Dempster's rule is known to be associative and commutative. For the example above, we obtain the following mass function m :

$$\begin{aligned}
 m(S) &= .4, & S &= \{W_1, W_2\} \\
 &= .1, & S &= \{W_3\} \\
 &= .4, & S &= \{W_1, W_2, W_4, W_5\} \\
 &= .1, & S &= \{W_3, W_6\} \\
 &= 0, & & \text{otherwise}
 \end{aligned}$$

It must be noted that Dempster's rule makes an assumption that translates to the independence assumption in the probabilistic case. We show an example that illustrates that the above interpretation degenerates to the interpretation in probabilistic databases under the independence assumption. Suppose, x_1, x_2 for the example above were both probabilistic:

² We have simplified the combination rule by eliminating normalization to account for conflict (usually denoted by K), because construction of the basic mass function ensures that we never encounter non-zero conflict.

$$\begin{aligned}
x_1(S) &= .5, & S &= \{(\text{Die Hard}, 1988)\} \\
&= .3, & S &= \{(\text{Die Hard}, 1989)\} \\
&= .2, & S &= \{(\text{Die Hard}, 1990)\} \\
&= 0, & \text{otherwise} \\
x_2(S) &= .5, & S &= \{(\text{Pulp Fiction}, 1994)\} \\
&= .5, & S &= \{?\} \\
&= 0, & \text{otherwise}
\end{aligned}$$

We would get the corresponding basic mass functions m_1, m_2 as:

$$\begin{aligned}
m_1(S) &= .5, & S &= \{W_1, W_4\} \\
&= .3, & S &= \{W_2, W_5\} \\
&= .2, & S &= \{W_3, W_6\} \\
&= 0, & \text{otherwise} \\
m_2(S) &= .5, & S &= \{W_1, W_2, W_3\} \\
&= .5, & S &= \{W_4, W_5, W_6\} \\
&= 0, & \text{otherwise}
\end{aligned}$$

Hence, we get the mass function m for the generalized uncertain database as:

$$\begin{aligned}
m(S) &= .25, & S &= \{W_1\} \\
&= .15, & S &= \{W_2\} \\
&= .1, & S &= \{W_3\} \\
&= .25, & S &= \{W_4\} \\
&= .15, & S &= \{W_5\} \\
&= .1, & S &= \{W_6\} \\
&= 0, & \text{otherwise}
\end{aligned}$$

Notice that this mass function m is probabilistic. In fact, it corresponds directly to the semantics of probabilistic databases. We formalize the intuition suggested by this example below:

Observation 5 *When the x -mass functions for all x -tuples in a generalized uncertain database are probabilistic, the mass function for the generalized uncertain database is also probabilistic, and corresponds to the probability distribution defined in probabilistic databases.*

Similarly, we also have:

Observation 6 *When the x -mass functions for all individual x -tuples in a generalized uncertain database correspond to uncertain databases without confidences, the mass function for the generalized uncertain database also corresponds to an uncertain database without confidences.*

Based on results from [11], we can immediately see that the model presented so far for generalized uncertain databases is not *complete*, or even closed under select-project-join queries. Hence, we incorporate the notion of *lineage* [19] from Trio into the model.

- *Lineage*: Informally, lineage is a function $\lambda : T \rightarrow \text{Bool}(T)$ where $T = \cup_{W \in PW} W$. For each alternative in a database, lineage provides a Boolean formula over alternatives. Consider an additional x -tuple t_3 in the example database above with:

t_3	1989, 1994	?
-------	------------	---

$$\lambda(t_3, 1) = (t_1, 2) \wedge (t_2, 1)$$

We refer the reader to [19] for a comprehensive description of lineage. As in [19], we use (t_i, j) to reference the j^{th} alternative of t_i . x -tuples may be *base x -tuples* when none of their alternatives have lineage defined (t_1, t_2 in the example) or *derived* when all alternatives have lineage defined (t_3 in the example). The possible worlds PW and mass function m over possible worlds are interpreted based on the base x -tuples only, as described above. An alternative a from a derived x -tuple is added to a possible world W if its (transitive) lineage formula [19] $\lambda^*(a)$ is “**true**” in the world W . Since $(t_1, 2)$ and $(t_2, 1)$ are both present in W_2 , lineage of $(t_3, 1)$ is **true** in W_2 . Similarly, lineage of $(t_3, 1)$ is **true** in W_1, W_2 , and lineage of $(t_3, 2)$ is **true** in W_2 and W_5 . Hence, the possible worlds for the relation with x -tuples t_1, t_2, t_3 are:

W_1	W_2	W_3
Die Hard, 1988	Die Hard, 1989	Die Hard, 1990
Pulp Fiction, 1994	Pulp Fiction, 1994	Pulp Fiction, 1994
	1989, 1994	
W_4	W_5	W_6
Die Hard, 1988	Die Hard, 1989	Die Hard, 1990

Our model requires that an x -mass function be provided for each base x -tuple, while no x -mass functions may be explicitly provided for derived tuples (just like confidences in Trio). Hence, the mass function over possible worlds is not affected due the presence of lineage, and is constructed based on base x -tuples only. The mass function for the generalized uncertain database with x -tuples t_1, t_2, t_3 is as shown earlier.

We can now state the following theorem:

Theorem 1. *The representation scheme presented above is complete: there exists a way to represent every generalized uncertain database instance using the representation scheme.*

A complete representation scheme is necessarily closed under any class of queries, when semantics are defined according to Definition 2. We omit the proof of the above theorem, but note that a representation instance can be constructed for each model instance in a manner very similar to [11]. The constructive proof creates a “dummy” base x-tuple which has an alternative t_W corresponding to each possible world W in the model instance to be represented. Corresponding to each tuple t in the model instance, a derived x-tuple $t|?$ is created; the lineage of t is the disjunction over all alternatives t_W in the dummy tuple such that $t \in W$.

4 Query Processing

We consider select-project-join queries over generalized uncertain databases. We adapt Trio’s query processing techniques to operate over our representation for generalized uncertain databases. We reuse without modification the *eager data processing* aspects of Trio query processing. Query results include lineage formulas identifying the data from which each result alternative is derived in the same way as [19]. Hence, lineage plays the dual role of allowing tracing origins of result data, while at the same time properly representing the semantics for generalized uncertain databases as described in Section 3. The notion of lineage used by Trio is sufficient to enable the extended semantics of generalized uncertain databases. We do not describe this part of query processing, and instead refer the reader to [19].

We generalize the *lazy confidence computation* problem in Trio to a *lazy uncertainty evaluation* problem for generalized uncertain databases, to agree with the new semantics.

4.1 Uncertainty Evaluation

Uncertainty evaluation occurs after result data and lineage have been computed during the data processing phase. It involves computing mass, belief, or plausibility values for a result tuple, henceforth collectively referred to as *uncertainty values*. More generally, we allow computing uncertainty values for boolean formulas over tuples. The formalization below states that uncertainty values for a boolean formula are the corresponding values for the set of possible worlds where the formula is satisfied.

Definition 3 *Consider Boolean formula f consisting of variables corresponding to tuples in a generalized uncertain database: $f \in \text{Bool}(T)$ where $T = \cup_{W \in PW} W$. We define the mass, belief, and plausibility values for the formula f*

as follows:

$$\begin{aligned} m(f) &= m(\{W \in PW : f \text{ is true in } W\}) \\ \text{bel}(f) &= \text{bel}(\{W \in PW : f \text{ is true in } W\}) \\ \text{pl}(f) &= \text{pl}(\{W \in PW : f \text{ is true in } W\}) \end{aligned}$$

Uncertainty evaluation is the problem of computing an uncertainty value for an input boolean formula. This problem is very general because it lets us ask about the uncertainty in: (1) individual tuples, using a degenerate boolean formula; (2) possible worlds, posed as a conjunction of all tuples; (3) any collection of tuples, and their relationships expressed as an arbitrary formula.

The first part of uncertainty evaluation involves transforming the input boolean formula f to another formula $f_b \in \text{Bool}(T_b)$, $T_b = \{t \in \cup_{W \in PW} W : t \text{ is a base tuple}\}$ over base tuples. Lineage is used in making this transformation, essentially by replacing any derived tuple by its lineage formula, transitively. We need no modifications to this part of the algorithm, and refer the reader to [19] for details.

Now we discuss the uncertainty evaluation problem for boolean formulas over base tuples. We observe that this problem generalizes confidence computation over boolean formulas in Trio: when input mass functions are probabilistic, all three of mass, belief, and plausibility functions degenerate to probabilities. Since confidence computation is known to be #P-hard, this observation immediately establishes that the uncertainty evaluation problem is also #P-hard.

Theorem 2. *Uncertainty evaluation for boolean formulas over base tuples is #P-hard.*

Hence, uncertainty evaluation can be expensive in general. The naive algorithm enumerates the “truth table” E for the boolean formula, and computes the mass function over 2^E by combining individual mass functions using Dempster’s rule. Efficient algorithms have been proposed for computation of belief and plausibility functions using a mass function based on fast Möbius transform [13].

It is well known that in probabilistic databases, when the boolean formula over base tuples is conjunctive, the confidence can be evaluated efficiently. This result carries over to the uncertainty evaluation problem:

Theorem 3. *Uncertainty evaluation for conjunctive boolean formulas over base tuples is PTIME.*

We omit the proof for this theorem, but make the following comment. Belief, plausibility, and mass can each be treated as probabilities (lower, upper, and mass, respectively) when the boolean formula over base tuples is conjunctive, and the confidence computation module of Trio can be used directly for uncertainty evaluation.

Approximation techniques have been proposed for the confidence computation problem [10]. The observations above suggest that such techniques can be

adapted to solve the uncertainty evaluation problem. Thorough investigation of the uncertainty evaluation problem is left as future work; specific directions are listed in Section 6.

5 Related Work

Generalized uncertain databases are based on Dempster-Shafer theory [17], whereas probabilistic databases are based on Bayesian probability theory. There has been a lot of work on Dempster-Shafer theory: [20] provides a good selection of research. Dempster-Shafer theory doesn't satisfy [8] the assumptions of Cox's Theorem [9], and hence the argument that probability theory provides the only "consistent" way of managing uncertainty doesn't apply.

Extended relational models based on Dempster-Shafer theory have been previously proposed [6, 7, 14–16, 21]. However, these models are incomplete, and do not enable efficient query processing. More recently, there has been a flurry of research focused on modeling and managing aleatory uncertainty in databases [2, 5, 12, 18, 19]. Techniques for efficient query processing have been developed and implemented as a result of this research. Our approach bridges the gap between these two lines of research.

There has also been recent research on possibilistic databases [4] based on possibility theory. Like our approach, this proposal also enables managing of data where no exact probability values are available. Possibility theory can be captured in Dempster-Shafer theory in terms of representation, but it uses a cautious combination operator instead of Dempster's rule.

6 Future Work

This paper provides first steps towards enabling databases to manage uncertain data that may not have exact confidence values. In doing so, it exposes numerous challenges for managing data with incomplete uncertainty. We identify the following as some interesting directions for future work:

- Adding information can decrease epistemic uncertainty, giving more precise answers. Towards this end, we are investigating the data integration problem for generalized uncertain databases. Some early results from this line of investigation are presented in [1].
- We have not extensively explored the problem of efficient uncertainty evaluation for generalized uncertain databases. We expect that efficient techniques can be derived by leveraging past research on Dempster-Shafer theory, possibly by making restrictions on the structure of mass functions and lineage.
- Approximation techniques have been proposed for confidence evaluation in uncertain databases. It will be interesting to investigate whether they can be extended to generalized uncertain databases.
- Implementation of these techniques in the Trio system is a natural next step.

Acknowledgements

We thank the anonymous reviewers for their helpful comments.

References

1. P. Agrawal, A. Das Sarma, J. Ullman, and J. Widom. Foundations of uncertain-data integration. In *Proc. of VLDB*, 2010.
2. L. Antova, C. Koch, and D. Olteanu. MayBMS: Managing Incomplete Information with Probabilistic World-Set Decompositions. In *Proc. of ICDE*, 2007.
3. D. Barbara, H. Garcia-Molina, and D. Porter. A probabilistic relational data model. In *Proc. of EDBT*, 1990.
4. P. Bosc and O. Pivert. About projection-selection-join queries addressed to possibilistic relational databases. *IEEE T. Fuzzy Systems*, 2005.
5. 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.
6. S. Choenni, H. E. Blok, and M. Fokkinga. Extending the relational model with uncertainty and ignorance. *Internal Report, University of Twente*, 2004.
7. S. Choenni, H. E. Blok, and E. Leertouwer. Handling uncertainty and ignorance in databases: A rule to combine dependent data. In *DASFAA*, 2006.
8. M. Colyvan. Is probability the only coherent approach to uncertainty? In *Risk Analysis*, 2008.
9. R. T. Cox. Probability, frequency and reasonable expectation. In *Readings in uncertain reasoning*, 1946.
10. N. Dalvi and D. Suciu. Efficient Query Evaluation on Probabilistic Databases. In *Proc. of VLDB*, 2004.
11. A. Das Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working Models for Uncertain Data. In *Proc. of ICDE*, 2006.
12. A. Deshpande, L. Getoor, and P. Sen. Graphical models for uncertain data. In *Managing and Mining Uncertain Data*. Springer, 2009.
13. R. Kennes and P. Smets. Fast algorithms for dempster-shafer theory. In *Uncertainty in Knowledge Bases*. Springer, 1991.
14. S. K. Lee. An extended relational database model for uncertain and imprecise information. In *Proc. of VLDB*, 1992.
15. S. K. Lee. Imprecise and uncertain information in databases: An evidential approach. In *Proc. of ICDE*, 1992.
16. E.-P. Lim, J. Srivastava, and S. Shekhar. An evidential reasoning approach to attribute value conflict resolution in database integration. *IEEE Trans. on Knowl. and Data Eng.*, 1996.
17. G. Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, 1976.
18. 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.
19. J. Widom. Trio: A system for data, uncertainty, and lineage. In *Managing and Mining Uncertain Data*. Springer, 2008.
20. R. R. Yager and L. Liu. *Classic Works of the Dempster-Shafer Theory of Belief Functions*. Springer, 2008.
21. L. A. Zadeh. A simple view of the dempster-shafer theory of evidence and its implication for the rule of combination. In *AI Magazine*, 1986.

Tuple Merging in Probabilistic Databases

Fabian Panse and Norbert Ritter

Universität Hamburg, Vogt-Kölln Straße 33, 22527 Hamburg, Germany
{panse,ritter}@informatik.uni-hamburg.de
<http://vsis-www.informatik.uni-hamburg.de/>

Abstract. Real-world data are often uncertain and incomplete. In probabilistic relational data models uncertainty can be modeled on two levels. First by representing the uncertain instance of a tuple by a set of possible instances and second by assigning each tuple with its degree of membership to the considered relation. To overcome incompleteness, data from multiple sources need to be combined. In order to combine data from autonomous probabilistic databases, an integration of probabilistic data has to be performed. Until now, however, data integration approaches have focused on the integration of certain source data (relational or XML). There has been only less attention on the integration of uncertain (esp. probabilistic) source data so far. In this paper, we consider probabilistic tuple merging being an essential step in the integration of probabilistic data. We present techniques for merging uncertain instance data as well as for merging different degrees of tuple membership.

Key words: data integration, tuple merging, probabilistic database

1 Introduction

The increasing need for applications that produce uncertain data (e.g. in the area of astronomy [24]) has attracted high attention of uncertain data management in the database research community in recent years. Thus, several probabilistic data models have been proposed [9, 4, 16, 5, 26] and several probabilistic database prototypes have been designed [2, 17, 8].

Nevertheless, current approaches for data integration mostly consider a probabilistic handling of uncertainty emerging during the integration of certain source data (e.g. [13, 25, 26]). Integration of uncertain (esp. probabilistic) source data has only been rarely addressed so far [1]. However, to combine probabilistic data from multiple sources, for example for unifying data produced by different space telescopes, an integration of probabilistic source data is required.

In general, a data integration process consists of two steps: (a) duplicate detection [15] for identifying multiple representations of same real-world entities and (b) tuple merging [18, 23, 6] (also known as data fusion [7]) to consolidate multiple representations to a single one.

In [21] we already adapted existing techniques for duplicate detection to probabilistic data. From this duplicate detection process a set of tuple cluster

(one cluster for one real-world entity) results. In this paper, we present techniques for merging the tuples of each cluster to a single probabilistic representation.

The contributions of this paper are: (i) we define requirements for an ideal tuple merging, (ii) we present an ideal merging of instance data and (iii) we present strategies for merging tuple memberships.

The paper is structured as follows. First we present the concept of probabilistic tuples (Section 2). Then we formalize the problem of merging multiple probabilistic tuples in Section 3. In Section 4, we introduce a strategy for tuple merging in probabilistic databases. We present techniques for merging the possible instances of two probabilistic tuples (Section 4.1) as well as techniques for merging the tuples' degrees of membership (Section 4.2). Related work is presented in Section 5. Section 6 concludes the paper. Finally, in Section 7 open problems and future challenges are discussed.

2 Probabilistic Tuples

In probabilistic relational models, uncertainty is modeled on two levels: (a) each tuple t is assigned with a probability $p(t)_{\mathcal{R}} \in (0, 1]$ denoting the likelihood, also called membership degree, that t belongs to the corresponding relation \mathcal{R} (membership level), and (b) alternatives for attribute values are given (instance level).

In earlier approaches, alternatives of different attribute values are considered to be independent (e.g. [4]). In these models, each attribute value can be considered as a separate random variable with its own probability distribution. Newer models like ULDB [5] or MayBMS [17] support dependencies by introducing new concepts like ULDB's x-tuple and MayBMS's U-relation.

To get a general representation of the uncertain information captured by a single tuple, we consider a representation model which is defined within the *possible world semantics*. Theoretically, a probabilistic tuple can be considered as a set of possible instances together with the probability of each instance to be the true instance (uncertainty on instance level) and the tuple's degree of membership to the considered relation (uncertainty on membership level).

Definition 1 (Probabilistic Tuple): Let \mathcal{R} be a probabilistic relation and $\mathcal{A} = \{A_1, A_2, \dots, A_n\}$ its finite set of attributes where each attribute $A_i \in \mathcal{A}$ has a finite domain D_i . A probabilistic tuple $t \in \mathcal{R}$ is defined as $t = ((\mathcal{I}, P), p(t)_{\mathcal{R}})$ where $\mathcal{I} = \{I_1, \dots, I_k\} \subseteq D_1 \times D_2 \times \dots \times D_n$ is the set of possible instances of t , $P : \mathcal{I} \rightarrow (0, 1]$, $\sum_{I \in \mathcal{I}} P(I) = 1$ is the probability distribution over these instances, and $p(t)_{\mathcal{R}}$ is degree of membership of t to \mathcal{R} .

Each possible instance can be considered as a certain tuple. Along with its probabilities the set of possible instances is in the following denoted as instance data. Maybe-tuples are tuples for which the membership to the considered relation is uncertain and hence have a membership degree smaller than 1.

In the following, we present a probabilistic tuple by an own table, where each row represents a possible instance together with its probability. An extra column specifies the membership of the whole tuple. An example is depicted in Figure 1.

	$A_1 : D_1$	$A_2 : D_2$	\dots	$A_n : D_n$	$P(I)$	$p(t)_{\mathcal{R}}$
I_1	v_{11}	v_{12}	\dots	v_{1n}	0.6	0.8
I_2	v_{21}	v_{22}	\dots	v_{2n}	0.4	

Fig. 1. Representation of a probabilistic tuple $t = ((\{I_1 = (v_{11}, v_{12}, \dots, v_{1n}), I_2 = (v_{21}, v_{22}, \dots, v_{2n})\}, P : \{I_1 \mapsto 0.6, I_2 \mapsto 0.4\}), 0.8) \in \mathcal{R}$ based on the *possible world semantics*

2.1 Tuple Membership

In certain and complete data, to each entity an exact position within the real-world is assigned (not necessarily the correct one) and hence its membership to any specific extension can be exactly derived (as an example see the red dot in Figure 2(i)). In contrast, in uncertain and/or incomplete data, an entity's position only can be restricted on a subarea of the real-world (as an example see the red area in Figure 2(ii)). Consequently, for some extensions, the entity's membership cannot be exactly determined.

Definition 2 (Real-World): The real-world, denoted \mathfrak{W} , is the set of all existing real-world entities. The mapping $\omega : \mathcal{R} \rightarrow \mathfrak{W}$ maps each tuple of any relation \mathcal{R} on an entity of \mathfrak{W} .

Definition 3 (Relation's Extension): The extension of a relation \mathcal{R} , denoted $Ext(\mathcal{R})$, is the part of the real-world which is actually modeled by \mathcal{R} : $Ext(\mathcal{R}) = \bigcup_{t \in \mathcal{R}} \omega(t) \subseteq \mathfrak{W}$

The membership of a tuple to a specific relation generally depends on the relation's intended universe of discourse.

Definition 4 (Reference Extension): The reference extension of a relation \mathcal{R} , denoted $E_{\mathcal{R}}$, is the part of the real-world which has to be intendedly modeled by \mathcal{R} : $E_{\mathcal{R}} \subseteq \mathfrak{W}$

Note, data can be incorrect and/or uncertain. Thus, the actual set of real-world entities modeled by a relation's tuples is not necessarily a subset of the relation's *reference extension* ($Ext(\mathcal{R}) \not\subseteq E_{\mathcal{R}}$).

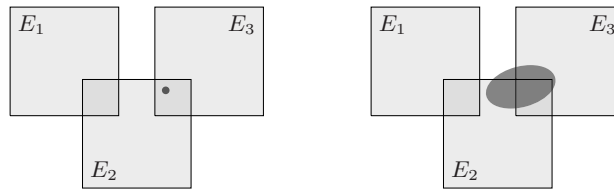


Fig. 2. Real-world position of an entity modeled in a certain and complete data source (left) and modeled in an uncertain and/or incomplete data source (right)

3 Problem Description

In the first data integration step duplicate representations of same real-world entities have been detected. The result of the duplicate detection step is a clustering of the tuples of all source relations (one cluster for each real-world entity).

Definition 5 (Clustering): Let \mathcal{R} be a relation. A clustering \mathcal{C} of \mathcal{R} is a partition $\{C_1, C_2, \dots, C_k\}$ of \mathcal{R} (meaning that the C_i are disjoint and their union equals \mathcal{R}), where each C_i is called a cluster, such that for each cluster all its tuples refer to the same real-world entity: $(\forall C_i \in \mathcal{C}) : (\forall t_1, t_2 \in C_i) : \omega(t_1) = \omega(t_2)$.

The goal of tuple merging is to combine all tuples of one cluster into a single one. As we will discuss in Section 4.2, tuple merging essentially depends on the integration context. Let $C = \{t_1, t_2, \dots, t_k\}$ be the considered cluster, where each probabilistic tuple t_i belongs to a source relation \mathcal{R}_i (the source relations of different tuples can be equal). For simplification, we consider the schema of all source relations to be identical. Let $\mathcal{S} = \{A_1 : D_1, \dots, A_n : D_n\}$, $D = D_1 \times \dots \times D_n$ be the common instance schema and E' the *reference extension* of the integration result. Thus, merging the probabilistic tuples of C can be formalized as (Please note: $\mathcal{P}(S)$ is the power set of the set S):

$$t_C = \mu(C, E'), \quad \mu : \mathcal{P}(\mathcal{P}(D \times (0, 1]) \times (0, 1]) \times \mathcal{E} \rightarrow \mathcal{P}(D \times (0, 1]) \times (0, 1]) \quad (1)$$

For reasons of clarity and comprehensibility, in the following examples, the index of a merged tuple is an ordered concatenation of the indexes of the tuples it is merged from. For example, $\mu(\{t_1, t_2, t_3\}, E')$ is denoted by t_{123} . Moreover, given the tuple $t_C = \mu(C, E')$, the tuples $\{t \in C\}$ are denoted as the base-tuples of t_C .

3.1 Requirements for an Ideal Merging

The tuple resulting from merging multiple base-tuples should properly combine the information of all these tuples. For that reason, we define a set of requirements for an ideal tuple merging. We denote a merging function μ to be ideal, if the following four conditions hold:

- (1) The merging result is independent from the representation of the source data and hence independent of the used probabilistic data model.
- (2) The function μ is associative. Thus, the tuple resulting from merging the base-tuples t_1 , t_2 and t_3 w.r.t. the considered *reference extension* E' is independent of the merging order:

$$\mu(\{\mu(\{t_1, t_2\}, E'), t_3\}, E') = \mu(\{\mu(\{t_1, t_3\}, E'), t_2\}, E') = \mu(\{t_1, t_2, t_3\}, E')$$

This requirement is important if data integration is considered in a pairwise fashion.

- (3) The function is idempotent ($\mu(\{t \in \mathcal{R}\}, E') = t$), if it is considered within the tuple's original context ($E' = E_{\mathcal{R}}$). This requirement ensures that the result from deduplicating a duplicate free relation is the relation itself.

- (4) The information of all base-tuples is sufficiently captured in the merged tuple. In this case, sufficiently means that no information is lost and no information is incorrectly introduced by the merging function.

The last requirement is based on intuitive perceptions and hence its formulation is rather vague. A goal of future work is to formalize this requirement (maybe by a quantification of information loss [10]). In order to satisfy the first requirement, we generally consider tuple merging within the *possible world semantics*. The other requirements are discussed during the following sections.

4 Merging of Probabilistic Tuples

In relational data, each attribute value represents a property of the real-world entity modeled by the tuple this attribute value belongs to. For simplification, we consider the properties of a real-world entity to be independent from the membership of this entity to a specific extension. Thus the instance data of a tuple is considered to be independent from its membership to a specific relation (this is actually not always the case, see discussion in Section 7). As a consequence, we consider the merging of possible instance data and the merging of tuple memberships each as a separate process and divide the probabilistic tuple merging into two independent steps:

- (1) Merging of instance data (see Section 4.1).
- (2) Merging of tuple memberships depending on the given source context and the considered target context (see Section 4.2).

For that purpose, we decompose the merging function μ into a function for merging instance data (μ_{ID}) and a function for merging tuple memberships (μ_{TM}). The function $\mu(C, E') = (\mu_{\text{ID}}(C), \mu_{\text{TM}}(C, E'))$ is ideal, if μ_{ID} and μ_{TM} are ideal.

$$\begin{aligned} t_C = \mu(C, E') &= \mu\left(\bigcup_{t_i \in C} \{(\mathcal{I}_i, P_i), p(t_i)_{\mathcal{R}_i}\}, E'\right) = ((\mathcal{I}_C, P_C), p(t_C)_{\mathcal{R}_C}) \\ &= (\mu_{\text{ID}}(\bigcup_{t_i \in C} \{(\mathcal{I}_i, P_i)\}), \mu_{\text{TM}}(\bigcup_{t_i \in C} \{p(t_i)_{\mathcal{R}_i}\}, E')) \end{aligned}$$

As a running example throughout this paper, we consider the two probabilistic tuples $t_1 = ((\mathcal{I}_1, P_1), p(t_1)_{\text{student}})$ and $t_2 = ((\mathcal{I}_2, P_2), p(t_2)_{\text{author}})$ as presented in Figure 3. Both tuples are maybe-tuples. Moreover, each tuple has three possible instances. Two of these instances are the same in both tuples.

	name	surname	location	$P_1(I)$	$p(t_1)$		name	surname	location	$P_2(I)$	$p(t_2)$	
I_1	John	Do	New York	0.3	0.8	\longleftrightarrow	I_2	John	Doe	Albany	0.4	
I_2	John	Doe	Albany	0.25			I_3	Johan	Doe	New York	0.35	0.5
I_3	Johan	Doe	New York	0.45			I_4	Jon	Ho	York	0.25	

$t_1 = ((\mathcal{I}_1, P_1), p(t_1)_{\text{student}} = 0.8)$

$t_2 = ((\mathcal{I}_2, P_2), p(t_2)_{\text{author}} = 0.5)$

Fig. 3. Probabilistic tuples $t_1 \in \text{student}$, $t_2 \in \text{author}$

4.1 Merging of Instance Data

In certain data, instance merging is considered on an attribute by attribute basis. Since in probabilistic data dependencies between attribute values can exist, we consider merging techniques always for whole instances. In contrast to a merging of certain tuples [7], conflict resolution by choosing one of the conflicting items (deciding strategy) or by creating a new representative (mediating strategy) is generally not required. Instead, each kind of uncertainty can be stored in the resulting data by taking multiple possible instances into account.

An ideal instance merging results from the union of all possible instances of all base-tuples. Since the merging is not associative, if a simple average of the individual probabilities is calculated, we assign a weight q_i to each tuple t_i and define that the weight of a merged tuple results from the sum of the weights of its base-tuples ($q_{ij} = q_i + q_j$). If tuple merging is considered within the context of data integration, the reliabilities of the corresponding sources can be used as tuple weights. In conclusion, an ideal function for merging instance data can be formalized as:

$$\mu_{\text{ID}}(\bigcup_{i \in [1, k]} \{(I_i, P_i)\}) = (\bigcup_{i \in [1, k]} I_i, \frac{\sum_{i \in [1, k]} q_i P_i}{\sum_{i \in [1, k]} q_i}) \quad (2)$$

It is obvious, that μ_{ID} is idempotent and associative. Moreover, the function takes each instance into account which is possible for at least one base-tuple and does not add an instance which is impossible for all base-tuples. Thus, intuitively this function also satisfies the requirement of sufficiently capturing the information of the given set of base-tuples.

Running Example: The instance data of the tuple $t_{12} = \mu(\{t_1, t_2\}, E')$ resulting from merging the two base-tuples t_1 and t_2 of Figure 3 by using the tuple weights $q_1 = 0.6$ and $q_2 = 0.4$ is presented in Figure 4.

	name	surname	location	$P_{12}(I)$
I_1	John	Do	New York	0.18
I_2	John	Doe	Albany	0.31
I_3	Johan	Doe	New York	0.41
I_4	Jon	Ho	York	0.1

$$(I_{12}, P_{12}) = \mu_{\text{ID}}(\{(I_1, P_1), (I_2, P_2)\})$$

Fig. 4. Instance data (I_{12}, P_{12}) resulting from merging (I_1, P_1) and (I_2, P_2)

User-defined Aggregation Functions. To enable the usage of additional domain knowledge, for each attribute a specific aggregation function can be defined by the user (resp. a domain expert). This is an important property, because in some scenarios a simple union of all possible instance values of an attribute is not adequate (e.g. in merging sales data) or other information for reducing the set of possible instances is available (e.g. the domain expert knows that the address field of a specific tuple is the correct one).

As an example, we consider a relation *inventory* with the three attributes *name*, *producer* and *stock* (see Figure 5). The two tuples $t_3 = ((\mathcal{I}_3, P_3), p_3)$ and $t_4 = ((\mathcal{I}_4, P_4), p_4)$ represent the same product, but the stock information of each tuple belongs to different orders. Therefore, in this scenario neither 15, 20 nor 6 items of this product but rather 21 or 26 items are available. As a consequence, the true stock value of this product results from the sum of stocks of both base-tuples instead of being the stock of one of them.

In general, the values of each possible combination of instances belonging to different base-tuples have to be aggregated. If for all attributes an aggregation function is defined, for each of these combinations a single possible instance of the merged tuple results by aggregating the values for each attribute. Otherwise, for each combination two instances result (one for each of the combined instances).

In our example, two aggregation functions are defined. As mentioned above, the true stock value is specified by the *sum*-function (mediating strategy). Moreover, the user knows that the *producer* value of the second instance data (\mathcal{I}_4) is correct. For that reason, always the *producer* of this tuple is chosen (deciding strategy). For the attribute *name*, no function is specified. Thus, all possible values are taken into account (see Figure 5). Note, instance merging is not ideal, if at least one non-associative aggregation function (e.g. average) is used.

name	producer	stock	P(I)
Twix	Maas Inc.	15	0.8
Twux	Nestle	20	0.2

(\mathcal{I}_3, P_3)

name	producer	stock	P(I)
Raider	Mars Inc.	6	1.0

(\mathcal{I}_4, P_4)

$\mu_{in}(\{(\mathcal{I}_3, P_3), (\mathcal{I}_4, P_4)\}) \Rightarrow$

name	producer	stock	P(I)
Twix	Mars Inc.	21	0.4
Raider	Mars Inc.	21	0.4
Twux	Mars Inc.	26	0.1
Raider	Mars Inc.	26	0.1

$(\mathcal{I}_{34}, P_{34})$

Fig. 5. Example for instance merging with user-defined aggregation functions

4.2 Merging of Tuple Memberships

The degree of membership of a merged tuple to the result relation depends on two factors: (i) the overlap scenario of the source relations' *reference extensions* (source context) and (ii) the intended scope of the result relation (target context).

Source Context: The *reference extensions* $E_{\mathcal{R}_1}$ and $E_{\mathcal{R}_2}$ of two relations \mathcal{R}_1 and \mathcal{R}_2 can be in four different overlap situations (see Figure 6(i)-(iv)).

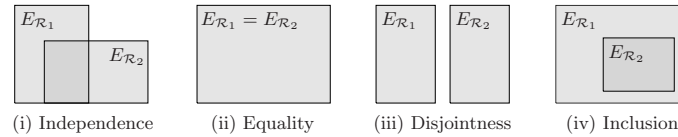


Fig. 6. The four different overlap situations of two *reference extensions*

Both *reference extensions* can be independent from each other (partially overlapping), equal, disjoint or one extension can be a subset of the other one (quantified overlap is supposed to be considered in future work). A set of pairwise overlap situations is called an overlap scenario.

To detect the given overlap scenario of different *reference extensions*, additional meta information on the individual sources and general information on real-world relationships is required. In our research, based on semantic concepts as ontologies and thesauri, we aim to identify overlap scenarios only by using the source relations' names (see example shown in Figure 7).

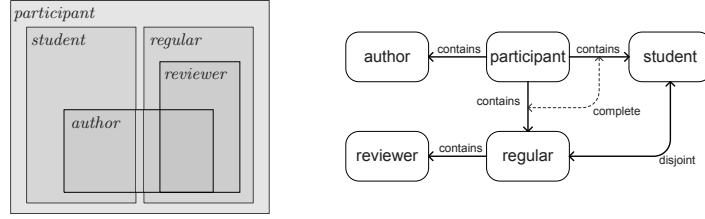


Fig. 7. Overlap Scenario of *reference extensions* (left), corresponding ontology (right)

Target Context: The membership of a merged tuple to the result relation generally depends on the integration's intended universe of discourse. In the following, the *reference extension* of the integration's result relation is denoted as *target reference extension*.

Let \mathcal{R}_3 be the relation which results from integrating the relations \mathcal{R}_1 and \mathcal{R}_2 . In general, each part of the real world ($E_{\mathcal{R}_3} \subseteq \mathfrak{W}$) can be considered as *target reference extension* (e.g., $E_{\mathcal{R}_3} \supset E_{\mathcal{R}_1} \cup E_{\mathcal{R}_2}$). However, the most cases are very unusual and there is not enough information available for predicting an adequate membership degree of the resulting tuples. Moreover, for the most integration processes an intuitive *target reference extension* can be implied by the used merge operators. For example, in [20] the four merge operators *merge join* \sqcap , *left outer merge join* \sqsubset , *right outer merge join* \sqsupset and *full outer merge join* \sqcup are introduced. An intuitive *target reference extension* $E_{\mathcal{R}_3}$ for each of these operators can be defined as:

$$\begin{array}{lcl} \mathcal{R}_3 & = & \left| \begin{array}{c} \mathcal{R}_1 \sqcap \mathcal{R}_2 \\ \mathcal{R}_1 \sqcup \mathcal{R}_2 \\ \mathcal{R}_1 \sqsubset \mathcal{R}_2 \\ \mathcal{R}_1 \sqsupset \mathcal{R}_2 \end{array} \right| \\ E_{\mathcal{R}_3} & = & \left| \begin{array}{c} E_{\mathcal{R}_1} \cap E_{\mathcal{R}_2} \\ E_{\mathcal{R}_1} \cup E_{\mathcal{R}_2} \\ E_{\mathcal{R}_1} \\ E_{\mathcal{R}_2} \end{array} \right| \end{array}$$

Membership Merging in Consistent and Complete Data: Given the definitions above, the membership merging function μ_{TM} can be formalized as:

$$\mu_{\text{TM}} : \mathcal{P}((0, 1]) \times \mathcal{P}(\mathfrak{W}) \rightarrow (0, 1], \quad p(t_C)_{\mathcal{R}_C} = \mu_{\text{TM}}\left(\bigcup_{t_i \in C} \{p(t_i)_{\mathcal{R}_i}\}, E'\right) \quad (3)$$

where $\mathcal{P}(\mathfrak{W})$ is the set of all possible parts of the real-world and E' is the considered *target reference extension*. As mentioned above, the quality of membership

merging can be enhanced if the overlap scenario of the source relations' *reference extensions* is given. Otherwise, situations of independence have to be assumed.

Target reference extension:	Membership merging function: $p(t_{12})_{\mathcal{R}_3} = \mu_{\text{TM}}(\{p(t_1)_{\mathcal{R}_1}, p(t_2)_{\mathcal{R}_2}\}, E_{\mathcal{R}_3})$
$E_{\mathcal{R}_3} = E_{\mathcal{R}_1} \cup E_{\mathcal{R}_2}$	$P(e \in E_{\mathcal{R}_3}) = P(e \in E_{\mathcal{R}_1}) + P(e \in E_{\mathcal{R}_2}) - P(e \in E_{\mathcal{R}_1} \wedge e \in E_{\mathcal{R}_2})$ $\Rightarrow p(t_{12})_{\mathcal{R}_3} = p(t_1)_{\mathcal{R}_1} + p(t_2)_{\mathcal{R}_2} - p(t_1)_{\mathcal{R}_1} \cdot p(t_2)_{\mathcal{R}_2}$
$E_{\mathcal{R}_3} = E_{\mathcal{R}_1} \cap E_{\mathcal{R}_2}$	$P(e \in E_{\mathcal{R}_3}) = P(e \in E_{\mathcal{R}_1} \wedge e \in E_{\mathcal{R}_2})$ $\Rightarrow p(t_{12})_{\mathcal{R}_3} = p(t_1)_{\mathcal{R}_1} \cdot p(t_2)_{\mathcal{R}_2}$
$E_{\mathcal{R}_3} = E_{\mathcal{R}_1}$	$P(e \in E_{\mathcal{R}_3}) = P(e \in E_{\mathcal{R}_1})$ $\Rightarrow p(t_{12})_{\mathcal{R}_3} = p(t_1)_{\mathcal{R}_1}$
$E_{\mathcal{R}_3} = E_{\mathcal{R}_2}$	$P(e \in E_{\mathcal{R}_3}) = P(e \in E_{\mathcal{R}_2})$ $\Rightarrow p(t_{12})_{\mathcal{R}_3} = p(t_2)_{\mathcal{R}_2}$

Fig. 8. Membership merging in case of two independent *reference extensions*

As a demonstrating example, we consider all four merge operators ($\sqcup, \sqcap, \sqsubset, \sqsupset$) w.r.t. the two independent *reference extensions* $E_{\mathcal{R}_1}$ and $E_{\mathcal{R}_2}$ (see Figure 8). Both base-tuples t_1 and t_2 represent the same real-world entity $e = \omega(t_1) = \omega(t_2)$. If for example, the full outer join merge is used ($E_{\mathcal{R}_3} = E_{\mathcal{R}_1} \cup E_{\mathcal{R}_2}$), the probability that e belongs to the *target reference extension* is equal to the probability that one of the two base-tuples t_1 and t_2 belongs to their corresponding relation ($P(e \in E_{\mathcal{R}_3}) = P(e \in E_{\mathcal{R}_1}) + P(e \in E_{\mathcal{R}_2}) - P(e \in E_{\mathcal{R}_1} \wedge e \in E_{\mathcal{R}_2})$).

In probability theory, the situation of independence can be specialized to the situations of equality, inclusion or disjointness by introducing some additional dependencies:

(Equality) $E_{\mathcal{R}_1} = E_{\mathcal{R}_2}$	Dep. 1.: $\omega(t_1) \in E_{\mathcal{R}_1} \Leftrightarrow \omega(t_2) \in E_{\mathcal{R}_2}$ $\Rightarrow p(t_1)_{\mathcal{R}_1} = p(t_2)_{\mathcal{R}_2}$
(Disjointness) $E_{\mathcal{R}_1} \cap E_{\mathcal{R}_2} = \emptyset$	Dep. 1.: $\omega(t_1) \in E_{\mathcal{R}_1} \Rightarrow \omega(t_2) \notin E_{\mathcal{R}_2}$ Dep. 2.: $\omega(t_2) \in E_{\mathcal{R}_2} \Rightarrow \omega(t_1) \notin E_{\mathcal{R}_1}$ $\Rightarrow p(t_1)_{\mathcal{R}_1} + p(t_2)_{\mathcal{R}_2} \leq 1$
(Inclusion) $E_{\mathcal{R}_1} \supset E_{\mathcal{R}_2}$	Dep. 1.: $\omega(t_2) \in E_{\mathcal{R}_2} \Rightarrow \omega(t_1) \in E_{\mathcal{R}_1}$ $\Rightarrow p(t_1)_{\mathcal{R}_1} \geq p(t_2)_{\mathcal{R}_2}$

If for the given membership degrees some of these dependencies are not valid, elementary principles of probability theory can be violated. As for example:

$$(\exists e \in \mathfrak{W}) : (\exists E_{\mathcal{R}} \subseteq \mathfrak{W}) : P(e \in E_{\mathcal{R}}) + P(e \notin E_{\mathcal{R}}) \neq 1$$

In this case, we call the given membership degrees to be inconsistent to each other. In a single probabilistic database we can assume that all membership degrees are defined in a consistent way. In data integration, however, we operate with degrees specified by different independent sources. Thus such an assumption is not reasonable.

As a consequence, deriving membership merging functions for the situations of equality, inclusion or disjointness from the functions defined for the situation of independence (see Figure 8) is not suitable.

Moreover, for queries defined on a single database, the *closed world assumption* (CWA) [22] applies. Thus, each relation is assumed to be complete and each real-world entity which is not represented by a relation's tuple is assumed to be definitely not belonging to the relation's *reference extension* ($p(t)_{\mathcal{R}} = 0 \Rightarrow \omega(t) \notin E_{\mathcal{R}}$). Nevertheless, one main purpose of integrating data is to join multiple incomplete sources to a complete result. Thus, for source relations completeness cannot be assumed and the hypothesis that each missing tuple definitely does not belong to the considered relation cannot be made. For dealing with missing membership degrees, we make a generalization of the CWA and assume that the membership of missing tuples is completely unknown.

In conclusion, the problem of membership merging is to determine an adequate probability that the considered entity belongs to the result relation despite of inconsistent and missing membership degrees given by the individual sources.

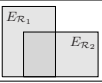
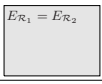
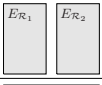
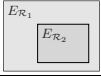
Overlap situation:		Membership merging function:
		$p(t_{12})_{\mathcal{R}_1 \sqcup \mathcal{R}_2} = \mu_{\text{TM}}(\{p(t_1)_{\mathcal{R}_1}, p(t_2)_{\mathcal{R}_2}\}, E_{\mathcal{R}_1} \cup E_{\mathcal{R}_2})$
Independence		$p(t_{12})_{\mathcal{R}_1 \sqcup \mathcal{R}_2} = p(t_1)_{\mathcal{R}_1} + p(t_2)_{\mathcal{R}_2} - p(t_1)_{\mathcal{R}_1} \cdot p(t_2)_{\mathcal{R}_2}$
Equality		$p(t_{12})_{\mathcal{R}_1 \sqcup \mathcal{R}_2} = (q_1 \cdot p(t_1)_{\mathcal{R}_1} + q_2 \cdot p(t_2)_{\mathcal{R}_2}) / (q_1 + q_2)$
Disjointness		$p(t_{12})_{\mathcal{R}_1 \sqcup \mathcal{R}_2} = \min(1, p(t_1)_{\mathcal{R}_1} + p(t_2)_{\mathcal{R}_2})$
Inclusion		$p(t_{12})_{\mathcal{R}_1 \sqcup \mathcal{R}_2} = \begin{cases} p(t_1)_{\mathcal{R}_1} & , \text{ if } p(t_1)_{\mathcal{R}_1} \geq 0 \\ (q_2 \cdot p(t_2)_{\mathcal{R}_2}) / (q_1 + q_2) & , \text{ else} \end{cases}$

Fig. 9. Four different overlap situations together with possible membership merging functions defined for the case of inconsistent membership degrees

Inconsistent Membership Degrees: In Figure 9 we present four possible functions defined for merging tuple memberships in a *full outer merge join* adapted to inconsistent membership degrees. Even though we defined these functions to be binary, a definition for more than two base-tuples is also possible. In the situations of equality and inclusion, we use the two weights q_1 and q_2 already defined in Section 4.1.

Note, the merging functions defined for each situation are individually associative. From a global point of view (e.g. an overlap scenario of multiple source extensions as shown in Figure 7(i)), however, the merging is not associative and hence not ideal. Defining a globally associative merging of inconsistent membership degrees seems very hard and is an interesting challenge of future research

(see Section 7). Nevertheless, in many integration scenarios the reference extensions of all source relations are equal (e.g. in unifying data resulting from different observations of same real-world phenomena), disjoint (e.g. in an integration of multiple local databases into a global one), almost independent (e.g. in integrating data resulting from observations of several independent real-world phenomena) or pairwise include each other (e.g. in integrating data for enhancing the quality of a specific database's part). Since each of the merging functions defined in Figure 9 is individually associative, for such scenarios an ideal membership merging can be guaranteed.

$E_{\mathcal{R}_3} = E_{\mathcal{R}_1} \cup E_{\mathcal{R}_2}$	$P(e \in E_{\mathcal{R}_3}) \geq P(e \in E_{\mathcal{R}_1}) \Rightarrow p(t_C)_{\mathcal{R}_3} \in [p(t_1)_{\mathcal{R}_1}, 1]$
$E_{\mathcal{R}_3} = E_{\mathcal{R}_1} \cap E_{\mathcal{R}_2}$	$P(e \in E_{\mathcal{R}_3}) \leq P(e \in E_{\mathcal{R}_1}) \Rightarrow p(t_C)_{\mathcal{R}_3} \in [0, p(t_1)_{\mathcal{R}_1}]$
$E_{\mathcal{R}_3} = E_{\mathcal{R}_2}$	$P(e \in E_{\mathcal{R}_3}) = P(e \in E_{\mathcal{R}_2}) \Rightarrow p(t_C)_{\mathcal{R}_3} \in [0, 1]$
$E_{\mathcal{R}_3} = E_{\mathcal{R}_1}$	$P(e \in E_{\mathcal{R}_3}) = P(e \in E_{\mathcal{R}_1}) \Rightarrow p(t_C)_{\mathcal{R}_3} = p(t_1)_{\mathcal{R}_1}$

Fig. 10. Membership merging w.r.t. two independent source relations \mathcal{R}_1 and \mathcal{R}_2 in case of a missing membership degree of e to $E_{\mathcal{R}_2}$ ($P(e \in E_{\mathcal{R}_2}) \in [0, 1]$)

Missing Membership Degrees: Independent from the duplicate detection result, the membership degree of each tuple has to be recalculated, if the *target reference extension* deviates from the *reference extension* of its source relation. For that reason, in an integration process working on disparate *reference extensions*, membership merging has to be applied on each duplicate cluster, whether this cluster contains multiple tuples or not.

If a cluster does not contain a tuple for each of the different source relations' *reference extensions*, the membership degree of the considered real-world entity to this extension is missing. Thus, in some contexts, the membership of the merged tuple to the *target reference extension* cannot be exactly determined as described above and instead of an exact value only a range of possible membership can be specified. Since, in this paper, we restrict ourselves to binary merging functions, we exemplarily consider the case of an integration of two relations \mathcal{R}_1 and \mathcal{R}_2 and a cluster C that only contains a single tuple $C = \{t_1 \in \mathcal{R}_1\}$. Depending on the considered *target reference extension* ($E_{\mathcal{R}_3}$), the resulting membership degree $p(t_C)_{\mathcal{R}_3}$ is more or less uncertain. As an example, we consider the situation of independence which is shown in Figure 10. In three of four target contexts only a range of possible membership results. In the last case, however, the *target reference extension* is equal to $E_{\mathcal{R}_1}$. Thus, in this case, none of the required membership degrees is missing and the resulting degree of membership can be exactly determined. Note, if in the probabilistic target model, probability ranges cannot be stored, using the expected probability seems most suitable.

Ideality: Membership merging without inconsistent and missing membership degrees is based on probability theory and hence seems to be sufficient. In contrast, a sufficient capturing of the information represented by the individual memberships cannot be intuitively defined if inconsistent or missing membership

degrees exist. Nevertheless, in case of inconsistent membership degrees, a preserving of the information represented by all the base-tuples' memberships is not reasonable, because the information on memberships is definitely not accurate.

Running Example: With respect to our running example, the reference extensions of both source relations are independent (see Figure 7). Thus, by using the *join merge*, the degree of membership of the merged tuple to the result relation representing all student authors results in:

$$p(t_{12})_{student\ authors} = p(t_1)_{student} \cdot p(t_2)_{authors} = 0.4$$

5 Related Work

Tuple merging in certain data is considered in different works [11, 7, 19, 6]. Since in certain data only single values can be stored, conflicts can only be resolved by choosing one of the conflicting values (e.g. by using *max*) or by creating a new representative (e.g. by using *avg*). With respect to the most conflict resolution functions tuple merging is not associative and hence not ideal. Moreover, our approach is more general, which can be specialized to conflict resolution as defined for certain data if for each attribute an aggregation function is specified.

Robertson et al [23] consider tuple merging within a transposition of certain data. Merging of two tuples with contrary instance data is not provided (in such cases both tuple are denoted to be *non mergeable*).

DeMichiel [12] and Tseng [25] use *partial values* (resp. *probabilistic values*) to resolve conflicts between certain values by taking multiple possible instances into account. Consequently, these approaches already produce uncertain data as result data. This is similar to our ideal instance merging if each base-tuple is considered to be certain and no aggregation functions are used. Nevertheless, both approaches consider conflict resolution on an attribute by attribute basis. Dependencies between possible attribute values are not considered.

Andritsos et al [3] define queries on multiple conflicting duplicates. Thus instead of merging the tuples of each cluster into a single one, query results are derived from sets of mutual exclusive base-tuples. Since to each cluster's tuple a probability can be assigned, this approach is mostly identical to our ideal instance merging without the additional offering of attribute specific aggregation functions.

None of the studies, however, allows uncertain (esp. probabilistic) data as source data. Membership merging is consequently not handled in these works.

A merging of tuples representing uncertain information (on instance as well as membership level) is proposed by Lim et al [18]. Nevertheless, instead of probability theory this approach is based on the *Dempster-Shafer theory of evidence*. For membership merging in a union of two relations the authors do not take different *target reference extensions* or different overlap scenarios of source *reference extensions* into account. Moreover, the authors explicitly specify a *membership derivation function* only for the relational selection operator.

In the publication of Agrawal et al [1], deduplication is not considered.

6 Conclusion

Many applications naturally produce uncertain data. For that reason, probabilistic databases have become a topic of interest in the database community in recent years. In order to combine the data from different probabilistic data sources, an integration process has to be applied. To obtain concise integration results, merging of duplicate tuples is an essential activity. We consider duplicate detection in probabilistic data in [21]. In this paper, we have investigated how a set of probabilistic tuples designated as duplicates can be merged to a single one. We have considered probabilistic tuples representing uncertainty on instance level and uncertainty on membership level. We have defined a set of requirements for an ideal tuple merging. Moreover, we have divided probabilistic tuple merging into a merging of instance data as well as a merging of membership degrees. Without additional domain knowledge, instance merging is realized by the union of the tuples' possible instances. Otherwise user-defined aggregation functions can be used. For defining an adequate membership merging, we take the overlap scenario of the real-world scopes modeled by all source relations as well as the intended scope of the integration result into account. Whereas the instance merging is always ideal if solely associative aggregation functions are used, an ideal membership merging only results if either the underlying membership degrees are consistent to each other or a scenario with only a single kind of overlap situation (independence, equality, disjointness, or inclusion) exists.

In conclusion, this paper gives first ideas in the large area of merging duplicate tuples in probabilistic databases. Nevertheless, open problems still exist. Thus, we discuss some future challenges in the following section.

7 Open Problems and Future Challenges

As already mentioned in Section 4.2, the presented membership merging functions are only associative, if a scenario with same situations of overlap is given. Otherwise associativity only can be guaranteed if the underlying membership degrees are consistent to each other.

Challenge 1 *Definition of an associative merging of inconsistent membership degrees beyond scenarios with same situations of overlap.*

Usually relations to be integrated have heterogeneous schemas and their sets of attributes only partially overlap. In this case, merging of instance data could be considered as a kind of full outer union in relational data where missing values are filled up with uniform distributions on corresponding attribute domains.

Challenge 2 *Techniques for merging probabilistic tuples defined on heterogeneous schemas.*

In contrast to the assumption made in Section 4, membership merging and instance merging is not always independent from each other. The instance of an entity and its membership to a specific extension depends on each other, if (a) only entities of the considered extension have a specific property (e.g. only

students have a study path or a student number) or if (b) the membership to the considered extension restricts the value of at least one entity's property on some special domain elements (e.g. each mathematics student has the study path 'mathematics' or each driver is older than 18 years). Given a relation \mathcal{R} , an attribute A defined in the domain D and the symbol \perp denoting the situation of nonexistence, corresponding functional dependencies can be specified as:

$$\text{existence dependency: } e \notin \text{Ext}(\mathcal{R}) \rightarrow (\forall t \in \omega^{-1}(e)) : t.[A] = \perp \quad (4)$$

$$\text{value dependency: } e \in \text{Ext}(\mathcal{R}) \rightarrow (\forall t \in \omega^{-1}(e)) : t.[A] \in X \subseteq D \quad (5)$$

Challenge 3 *Adaptation of the presented tuple merging to existence dependencies and value dependencies between instance data and membership degrees.*

We presented an ideal merging of instance data in Section 4.1. The requirements for ideality guarantee that the resulting instance data is as correct as possible. Nevertheless, by using an ideal merging function the instance data on a single real-world entity becomes more and more uncertain the more tuples are merged together. This, however, is most often not the actual purpose of data integration. Thus, in many applications using an ideal merging function is often not valuable and other merging strategies are required. Finding a merging strategy best fitting for a special application is generally a trade off between correctness and certainty. Most correct and also most uncertain data results, if all possible instances of all base-tuples are taken into account. In contrast, the result is most certain but most likely also incorrect, if only one of the possible instances is chosen. In many applications an adequate merging function has to be a compromise between these two extremes.

Challenge 4 *Definition of some non-ideal functions making a suitable trade-off between certainty and correctness possible.*

We do not address a merging of data lineage in this paper. In many probabilistic data models, e.g., ULDB, however, data lineage is an important concept which can be used for validating the consistence of given probabilities.

Challenge 5 *Techniques for merging the base-tuples' lineage in a way that the merged membership degree can be consistently derived from the merged lineage.*

A non-consideration of dependencies between individual data sources can impair the quality of the merged tuple. Usually, data sources are not independent from each other. In contrast, often the data of one source is copied from another source. Thus false instances can be spread through copying and are considered in tuple merging with high certainty. Techniques for detecting dependencies between individual sources are proposed in [14].

Challenge 6 *To figure out the role of source dependencies in merging probabilistic tuples.*

Finally, one of the most important challenges is to adapt the proposed tuple merging strategies to more succinct representation models on which probabilistic databases usually are based.

Challenge 7 *Adaptation of the presented merging functions to probabilistic data-models not storing each possible instance of a tuple separately.*

References

1. P. Agrawal et al. Foundations of Uncertain-Data Integration. In *VLDB 2010*. Stanford.
2. P. Agrawal et al. Trio: A System for Data, Uncertainty, and Lineage. In *VLDB*, pages 1151–1154, 2006.
3. P. Andritsos et al. Clean Answers over Dirty Databases: A Probabilistic Approach. In *ICDE*, page 30, 2006.
4. D. Barbará et al. The Management of Probabilistic Data. *IEEE Trans. Knowl. Data Eng.*, 4(5):487–502, 1992.
5. O. Benjelloun et al. ULDBs: Databases with Uncertainty and Lineage. In *VLDB*, pages 953–964, 2006.
6. O. Benjelloun et al. Swoosh: a generic approach to entity resolution. *VLDB J.*, 18(1):255–276, 2009.
7. J. Bleiholder et al. Data fusion. *ACM Comput. Surv.*, 41(1), 2008.
8. J. Boulos et al. MYSTIQ: a system for finding more answers by using probabilities. In *SIGMOD Conference*, pages 891–893, 2005.
9. R. Cavallo et al. The Theory of Probabilistic Databases. In *VLDB*, pages 71–81, 1987.
10. T. M. Cover et al. *Elements of Information Theory*. Wiley & Sons, 2006.
11. U. Dayal. Processing Queries Over Generalization Hierarchies in a Multidatabase System. In *VLDB*, pages 342–353, 1983.
12. L. G. DeMichiel. Resolving Database Incompatibility: An Approach to Performing Relational Operations over Mismatched Domains. *IEEE Trans. Knowl. Data Eng.*, 1(4):485–493, 1989.
13. X. Dong et al. Data Integration with Uncertainty. *VLDB J.*, 18(2):469–500, 2009.
14. X. Dong et al. Integrating Conflicting Data: The Role of Source Dependence. *PVLDB*, 2(1):550–561, 2009.
15. A. K. Elmagarmid et al. Duplicate Record Detection: A Survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.
16. N. Fuhr et al. A Probabilistic Relational Algebra for the Integration of Information Retrieval and Database Systems. *ACM Trans. Inf. Syst.*, 15(1):32–66, 1997.
17. J. Huang et al. MayBMS: a probabilistic database management system. In *SIGMOD Conference*, pages 1071–1074, 2009.
18. E.-P. Lim et al. An Evidential Reasoning Approach to Attribute Value Conflict Resolution in Database Integration. *IEEE Trans. Knowl. Data Eng.*, 8(5):707–723, 1996.
19. A. Motro et al. Fusionplex: Resolution of Data Inconsistencies in the Integration of Heterogeneous Information Sources. *Information Fusion*, 7(2):176–196, 2006.
20. F. Naumann et al. Completeness of Integrated Information Sources. *Inf. Syst.*, 29(7):583–615, 2004.
21. F. Panse et al. Duplicate Detection in Probabilistic Data. In *NTII*, pages 179–182, 2010.
22. R. Reiter. On Closed World Data Bases. In *Logic and Data Bases*, pages 55–76, 1977.
23. E. Robertson et al. Optimal Tuple Merge is NP-Complete. Technical report, 2004.
24. D. Suciú et al. Embracing Uncertainty in Large-Scale Computational Astrophysics. In *MUD*, pages 63–77, 2009.
25. F. S.-C. Tseng et al. Answering Heterogeneous Database Queries with Degrees of Uncertainty. *Distributed and Parallel Databases*, 1(3):281–302, 1993.
26. M. van Keulen et al. A Probabilistic XML Approach to Data Integration. In *ICDE*, pages 459–470, 2005.

Uncertain Databases in Collaborative Data Management ^{*}

Reinhard Pichler¹, Vadim Savenkov¹, Sebastian Skritek¹, Hong-Linh Truong²

¹ {pichler, savenkov, skritek}@dbai.tuwien.ac.at

² truong@infosys.tuwien.ac.at
Vienna University of Technology

Abstract. We discuss an approach to collaborative data management based on uncertain databases. Note that, in a collaborative data management system, users may have contradicting opinions about the correct values of data items. In our approach, we propose to store all conflicting data versions in parallel and to resolve conflicts based on user ratings. We show that such a collaborative data management system can be nicely represented in an uncertain database using U-relations.

1 Introduction

With the Web 2.0 paradigm invading more and more areas of life, from entertainment to enterprise workflows and even e-government (take the Gov 2.0 initiative of the US government as an example, see <http://www.gov2summit.com>), decentralized community-oriented architectures become increasingly important. With a vast amount of curated datasets available to-date, we are confronted with scenarios where the data are exported, updated, shared and used by people through numerous online services. Consequently, several *approaches to collaborative data management* have emerged over the past years to support such scenarios. In case of unstructured data, solutions based on the Wiki idea [14] have been very successful. For sharing scientific data, portals such as BIRN [5] and GEON [9] have been created. Recently, the Orchestra system [12] considered “collaborative update exchange” for structured data, where multiple data peers connected by data dependencies can publish and receive updates to their data. Conflicting updates are reconciled based on the inter-peer trust relationships, established a priori. Propagation of data updates forward and backward along such data dependencies was studied in the Youtopia project [13]. Considering trust relations similar to [12], [8] introduced an approach for conflict resolution whose result is independent of the order in which updates arrive at the system, allowing for globally consistent states.

While useful for scenarios with independent data peers, maintaining a single consistent database version [12, 13] is not always satisfactory for scenarios where some consistent global state of the data in the network is required [8]. Moreover, a more fine-grained notion of trust would be desirable. That is, it should be possible to distinguish between the trustworthiness of a data source or a user in

^{*} This work was supported by the Vienna Science and Technology Fund (WWTF), project ICT08-032. Vadim Savenkov is supported by the Erasmus Mundus External Co-operation Window Programme of the European Union

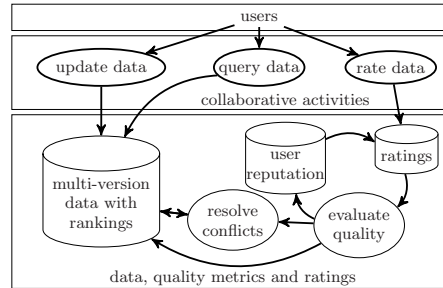


Fig. 1. A collaborative data management system

general and the acceptability of a concrete data item. Of course, in the long term, there must be a correlation between the two trust notions. Another approach for dealing with inconsistent data is a multi-versioned database concept, shared e.g. by uncertain databases [16] and the recent BeliefDB system [7], which allows different users to specify their own versions of each data tuple, called *beliefs*. There, also a belief-aware query language that allows e.g. to query for all users who share (or do not share) a particular belief is presented and analyzed from the complexity point of view.

In this paper, we present a new, *user-rating-based approach* to collaborative data management. Unlike Orchestra and Youtopia, we propose not to resolve such inconsistencies before storing the data, but to store all the different versions of the data in parallel, thus following the multi-versioned database model. We assume a scenario where all users work on the same data set, thereby updating and querying the data, but also rating its quality. In such a scenario, contradicting updates and inconsistencies (different beliefs of [7]) are unavoidable. However, unlike BeliefDB, we are not interested in tracking the beliefs of a particular user, but rather in combining the majority of common beliefs in a single consistent view on the data. The community feedback to various versions of each tuple (and, ultimately, various versions of the whole database) derived from beliefs of different users is being collected, in terms of ratings in the $[0, 1]$ scale. From this feedback the reputation of each user is derived, as a measure of alignment of her beliefs with the majority viewpoint. Figure 1 sketches our model of a collaborative data management system. It is similar to that of Wikipedia and P2P systems [10], but we focus on structured data, which have different query and update models.

Since various versions of data items have to be maintained in parallel, *uncertain databases* are a natural candidate for realizing such systems. Indeed, we show that *U-relations* (see [2]) with slight extensions are perfectly suited for this purpose. In U-relations, different versions of a data item are kept apart by assigning different values to some world set descriptor(s). A *world table* keeps track of all allowed value combinations for these descriptors. This allows one to store ratings for updates (in fact, for data versions created by updates) by annotating the various value assignments to world set descriptors. In addition

we also have to introduce a user table that stores the current reputation of each user and allows us to keep track of the initiator of each update.

Existing recommendation systems [1] show that user feedbacks are usually used for helping the user to select data suitable to the user's context. In this paper we assume that user ratings are made on the *data quality* of data tuples. The rationale behind this assumption is that data quality is critical in collaborative databases and currently there is a lack of techniques to evaluate data quality based on the user feedback and to combine such evaluated, community-based quality metrics into automatic, system-based quality metrics in traditional databases. We believe that if such a combination can be implemented in an integrated data management system, the quality of query answering can be improved. There exist a lot of data quality dimensions [4]. The approach suggested here is general enough to work with every quality dimension.

We believe that collaborative data management can become a perfect showcase for the emerging uncertain and probabilistic database technology, along with other application domains as scientific and sensoric data management [17] and information extraction [11] (see, e.g., [6] for a short survey, with a focus on probabilistic data processing).

Organization of the paper and summary of results.

- *System Model.* In Section 2, we describe the basic principles of our user-rating-based approach to collaborative data management. In particular, we describe how updates are incorporated into the database and how user ratings issued on an update can be aggregated to compute an overall rating of data items. We also explain how these ratings can be used to compute ratings for the entire database and to derive a reputation value for the user initiating an update.
- *Representation by U-relations.* We describe in Section 3 how uncertain databases can be used to implement a collaborative data management system. For this purpose, updates (or, equivalently, versions of data items) are annotated with reputation values. Care is taken that these reputation values can be computed incrementally and do not have to be recomputed every time a new rating arrives.
- *Extensions.* In Section 4, we discuss two important extensions of the basic model presented in Section 2, namely: We introduce the notion of “rigid updates” which allow the user to restrict the number of possible versions of tuples resulting from an update. Moreover, we describe how the deletion of tuples can be incorporated into our rating-based model. For both extensions, the required adaptations of the representation via U-relations are described. Directions for further extensions, which are left for future work, are discussed in Section 5.

2 Basic Collaboration Model

In this section, we describe the basic principles of our collaborative data management model. In the next section we will then show that this model can be very naturally implemented using uncertain databases. We note that there exists a huge body of rating and ranking systems in the literature. For the sake of presentation, we use a rather simple rating model, and show that it can be encoded in uncertain databases. We note that obviously also other methods could be used, without losing this nice property. The actual rating is not the main

contribution of this paper, but the observation that ratings in general fit nicely into the model of uncertain databases (this holds for several design decisions).

The fundamental idea of the approach is never to delete or overwrite any information once it has been added to the system, and to allow the insertion of conflicting and contradicting data. That is, rather than to reconcile updates and to maintain a single consistent version of the data, our database has multiple versions, each given by some non-conflicting combination of updates. A consequence of this approach is that the semantics of an update differs from the one usually assumed: In our case, *any update results in an insertion*. Throughout this paper, we assume every database relation to possess a key, and we define a *tuple* as an entry in the database identified by a key. Disagreement on the non-key values of a tuple leads to several *versions* of this tuple, which give rise (also in combination with the other tuples) to different *possible worlds*. That is, a version is a concrete value expression for some relation schema, while a tuple is a collection of versions for the same key. Every kind of update on a tuple is now mapped to the insertion of a new version.

Semantics Given the schema \mathcal{R} of some relation, we consider a partitioning of the attributes appearing in \mathcal{R} into sets of dependent attributes, which we call *blocks* (our notion of blocks should not be confused with the one in [6], where blocks refer to sets of tuples sharing the same key). We define the key attributes to always form a single block. In general, blocks are used whenever an object's property can be decomposed into several fields (like an address), but the values in these fields highly depend of each other. Therefore we allow as possible values for each block only those explicitly defined by updates, while different blocks, just as different tuples, are mutually independent, such that their values can be arbitrarily mixed. That is, for some *tuple* τ_i with the non-key attributes partitioned into blocks C_1, \dots, C_ℓ and a set of possible values $c_j = \{c_j^1, \dots, c_j^{k_j}\}$ for each C_j ($j \in \{1, \dots, \ell\}$), we define the set of possible *versions* of τ_i as $c_1 \times \dots \times c_\ell$. Furthermore, let $T = \{\tau_1, \dots, \tau_n\}$ be a set of tuples where for every τ_i ($i \in \{1, \dots, n\}$) there exist several possible versions $\tau_i^1, \dots, \tau_i^{k_i}$, i.e. τ_i itself is a set $\tau_i = \{\tau_i^1, \dots, \tau_i^{k_i}\}$. Then the set of possible worlds defined by T is $\tau_1 \times \dots \times \tau_n$. Note that values assigned to different blocks can be arbitrarily recombined, even if the resulting tuple was never inserted explicitly. Hence attributes of different blocks have to be completely independent of each other (This rather strong assumption is relaxed in Section 4.1). We are thus able to consider updates affecting a single block only, as splitting every update that changes more than one block into several “unary” updates has the same effect. This allows us to identify every possible value for a block with one update that inserted exactly this value. As every tuple is built up from a unique set of blocks, every tuple version can be identified by a uniquely defined set of updates.

Updates First recall that in our data model, we never overwrite or delete any information stored in the database. Instead, every update gives rise to one (or several) new possible world(s), unless this world is already present. Formally, we define updates as value-assignments on block level, where for a block B of attributes B_1, \dots, B_k , an update $u: (B_1, \dots, B_k) \leftarrow (v_1, \dots, v_k)[key]$ assigns to attribute B_i the value v_i for the tuple identified by *key*. One can distinguish

several types of updates: (1) Insertion of new tuples. (2) Deletion of tuples. (3) Update of non-key blocks. (4) Update of key blocks. Thereby, we defer the discussion of (2) to Section 4.2. An update affecting a key block corresponds either to the insertion of a new tuple, or, if the updated key is already present, to an update of the corresponding non-key blocks. The insertion of a tuple means just to insert a new tuple with exactly one version. For the update of a non-key block, assume an update on block C_j of tuple τ_i with versions $\tau_i^1, \dots, \tau_i^{k_i}$. Let V be the set of versions of τ_i restricted to the blocks $C_1, \dots, C_{j-1}, C_{j+1}, \dots, C_\ell$. Then the result of the update is a new version for every $v \in V$, where C_j is defined according to the update.

Example 1. Consider, similar to [7], a database for monitoring an animal population, where people report their observations. As different people may have different opinions about what they saw, there will be probably disagreement about the data to store. Let this database contain a table with schema $obs(T, A, B, S)$, divided into blocks $K = \{T\}$, $B_1 = \{A, B\}$, and $B_2 = \{S\}$. Thereby K is the key, consisting of the time of the observation (for the sake of simplicity, we assume that time would give a unique identifier), B_1 describes the color (A) of the animal and its probable kind (B), while B_2 stores its estimated size (S) (note that while kind and size of an animal are not independent, the observations of these properties are). Assume for some observation (say made by Alice) the database to contain the tuple τ_i with the single version (t_1, a_1, b_1, s_1) . Further assume that Bob made the same observation, but disagrees on the type (and color) of the animal seen. He issues an update $u_1: (A, B) \leftarrow (a_2, b_2)[t_1]$. This results in the two possible worlds $obs(t_1, \{(a_1, b_1)|(a_2, b_2)\}, s_1)$. If now John disagrees with the size of the seen animal, he performs an update $u_2: (S) \leftarrow (s_2)[t_1]$, resulting in the four possible versions $obs(t_1, \{(a_1, b_1)|(a_2, b_2)\}, \{s_1|s_2\})$ of τ_1 . \square

Ratings & User Reputation Next we describe how the quality of a tuple version is estimated. Users are allowed to rate either updates or tuple versions, where rating a tuple version is the same as giving this rating to all the (uniquely defined) updates that build up this version. The votes given on one update by different users are aggregated, where the influence of a rating is weighted according to the reputation of the user giving the vote. The reputation of a user is derived from the ratings given to the updates performed by the user, and as stated in the introduction, is interpreted as a score value normalized to $[0, 1]$. Every update is automatically rated with the reputation of the user who performed it. We fix the following notation. Let $U = \{u_1, \dots, u_n\}$ be a set of n updates, and let $R_i = \{r_1, \dots, r_{m_i}\}$ be a set of m_i ratings for every $u_i \in U$. With $user(u_i)$ and $user(r_j)$, we denote the user who performed the update u_i , or gave the rating r_j , respectively. Finally, let $rep(u_i)$ and $rep(r_j)$ be the reputation of $user(u_i)$ and $user(r_j)$, respectively.

We define the aggregated rating for u_i by

$$rating(u_i) = \frac{\sum_{\alpha=1}^{m_i} (r_\alpha \cdot rep(r_\alpha))}{\sum_{\alpha=1}^{m_i} (rep(r_\alpha))} \quad (1)$$

and the rating of a tuple version $\tau_i^j = (b_1, \dots, b_k)$ (where b_1, \dots, b_k are grouped to blocks c_1, \dots, c_ℓ) as $rating(\tau_i^j) = \sum_{\alpha=1}^\ell (w_\alpha \cdot rating(c_\alpha))$. Thereby $rating(c_\alpha) =$

$rating(u_i)$, with u_i being the update that sets the value for the block C_α in τ_i to c_α , and w_α is some weighting factor ($\sum_{\beta=1}^{\ell} w_\beta = 1$) that expresses the influence of each block. Denoting the number of attributes in a block C_α with $|C_\alpha|$, a reasonable value for w_α could be $(1/\sum_{\beta=1}^{\ell} |C_\beta|) \cdot |C_\alpha|$, which can be adopted accordingly if not all attributes are considered equally important. Obviously $rating(u_i)$ can be incrementally computed. For m_i votes, instead of storing them together with the user reputations, it suffices to store $\overline{rat}(u_i) = \sum_{\alpha=1}^{m_i} (r_\alpha \cdot rep(r_\alpha))$ and $\overline{rep}(u_i) = \sum_{\alpha=1}^{m_i} (rep(r_\alpha))$, from which the new rating after a new vote can be easily derived.

Concerning user reputation, we argue that it is advantageous not to consider all contributions of a user from the beginning, but only her more recent work. Thereby “recent” is defined in terms of a *time window* that defines which contributions to take into account. Leaving its concrete definition as a parameter to the system, beside its actual size it can be also defined either in terms of time (e.g. all the contributions from the last year), or in terms of contributions (e.g. the last 100 updates performed by this user). For the computation of a user’s reputation (say $user_\mu$), consider $U = \{u_1, \dots, u_n\}$ as the set of updates done by $user_\mu$ that fall into the time window. We define her reputation as

$$reputation(user_\mu) = \frac{\sum_{\alpha=1}^n (rating(u_\alpha) \cdot \overline{rep}(u_\alpha))}{\sum_{\alpha=1}^n \overline{rep}(u_\alpha)} = \frac{\sum_{\alpha=1}^n \overline{rat}(u_\alpha)}{\sum_{\alpha=1}^n \overline{rep}(u_\alpha)} \quad (2)$$

The reputation of a user can change because of two reasons. Either a new rating enters the time window, or some ratings (i.e. some updates) fall out of it. In both cases the reputation can be easily computed incrementally by storing $\overline{rat}' = \sum_{i=1}^n \overline{rat}(u_i)$ and $\overline{rep}' = \sum_{i=1}^n \overline{rep}(u_i)$. If a new rating arrives, the following steps are required for updating the user reputation: (1) Identify the user who performed the update just rated. (2) Check if the update is currently in the time window. (3) Update the user’s reputation and the values stored for the user. For performance reasons, the current time window for each user is explicitly stored as an index for the contained updates. From the description, it goes without saying that also several alternatives for aggregating the user reputation could be used, like the moving average to “fade out” older updates.

The initial reputation of a new user is 0 until she makes some contribution that is rated by other users. The system, however, can be easily adapted to support any other initial reputation. If a user is invited by some other user, she could get the reputation of the inviting user. In such cases, for aggregation, the initial reputation can be modeled as “rating” for some dummy update u_0 .

Example 2. Recall the scenario in Example 1, and consider the situation after u_1 . We split the initial update done by Alice into three smaller ones. One that inserted the key, one that set $B_1 = (A, B)$ to (a_1, b_1) and one update $B_2 = (S) \leftarrow (s_1)[t_1]$. We denote the updates with u_K , u_0 and u'_0 , respectively. Also, let $\tau^1 = (t_1, a_1, b_1, s_1)$, and $\tau^2 = (t_1, a_2, b_2, s_1)$. Assume $\overline{rat}(u_0) = 8.4$, $\overline{rep}(u_0) = 12$, $\overline{rat}(u'_0) = 2.4$, $\overline{rep}(u'_0) = 8$, $\overline{rat}(u_1) = 25.2$, $\overline{rep}(u_1) = 28$, hence, by (1), $rating(u_0) = \frac{8.4}{12} = 0.7$, $rating(u'_0) = 0.3$, and $rating(u_1) = 0.9$. We omit the influence of the key and assume a uniform influence of the non-key attributes

A, B, S , that is $w_K = 0$, $w_{B_1} = \frac{2}{3}$, and $w_{B_2} = \frac{1}{3}$. Then we get $rating(\tau^1) = \frac{2}{3} \cdot 0.7 + \frac{1}{3} \cdot 0.9 \approx 0.77$ and $rating(\tau^2) = \frac{2}{3} \cdot 0.3 + \frac{1}{3} \cdot 0.9 = 0.5$.

Now suppose that John performs update u_2 . Let $\overline{rat}'(John) = 7.5$ and $\overline{rep}'(John) = 15$, hence $reputation(John) = 0.5$. His update is automatically rated by 0.5, which results in $rating(u_2) = \frac{0.5 \cdot 0.5}{0.5} = 0.5$. (It is easy to check that his reputation remains unchanged.) Now let some user $user_3$ with a high reputation of 0.9 disagree with u_2 , which she expresses by giving a rating of 0.3 on u_2 . This results in $\overline{rat}(u_2) = 0.52$ and $\overline{rep}(u_2) = 1.4$, hence $rating(u_2) = \frac{0.25 + 0.27}{1.4} \approx 0.372$. Because of this rating, the reputation of John changes to $reputation(John) = \frac{7.5 + 0.25 + 0.27}{15 + 0.5 + 0.9} \approx 0.489$. Yet another user, with reputation 0.2 agrees with u_2 and rates it 0.7. Then $\overline{rat}(u_2) = 0.66$ and $\overline{rep}(u_2) = 2.1$, hence $rating(u_2) = \frac{0.66}{1.6} = 0.4125$. For the reputation of John, this has the effect of $\overline{rat}'(u_2) = 8.16$ and $\overline{rep}'(u_2) = 16.6$, hence $reputation(John) \approx 0.492$.

That is, the resulting rating for $\tau^3 = (t_1, a_1, b_1, w_2)$, and $\tau^4 = (t_1, a_2, b_2, w_2)$ is $rating(\tau^3) = \frac{2}{3} \cdot 0.7 + \frac{1}{3} \cdot 0.4125 \approx 0.60417$ and $rating(\tau^4) = \frac{2}{3} \cdot 0.3 + \frac{1}{3} \cdot 0.4125 = 0.3375$. \square

Foreign Keys The semantics described in this section allows also for foreign keys. Consider a relation R_1 with a block A of key attributes, and another relation R_2 , that contains a block C_j with the same number and type of attributes as in A . Then C_j can be defined as a foreign key to A . To ensure referential integrity between C_j and A , it suffices to check for every update on C_j whether for the newly added values c_j there already exists a tuple τ of R_1 where the key A has the value c_j . If this is the case then referential integrity is ensured in all possible worlds. This is due to our assumption that different versions of τ are due to different values of the non-key attributes; however, the value c_j of the key attributes of τ is the same in all possible worlds.

Query Answering We define query answering in terms of the best rated world. That is, given a query from a user, in a first step, the best rated possible world is selected. Then the query is answered on this world. This approach has the advantage to provide the user a consistent view of the worlds, independent of the issued query. We therefore have to define the rating of a possible world. As each world is a set of tuples, we define the rating of a world as the average of the ratings over all tuple versions appearing in this world. That is, for a possible world W_i containing tuples $W_i = \{\tau_1, \dots, \tau_n\}$, $rating(W_i) = \frac{1}{n} \cdot \sum_{\alpha=1}^n rating(\tau_\alpha)$. We note that, in our model, every possible world has exactly the same number of tuples, as for every primary key, exactly one tuple version must be selected.

Obviously, the best rated world does not need to be unique. If this is the case, we prefer more recent updates: Going from the newest update to the oldest, if some of the best rated worlds contains the update (i.e. that have the corresponding block set to the value defined by the update), remove all worlds that do not contain this update, until only a single world is left. In the model described above, the most recent, best rated world can be found easily. Just pick for every block the best rated update. If there are several best rated updates for one block, then pick the youngest one. Of course, our approach could be easily adapted to other preference criteria: e.g., the most often rated update or the update with highest \overline{rep} .

3 Representation in U-relations

In this section we show that the data model described above can be encoded in U-relations. We use the database schema for U-relations as proposed in [2], consisting of the vertical decomposition tables (VDTs) for every attribute and the world table (W). For every tuple τ_i and every block C_j , we use a unique world set descriptor (WSD) $x_{i,j}$, and define in W one distinct assignment to $x_{i,j}$ for every possible value for C_j in any version of τ_i . That is, let $\tau_i = \{\tau_i^1, \dots, \tau_i^k\}$ be a tuple with k versions, $C_j = B_1, \dots, B_{s_j}$ a block, and $V = \bigcup_{\alpha=1}^k \pi_{C_j}(\tau_i^\alpha)$, where $\pi_{C_j}(\tau_i^\alpha)$ denotes the projection of τ_i^α onto the values for the attributes in C_j . In the vertical decomposition table (VDT) of every attribute $B_\beta \in C_j$, there exists exactly one distinct assignment to $x_{i,j}$ for every $v \in V$. For the vertical decompositions of the key attributes, there exists exactly one variable $x_{i,A}$ for every tuple τ_i , with exactly one assignment for $x_{i,A}$.

Example 3. τ_1 from Example 1 could be represented in U-relations as follows:

tid	WSD	T	WSD	A	WSD	B	WSD	S
τ_1	$x_{1,K} \rightarrow 1$	τ_1	$x_{1,1} \rightarrow 1$	a_1	$x_{1,1} \rightarrow 1$	b_1	$x_{1,2} \rightarrow 1$	s_1
			$x_{1,1} \rightarrow 2$	a_2	$x_{1,1} \rightarrow 2$	b_2	$x_{1,2} \rightarrow 2$	s_2

(Except for T , we discard the column holding the tuple id in the relations, as we consider only a single tuple. The corresponding world table is depicted in Example 4.) Note that every update is identified by an assignment to a WSD, e.g. $x_{1,1} \rightarrow 2$ corresponds to u_1 , or $x_{1,2} \rightarrow 2$ to u_2 . As discussed in Example 2, the initial update is split into the three updates u_K , u_0 , and u'_0 . For u_K , a WSD $x_{1,K}$ with the single assignment 1 is created, and added together with t_1 into the corresponding table. Similar for u_0 and u'_0 , where the WSDs $x_{1,1}$ and $x_{1,2}$ are used. For u_1 , a new assignment for $x_{1,1}$ is created, and corresponding entries are added to the VDTs for A and B . Similarly for u_2 and $x_{1,2}$. \square

As sketched in the example, every update operation identified above can be easily mapped to this representation: For the update of a non-key block C_j of some existing tuple τ_i , as already stated above, we only need to consider updates of a single block of a single tuple. The update process consists of creating a new assignment for the unique variable $x_{i,j}$, and adding a new entry for τ_i , the new assignment for $x_{i,j}$, and the new value to the VDT for every $B_\beta \in C_j$. If the value is already present for this block, then the update is ignored. The insertion of a new tuple is similar to the above operation. To insert a new tuple $(a_1, \dots, a_k, b_1, \dots, b_n)$, first insert the key (a_1, \dots, a_k) with a new WSD, and then perform ℓ updates on this tuple, by setting the values for the blocks c_1, \dots, c_ℓ .

To keep track of the ratings and user reputations, the information described above can be stored as follows. The information about ratings on updates can be stored in the world table W , as every update corresponds to the combination of a variable and an assignment, i.e. to one row in W . We extend $W(WSD, WSDvalue)$ to $W(WSD, WSDvalue, \overline{rat}, \overline{rep}, rating, timestamp [, user])$. Thereby $user$ stores a reference to the user who initiated the update. If the time window is stored explicitly for every user, this reference can be omitted. The information \overline{rat} , \overline{rep} , and $rating$ need to be present for every quality dimension

tracked. The current time window for each user can be maintained using a relation $tw(user, tid, WSD, WSDvalue, (count|timestamp))$. We can further log which user gave a rating to which update. Finally, in an additional table we store for every user her current reputation as well as \overline{rat}' and \overline{rep}' .

Example 4. The left table shows the world table to Example 3, while the right one stores the user information. We omit the representation of the time windows.

WSD	ass.	\overline{rat}	\overline{rep}	rating	user	\overline{rat}'	\overline{rep}'	reputation
$x_{1,K}$	1	1	1	1	John	8.16	16.6	0.492
$x_{1,1}$	1	8.4	12	0.7	$user_3$	20.34	22.6	0.9
$x_{1,1}$	2	2.4	8	0.3	$user_4$	2.46	12.3	0.2
$x_{1,2}$	1	25.2	28	0.9				
$x_{1,2}$	2	0.66	2.1	0.4125				

□

Query Answering We defined the semantics for query answering as answering the query on the most recently updated, top rated world. On the representation level, every possible world is identified by a (total) assignment to the WSDs [2]. As described above, in the world table W , we store every possible assignment to a WSD along with its rating (which corresponds to a rating on some update). Choosing one assignment to every WSD defines one set of tuples. For each of these tuples, we already defined their ratings, hence the rating of the world can be easily computed.

4 Extensions

In this section, we consider two important extensions of the basic scenario, namely, tuple updates with inter-block dependencies and tuple deletions. First we discuss the required adaptations of our basic model described in Section 2. We then also discuss the impact on the representation by U-relations.

4.1 Inter-block dependencies

In the basic scenario, updates of the attribute blocks were independent of each other. Assume now that the user wants to update the values of several blocks at the same time, e.g., by inserting new values for all three non-key attributes in our running example. Moreover, one may want to exclude from any possible world the combination of the new size with any of the prior values for kind and color of the animal. We call such updates spanning several blocks *rigid*.

In principle, such updates could be handled by simply merging the two blocks into one. However, this naive approach would lead to an explosion of the number of tuples needed to represent the possible worlds stored in the previously independent blocks. We therefore propose a more expressive representation scheme here.

Example 5. Consider the following shorthand notation of the schema of our running example: $obs = \overline{KB_1B_2}$ where K is the key block and B_1 and B_2 are the blocks of attributes as defined in Example 1. Let obs contain the tuple τ_1 with the uncertain value $\tau_1 = R(k, \{b_1|b'_1\}, \{b_2|b'_2\})$, giving rise to 4 possible worlds and the following decomposition into three partitions:

$\mathbf{obs_K}$	T	WSD	K
τ_1	x_K	k	

$\mathbf{obs_{B_1}}$	T	WSD	B_1
τ_1	$x_1 \rightarrow 1$	b_1	
τ_1	$x_1 \rightarrow 2$	b'_1	

$\mathbf{obs_{B_2}}$	T	WSD	B_2
τ_1	$x_2 \rightarrow 1$	b_2	
τ_1	$x_2 \rightarrow 2$	b'_2	

Suppose that some observer now wants to ensure that the size added only appears in conjunction with the animal type she also added. That is, she needs to insert a pair of block values b''_1, b''_2 as a rigid update of the tuple τ_1 . If one now chooses to merge respective blocks, it will be necessary to first explicitly specify all possible worlds compactly represented in the partitions $\mathbf{obs_{B_1}}$ and $\mathbf{obs_{B_2}}$:

$\mathbf{obs_K}$	T	WSD	A
τ_1	$x_K \rightarrow 1$	k	

$\mathbf{obs_{B_1}}$	T	WSD	B_1
τ_1	$x_1 \rightarrow 1$	b_1	
τ_1	$x_1 \rightarrow 2$	b_1	
τ_1	$x_1 \rightarrow 3$	b'_1	
τ_1	$x_1 \rightarrow 4$	b'_1	
τ_1	$x_1 \rightarrow 5$	b'_1	

$\mathbf{obs_{B_2}}$	T	WSD	B_2
τ_1	$x_1 \rightarrow 1$	b_2	
τ_1	$x_1 \rightarrow 2$	b'_2	
τ_1	$x_1 \rightarrow 3$	b_2	
τ_1	$x_1 \rightarrow 4$	b'_2	
τ_1	$x_1 \rightarrow 5$	b'_2	

Clearly, the total number of tuples needed for such “decompression” of the succinct representation with independent partitions is exponential in the number of blocks that have to be merged. Therefore, the succinctness of representation due to partitioning in U-relations is deteriorated by rigid updates. \square

Note that in [3] it is also observed that updates might make the decompression of the succinct representation by U -relations necessary. However, in our case, the blow-up of the U -relations is even more problematic since, in contrast to [3], we want to be able to increase the number of possible worlds as a consequence of an update (as we never overwrite any data). We therefore propose to use *compound* (that is, non-normalized [3]) WSDs for blocks. To express arbitrary dependencies between N blocks in a tuple, it is sufficient that the WSD of each block contain N variables.

Example 6. The final state of Example 5 has the following representation:

$\mathbf{obs_K}$	T	W_K	K
τ_1	$x_K \rightarrow 1$	k	

$\mathbf{obs_{B_1}}$	T	W_{B_1}	W_{B_2}	B_1
τ_1	$x_1 \rightarrow 1$			b_1
τ_1	$x_1 \rightarrow 2$			b'_1
τ_1	$x_1 \rightarrow 3$	$x_2 \rightarrow 3$		b''_1

$\mathbf{obs_{B_2}}$	T	W_{B_1}	W_{B_2}	B_2
τ_1			$x_2 \rightarrow 1$	b_2
τ_1			$x_2 \rightarrow 2$	b'_2
τ_1	$x_1 \rightarrow 3$	$x_2 \rightarrow 3$		b''_2

Since only the value combinations connected to consistent variable assignments are admitted, one can check that the U-relations above define exactly the desired five possible worlds: E.g., the tuple (b_1, b'_2) is not part of the (uncertain) projection $\pi_{B_1, B_2}(\mathbf{obs})$, as the variable assignment $x_1 \rightarrow 1, x_1 \rightarrow 3$ is inconsistent, whereas (b_1, b'_2) , corresponding to the assignment $x_1 \rightarrow 1, x_2 \rightarrow 2$, is a possible answer. \square

The following procedure can then be used to accommodate new inserts into the database: Let C be an attribute block, and assume that each attribute belongs to a separate U-relational partition. Thus, each attribute in C has a compound WSD of at most $|C|$ variables. Consider a rigid update u for a tuple τ introducing a new value assignment for attributes $C' \subseteq C$; it can be accommodated in the following two steps:

1. *Build a WS descriptor \bar{w}_u for u :* For each attribute $A \in C'$ with a corresponding variable x_A in the WS descriptor of $\tau.C$, check if the value $u.A$ is already present in $\tau.A$. If yes, re-use the found assignment for x_A in \bar{w}_u ; otherwise, take a fresh domain value not occurring in the WS descriptor of $\tau.A$ as an assignment for x_A in \bar{w}_u .
2. *Perform the insert:* for each attribute $A \in C'$, insert the tuple $(id(\tau), \bar{w}_u, u.A)$ in the partition P_A of the U-relation.

For example, a rigid update b_1, b_2^* for the blocks B_1, B_2 of the relation *obs* in its final state as shown in the Example 5 will be assigned a WSD $x_1 \rightarrow 1, x_2 \rightarrow 4$.

Note that new updates can be composed of the values already occurring in other updates: the update $\tau_1.B_1 = b_1'$ can be admitted (and assigned a WSD $x \rightarrow 3$), despite the rigid update $B_1B_2 \leftarrow (b_1', b_2')$ being already present in the table. A possible meaning of such new insert is “the value b_1' can be combined not only with b_2' , but with any other value of the block B_2 ”.

Ratings of rigid updates In the basic scenario, each update is identified by an assignment to its WSD variable. The aggregated ratings associated with each update are summarized in the world table. If rigid updates are allowed, the world table needs to be extended to accommodate compound WSDs: the number of variable/value column pairs equals the maximal number of blocks in any relation described by the world table; representation of ratings of updates remains the same as in the Example 3. Deriving ratings of tuples from the update ratings must be redefined, however. We address this issue in Section 4.3.

4.2 Deleting tuples

So far we have only considered disagreements on the correct values for the tuples stored in the database. We now extend the system to also allow to express that some tuple should not be present at all. This is modelled by introducing a special tuple version, namely \emptyset , to express that there exist possible worlds that do not contain any version of this tuple at all. Under this semantics, deletion of a tuple corresponds to adding \emptyset to the set of possible versions for this tuple. Concerning our definition of updates, we model a deletion as $u: (A_1, \dots, A_k) \leftarrow ()[key]$, where A_1, \dots, A_k are the key attributes. We do, however, not allow \emptyset to be the only version of a tuple, but require at least one other version to exist.

Representing \emptyset in U-relations can be done easily. Given a relation R with the key-block K , for every tuple $\tau_i \in R$, so far the WSD consists of exactly one variable $x_{\tau_i, K}$ with a unique assignment (say $x_{\tau_i, K} \rightarrow 1$) that encodes the value of the key for τ_i . Inserting the version \emptyset for τ_i can be done by adding another assignment for $x_{\tau_i, K}$ (say $x_{\tau_i, K} \rightarrow 0$) to the world table, without inserting an entry for this assignment to the VDTs of the key attributes. With non-normalized U-relations, it can be further ensured that this (empty) “selection” for the key values cannot be combined with any other non-null values of the non-key attributes: It suffices to add for every tuple τ_i the assignment $x_{\tau_i, K} \rightarrow 1$ to the WSDs of every non-key attribute block. That is, the VDT of each non-key block is extended by a column WSD_{key} that contains $x_{\tau_i, K} \rightarrow 1$ (resp. by columns $WSD_{key}, WSD_{key}val$ with $(x_{\tau_i, K}, 1)$). Hence selecting $x_{\tau_i, K} \rightarrow 0$ will return no value for any attribute of τ_i .

Example 7. Consider the representation in Example 6. To support deletion, the WSD of the key block is added to each partition.

obs _K			obs _{B₁}					obs _{B₂}				
T	W _K	K	T	W _K	W _{B₁}	W _{B₂}	B ₁	T	W _K	W _{B₁}	W _{B₂}	B ₂
τ ₁	x _K → 1	k	τ ₁	x _K → 1	x ₁ → 1		b ₁	τ ₁	x _K → 1		x ₂ → 1	b ₂
			τ ₁	x _K → 1	x ₁ → 2		b' ₁	τ ₁	x _K → 1		x ₂ → 2	b' ₂
			τ ₁	x _K → 1	x ₁ → 3	x ₂ → 3	b'' ₁	τ ₁	x _K → 1	x → 3	x ₂ → 3	b'' ₂

Deletion of τ₁ is allowed by leaving the above tables unchanged, but adding the assignment x_k → 0 to the world table. □

Rating tuple deletions The deletion of a tuple can be rated like every other update on the database. As ∅ is expressed by an entry in the world table, it is also no problem to store the rating there, hence also just like the rating on every other update. The definition of the rating of a tuple version τ_i^k requires a slight modification: If τ_i^k ≠ ∅, then it is as defined in Section 2. Otherwise, if τ_i^k = ∅, then $rating(\tau_i^K) = rating(u_\alpha)$, where u_α is the update that inserted ∅.

4.3 Adapting the top database semantics

So far in this section we have concentrated solely on the representation of updates and possible worlds defined by them. We now shift our focus to user ratings and, consequently, to the reconciliation of conflicting updates, which is an essential task from the user point of view. In the following, we assume each relation to contain at least one non-key attribute block. Note that this is no real restriction, as otherwise there could not be multiple versions for this relation anyway.

Conceptually, the conflict-resolution approach remains exactly the same as in the basic scenario: We first look for WSD variable assignments maximizing the overall world rating, and then we use the corresponding top rated world for query answering. However, the extensions discussed earlier in this section have several important implications:

Maximal feasible variable assignments Because of the introduction of rigid updates, variable assignments for different variables are no longer independent of each other, as it was the case in the basic scenario. Due to the fact that the WSD of each rigid update is now a set of variable assignments, the dependencies between these assignments have to be enforced. We must thus speak of *maximal feasible variable assignments* defined by consistent sets of WSDs.

Example 8. Consider the setting from Example 7. There, x_k → 1, x₁ → 3, x₂ → 3 is an example of a maximal feasible variable assignment. The assignment x_k → 1, x₁ → 1 is feasible but not maximal, whereas x₁ → 3, x₂ → 2 and x_k → 0, x₁ → 1, x₂ → 1 are not feasible: No valid combination of updates issued to the database gives such a variable assignment on the WSD. □

Definition 1. A set **U** of WSDs (resp. the set **u** of updates identified with **U**) is proper if it satisfies the following conditions:

— Consistency: No two WSDs in **U** contain different assignments for the same variable (resp. no two updates in **u** have different values for the same block).

— Maximal coverage. Let domain $\text{dom}(\mathbf{U})$ denote the set of all variables in \mathbf{U} . We request that the domain of \mathbf{U} is maximized in the following sense: there is no update in the database identified with the WSD U , such that $\mathbf{U} \cup U$ is consistent and $\text{dom}(\mathbf{U}) \subset \text{dom}(\mathbf{U} \cup \{U\})$ (resp. every tuple is either deleted by \mathbf{u} or the values for all blocks of the tuple are specified by \mathbf{u}).

— Irreducibility. Let $\text{nkdom}(\mathbf{U})$ denote $\text{dom}(\mathbf{U})$ restricted to non-key variables: that is, variables not occurring in the WSD of any key block. We request that for each WSD $U \in \mathbf{U}$, $\text{nkdom}(\mathbf{U} \setminus \{U\}) \subset \text{nkdom}(\mathbf{U})$ (resp. no update in \mathbf{u} spans only the blocks which are also spanned by other updates).

It can be easily shown that any proper set of WSDs (resp. proper set of updates) defines a maximal feasible variable assignment.

Example 9. Let a relation R have a schema \overline{KABC} where each letter defines an attribute block, the key one denoted by K . Suppose that a tuple identified by a key value k is comprised by the following four rigid updates: $\{u_1^{r=0.3}: AB \leftarrow (a, b), u_2^{r=0.45}: C \leftarrow c, u_3^{r=0.5}: AC \leftarrow (a, c), u_4^{r=0.2}: B \leftarrow b\}$ (where superscripts denote update ratings) which give rise to a single possible tuple value (k, a, b, c) , and one update u_5 , where the tuple was deleted.

According to the procedure of U-relational updates from Section 4.1, a value of a non-key block determines the corresponding WS variable assignment: Hence, there are three assignments to non-key variables $x_a \rightarrow v_a$, $x_b \rightarrow v_b$, and $x_c \rightarrow v_c$, and two assignments to the key variable, namely $x_k \rightarrow 1$ and $x_k \rightarrow 0$. The variable assignments of the updates are $U_1 = x_k \rightarrow 1 \wedge x_a \rightarrow v_a \wedge x_b \rightarrow v_b$, $U_2 = x_k \rightarrow 1 \wedge x_c \rightarrow v_c$, $U_3 = x_k \rightarrow 1 \wedge x_a \rightarrow v_a \wedge x_c \rightarrow v_c$, and $U_4 = x_k \rightarrow 1 \wedge x_b \rightarrow v_b$.

One can check that there are three proper update sets assigning a non-empty value to the tuple k : namely, $\mathbf{u}_1 = \{u_1, u_2\}$, $\mathbf{u}_2 = \{u_3, u_4\}$ and $\mathbf{u}_3 = \{u_1, u_3\}$. We say that all of them *define* a maximal variable assignment $\mathcal{A} = \{x_k \rightarrow 1, x_a \rightarrow v_a, x_b \rightarrow v_b, x_c \rightarrow v_c\}$. By definition, \mathcal{A} is feasible (as all three update sets are consistent). Yet another maximal feasible variable assignment is $x_k \rightarrow 0$ implying that tuple k must not be part of the respective worlds. \square

Finding a best rated world then amounts to selecting a proper update set that maximizes the world rating. It turns out, however, that the computation of the rating of some world defined by a given set of updates also requires adaptation if rigidity is allowed.

Composing update ratings In the basic scenario, the rating of a tuple value was given by: $\text{rating}(\tau_i^j) = \sum_{\alpha=1}^{\ell} (w_{\alpha} \cdot \text{rating}(c_{\alpha}))$ where every c_{α} is a block value corresponding to some update (recall that in absence of rigid updates each block value is given by exactly one update) and weight w_{α} depends on the attributes comprising the respective block C_{α} . We call this *the composition formula*.

In presence of rigid updates, we no longer have ratings for each block straight away, but rather the rating for each rigid update. We re-define the rating of a block value c_{α} as the maximal rating of an update in which c_{α} is contained.

Example 10. As an example, consider the update set $\mathbf{u}_3 = \{u_1, u_3\}$. We take $r(A \leftarrow a) = \max(r(u_1), r(u_3)) = 0.5$, $r(B \leftarrow b) = r(u_1) = 0.3$, and $r(C \leftarrow c) = r(u_3) = 0.5$ (we shorten $\text{rating}(\cdot)$ as $r(\cdot)$ for the sake of readability). According

to the composition formula, $rating(\mathbf{u}_3) = \frac{1}{3}(0.5 + 0.3 + 0.5) \approx 0.43$ (the weight w_α of each block is taken equal to $\frac{1}{3}$, so that the tuple ratings fall into $[0, 1]$).

As shown in Example 6, \mathbf{u}_3 is not the only proper update set defining the variable assignment \mathcal{A} . Its rating — and the rating of the corresponding tuple value (k, a, b, c) — is thus not uniquely determined: other possible candidates are $rating(\mathbf{u}_1) = \frac{1}{3}(0.3 + 0.3 + 0.45) = 0.35$ and $rating(\mathbf{u}_2) = \frac{1}{3}(0.5 + 0.5 + 0.2) = 0.4$, obtained by the composition formula. Similarly to selection of block ratings, we choose the maximum (i.e. $rating(\mathbf{u}_3) \approx 0.43$) as the rating of the tuple value. \square

Let \mathcal{A} be a maximal feasible variable assignment, corresponding to a world $\mathcal{W}_{\mathcal{A}}$. Its rating can be found in two steps:

1. For each tuple value τ determined by \mathcal{A} , take $rating(t)$ to be the maximum rating computed according to the composition formula from any proper update set defining \mathcal{A} .
2. Take the maximum of the tuple ratings as the world rating (Other aggregates like sum or average can be used instead).

The top world is then chosen as the most recently updated world from the worlds with the highest rating, as described in Section 2.

5 Conclusion and Future Work

In this paper we presented collaborative data management as a relevant application area for uncertain databases. We proposed an update storage model in which user contributions are never overwritten or deleted, but persist in the database in their original form, leading to different possible worlds. We described a framework in which community feedback is used to reconcile contradicting updates and to compute reputation values for each user according to the feedback on her updates. One of the main points of the paper was to describe a relevant use case for uncertain databases, in particular based on U-relations. Due to the lack of space, we have left out some important questions that have to be addressed in a real-world system: e.g. how to prevent creation of virtual users to rate dummy updates for the sake of earning high reputation [15].

Directions for future work are thus manifold. A particularly important question is the choice of the semantics for query answering. Unlike the approach chosen here, in probabilistic databases queries are conceptually evaluated over all possible worlds, and then the “best” possible answer is returned. It would be thus interesting to extend our framework to other semantics for query answering and to design efficient methods for query evaluation under these semantics. Another open issue is a precise complexity analysis of query answering.

Besides determining the best choices for all these problems, there are also several open questions concerning techniques of the underlying uncertain databases. A potential shortcoming of the semantics described in Section 2 is the arbitrary combination of all available, compatible updates. Roughly speaking, every update is applied to all possible worlds in parallel. The user, however, might just want to introduce exactly one new possible world, or to be able to specify the set of possible worlds to which the update is applied. Hence we do no longer

make any assumptions about dependent and independent attributes, but the set of possible worlds is just defined by the updates performed. To the best of our knowledge, no algorithm for these kinds of inserts in U-relations are known. So far, only updates that really overwrite the old data have been considered [3], but the insertion of new possible worlds has not been addressed.

Another problem arises from the assumption that data is never lost. In practice, this will most probably not be possible or maybe not even desirable. While defining some agreement which data to delete can be easily done, to the best of our knowledge, also the problem of efficiently deleting possible worlds from U-relations has not yet been completely solved.

Finally we plan an implementation and experimental evaluation of the proposed framework.

References

1. G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734 – 749, june 2005.
2. L. Antova, T. Jansen, C. Koch, and D. Olteanu. Fast and simple relational processing of uncertain data. In *Proc. ICDE 2008*, pages 983–992. IEEE, 2008.
3. L. Antova and C. Koch. On APIs for probabilistic databases. In *Proc. of QDB/MUD '08*, pages 41–56, 2008.
4. C. Batini, C. Cappiello, C. Francalanci, and A. Maurino. Methodologies for data quality assessment and improvement. *ACM Comput. Surv.*, 41(3), 2009.
5. BIRN. <http://nbirn.net/cyberinfrastructure/portal.shtm>.
6. N. Dalvi, C. Ré, and D. Suciu. Probabilistic databases: diamonds in the dirt. *Commun. ACM*, 52(7):86–94, 2009.
7. W. Gatterbauer, M. Balazinska, N. Khoussainova, and D. Suciu. Believe it or not: adding belief annotations to databases. *Proc. VLDB Endow.*, 2(1):1–12, 2009.
8. W. Gatterbauer and D. Suciu. Data conflict resolution using trust mappings. In *Proc. of SIGMOD 2010*, pages 219–230. ACM, 2010.
9. GEON. <http://www.geongrid.org/>.
10. T. Gruber. Collective knowledge systems: Where the social web meets the semantic web. *J. Web Sem.*, 6(1):4–13, 2008.
11. R. Gupta and S. Sarawagi. Creating probabilistic databases from information extraction models. In *VLDB*, pages 965–976, 2006.
12. Z. G. Ives, T. J. Green, G. Karvounarakis, N. E. Taylor, V. Tannen, P. P. Talukdar, M. Jacob, and F. Pereira. The ORCHESTRA collaborative data sharing system. *SIGMOD Rec.*, 37(3):26–32, 2008.
13. L. Kot and C. Koch. Cooperative update exchange in the Youtopia system. *PVLDB*, 2(1):193–204, 2009.
14. B. Leuf and W. Cunningham. *The Wiki Way – Quick Collaboration on the Web*. Addison-Wesley, 2001.
15. B. N. Levine, C. Shields, and N. B. Margolin. A Survey of Solutions to the Sybil Attack. Tech report 2006-052, University of Massachusetts Amherst, October 2006.
16. A. D. Sarma, M. Theobald, and J. Widom. Live: A lineage-supported versioned dbms. In *Proc. of SSDBM 2010*, volume 6187 of *Lecture Notes in Computer Science*, pages 416–433. Springer, 2010.
17. S. Singh, C. Mayfield, S. Mittal, S. Prabhakar, S. Hambrusch, and R. Shah. Orion 2.0: native support for uncertain data. In *SIGMOD '08*, pages 1239–1242, New York, NY, USA, 2008. ACM.

Handling Uncertainty and Correlation in Decision Support

Katrin Eisenreich and Philipp Rösch

SAP Research Center Dresden, Germany

`katrin.eisenreich@sap.com`, `philipp.roesch@sap.com`

Abstract. Many models and approaches have been developed to tackle the management of diverse uncertain data in different application fields. However, support for flexible handling of such data in the context of decision support has not received dedicated attention so far. We argue that this application field necessitates special support for the introduction and modification of uncertainty to incorporate a user's assumptions. Therefore, this paper addresses the representation, analysis, and modification of uncertainty with the primary goal to provide means for efficient analysis and scenario creation for decision support. This includes the modeling of correlated uncertain data on a suitable level of flexibility, such as to enable a broad range of types of uncertain data and dependencies in such data. At the same time, we preserve a reasonable level of “simplicity” for users to interpret and apply the provided functionality. We discuss features and propose a data model and an operator set for introducing, analyzing, and modifying uncertainty information. Furthermore, we show how the presented solution can help experts to examine and track uncertainty in their data throughout the analysis process and thus consider different assumptions to make better founded decisions.

1 Introduction

Analysis of historic data and forecasting of future developments are both central tasks in decision support. Based on previous business results and assumptions about the future, a decision maker must decide on specific actions to take, aiming to achieve an optimal outcome with respect to some target. However, analysis and planning processes that result in crisp numbers can disguise the uncertainty inherent both in underlying data and in assumptions about future developments. To account for these uncertainties, they must be represented in the data and considered in operations applied to the data. Existing uncertainty management approaches, such as those reported in [1, 2] mostly address application scenarios in the field of scientific and sensor data processing, spatial databases, information extraction, or data cleansing. Decision support over large volumes of data including uncertainty of measure values, however, has not received broad attention so far, although it imposes similar challenges.

To address this topic we investigate the representation, analysis, and modification of uncertain data with the primary goal to provide means for powerful

and flexible creation and evaluation of what-if scenarios. As a starting point for such scenarios, we consider assumptions users make about future developments (which, by definition, are uncertain). Firstly, we target the question of how such uncertain information is introduced to the system. While in some cases users want to provide empirical values, probably not all users are able to specify the nature of the distribution of a value. Therefore, they should be empowered as far as possible to compute distributions from available data using suitable operators. Secondly, users should be able to modify uncertain values based on assumptions, e.g., about changing conditions or effects of business decisions they aim to evaluate. In a next step, they should then be able to compare scenarios derived from different assumptions. A third important aspect that relates both with the nature of uncertain data and the introduction of assumptions is dependency, or correlation, in data. Previous research such as [1, 3, 4] mainly investigates means to keep track of dependencies introduced by relational algebra operators such as joins. All those approaches consider *input* data independent. In particular, given two (separately provided) input values and a user's "knowledge" about their correlation, they do not provide any functionality for introducing this information to the system. We argue that available input data, such as forecast values of various economic figures, might indeed be highly correlated. Also, users might wish to evaluate the influence of different correlation patterns between values of interest. Such functionality necessitates means to provide information about assumed correlation to the system.

We hold that the introduction and modification of uncertainty, scenario creation, and proper handling of correlation are particularly relevant when it comes to interactive processes—processes, where users create and adjust assumptions to eventually derive "new" data and decisions from such assumptions. The following are the primary aspects we address in the remainder of this paper to tackle those issues:

- **Expressivity of probabilistic data model:** To meet the requirements imposed by the decision support context, our data model supports the representation of attribute-level uncertainty over discrete and continuous domains (the former can be represented as special case of the latter). Further, we address the modeling of correlation between uncertain attributes and the representation of alternative scenarios implementing a user's assumptions.
- **Versatile functionality for analyzing and processing uncertain data:** We provide operators for the analysis over uncertain data as well as the introduction and modification of uncertainty, particularly enabling the introduction of correlation information and the derivation of different scenarios. The provided set of high-level operators is closed. A set of basic functions serves to retrieve characteristics of uncertain values and is used in the computation of those operators.

The remainder of the paper is structured as follows: In Section 2, we discuss fundamental concepts for representing and processing uncertainty in data; we introduce two example tasks to motivate and illustrate our concepts throughout

the paper. Section 3 describes the data model and operator set implementing the proposed features. We illustrate exemplary compositions of our operators and address specific details applying them to our example tasks in Section 4. Section 5 relates the presented work to findings of previous research. In Section 6, we conclude the paper and give some indications of future work.

2 Analyses and Uncertainty

Before we introduce our data model and the operators for handling uncertainty, we give a brief overview over some fundamental concepts of representing and processing uncertainty. We focus on features that are particularly interesting for what-if scenario analyses over large-scale datasets. To keep it simple and more concrete, consider a business case where an expert investigates potential markets for a company's products and prospective revenue generated within different regions. As running examples, we consider the following two tasks:

- **T1: Regional Revenue Forecast** An expert wants to evaluate next year's probable revenue in a newly developed region R_{new} . For lack of long-term historic data for this region, he wants to predict the revenue development based on the past development of a similar region R_{ref} , assuming the revenue in R_{new} of the last quarter in 2009 as baseline value. Additionally, he wants to take into account forecasts about the general economic development.
- **T2: Product Portfolio Management** The expert wants to investigate the change in sales of a product P_A after the launch of product P_B , given knowledge about the effect of a similar product P_C on sales of P_A . He evaluates different assumptions with respect to possible sales numbers of P_B to evaluate marketing strategies.

The first task addresses the aspects of introducing uncertainty and correlation; in the second task, we additionally consider the modification of uncertainty to create alternative scenarios.

2.1 The Analysis Process

The analysis processes associated with tasks as described above involve both operations similar to those known from OLAP (slicing, dicing, roll-up, drill-down) and operations for derivation, introduction, and modification of uncertain data. Figure 1 illustrates on an abstract level how operations are applied in such a process in a sequential and iterative fashion. After loading and analyzing underlying data, different operations for the evaluation of assumptions can be applied before the user saves one or more resulting scenarios. In the graph, operators are shown as rectangles with round corners. Rectangles depict data consumed and produced by the steps (i.e., the applied operators) in the process. We consider both certain (historic) and uncertain (approximated or forecast) data loaded from a data warehouse (Step 1). In the figure, uncertain data is distinguished

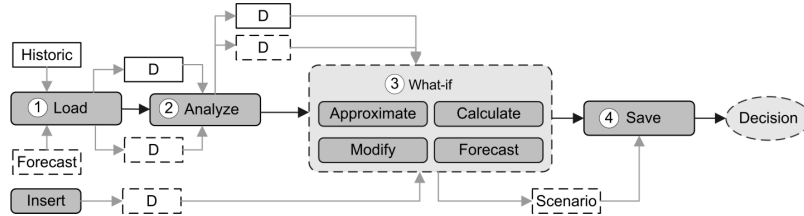


Fig. 1. A schematic description of the what-if analysis process

from certain data by dashed lines. Users select and analyze data to view it on an appropriate aggregation level (Step 2). To evaluate a possible development of the current situation those views can then be processed through several steps to create an (uncertain) what-if scenario (Step 3); e.g., users can select and calculate statistic measures of past data and use the result as input for a forecast. Additionally, they might insert new values assumed to hold for some attribute. They can iterate steps 2 and 3 to derive, explore, and subsequently compare several scenarios based on previous intermediate results. For example, data resulting from an analysis in step 2 can serve as input for several what-if analysis steps 3a, 3b, etc. The final result of the illustrated process is uncertain data of one or several scenarios. Users can investigate those scenarios to make appropriate business decisions and possibly store them for later reference (Step 4).

2.2 Uncertainty in Decision Support Scenarios

As a basis for our data model and the operators presented in Section 3, we give some general remarks on representing and handling uncertainty.

Representing Uncertainty To appropriately handle uncertainty in decision support systems, we have to observe the specific characteristics of the data and tasks involved. First, the domain of a value's distribution can be either continuous, like forecast results provided as an expected value and an associated confidence interval modeled, e.g., as a Gaussian distribution, or discrete, like for cases where users want to model only a number of discrete alternatives with their respective probabilities. Here, an example is the evaluation of a company's potential revenue increase of +3%, +5%, and +10% with associated probability values of 0.5, 0.4, and 0.1, respectively. Second, uncertainty can occur both with respect to measure values describing a relevant key figure, and regarding the dimensional characterization of data. In the context of planning and forecasting, this relates especially to the temporal allocation of facts, which can hold during some indeterminate time interval. Examples are uncertain product launch times or a project duration which cannot be determined with certainty. We want to enable users to incorporate the aspect of temporal indeterminacy in their analyses, e.g., by aggregating over indeterminate "events". However, this aspect is rather complex and therefore will not be addressed in detail in this paper.

Correlation In addition to uncertainty associated with individual values, real-world data is often subject to correlation. This means that values can not be handled as independent from each other as they covary and therefore exhibit a dependent behavior which must be considered in computations. If such dependencies are not reflected in the data, analyses can yield incorrect results in terms of misleading probabilities, possibly leading to an underestimation of risks or chances. As a simple example, in task T1 the data considered will probably be subject to some correlation in the sense that the (predicted) revenue development in region R_{ref} is positively correlated with the economic growth forecast. Since data can exhibit dependency structures other than linear correlation, we need means for flexible representation of different correlation structures between arbitrarily distributed values.

Handling Uncertainty During the analysis process, users do not only want to be informed about the present uncertainty but also need to evaluate the effect of new assumptions or modified uncertain information.

Introduction of uncertainty At some point, users need to introduce uncertainty derived from experience or as approximations from initially certain data. For example, to derive prospective future trends, they want to incorporate historic knowledge approximated from a set of facts (e.g., sales from different cities) capturing a value's distribution over those facts rather than the exact values. The intuition is that for future developments we will likely observe values which follow a similar distribution as those recorded in the past. Additionally, if there exists information about correlation in data which can not be derived from fact data but is based on users' assumptions, it should be—and with our approach it is—possible to introduce information about diverse correlation structures to the present values. For example, if users assume a linear correlation of 0.8 between the economic growth and the regional revenue development, they can introduce this information to the existing data. Similarly, they can apply different dependency structures and subsequently compare their influence on the result.

Modification of uncertainty When users want to analyze the influence of different assumptions about an aspect of the future, they need to apply changes to uncertain data, e.g., an expected revenue value. We treat such modifications as creation of new (delta) versions of the original data. This enables users to derive alternative scenarios and thus to relate and track the derivation of initial and modified values throughout the process.

The next section describes how we address the motivated features, providing details on the data model and operators used for uncertainty representation, analysis and modification.

3 Data Model and Operators

The central goal in the design of our model for uncertainty is versatility both with respect to the form of uncertainty representable and the operations we

can execute using our model. Users can independently choose different forms for (i) the concrete representation for describing uncertain values and (ii) the access to such data. Currently, we support a symbolic and a histogram-based representation. A basic interface to a distribution offers access to its statistic characteristics, such as its moments and function values of the density, distribution, and inverse distribution functions. The statistics are accessed independently from the concrete underlying representation form, while their computation is, of course, representation-specific.

3.1 Data Model for Representing Uncertain Values and Correlation

We now describe our model for the representation of uncertainty. As already stated, a number of basic continuous distributions can be modeled symbolically. This form is optimal in terms of storage and also allows for highly efficient execution of a range of operations for some distributions. Most real-world data, however, cannot be fit exactly by a basic distribution function. Moreover, application of symbolic computations is restricted to only a few distribution functions and some of the involved operations are very expensive. The focus of this section therefore is the application of the more generic, histogram-based representation of distributions. This representation will be assumed implicitly in the remainder of the paper when we talk about a distribution unless explicitly stated otherwise.

Histogram-based representation of arbitrary distributions Similar to [1], we use compact histograms representing continuous distribution functions to circumvent the abovementioned restrictions of symbolic representations. For an attribute X , we consider uncertain attribute values x_i distributed according to a distribution P_i within a support interval $I_i = [l_i, h_i]$. The bounds l_i and h_i indicate the lowest and highest value x_i may take. The distribution P_i is represented by a set of pairs $\{(v_j, f_j)\}$ where v_j are discrete values from I_i and f_j are the respective frequencies in the underlying data. A histogram \bar{P}_i^β is built by partitioning the concrete value instances for x_i into β mutually disjoint subsets (bins) $B = \{b_1, \dots, b_\beta\}$. Each bin b_k represents a sub-interval $[l_k, h_k]$ of I_i and is associated with the relative frequency \bar{f}_k computed by aggregating frequencies f_j of all values lying in $[l_k, h_k]$. Similarly, we can approximate a continuous distribution function (pdf_i), associating each bin b_k of the result histogram with a density value computed as the integral $\int_{l_k}^{h_k} pdf_i$. Figures 2(a) and 2(b) show two histograms; \bar{P}_{inc_e} represents the density function of the predicted (relative) economic growth inc_e distributed according to a Gaussian with mean 0 and variance 1, while $\bar{P}_{inc_{ref}}$ represents the (relative) regional revenue increase inc_{ref} calculated from historic revenue numbers per city in region R_{ref} . Histograms can be constructed flexibly by applying different partitioning schemes which we do not want to investigate at this point (see, e.g., the classification in [5]). At the moment, we use equi-width histograms due to their construction and update efficiency. Other forms such as equi-depth histograms could be introduced but

would require expensive conversions for some of our operators to process them correctly.

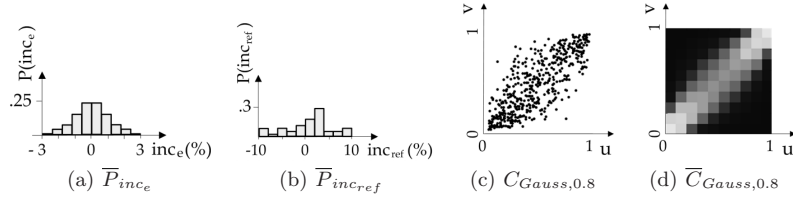


Fig. 2. Representation of univariate distributions, a copula (500 samples), and a corresponding ACR. In (d), light levels of gray indicate high density values.

Correlation Similar to one-dimensional distributions, multivariate distributions can be approximately represented using multidimensional histograms. If we dispose of (historic) fact data containing values for two (or more) attributes, we can compute a multidimensional histogram¹ to represent the common distribution over the attribute values. However, this approach is not applicable in cases where only “independent” values are available (e.g., values inc_e and inc_{ref}) or the available data is too sparse. If a user wants to evaluate an *assumption* that there is a certain correlation, say, a linear correlation of $\rho = 0.8$, between those values, we must provide means to introduce this correlation first. A well-known approach to address this issue is to use Monte Carlo simulation (see, e.g., [6]) to “generate” samples from the uncorrelated distributions such that the samples have marginal distributions equal to the distributions of the values to correlate and exhibit the desired correlation. Then, we can build a multidimensional histogram over those samples. Eventually, the correlation will be represented both encapsulated in the chosen sampling function and the resulting multidimensional histogram. One drawback of this approach is that—depending on the marginal distributions as well as the structure of dependency in the resulting distribution—the sampling process can incur high computational costs at run time. Additionally, the approach is restricted regarding the correlation structures we can impose and the marginal distributions of the values to correlate.

To circumvent the given restrictions and provide a flexible method for correlation introduction within our solution, we propose to represent correlation information as a separate artifact, independent from the concrete values to be correlated. This artifact is constructed as an n -dimensional histogram over an instance of special functions known as *copulas*. In brief, a copula is an n -dimensional distribution function $C : [0, 1]^n \rightarrow [0, 1]$ with uniform marginals, representing the functional relation between n individual distributions and their

¹ Note that the construction of equi-width histograms can be easily extended to the multi-dimensional case.

joint distribution. In this paper, without restriction of generality, we consider the case $n = 2$. That is, we link two distributions using a two-dimensional copula with marginals u and v uniformly distributed over $[0, 1]$. Formally, the concept of copulas is founded on Sklar's Theorem [7]. It states that, given H as a bivariate distribution with F and G as univariate marginal distributions, there exists a (copula) function $C : [0, 1]^2 \rightarrow [0, 1]$ so that $H(x, y) = C(F(x), G(y))$. Conceptually, C is a mapping assigning the value of the joint distribution function H to each ordered pair of values of the marginal distribution functions F and G . Using the inversion approach (see, e.g., [8]), we can construct a copula $C(u, v) = H(F^{-1}(u), G^{-1}(v))$, the structure and degree of the correlation being determined by the distribution H used to build the copula. We write $C_{H,d}$ to denote a copula constructed based on a bivariate distribution H with correlation d . For example, we can compute $C_{Gauss,0.8}$ based on a bivariate Gaussian distribution with a correlation of 0.8, as displayed in Figure 2(c). In order to use such a copula to correlate two arbitrary distributions P_x and P_y , we again apply Sklar's Theorem, substituting F and G with the desired marginals P_x and P_y . For further details on the copula approach, see, e.g., [8].

Copulas have been widely used, e.g., for stochastic modeling in the field of risk management [9]. However, to the best of our knowledge, they have not yet been applied in the database field. We propose to pre-compute copulas and store them in the form of histogram-based approximate correlation representations (ACRs, denoted by $\bar{C}_{H,d}$) for a number of possible correlation structures and degrees. For example, Figure 2(d) displays the ACR for the copula shown in Figure 2(c). ACR histograms induce only small memory costs, depending on the parameters of the histogram construction. Further, when using ACRs to introduce correlation between distributions at run time, we can avoid expensive sampling and function calls. In the next section, we describe an operator for introducing the correlation represented by an ACR between two concrete values.

3.2 Operators for Handling Uncertainty

We now present our set of logical operators supporting the features motivated earlier. We group operators according to whether they support functionality for representation (and converting representations), analysis, or modification of uncertain data.

Representation of Uncertainty As discussed above, we can use either symbolic or histogram-based representations of uncertainty. To enable users to flexibly use the representation that best fits their needs, we provide the following functionality.

Statistic properties: We provide a basic interface for retrieving moments of a distribution and the function values of the probability density $pdf(x)$, distribution $cdf(x)$, and inverse distribution $invcdf(p)$ functions. Apart from gaining insight into the statistic characteristics of data, those functions are

used in the computation of our operators. In particular, for introducing correlation information we make use of $invcdf(p)$ as will be described below.

Derive uncertainty $DRV(V, tgt) = x_{new}$: Given input values V from a set of fact data, users can introduce uncertainty by approximating the discrete data by a distribution P_{new} . Using DRV , they can either build a histogram \bar{P}_{new}^β or derive parameters for an (assumed) distribution function, depending on the specified target distribution tgt . Histograms are built based on parameters specifying the desired type (e.g., equi-width) and the number of bins within a support interval. As an example in the context of T1, a user can apply DRV on the quarterly revenue growth numbers collected for cities in region R_{ref} . The resulting value inc_{ref} reflects the distribution of those numbers which then can be assumed as reference distribution for other regions. Figure 2(b) displays the associated result histogram $\bar{P}_{inc_{ref}}$.

Convert representations $CNV(x_1, tgt) = x_2$: The CNV operator converts any symbolic representation of x_1 into its approximate histogram-based form, or vice versa, depending on the parameters specifying the target distribution tgt . Note that the accuracy of a conversion from an histogram-based into a symbolic representation depends on the requested target distribution. That is, the user needs knowledge about the nature of the distribution (e.g., whether the data roughly follows a Gaussian distribution) to compute the appropriate parameters from underlying values.

Analysis and Computation over Uncertain Values We support standard analysis operators similar to relational algebra such as selection and aggregation, incorporating the uncertainty in data.

Filter $FIL(X, cond, prob) = \{x_i | P(cond(x_i)) > prob\}$: The user can perform selections similar to selections known from relational algebra, providing an attribute of interest X , a selection condition $cond$ in terms of an operator and a value for comparison, and a probabilistic threshold value $prob$. The value of $prob$ determines the lowest admissible probability of the predicate holding. For example, for an uncertain attribute value x_i uniformly distributed over the interval $[0, 5]$ the condition $x_i < 4.5$ with a required minimum probability of $prob = 0.8$ evaluates to *true*, and x_i passes the selection filter.

Aggregate $AGG(\{x_1, \dots, x_n\}) = x_{AGG}$: Similar to aggregation over certain data we support the computation of aggregates over uncertain values, such as their sum (AGG_{SUM}) or maximum (minimum) (AGG_{MAX} , AGG_{MIN}). Summation is performed by convolution over the associated histograms \bar{P}_i^β (or symbolically, if applicable). The basic strategy for computation of MAX or MIN is to compute the maximum or minimum of the expected values $E(x_i)$. As an alternative approach, we could, e.g., consider the complete support interval I_i of each x_i , thus taking into account all realizations of involved uncertain values (with the negative effect of including possible outliers in the distribution).

Calculate $CLC(formula, attributes) = x^{result}$: We allow the use of custom operators to calculate a result value x^{result} from values of a set of input

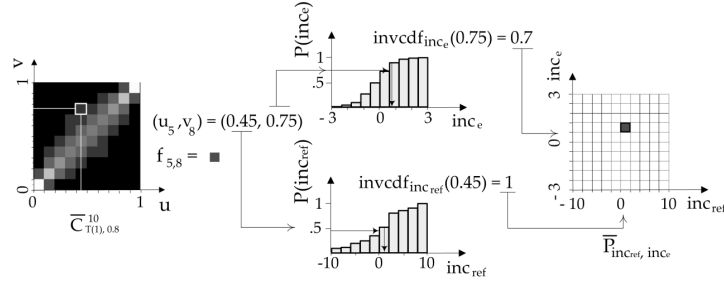


Fig. 3. Applying inversion to introduce correlation between inc_e and inc_{ref} based on an approximate copula representation (in this case using a copula $C_{Gauss,0.8}$)

attributes. For example, we can calculate the product of a key figure and a factor representing the prospected increase of this figure. This is applied to compute the revenue forecast rev_{new} for region R_{new} in T1.

Modification of Uncertainty Introducing and changing assumptions about developments, and thus, uncertainty in data, is a central task in the process of analyses for decision support. Therefore, we introduce the following operators allowing the modification of values and the introduction of correlation for creation of new scenarios.

Modify uncertainty $MOD(x_{old}, x_{new}) = x_{\Delta}$: We allow the modification of an uncertain value represented by x_{old} to a value x_{new} . We do not perform in-place modification but rather store the new value and a reference to the initial (modified) value. This form of modification management facilitates the traceability of modifications as well as comparison of scenarios derived through modifications in an analysis process. The histogram \bar{P}_{Δ} is computed as a delta (histogram) of \bar{P}_{new} to \bar{P}_{old} , i.e., we keep for each bin of \bar{P}_{old} the difference of density values to the respective bin of \bar{P}_{new} .

Introduce correlation $COR(x, y, H, d) = (x, y)$: This operator enables users to introduce a specified correlation between two distributions P_x and P_y . The approach is illustrated in Figure 3. Conceptually, the result of this operator is a bivariate distribution $P_{x,y}$ with P_x and P_y as marginal distributions and the specified correlation. Internally, we use a suitable ACR $\bar{C}_{H,d}$ to compute the concrete distribution $P_{x,y}$, represented by histogram $\bar{P}_{x,y}$. We do so by applying the inversion method to each of the bins $b_{i,j}$ of $\bar{C}_{H,d}$, considering each bin as a representative of the accumulated samples from the copula $C_{H,d}$. In the basic approach, we assume the center of $b_{i,j}$ as its coordinates (u_i, v_j) ; the associated frequency $f_{i,j}$ indicates the relative number of represented samples with those coordinates. Then, we apply the $invcdf$ functions of P_x and P_y to compute samples of the target joint

distribution function as $(invcdf_x(u_i), invcdf_y(v_j))$. Finally, we add the frequency $f_{i,j}$ to that bin of the resulting joint distribution histogram $\bar{P}_{x,y}$ to which $(invcdf_x(u_i), invcdf_y(v_j))$ belongs. For example, if we want to introduce a linear correlation of $d = 0.8$ between the distributions $P_{inc_{ref}}$ and P_{inc_e} , we use an ACR $\bar{C}_{Gauss,0.8}$. We apply the inversion method, accessing $\bar{P}_{inc_{ref}}$ and \bar{P}_{inc_e} to compute $(invcdf_{inc_{ref}}(u_i), invcdf_{inc_e}(v_j))$ as samples of the joint distribution P_{inc_{ref},inc_e} for each $b_{i,j}$ in $\bar{C}_{Gauss,0.8}$. To introduce a different structure of dependency, we could simply apply a different ACR, such as $\bar{C}_{T(1),0.8}$ which models the effect of highly dependent extreme values based on a bivariate T distribution with one degree of freedom. In a first implementation, we extended the described basic approach by drawing a number (relative to $f_{i,j}$) of uniformly distributed samples per bin $b_{i,j}$, rather than simply assigning $f_{i,j}$ to the inverted center of $b_{i,j}$. This approach proved to yield a much higher accuracy of the resulting joint distribution.

4 Implementation and Scenario Evaluation

We have implemented a prototype to evaluate the described data model and provide a basic operator set. The introduction of generic correlation other than linear correlation is subject to current work. A detailed description of the implementation is out of the scope of this paper; instead, we point out some of its central concepts to give a coarse understanding of how operators and their composition are implemented.

Architecture We extended an existing proprietary engine that computes complex analytical queries by means of so-called *Calculation Views* (CV). Those views enable OLAP analysis functionality as well as application of custom operations provided as Python or C++ implementations. Each CV can be queried like a base table or serve as input to further CVs. The topmost CV along with its ancestors forms a directed acyclic graph and is stored as a *Calculation Scenario*.

Model and Operator implementation In the graph introduced in Figure 1, the phase of what-if analysis is preceded by steps for loading data and analyzing selected data on an appropriate level of granularity. To implement those steps, we create CVs specifying the required input tables, attributes, aggregation functions, and filter conditions indicating the grouping, aggregation, and selected scope of the resulting CV. The described operators are implemented either as stacked CVs using only the native analysis functionality, or their computation is (partially) implemented by means of custom operator functions.

4.1 Scenario implementation

Revenue forecasting (T1) In Figure 4, we show the composition of operators implementing task T1. The final goal is to derive a forecast for the revenue rev_{new} in region R_{new} . As described, operators are composed by creating stacked

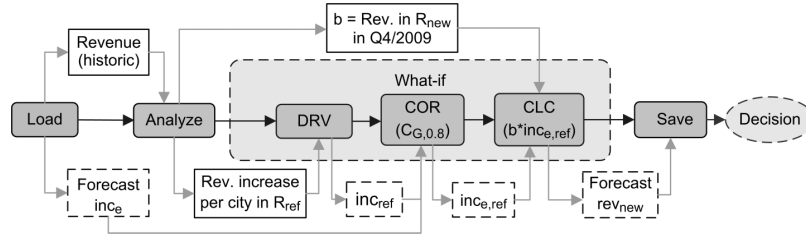


Fig. 4. Implementation of task T1 as an example of scenario creation in what-if analyses

views, their intermediate results being indicated in Figure 4 as input and output data items. In a first step, a user loads relevant historic revenue data as well as data reflecting the forecast economic growth inc_e . The user then views the revenue data on the level of quarterly revenue increases per city of R_{ref} . Based on this information, he derives inc_{ref} , which represents the distribution of the quarterly regional revenue increase for R_{ref} . To apply the assumption that a linear correlation of 0.8 holds between inc_e and inc_{ref} , the user applies *COR* to calculate a bi-dimensional distribution over those values. The revenue of quarter $Q4/2009$ for region R_{new} serves the user as a base value b for the forecast of rev_{new} , which is derived by applying *CLC* to increase b by the quarterly increase reflected by $inc_{e,ref}$. Finally, the expert stores the scenario for later reference.

Assuming alternative buying patterns The created scenario only represents *one* possible future development. The forecast (or further steps succeeding the forecast, for that matter) could yield different results if, e.g., a different correlation or different economic data was incorporated. For an alternative scenario, consider a situation where the expert wants to forecast revenues for the special class of high-end products. He could make the assumption that consumers' buying pattern for such goods is not linearly correlated with economic growth. Rather, given a very high economic growth, a particularly large part of consumers might be inclined to "treat themselves" to some luxurious products. Conversely, with a very strong economic decline, even those who earn well might avoid unnecessary spendings. Moderate changes of the economy, on the other hand, might have no directly visible effects on consumers' behavior. To consider such a case, a user must incorporate high dependencies of extreme values of the marginal distributions (i.e., those values close to the lower and upper bounds of I_{inc_e} and $I_{inc_{ref}}$, respectively). Therefore, in such a case the expert might apply an ACR representing, e.g., the copula $C_{T(1),0.8}$ rather than a Gauss-copula.

Product portfolio planning (T2) Task T2 also involves many assumptions and thus uncertainty. Its implementation is displayed in Figure 5. The user wants to evaluate which effect the launch of a product P_B at $t_{launchB}$ has on sales of product P_A . To do this, he first assumes that P_B is similar to a previously

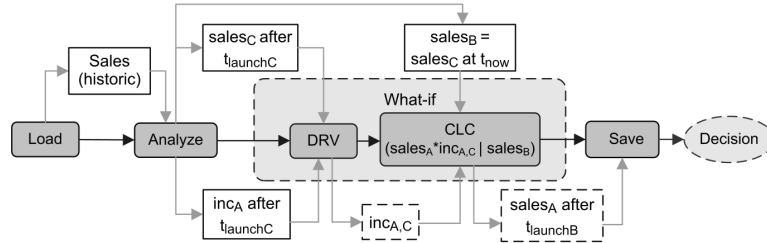


Fig. 5. Process graph depicting the implementation of T2

launched product P_C as regards their effect on $sales_A$. To provide the information representing this correlation, he derives a bivariate distribution from the sales of P_C ($sales_C$) and the sales increase (or decrease) of product P_A (inc_A) recorded for corresponding stores and times after the launch of P_C ($t_{launchC}$). Second, the expert assumes the current sales of P_C ($sales_C$) as initial value for $sales_B$. Third, he wants to evaluate different scenarios applying changes to the assumed value of $sales_B$. For simplicity, this is not shown in the figure. As example, imagine that the expert is positive that appropriate marketing measures will be taken to decrease the risk of very low sales numbers of product P_B . This assumption could be introduced by applying operator MOD to decrease the probabilities of low values in the distribution of $sales_B$.

5 Related Work

Uncertainty management has been an important field of database research during the last decade. Most approaches, such as the prominent TRIO [2] and MayBMS [4] systems, address the modeling and querying of discrete uncertainty relying on model extension approaches. Alternative tuples and attribute values are associated with a probability of occurrence and lineage information is recorded to enable correct handling of introduced dependencies. In recent developments, one can also observe growing support for continuous distributions, e.g., in the Orion 2.0 system [1] and in proposals described in [3]. We take a similar approach to [1], modeling continuous uncertainty in a generic way using histograms as an approximation. However, we additionally allow the independent representation of correlation information, while [1] only represents this information implicitly in multidimensional histograms over concrete value instances. An alternative approach to model extension is the sample-first approach employed, most prominently, by [6] which relies completely on Monte Carlo simulation. Here, stochastic models are implemented through variable generation functions, which are used to generate samples according to the assumed model. Queries are then evaluated over the samples. This approach is highly flexible and enables the computation of complex stochastic models. However, the models are completely encapsulated in generation functions and intermediate (uncertain)

results can not be retained for later use, as opposed to the histogram-based approach described in this paper.

The introduction and modification of uncertainty for what-if scenario creation have not received attention in most previous research. Most systems such as [2] focus on querying uncertain data but do not allow its flexible handling for forecasting and scenario analysis. The approach presented in [6] is situated in the decision support context and allows modeling and evaluation of scenarios, but as stated above relies completely on sampling while we represent uncertain values and assumptions (such as modifications and introduced correlation) as concrete artifacts which we can later reuse. The MayBMS [4] system includes a “repair-key” operator to provide for introduction of uncertainty to initially certain data, but addresses only the management of discrete probabilities on tuple level while we allow modifications over (histogram-based representations) of continuous uncertain values. Similar to our work, [10] addresses the special relevance of correlation in uncertain data. The authors describe an approach for representing tuple correlations using probabilistic graphical models and factored representations of joint probabilities. They redefine the relational operators to process factors during query evaluation. Their approach does, however, not address the separate representation and application of arbitrary correlations to independently represented continuous distributions. The system described in [1] can also represent joint distributions over several values, but does not enable the introduction of correlation information. Nor does any of the other approaches evaluated provide this feature we described as part of our work.

6 Conclusion and Future Work

In this paper, we addressed the modeling of uncertain data and correlation in such data with the special focus on enabling flexible what-if analysis in decision support. Our major objectives were the expressivity of the probabilistic data model as well as versatile functionality for analyzing and processing uncertain data. With focus on the iterative process of analysis and what-if scenario creation we discussed different characteristics of uncertainty. For the representation of uncertainty—and more specifically, for the support of *arbitrary* distributions—we described a generic histogram-based uncertainty representation. We further introduced a set of operators including functionality such as the derivation and analysis of uncertain values, the modification of such values, and the introduction of different correlation structures to previously uncorrelated values (i.e., values for which no correlation information is present in the database). The last aspect has not yet been addressed for uncertain data management, although experts’ knowledge or assumptions about correlation indeed can be important information when analyzing risks and chances of business decisions. Based on two example tasks, we illustrated the application and composition of our operators and explained special aspects of introduced assumptions. In particular, we showed where the introduction of correlation comes into the picture and

provided simple examples for applying different correlation structures, e.g., to evaluate alternative consumption patterns.

A first prototype implements most of the described features. Currently, the introduction of correlation is restricted to basic linear correlation. The implementation of the generic correlation introduction as well as an evaluation of its performance with respect to both run times and accuracy is subject of current work. Further, to support the iterative nature of the decision support process, we aim to provide a graphical visualization of (intermediate) results of applied operators in the analysis process. This way, we could present the user with an intuitive view of how information in a given scenario was derived or how values (more precisely, the distribution of those values) in alternative scenarios differ. Another interesting aspect would be the application of the discussed functionalities on real-world data (e.g., sales data). Such data will most probably exhibit more diverse distributions than artificially generated data and thus open up better possibilities to investigate the flexible analysis process.

References

1. Singh, S., Mayfield, C., Mittal, S., Prabhakar, S., Hambrusch, S., Shah, R.: Orion 2.0: Native Support for Uncertain Data. In: SIGMOD '08: Proceedings of the 2008 ACM SIGMOD, New York, NY, USA, ACM (2008) 1239–1242
2. Agrawal, P., Benjelloun, O., Sarma, A.D., Hayworth, C., Nabar, S., Sugihara, T., Widom, J.: Trio: A System for Data, Uncertainty, and Lineage. In: VLDB '06: Proceedings of the 32nd International Conference on Very Large Data Bases, VLDB Endowment (2006) 1151–1154
3. Agrawal, P., Widom, J.: Continuous Uncertainty in Trio. In: MUD, Stanford InfoLab (2009)
4. Huang, J., Antova, L., Koch, C., Olteanu, D.: MayBMS: A Probabilistic Database Management System. In: SIGMOD '09: Proceedings of the 35th SIGMOD International Conference on Management of Data, New York, NY, USA, ACM (2009) 1071–1074
5. Poosala, V., Haas, P.J., Ioannidis, Y.E., Shekita, E.J.: Improved Histograms for Selectivity Estimation of Range Predicates. SIGMOD Rec. **25**(2) (1996) 294–305
6. Jampani, R., Xu, F., Wu, M., Perez, L.L., Jermaine, C., Haas, P.J.: MCDB: A Monte Carlo Approach to Managing Uncertain Data. In: SIGMOD '08: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, New York, NY, USA, ACM (2008) 687–700
7. Sklar, A.: Fonctions de repartition à n dimensions et leurs marges. Publications de l'Institut de Statistique de L'Université de Paris **8** (1959) 229–231
8. Nelsen, R.B.: An Introduction to Copulas. Springer Series in Statistics. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2006)
9. Embrechts, P., Lindskog, F., McNeil, A.: Modelling Dependence with Copulas and Applications to Risk Management (2001)
10. Sen, P., Deshpande, A.: Representing and Querying Correlated Tuples in Probabilistic Databases. In: Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on. (2007) 596–605

