The first international VLDB workshop
on
Management of Uncertain Data

edited by
Ander de Keijzer, Maurice van Keulen
University of Twente
Alex Dekhtyar
University of Kentucky

CTIT Workshop Proceedings Series WP07-08

Sponsor

Centre for Telematics and Information Technology (CTIT)

Publication Details

Proceedings of the First VLDB workshop on Management of Uncertain Data
Edited by Ander de Keijzer, Maurice van Keulen and Alex Dekhtyar
Published by the Centre for Telematics and Information Technology (CTIT),
University of Twente
CTIT Workshop Proceedings Series WP07-08
ISSN 1574-0846

An electronic version of the proceedings is available at `http://mud.cs.utwente.nl/`
September 2007, Enschede, The Netherlands

Preface

After the Twente Data Management workshop on *Uncertainty in Databases* held at the university of Twente in June 2006, the speakers and participants expressed their wish for a workshop on the same topic colocated with a large, international conference. This Management of Uncertain Data workshop, colocated with the international conference on Very Large DataBases (VLDB) is the result of this wish.

We received 9 submissions from all over the world. Each of these submissions was reviewed by at least 3 different reviewers, resulting in 6 accepted papers for the workshop. In addition, we have 2 invited talks. The first talk *Combining Tuple and Attribute Uncertainty in Probabilistic Databases* by Lise Getoor from the University of Maryland, and the second talk *Supporting Probabilistic Data in Relational Databases* by Sunil Prabhakar from Purdue University.

We would like to thank the PC members for their effort in reviewing the papers and of course the authors of all submitted papers for their work. We also would like to thank the Centre for Telematics and Information Technology (CTIT) for sponsoring the proceedings. Last, but not least, we would like to thank the VLDB organizers for their support in organizing this workshop.

Ander de Keijzer
Maurice van Keulen
Alex Dekhtyar

Program co-chairs

Ander de Keijzer (University of Twente)
Maurice van Keulen (University of Twente)
Alex Dekhtyar (University of Kentucky)

Program Committee

Patrick Bosc (IRISA/ENSSAT, France)
Nilesh Dalvi (University of Washington, USA)
Alex Dekhtyar (University of Kentucky, USA)
Maarten Fokkinga (University of Twente, The Netherlands)
H.V. Jagadish (University of Michigan, USA)
Lise Getoor (University of Maryland, USA)
Ander de Keijzer (University of Twente, The Netherlands)
Maurice van Keulen (University of Twente, The Netherlands)
Laks V.S. Lakshmanan (University of British Columbia, Canada)
Thomas Lukasiewicz (Unversita di Roma La Sapienza, Italy)
Fatma Özcan (IBM Almaden, USA)
Gabriella Pasi (Univ. of Milano, Italy)
Giuseppe Psaila (University of Bergamo, Italy)
Olivier Pivert (IRISA/ENSSAT, France)
V.S. Subrahmanian (University of Maryland, USA)
Dan Suciu (University of Washington, USA)
Martin Theobald (Stanford University, USA)
Guy De Tré (Ghent University, Belgium)
Jef Wijsen (University of Mons-Hainaut, Belgium)
Vladimir Zadorozhny (University of Pittsburgh, USA)

Advisory Member
Jennifer Widom (Stanford University)

Workshop Program

Monday, September 24th, 2007
University of Vienna, Vienna, Austria

09.15 Opening
Ander de Keijzer, Maurice van Keulen, Alex Dekhtyar

09.30 **Session 1: Invited Talk**
Combining Tuple and Attribute Uncertainty in Probabilistic Databases
Lise Getoor

10.30-11.00 **Coffee break**

11.00 **Session 2: Applications of Uncertain Data**
Flexible matching of Ear Biometrics
Antoon Bronselaer, Joan De Winne, and Guy De Tré
Uncertainty in data integration: current approaches and open problems
Matteo Magnani and Danilo Montesi
A New Language and Architecture to Obtain Fuzzy Global Dependencies
Ramón Alberto Carrasco, Mari Amparo Vila, and Mari Ángeles Aguilar

12.30-14.30 **Lunch break**

14.30 **Session 3: Invited Talk**
Supporting Probabilistic Data in Relational Databases
Sunil Prabhakar

15.30 **Session 4: Querying Uncertain Data**
About the Processing of Division Queries Addressed to Possibilistic Databases
Patrick Bosc, Nadia Iben Hssaien, and Olivier Pivert

16.00-16.30 **Coffee break**

Session 4: Querying Uncertain Data
Consistent Joins Under Primary Key Constraints
Jef Wijsen
Making Aggregation Work in Uncertain and Probabilistic Databases
Raghotham Murthy and Jennifer Widom

17.30 **Discussion & Closing**

Table of Contents

Combining Tuple and Attribute Uncertainty in Probabilistic Databases

Lise Getoor

Computer Science Department
University of Maryland, College Park

There has been a long history of work in the database community on probabilistic databases. There is also a long tradition of work within the machine learning and reasoning under uncertainty communities on tractable factored representations of probability distributions, such as probabilistic graphical models. Recently, research from these two communities are starting to share more and more commonalities, as the probabilistic database work incorporates richer probabilistic dependencies and the machine learning work incorporates richer relational models.

In this talk, I will survey some of the recent work, including work on probabilistic relational models. I will show how these models can capture both tuple and attribute uncertainty, and discuss effective query methods. Joint work with Prithviraj Sen and Amol Deshpande.

Supporting Probabilistic Data in Relational Databases

Sunil Prabhakar

Department of Computer Science
Purdue University

Many applications domains are faced with the need to store and manipulate uncertain or imprecise data. Examples include sensor databases, data cleansing, scientific data, and information retrieval systems. Probabilistic modeling of this uncertainty is an attractive option for these applications. There is current interest in developing database management systems for uncertain data and several projects have begun to address this need.

In this talk we discuss some of the challenges and emerging solutions for supporting probabilistic data in relational databases. The challenges extend from the design of probabilistic relational models to performance, and user interfaces. Models are an essential first step with several design choices including the types of uncertainty handled (discrete, continuous, tuple, attribute, etc.) and semantics of the operators. Implementation issues include the choice of implementing the probabilistic support in the core or as an external wrapper. We will discuss some performance issues including indexing and optimization. The talk will draw upon experience with developing the Orion uncertain data management system.

Flexible matching of Ear Biometrics

Antoon Bronselaer[(1)], Joan De Winne[(2)], and Guy De Tré[(1)]

[(1)] Department of Telecommunications and Information Processing, Ghent University, Sint-Pietersnieuwstraat 41, B-9000 Gent, Belgium
[(2)] Disaster Victim Identification Team, Federal Police Belgium, Ruiterijlaan 2, B1040 Brussels, Belgium
Antoon.Bronselaer@ugent.be

Abstract. When identifying found bodies at the scene of a large-scale disaster, a technique for fast and cheap identification is very useful. Many techniques have been proposed in the past for this purpose. The method developed here allows flexible matching of ante mortem and post mortem pictures taken of the human ear and faces a challenge left unhandled before: the low quality control on the pictures of missing persons. As these pictures can be unsharp and out of profile, the comparison becomes more difficult and a flexible approach is required.
Keywords: Flexible Object Matching, Ear Biometrics

1 Introduction

When severe disasters occur, the Disaster Victim Identification (DVI) team, has the task of identifying the found bodies at the scene. In case of large scale disasters such as a tsunami or an earth quake, their mission becomes very time-critical. Further on, they often reside in countries where no DNA-databases or fingerprint databases are at hand. In such cases, the team is in need for a fast and accurate strategy to identify found bodies. To help them do this, a method is developed that allows such an identification based on pictures of the ears. Next to the scenario described in this paper, there are many other applications where identification techniques can be very usefull, for example in analyzing images of security cameras. The idea is that first, a database is filled with information of found bodies (Post Mortem Cases or for short PMC's). Next, a database with information on missing persons is filled (Ante Mortem Cases or for short AMC's). Finally, a flexible matching of AMC's and PMC's allows the identification of the found bodies. The biggest problem at hand is the poor quality of some AM pictures which are for example retrieved from relatives (e.g.: family pictures). In fact, there is no quality control of any kind at all. The main consequences are that the AM pictures are often unsharp (Figure 1) and out of profile. These two difficulties are the main challenges for our technique to cope with, but are already recognized in other techniques. In [8] a multiple identification method is presented and in [2] and [3], Voronoi diagrams are used. In both methods, the quality control is recognized as the major pitfall in ear identification with arbitrary pictures. The best known technique up to now is given in both [6]

and [7] which works good on noisy pictures. The remainder of the paper will be as follows. In section 2 the problem is described in detail. In section 3 a solution is given to this problem by describing the Ear Identification System (EIS). The system is discussed in 4 and some future work is discussed. Finally, some concluding remarks are given in section 5.

Fig. 1. Example of low quality picture of an ear.

2 Problem description

When a body is found, the DVI-team will take a picture of each ear. These pictures are labeled as follows:

$$PMXXX/EGR/YYYY$$

where XXX is a unique identification number for the found body, E signifies the ear (Left (L) or Right (R)), G signifies the gender (Female (F), Male (M) or Unknown (U)), R signifies the race (Caucasoid (C), Negroid (N), Mongoloid (M) or Unknown (U)) and $YYYY$ signifies the year when the picture was taken. PM signifies 'Post Mortem'. So an example for a label is:

$$PM001/LFN/2006$$

The labels combined with the two pictures present a Post Mortem Case (PMC). When a person is missing, the DVI-team will ask the relatives for pictures of the missing person's ears. These pictures are labeled as follows:

$$AMXXX/EGR/yyyy - YYYY/N$$

where AM signifies 'Ante Mortem', $yyyy$ signifies the birth year of the missing person and N signifies the number of AM-pictures received from the relatives. The other signs have the same meaning as with the PMC labels. So an example for a label is:

$$AM005/RMM/1985-2006/4$$

The labels combined with the pictures present an Ante Mortem Case (AMC). Considering the set of AMC's and PMC's, the challenge is now to match the AMC's to the correct PMC's, or otherwise to match the PMC's to the correct AMC's. The chosen direction of identification changes nothing to the technique and for the sake of simplicity, only the matching of AMC's to PMC's is considered. Hence, our problem can be stated as: *Given an AMC, find the PMC that refers to the same person.* The solution to this problem is described next.

3 Ear Identification System

We will now present the Ear Identification System (EIS) which is capable of matching a given AMC to a resulting set of PMC's, called R from now on. The structure of R is specified later on. As a founding for the methods described in this paper, a generic framework for the matching of objects is chosen [5]. The framework allows a generic comparison scheme to be tailored onto a given situation by specifying the configuration of the scheme. This paper focuses on the specification of this configuration to achieve an accurate comparison scheme.

3.1 Feature extraction

The first step towards comparison of cases is to somehow extract features from the pictures in each case. More specific, a finite and ordered list of points $L =< p_1, p_2, ..., p_n >$ with $p_i = (x_i, y_i)$, $i = 1, 2, .., n$, *uniquely* describing the ear on a picture, is required. Since it is impossible to prove that a certain set P uniquely describes a given ear, the number of points and the position of points is based on expert knowledge and former scientific work. Therefore, the exact number n and the position of the points is not specified here. They are considered as variables for the system. The only restriction is that the points need to be ordered to allow a point-to-point comparison. The question remains how to extract the points from the picture. This could be done manually through clicking the points or automatically, by means of an image processing algorithm. This choice does not poses any restriction on the system. It is assumed that points are extracted somehow from a (low quality) picture resulting in a list L. Once this is done, each picture is replaced in the case by it's corresponding list L. However, the automated and ordered extraction of the points is a complex problem [2].

3.2 Evaluation domain

The choice of the evaluation domain is very important with respect to the accuracy of the system. The choice here is influenced by two aspects. Firstly, the goal of EIS is to tell if two cases are describing the same person. According to [5], possibilistic truth values are preferred in this case because they allow elegant modeling of uncertainty about truth. Motivated by this, the evaluation domain

used within EIS is the domain of possibilistic truth values. Consequently, the result of evaluation operators and aggregation operators must be a possibilistic truth value, which is an extension of the classical two-valued boolean logic [4], [9]. A formal definition of a possibilistic truth value is given next.

Definition 1 (Possibilistic truth value)
With the understanding that $\tilde{\wp}(I)$ represents the set of all fuzzy sets defined over the universe $I = \{T, F\}$ and P represents the set of all propositions, the possibilistic truth value $\tilde{t}(p)$ of a proposition $p \in P$ is formally defined by means of the function $\tilde{t}$:

$$\tilde{t} : P \rightarrow \tilde{\wp}(I) : p \mapsto \tilde{t}(p)$$

which associates a fuzzy set $\tilde{t}(p)$ with each $p \in P$. This fuzzy set presents a possibility distribution, i.e. it's membership grades are interpreted as degrees of uncertainty:

$$\forall x \in I : \pi_{t(p)}(x) = \mu_{\tilde{t}(p)}(x)$$

which means that

$$\forall p \in P : \pi_{t(p)} = \tilde{t}(p)$$

where $t : P \rightarrow I$ is the mapping function which associates the value T with p if p is true and associates the value F with p otherwise. In conclusion, a possibilistic truth value is a fuzzy set with the following general form:

$$\tilde{t}(p) = \left\{ \left(T, \mu_{\tilde{t}(p)}(T)\right), \left(F, \mu_{\tilde{t}(p)}(F)\right) \right\}$$

According to definition 1, the possibilistic truth value $\tilde{t}(p)$ of a proposition $p \in P$ must be interpreted as follows:

$$Pos[t(p) = T] = \mu_{\tilde{t}(p)}(T)$$
$$Pos[t(p) = F] = \mu_{\tilde{t}(p)}(F)$$

When modeling uncertainty by use of a possibilistic truth value, it is assumed here that there always is one possibility completely possible. This means that in this work, the fuzzy set $\tilde{t}(p)$ is assumed to be normalised. If not, one should assume that there is *another* possibility, which we don't know, that is completely possible.

3.3 Evaluation operators

The comparison of two cases start with comparison of elementary aspects of the cases. These elementary aspects are called *attributes* of a case. There are four attributes to be compared:

- gender
- ear

– race
– list L of points describing the ear

For each of these attributes, an evaluation operator $E(.)$ is needed. The result of the evaluation operator must be a possibilistic truth value (PTV) expressing the possibility that the arguments are the same and the possibility that the arguments differ. The first evaluation operator described here is the evaluation operator $E_{ear}(e_1, e_2)$ for the comparison of the attribute 'Ear'. The value for this attribute is never unknown and the domain contains two values: $dom_{ear} = \{L, R\}$. Hence, the evaluation operator is the standard equality operator adjusted to the domain of PTV's.

$$E_{ear}(e_1, e_2) = \begin{cases} \{(T, 1)\}, e_1 = e_2 \\ \{(F, 1)\}, e_1 \neq e_2 \end{cases}$$

The second evaluation operator is the operator to compare the attribute 'gender'. Since the gender can be unknown, a model that incorporates uncertainty is adopted here. The domain for attribute 'gender', where M signifies 'male' and F signifies 'female', is $dom_{gender} = \{\{(M, 1)\}, \{(F, 1)\}, \{(M, 1), (F, 1)\}\}$. The element $\{(M, 1), (F, 1)\}$ models the case where the genders 'male' and 'female' are equally possible, hence representing the case where the gender is unknown. The domain consists of three elements which are *possibility distributions* defined over the set $\{F, M\}$. The first two distributions have membership degrees 0 for respectively F and M. For a formal definition of possibility distributions, the reader is referred to [11]. The evaluation operator for equality of possibility distributions defined here is based on the Extension Principle of Zadeh [10]. The generic operator is formally given by the following definition.

Definition 2 (Equality of possibility distributions)
With the understanding that $\tilde{\wp}(I)$ represents fuzzy power set defined over the universe $I = \{T, F\}$ and $\tilde{\wp}(U)$ represents the fuzzy power set defined over an arbitrary universe U, the equality operator for possibility distributions is defined as:

$$E_\pi : \tilde{\wp}(U) \times \tilde{\wp}(U) \rightarrow \tilde{\wp}(I) : \tilde{S}_1 \times \tilde{S}_2 \mapsto E_\pi(\tilde{S}_1, \tilde{S}_2)$$

where $E_\pi(\tilde{S}_1, \tilde{S}_2)$ is calculated by applying the extension principle of Zadeh [10] for equation:

$$\mu_{E_\pi(\tilde{S}_1, \tilde{S}_2)}(T) = \sup_{(x,y) \in \{(x,y) | \, (x,y) \in U \times U \wedge x = y\}} \left(\min(\mu_{\tilde{S}_1}(x), \mu_{\tilde{S}_2}(y))\right)$$

and

$$\mu_{E_\pi(\tilde{S}_1, \tilde{S}_2)}(F) = \sup_{(x,y) \in \{(x,y) | \, (x,y) \in U \times U \wedge x \neq y\}} \left(\min(\mu_{\tilde{S}_1}(x), \mu_{\tilde{S}_2}(y))\right)$$

This definition allows for a comparison of possibility distributions. The evaluation operator for 'gender' simplifies to:

$$E_{gender}(g_1, g_2) = \left\{(T, \mu_{gender}(T)), \left(F, \mu_{gender(F)}\right)\right\}$$

with

$$\mu_{gender}(T) = \max_{x \in \{F,M\}} (\min(\mu_{g_1}(x), \mu_{g_2}(x)))$$

and

$$\mu_{gender}(F) = \max_{x,y \in \{F,M\} \wedge x \neq y} (\min(\mu_{g_1}(x), \mu_{g_2}(y)))$$

The same considerations can be made for the attribute 'race'. We consider three basic *Races of craniofacial anthropology*: Negroid (N), Caucasoid (C) and Mongoloid (M). As domain of the attribute 'gender' the set of all possibility distributions over $\{C, N, M\}$ is considered. The evaluation operator for 'race' is again based on definition 2 which simplifies to:

$$E_{race}(r_1, r_2) = \{(T, \mu_{race}(T)), (F, \mu_{race}(F))\}$$

with

$$\mu_{race}(T) = \max_{x \in \{C,N,M\}} (\min(\mu_{r_1}(x), \mu_{r_2}(x)))$$

and

$$\mu_{race}(F) = \max_{x,y \in \{C,N,M\} \wedge x \neq y} (\min(\mu_{r_1}(x), \mu_{r_2}(y)))$$

The last and most complex evaluation operator is the operator for the attribute L: the list of points describing a human ear. The reason why an operator is considered for the whole list and not for individual points is a theoretical subtlety. There is a need for an operator that measures the similarity between two points. In [5] the best solution for expressing similarity between two objects/attributes is a similarity degree. However, the logical framework does not allow two different evaluation domains within the same comparison scheme. Consequently, we consider the operator $E_{points}(L_1, L_2)$ that compares two lists of points L_1 and L_2 and results in a PTV.

The comparison of L_1 and L_2, each of length l, starts by rescaling points of one list, so that points are matched on the same scale. Since it is possible that the pictures do not show the ear in good profile, a 3-dimensional affine transformation of the points is needed, which is represented by the 3×3-transformation matrix A:

$$\begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{pmatrix}$$

The transformation equations θ are given by the equation:

$$X = A.X'$$

where X is a reference point of L_1 and X' is a reference point of L_2, or otherwise. Since the points are two-dimensional, the elements c_1, c_2 and c_3 are not important and assigned the next values:

$$c_1 = 0,\ c_2 = 0,\ c_3 = 1$$

The other elements of A are determined by three reference points, resulting in a linear uniquely solvable system [1] of six equations and six variables. The theoretical linear system is presolved so that for three given reference points of both lists, A is uniquely determined. The question remains if there in fact are three points that are always unambiguously defined on the ear. We consulted the DVI-team with this question. The three points are (1) the tip of the tragus, (2) the intersection of the helix and antihelix the point as constructed on Figure 2. Once the

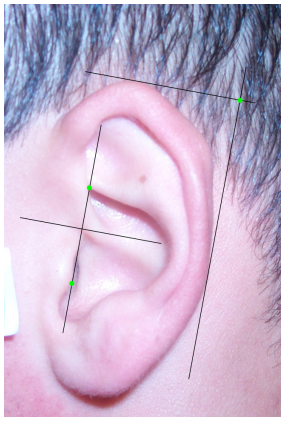

Fig. 2. Reference points of the transformation.

transformation matrix is known, points from one list can be transformed onto points of another list. The point-to-point transformation θ is given by:

$$\theta(X) = A.X$$

For further reasoning, it is assumed that list L_1 is preserved and list L_2 is transformed. Hence, the transformation operator $\Theta_{L_1}(L_2)$ will transform L_2 and the transformation matrix A depends on L_1. This is formally given by:

$$\Theta_{L_1}(L_2) : (\Re^3)^l \rightarrow (\Re^3)^l : [x, y, 1]^T \mapsto \theta([x, y, 1]^T)$$

[1] It is assumed that the reference points are not colinear in order to obtain a unique solution.

When the points are transformed, the next step is comparison of individual points. The similarity between two points used here is based on the 2D-gaussian distribution:

$$G = Ae^{\frac{(x-x_m)^2}{-2\sigma_x^2}} e^{\frac{(y-y_m)^2}{-2\sigma_y^2}}$$

where A is a scaling factor, σ_x and σ_y are the standard deviations and (x_m, y_m) is the mean of the distribution. An example of the distribution is shown in Figure 3. The similarity for two points $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$ (notice that the third

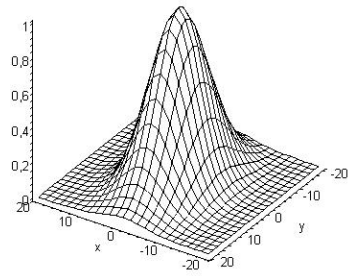

Fig. 3. 2D Gaussian distribution with $A = 1$, $\sigma_x = 5$, $\sigma_y = 8$ and mean $(0, 0)$

component of the points is skipped since the affine coordinates are normalised) is given by the formula:

$$sim_{\sigma_x,\sigma_y} = e^{\frac{(x_1-x_2)^2}{-2\sigma_x^2}} e^{\frac{(y_1-y_2)^2}{-2\sigma_y^2}}$$

This similarity measure is parameterised with the standard deviations σ_x and σ_y, which determine the $e^{-\frac{1}{2}}$ similarity interval. This is illustrated on Figure 4 where the grey zone represents the $e^{-\frac{1}{2}}$ similarity interval of the center of the ellipses. This means that points lying in this interval have a similarity with the mean of the distribution of more than $e^{-\frac{1}{2}} \approx 0.60653$. The combination of parameters can thus be interpreted as a degree of inaccuracy that is allowed in the assignment of the points. A larger value for the standard deviances implies a larger $e^{-\frac{1}{2}}$ similarity interval. This means that more inaccuracy is allowed and the restriction for two points to be similar is less severe. Since the scale of the picture is not predetermined, it is important that the parameters of the similarity measure are dynamically adapted. More precise, σ_x and σ_y are rescaled depending on the scale of the constellation in which points are compared. With linear rescaling the formula is:

$$\sigma = \sigma^*(1 - \frac{S^* - s}{S^*})$$

where s represents the scale of a given picture and σ^* and S^* represent the parameter value and scale of a referential picture. When the points are compared

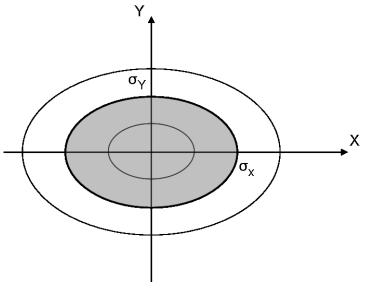

Fig. 4. Projected similarity intervals in XY-plane

one on one, the combination of the results is the next step. This can be done in different ways. Possibly choices are the (weighted) average operator or an arbitrary t-norm. All these operators result in a similarity degree d in the unity interval $[0, 1]$. The results can be combined by taking the (weighted) average of the individual similarity degrees. This results in the following intermediary evaluation operator for lists of points:

$$\epsilon_{points}(L_1, L_2) = \frac{1}{l} \sum_{i=1}^{l} e^{\frac{L_1[i].x - \Theta_{L_1}(L_2)[i].x}{-2\sigma_x^2}} * e^{\frac{L_1[i].y - \Theta_{L_1}(L_2)[i].y}{-2\sigma_y^2}}$$

where $L[i].x$ signifies the abscis of the i-th point of L and $L[i].y$ signifies the ordinate of the i-th point of L. Notice that a weighting average offers the modelling of uncertainty on the level of point assignment, which may be caused by the fact that the point is hard to identify. The parameter scaling is assumed to be implicit here. The result of $\epsilon_{points}(L_1, L_2)$ is a similarity degree. However, the evaluation operator should result in a PTV. This considerations results in the final evaluation operator for lists of points:

$$E_{points}(L_1, L_2) =$$
$$\left\{ \left(T, \frac{\epsilon_{points}(L_1,L_2)}{\max(\epsilon_{points}(L_1,L_2), 1-\epsilon_{points}(L_1,L_2))} \right), \left(F, \frac{1-\epsilon_{points}(L_1,L_2)}{\max(\epsilon_{points}(L_1,L_2), 1-\epsilon_{points}(L_1,L_2))} \right) \right\}$$

3.4 Aggregation operators

In the previous section, four evaluation operators are proposed, all resulting in a possibilistic truth value. The second step of the system is to aggregate these results into a general result. Given that *all* the attributes should be satisfied in order to conclude a positive identification, conjunctive behaviour of the aggregator is needed here. A weighted conjunction operator for PTV's will be introduced here based on a (t-norm,t-conorm)-pair. Which pair is best used for optimal identification is currently under research. The required operator is explained step by step. The first step induces a dynamic calculation of weights for the four basic evaluations. This mechanism determines the weights depending

on the outcome of the evaluations [1]. Since these outcomes are all PTV's, they represent the possibilities that the outcome is true or false. In order to model this dependence, a weight point (w_t, w_f) in a two dimensional weight space is predetermined for each evaluation. This weight point reflects the importance of the evaluation in case the outcome is true (w_t) and in case the outcome is false (w_f). Next to these weights this mechanism needs a measure for the degree to which an evaluation is *certainly* true or false. An appropriate measure is the *necessity* measure for possibility distributions, which in case of PTV's simplifies to:

$$Nec(E = T) = 1 - \mu_E(F)$$
$$Nec(E = F) = 1 - \mu_E(T)$$

where E represents the outcome of an evaluation operator, hence a PTV. The weight w_i^* for the evaluation with outcome E is given by:

$$w^* = w_t * (1 - \mu_E(F)) + w_f * (1 - \mu_E(T))$$

Once the weights are calculated, an operator is needed to model the impact of the weight on the outcomes of the evaluations. For this purpose, an implicator for PTV's is used. The implicator $\tilde{\Rightarrow}_f$ should only be used in combination with a conjunction operator and is defined as:

Definition 3 (Implication operator for combination with conjunction) *Assume a fuzzy implicator $\Rightarrow_f$ is given and that $\tilde{\wp}(I)$ is the fuzzy powerset of $I = \{T, F\}$. The generic implication operator for combination with a conjunction operator is defined as:*

$$\tilde{\Rightarrow}_f : [0,1] \times \tilde{\wp}(I) \to \tilde{\wp}(I) : (w, \tilde{V}) \mapsto \tilde{\Rightarrow}_f\ (w, \tilde{V})$$

where:

$$\mu_{\tilde{\Rightarrow}_f(w,\tilde{V})}(T) = w \Rightarrow_f \mu_{\tilde{V}}(T)$$

and

$$\mu_{\tilde{\Rightarrow}_f(w,\tilde{V})}(F) = \neg\,(w \Rightarrow_f \neg(\mu_{\tilde{V}}(F)))$$

Next to the impact of weight, a conjunction operator for PTV's is required in order to combine PTV's. An appropriate operator could be found in the application of the extension principle of Zadeh. However, a more simple approach is preferred here. The conjunction operator used is a direct extension of a given t-norm t.

Definition 4 (Conjunction for possibilistic truth values) *Assume a generic t-norm t and a t-conorm s is given and that $\tilde{\wp}(I)$ is the fuzzy powerset of $I = \{T, F\}$. The conjunction operator $\tilde{\wedge}_t$ is formally defined as:*

$$\tilde{\wedge}_t : \tilde{\wp}(I) \times \tilde{\wp}(I) \to \tilde{\wp}(I) : (\tilde{U}, \tilde{V}) \mapsto \tilde{U} \tilde{\wedge}_t \tilde{V}$$

with

$$\mu_{\tilde{U}\tilde{\wedge}_t\tilde{V}}(T) = t(\mu_{\tilde{U}}(T), \mu_{\tilde{V}}(T))$$

and

$$\mu_{\tilde{U}\tilde{\wedge}_t\tilde{V}}(F) = s(\mu_{\tilde{U}}(F), \mu_{\tilde{V}}(F))$$

All the theoretical pieces are now available to define the weighted conjunction operator that is used within the EIS system.

Definition 5 (Weighted conjunction for possibilistic truth values) *Assume a implicator for conjunction $\overset{\wedge}{\Rightarrow}_f$, a conjunction operator for PTV's $\tilde{\wedge}_t$ and let $\tilde{\wp}(I)$ be the fuzzy powerset of $I = \{T, F\}$. The weigthed conjunction operator is then formally given by:*

$$\wedge_t^w : ([0,1] \times \tilde{\wp}(I))^2 \to [0,1] \times \tilde{\wp}(I) :$$
$$((w_1, \tilde{V}_1), (w_2, \tilde{V}_2)) \mapsto (\max(w_1, w_2), \overset{\wedge}{\Rightarrow}_f (w_1, V_1) \tilde{\wedge}_t \overset{\wedge}{\Rightarrow}_f (w_2, V_2))$$

Definition 5 is the operator used in the EIS system to combine the outcome of all evaluations into a single PTV representing the result of the match.

3.5 Result set R

When the results are known, the best matching PMC's need to be identified. For this purpose an ordering function on PTV's is needed. The ordering function o considered here is:

$$o : \tilde{\wp}(I) \to \Re : \{(T, \mu_T), (F, \mu_F)\} \mapsto \frac{\mu_T + (1 - \mu_F)}{2}$$

This function can be applied on each PTV that is the result of a case-to-case match. Next, the ordering on the set of real numbers is used to identify the best matching cases. There are several strategies for the decision of which cases are returned. A first possibility is a threshold τ on the outcome of the ordering function o. Consequently, a PM-case with the PTV p as result of the match will be added to R if:

$$o(p) > \tau$$

Another strategy is a top-n result set, meaning that the n best matches will be represented in R. The precise strategy is of low theoretical importance and is quite user-dependent.

4 Discussion and future work

A first implementation of the system based on the probabilistic t-norm/t-conorm pair has been achieved. A first small test, being the matching of twenty cases yielded an accuracy of 65%. In 90% of the cases the correct case was retrieved within the top three. The main accuracy problem is the bias in the decision criteria caused by the most general affine transformation. However, this is an issue at the level of feature extraction while the system focusses on intelligent matching. Due to this, the feature extraction system must be expanded first

to cope with cases where perspective is a problem. This requires a number of restriction posed over the affine transformation used. This is immediatly the main reason why the test scope is limited in this work.

The current system implements the theoretical framework described in [5] for identification of humans. This identification is based mainly on the biometrics of the ear. However, next to the biometrics of the ear, there are several non-biometric features that can assist to a better identification. Adding extra features for which the value can easily be given (like gender and race) narrows down the search space very quickly. The automated combination of biometric and non-biometric data offers a very fast and cheap strategy in identification processes.

As for the biometrics matching the solution presented here uses a general affine transformation to rescale sets of points. Because there is no control on the angle from which the picture is taken, nor the distance from which it is taken, this affine transformation might bias the similarity measure on points. Hence, the technique as it is now is invariant to rotation, scaling and translation. It allows for minor variations in the perspective but as this variation becomes large, the matching process becomes tedious. For this purpose a dynamic method may be used to assign the points incalculating the perspective to some extent. However, better methods might be available and this should be the base for research in the future. Following this direction, it might be usefull to extend the system towards the matching of curves describing the biometrics. However, it should be tested if these methods work well when large amounts of noise are present.

5 Conclusions

We have proposed a method to identify victims of severe disasters quick and easily, based on (low quality) pictures of the victim's ears. The method offers flexibele comparison of *cases*, where a case consist of information drawn from the picture of the ear, gender, race and the side of the ear. The comparison of cases includes two major steps: *evaluation* of the main attributes of two cases and *aggregation* of the results of the evaluations. These two steps are theoretically guided by an underlying mathematical framework proposed in [5]. We defined appropriate evaluation and aggregation operators to implement an identification system.

References

1. A. Bronselaer, G. De Tré, A. Hallez. Dynamic preference modelling in flexible object matching. In *Proceedings of the EuroFuse workshop on New Trends in Preference modelling*, 2007.
2. Burge, M. and Burger, W. Ear Biometrics. In *A. Jain R. Bolle and S. Pankanti, editors, BIOMETRICS: Personal Identification in a Networked Society*, pp. 273-286. Kluwer Academic, 1998.
3. Burge, M. and Burger, W. Ear Biometrics in Computer Vision. In *15th International Conference of Pattern Recognition*, ICPR 2000, pp. 826-830.

4. G. De Tré. Extended Possibilistic Truth Values. In *International Journal of Intelligent Systems*, vol. 17, 427-446, 2002.
5. A. Hallez. 'A Hierarchical approach to object comparison'. In *Proceedings of IFSA 2007 World Congres*, 2007.
6. Hurley, D.J., Nixon, M. S., Carter, J.N. Automated Ear Recognition by Force Field Transformations. In *Proceedings IEE Colloquium: Visual Biometris* (00/018), 2000, pp. 8/1-8/5.
7. Hurley, D.J., Nixon, M.S., Carter, J.N. A New Force Field Transform for Ear and Face Recognition. In *Proceedings of the IEEE 2000 International Conference on Image Processing ICIP* 2000b, pp. 25-28.
8. Moreno, B., Sánchez, Á., Vélez. J.F. On the Use of Outer Ear Images for Personal Identification in Security Applications. In *IEEE 33rd Annual International Carnahan Conference on Security Technology*, 1999, pp. 469-476.
9. H. Prade. Possibility sets, fuzzy sets and their relation to Lukasiewicz logic. In *Proc 12th Int Symp on Multiple-Valued Logic*,223-227, 1982.
10. L.A. Zadeh. "The concept of a linguistic variable and its application to approximate reasoning I". In *Information Sciences*, vol. 8, 199-251, 1975
11. L.A. Zadeh. "Fuzzy sets as a basis for a theory of possibility". In *Fuzzy sets and Systems*, vol. 100, 9-34, 1999

Uncertainty in data integration: current approaches and open problems*

Matteo Magnani[1] and Danilo Montesi[2]

[1] University of Bologna, Italy,
matteo.magnani@cs.unibo.it
[2] University of Bologna, Italy,
danilo.montesi@unibo.it

Abstract. Uncertainty is an intrinsic feature of automatic and semi-automatic data integration processes. Although many solutions have been proposed to reduce uncertainty, if we do not explicitly represent and keep it up to the end of the integration process we risk to lose relevant information, and to produce misleading results. Models for uncertain data can then be used to represent integrated data sources resulting from uncertain data integration processes. In this paper we present a survey of existing approaches directly dealing with uncertainty in data integration, define a generic data integration process that explicitly represents uncertainty during all its steps, and present some preliminary results and open issues in the field.

Keywords: Uncertainty, Data integration, Data models

1 Introduction

This paper concerns the management of uncertainty in the field of data integration. We introduce the problem defining a generic probabilistic data integration process, present the state of the art, give some original contributions and outline open issues. In addition, we discuss the relationships between data integration and models for uncertain data.

Information integration is the general process of producing a single information source out of some local information sources [1–13]. Many studies regard structured information sources, for which we can define two sub-problems: schema integration and instance integration. The term *data integration* is often used to refer to information integration applied to structured data (both schema and instances), and we will use it in this acception in the following.

While uncertainty is unavoidable in data integration, and the majority of existing methods and systems deal with uncertain information, specific studies on this topic have emerged only recently, and to the best of our knowledge

* This work has been supported by projects Prin 2005 "Middleware basato su Java per la fornitura di servizi interattivi di TV digitale" and CIPE 4/2004 "Innovazione e centri di ricerca nelle Marche".

they focus on specific tasks of the data integration process, without describing how to use uncertain information outside those tasks. In fact, many existing methods remove the uncertainty at some point, through defuzzification or a (semi-)automatic choice of the most likely outcomes.

To make an analogy with the field of data modeling, database theory was initially developed with a crisp-and-closed-world assumption, and early data models could not represent uncertain information. However, it soon became clear that uncertain information could be valuable, and sometimes more useful than missing information. This leaded to the study of several formalisms, from null values and C-tables to probabilistic and possibilistic data models.

In the field of information integration we are experiencing a similar process. In a 2003 survey paper about data integration, the problem of uncertain data management was not mentioned, and the main difficulty was to identify correct semantic relationships between schema objects, i.e., to remove uncertainty about the relationships [14]. Later, on another survey paper, the problem of dealing with *imprecise mappings* was mentioned, without explicitly referring to uncertainty management [10]. However, it was recognized that we will never be able to find all correct matches between all schema objects we compare, and we must therefore be aware of possible errors and find ways to use partially incorrect results. In a recent survey paper by the same author, uncertainty management has been explicitly indicated as one of the future challenges in the field [11].

In the next section we define a **generic data integration process explicitly representing uncertainty**, and indicate where the management of uncertainty is more critical, pointing out the difficulties in implementing this approach. As we will see, there are both computational problems and representation problems, i.e., information that is difficult to formalize. This process extends existing methods that at some point during the integration activity transform uncertain information into exact one, and provides a common general context for further research. Then, we present some **preliminary results** we have obtained trying to tackle the aforementioned problems. In particular, we show that top-K mappings can be used to increase the recall of a data integration process, justifying the usage of less probable outcomes. However, we also see that cutting low-probability mappings may result in the loss of correct information. In addition, we show that probabilistic dependencies can provide information on how to improve the schema matching phase. Section 5 presents an up-to-date **survey** of existing data integration methods that explicitly deal with uncertainty. Finally, we conclude the paper with a brief discussion of **open issues**.

2 A probabilistic data integration approach

In Figure 1 we have represented the tasks composing a general data integration process, with 1:1 mappings and two input data sources, like in [13]. This is made of three main tasks: wrapping, matching and merging. First the data sources are translated into homogeneous data models (**wrapping**), that allow

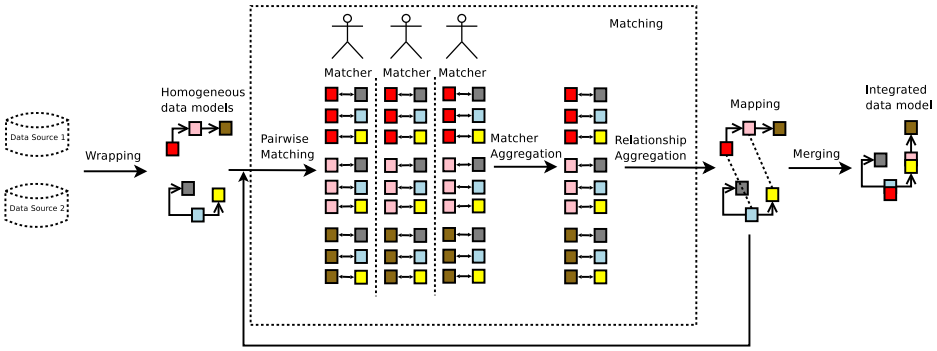

Fig. 1. A generic data integration process

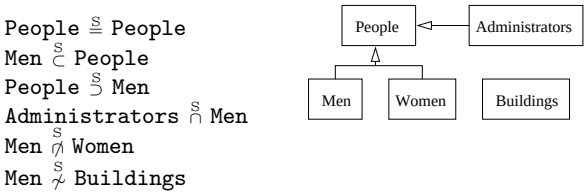

Fig. 2. A visual description of some possible semantic relationships between two entities. In this figure we have considered equivalence ($\overset{S}{\equiv}$), subset-subsumption ($\overset{S}{\subset}$), superset-subsumption ($\overset{S}{\supset}$), overlapping ($\overset{S}{\cap}$), disjointness ($\overset{S}{\oslash}$), and incompatibility ($\overset{S}{\not\sim}$). However, the majority of systems uses only two relationships, i.e., *match* and *not match* (M and $\not{M}$)

the comparison of otherwise heterogeneous data models with different information representation constructs. For now, we can model these information sources as sets of schema objects $S_1 = \{O_{11}, \ldots, O_{1M}\}$ and $S_2 = \{O_{21}, \ldots, O_{2N}\}$. Then, each object from the first data source is compared with all or some of the objects of the second (**matching**). The objective of the matching phase is to find one or more candidate mappings between S_1 and S_2. A *mapping* associates each pair of objects from the two input data sources to a semantic relationship[3], like the ones represented in Figure 2.

Definition 1. *Let R be a set of mutually exclusive semantic relationships, and S_1 and S_2 two sets of schema objects. A mapping is a function $m : S_1 \times S_2 \rightarrow R$.*

An *uncertain mapping* between data sources S_1 and S_2 is a probability distribution over the set $M(S_1, S_2)$ of all mappings between them. Finally, the discovered mapping is used to merge the data sources, i.e., to generate the integrated database (**merging**). Before merging, many methods implement user feedback cycles or other learning approaches to improve the quality of the mapping. The objective of these technique is to remove the uncertainty on the mapping, but in many cases we can only reduce it, and must find ways to manage it during the rest of the data integration process. The merging phase depends on the set R of semantic relationships. Many approaches look for two simple kinds of relationships (*match* and *not match*), put together matching objects, and evaluate a data mapping between them (also called record linkage, deduplication, and other names from different fields). In our vision, the result of the merging phase should be an uncertain data model, to provide to the final users/applications a complete view of the result of the integration. In particular, this will be necessary in all applications where human intervention is difficult or impossible to perform, like in peer-to-peer data integration/management, and in our opinion should be also used after human intervention to represent all the information for which it has not been possible to identify the correct relationships for sure. As a last consideration, there is still an assumption that we need to relax: in the process described so far, the input data sources were traditional databases. However, we can think of not only producing an uncertain data model, but also starting from uncertain data models. This assures the closure property of the process we are defining, that can be applied iteratively also to data sources obtained as integration of others, to merge more than two input databases.

To identify the critical points of the process, we must expand the most difficult task, i.e., the matching phase, to see how it is organized in existing methods. Modern systems do not use a single monolithic matcher, but a pool of matchers, each with specific expertise on some properties of the analyzed objects, like type, name or structure. After all matchers have compared the required objects, for each pair of objects their outcomes are combined (**matcher aggregation**) to

[3] *Semantic* relationships are not relationships between data instances, like strings or numbers, but between the real world objects they represent. This is one of the main causes of uncertainty in automatic schema matching. In the following we will always consider *semantic* relationships

produce a single relationship. If we do not need to represent uncertainty, the result of a matcher M comparing two objects O_1 and O_2 can be modeled by a tuple $\langle \mathrm{O}_1, \mathrm{O}_2, r \rangle$, where $\mathrm{r} \in R$. For example, the relationship between schema objects `S1.Country` and `S2.State` can be modeled as $\langle$`Country`, `State`, $\overset{\mathrm{S}}{\cap}\rangle$, stating that they overlap each other. To represent the inherent uncertainty of automatic schema matching algorithms we are going to use the following extended definition, that we call *probabilistic uncertain semantic relationship* (pUSR).

Definition 2 (pUSR). *A* probabilistic uncertain semantic relationship *between two objects O_1 and O_2 is a tuple $\langle \mathrm{O}_1, \mathrm{O}_2, R, P \rangle$, where R is a set of mutually exclusive relationships and P is a probability distribution over R.*

As an example, we can substitute the aforementioned relationship between `Country` and `State` with the pUSR $\langle$`Country`, `State`, $\{\overset{\mathrm{S}}{=}, \overset{\mathrm{S}}{\subset}, \overset{\mathrm{S}}{\supset}, \overset{\mathrm{S}}{\cap}, \overset{\mathrm{S}}{\not\cap}, \overset{\mathrm{S}}{\not\sim}\}$, $P_{\mathrm{ex}}\rangle$, where $P_{\mathrm{ex}}(\overset{\mathrm{S}}{\cap}) = .8$, $P_{\mathrm{ex}}(\overset{\mathrm{S}}{=}) = .2$ and $P_{\mathrm{ex}}(\overset{\mathrm{S}}{\subset}) = P_{\mathrm{ex}}(\overset{\mathrm{S}}{\supset}) = P_{\mathrm{ex}}(\overset{\mathrm{S}}{\not\cap}) = P_{\mathrm{ex}}(\overset{\mathrm{S}}{\not\sim}) = 0$. This means that we are no longer certain that $\overset{\mathrm{S}}{\cap}$ is the correct relationship, assigning a probability of .8 to it and a probability of .2 to the alternative hypothesis $\overset{\mathrm{S}}{=}$.

The second step of the matching phase is the production of the mapping (**relationship aggregation**), which corresponds to a combination of all the identified uncertain semantic relationships and the production of an uncertain mapping, as previously defined. The **merging** phase depends on the data modeling formalism and on the set of semantic relationships used during the matching phase.

2.1 Critical points in the uncertain data integration process

The critical points of a probabilistic data integration approach are the same as in many other applications of probability theory: probabilities must be *produced* and then *aggregated*. In particular, each matcher must return probabilities that can be compared and aggregated with the ones produced by other matchers. It should be therefore clear which interpretation of the theory is used inside each matcher, e.g., classical, frequency or subjective, and also if different probability distributions can be aggregated as they are. For example, assume a matcher finds some common instances inside two schema objects, supporting a *match* relationship, and another matcher thinks that the names of the two objects are not related, supporting a *not match*. How much probability mass should we assign to the two hypotheses? Surely we can tune these values after some experiments, but this is very different from having an underlying theory — which is the main reason to use probability theory.

Even if the matchers produce probabilities that can be aggregated, the aggregation itself is problematic. The first aggregation of probabilities concerns the outcomes of different matchers about the same pair of schema objects. Some matchers may be independent, meaning that we can change the features analyzed by one of them without affecting the outcome of the other. As an example, consider two matchers comparing respectively the number of instances in a schema

object and its name. If you change the name, but not the cardinality, the outcome of the cardinality matcher will not change, and vice versa. However, in general different matchers can be interdependent. As a consequence, the method that performs the combination needs to know the pUSRs it must merge *and* additional information about their dependencies.

The second aggregation of probabilities is problematic as well. Also in this case there can be dependencies between different pUSRs. For example, the matchers can believe that schema objects A and B are equivalent, that schema objects B and C are equivalent as well, but A and C are incompatible (A $\overset{S}{\equiv}$ B, B $\overset{S}{\equiv}$ C, C $\overset{S}{\not\equiv}$ A). Evidently, this is not possible: the probability of this mapping would be 0, and not the product of the probabilities locally assigned to the three relationships. Depending on the set of semantic relationships under consideration, some relationships between different pairs of schema objects can be mutually exclusive, referring to common schema objects, and there can also be other dependencies to be studied.

3 Preliminary results

The first result we present is an empirical evaluation of the effect of keeping (or not) uncertainty in the mapping produced by the matching phase. Figure 3 illustrates two portions of schemata obtained as views over a database containing political and geographical information about our Planet [15], that we use in the following experiments. These schemata contain many schema objects that are difficult to be matched. For example, the two tables `Country` and `State` contain some common information, despite their different names. The two tables named `Organization` are exactly the same, while one of the two tables called `City` refers only to Country capitals, despite their common name. Then, there are tables that are completely unrelated, like `Lake` and `Language`, although they share some column names. Even if we look at the instances (values) contained inside some columns, we see that automatic matchers could make the wrong decision. For example, both columns `area` and `population` contain integers, and probably even similar values.

In our experiments we compared subsets of the input schemata using the approach described in the previous section, and obtaining an uncertain mapping. Then, we extracted and analyzed the top-K (K=50) mappings. When we consider the most likely mapping, we retrieve $N_1 = m \cdot n$ relationships, where m and n are the sizes of the two input schemata. Of these, only N_1^t will be correct. If we consider the first two most likely mappings, they will contain N_2 mappings, of which only N_2^t correct. We will therefore evaluate the precision ($\frac{N_i^t}{N_i}$) and recall ($\frac{N_i^t}{N_1}$) of the uncertain mapping varying i, to see if using less likely mappings we can increase the recall without significantly decreasing the precision. If this is the case, we will know that considering uncertain information may generate better results. Similarly, we will check the maximum recall we obtain considering only a bounded number of mappings, i.e., not using all the uncertain information we

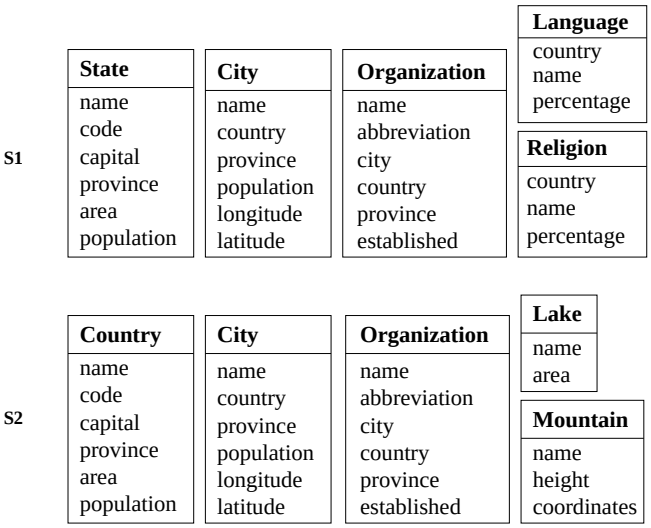

Fig. 3. Two schemata S1 and S2, containing geographical and political data

collected during the matching phase. This to show that even a top-K approach may lose some valuable information.

The matchers used in our experiments analyze the following features of the schema objects: name, data type, number of instances, value of instances, statistics on the values of instances. In addition, in the first set of tests we used only *match* and *not-match* relationships, with which there cannot be mutually exclusive semantic relationships and we can compute the top-K mappings in time linear to the number of pairs of schema objects. As the objective of this paper is not to describe a prototype and compare it with existing software, we do not provide the algorithms used by our matchers to compare the local databases.

In the following we perform a qualitative and a quantitative analyses. We will first describe some relevant examples manually chosen from the results of our tests, where the matchers failed in identifying the correct relationship in the top-1 mapping. These examples are very important, because they justify the discovery of additional mappings: the correct relationships can in fact be found in subsequent results produced by our method. Then, we will compute the increase in the recall of correct relationships, and study how this affects the precision of the integration. This analysis is partial, because we do not consider the probabilistic information associated to the mappings, but only their ordering. However, as we have already mentioned, this would be useful to compare our prototype with other methods, which is not an objective of this paper. In addition, one of the open problems highlighted in this paper concerns the representation and aggregation of probabilities, making the analysis of the probability of our results too preliminary.

We will now examine some relationships that have *not* been correctly identified in the first mapping retrieved by the matchers — we remind the reader that Figure 3 illustrates only part of the input schemata, for space reasons. In particular, we consider the relationships between a column `Country` in S1, referring to non-American Countries, and other columns of S2. The first case is the comparison with a table (primary key) named `Politics`. Here, the correct relationship is a *match*, because this table contains one row for each non-European country, and uses the code of the country as its primary key. However the Name matcher supports the *not match* relationship. The correct relationship is identified after a few mappings, according to the different opinion of the Instance matcher. This could make us think of considering instances more than other information. However, even if a detailed study of expert weighting is outside the scope of this paper, giving too much responsibility to the comparison of instances is not necessarily a good idea. In fact, we can be unlucky in the choice of the samples, or there can be unrelated instances with the same string representation, as it happens in the two columns `Country.Code` and `Organization.Abbreviation`.

Finally, as a more tricky example, consider a comparison between columns `Country` and `Population`. Here, the Instance matcher correctly identifies that the two columns do *not match*. However, the Name matcher finds that the two names have a common hypernym — both a country and a population are groups of people. Unfortunately, this is not the intended meaning of the columns named `Population` in our test databases, and the resulting relationship is wrong.

Also from a quantitative point of view the results we obtained with a limited number of relationships (177) to find are very good, and have been represented in Figure 4. In fact, we have been able to find all correct relationships, with an increase of 4% in the recall of the matcher, paying only a decrease of less than 1% in its precision. Also on a larger input (885 pairs of schema objects) we could increase the recall of a similar percentage. However, we could not identify all correct relationships.

In summary, the results obtained with these experiments show that it is worth keeping uncertainty after the matching phase, and that on complex integration problems the information loss caused by the removal of uncertainty becomes more relevant.

When we use 6 semantic relationships, different pUSRs are no longer independent. As an example, consider the pairs of schema objects `Borders.Country1`, `Economy.Country`, `State.Code`. During the schema analysis phase, the following relationships could be identified[4]:

- `Borders.Country1` $\overset{S}{\subset}$ `Economy.Country`
- `Borders.Country1` $\overset{S}{\cap}$ `State.Code`
- `Economy.Country` $\overset{S}{\subset}$ `State.Code`

Unfortunately, the relationship between schema objects `Borders.Country1` and `Economy.Country` is wrong, because the two objects overlap each other. In fact,

[4] To be more precise, one of the relationships is extracted from one of the input schemata, because two of the objects belong to the same schema. However, this is not relevant to our discussion

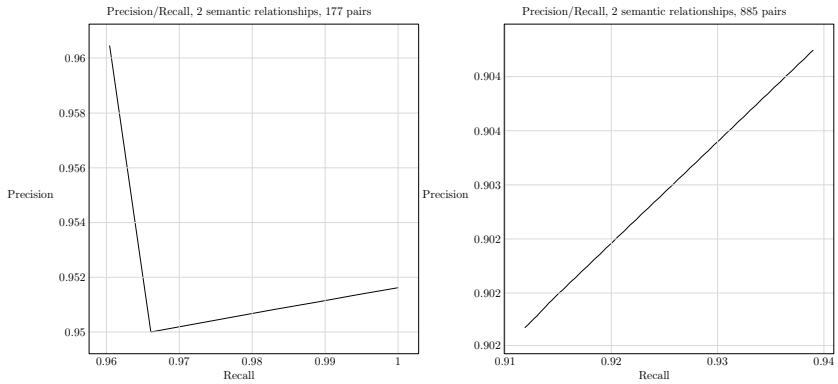

Fig. 4. Precision/Recall curve using two (match/not-match) relationships and 177/885 pairs of schema objects

the Instance matcher must have chosen a sample from `Borders.Country1` whose elements belonged to `Economy.Country`. However, this matcher was also aware that it was analyzing only a fraction of the instances, therefore it assigned some (smaller) probability to *overlapping* ($\overset{S}{\cap}$), depending on the relative size of the sample.

Now consider again the mappings ordered by their likelihood with an independence assumption (which we have already seen to be wrong in general). If unsatisfiable mappings have probability 0, they do not affect the ranking and the probability of the other discovered mappings. However, if there is some probability assigned to them, this is wrong, and should be decreased to 0 and distributed to other mappings. Therefore, with respect to the mappings obtained using the independence assumption the computed probabilities could differ from the real probabilities of a value depending on the probability incorrectly assigned to unsatisfiable mappings. However, the practical effect depends on the distribution of unsatisfiable mappings.

In Figure 5 we have plotted the number of unsatisfiable schemata found during the process — we stopped the algorithm after 500 hits, assuming of having found an intractable case. The top-K algorithm with the independence assumption determines the order with which the mappings are examined. Therefore, we may think of an ordered list of $6^{m \times n}$ mappings that we examine starting from the head — 6 is the number of relationships considered in our tests on inconsistencies. The situation we want to avoid is of having a distribution of the unsatisfiable mappings with a high density near the head of the list, and the reason of this undesired distribution can be the high probability assigned to a wrong relationship. In this case, this relationship will appear in many of the most likely mappings, creating inconsistencies with other (correct) relationships. This effect is likely to be exponential, because there would be $6^{(m \times n)-1}$ mappings containing this problematic relationship. This is the reason because intractable cases are more frequent when the number of pairs increases: with

more pairs it is more likely to find a pair where the matchers identify an inconsistent relationship. Therefore, for many of the schemata that we tried to match the distribution of unsatisfiable mappings has been near the top of the list obtained with the independence assumption. However, these cases can be tackled effectively, and these apparently negative results can be used to improve the outcome of the integration.

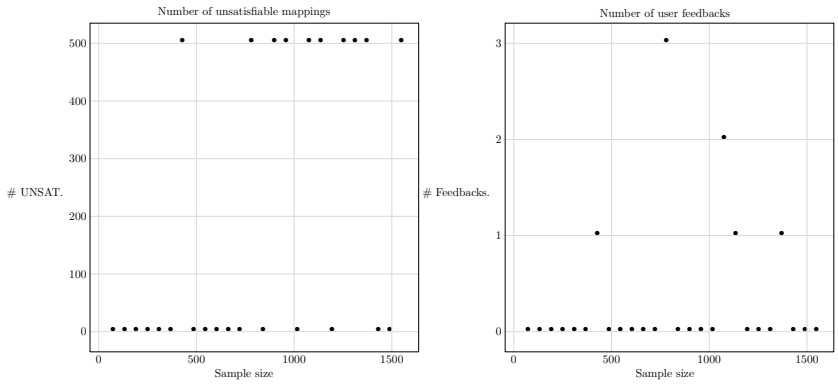

Fig. 5. Number of unsatisfiable mappings found during the computation of the top-K mappings, and number of feedbacks needed to reduce the intractable cases to tractable ones

To tackle this problem, consider that from the satisfiability-checking algorithm we can identify the relationships that caused the inconsistencies. Inconsistencies are due to wrong relationships, and this means that our software can focus our attention on the portion of the schemata where it fails in finding the correct relationship.

At this point, we can use this information to re-execute and improve the matching phase. If we can add a user feedback cycle to our approach, when an inconsistency is identified we can point the user to a few schema objects, which can be easily manually checked. The user feedback can be specified without altering the architecture of the software. We will simply add a new matcher to the pool, which allows a user to specify his/her opinion on some pairs of schema objects, through a probability assignment. This allows a human expert to give his/her contribution to the matching phase. Automatic matching is necessary because schemata may be too large to be manually analyzed. However, a user can focus on some pairs and "help" the matchers where they fail. These pairs are those identified as the cause of problems by the satisfiability-checking algorithm, and notified to the user. More in general, this approach can also be used in case a matcher behaves in an unexpected way, may be because of some features of the input schemata that had not been considered when the matcher was designed. In fact, when we use the unsatisfiability-checker to identify that some relationships have not been correctly identified, we can also ask the software to display the

probability assignments of the single matchers. As a result, we should be able to identify which matcher is causing troubles, and we may remove it from the pool or fix it, correcting its wrong behavior.

More interestingly, we may think of using this information to automatically improve the matching phase. In fact, the feedback can also be used by the software itself as a way to tune the matchers: when it identifies a wrong relationship, it can check which matcher(s) is causing troubles and try to reduce its impact on the results, to see if the problem persists.

To verify our analysis, we have run again the same integration processes. This time, we included a new manual matcher in the pool, and each time we encountered an intractable case we manually compared the pairs of schema objects highlighted by our software. Sometimes, after having solved an inconsistency, we have found that the same test databases needed additional feedbacks, as other inconsistencies were found.

In Figure 5 we have plotted the number of user-feedbacks needed to prevent an exponential number of steps to be performed by the top-K algorithm. Although we do not present the details of the algorithm used to retrieve the top-K mappings, it should be intuitively clear that given N pairs of schema objects and r semantic relationships there can be r^N different mappings, and therefore a top-K algorithm may need to check an exponential number of mappings before finding K inconsistent ones. According to this graph, we can make the following considerations:

- At most three user feedbacks where needed to reduce the execution to a linear time computation. Each user feedback involves the manual comparison of three schema objects. If we compare the results represented in the figure with the number of pairs of schema objects in our tests, it appears that human intervention is very limited and the majority of the process is carried on by the software.
- From a comparison of the two graphs of Figure 5 it may seem that some of the problems have been solved without any user feedback. This is due to two reasons: first, in many cases there was a single difficult pair of objects that caused the exponential behavior in different tests. We remind the reader that the tests have been performed on different subsets of the same databases, containing common schema objects. Therefore, the new matcher with the information collected in previous cases was already able to solve the problem without additional input. Moreover, the schema analysis phase is stochastic (it depends on the samples of instances chosen by the matchers), therefore it may happen that different executions on the same input databases produce different results.

The last steps can be repeated many times, if there is more than one problematic relationship. In summary, when the matchers perform well, i.e., find the correct or plausible relationships, the distribution of unsatisfiable mappings has a low impact on the probabilities computed with the independence assumption. If this does not happen, we are able to identify where the matchers are failing, and repeat the top-K algorithm providing user feedback — notice that we do not

need to re-execute the time-wasting schema analysis phase, because we already have the outcomes of the matchers and need only to incorporate the opinion of the user or automatically identify and manage badly-behaving matchers.

4 Open problems

In this section we briefly summarize the open issues regarding the implementation of a probabilistic data integration approach like the one described in Section 2. First, the nature and compatibility of the **probabilities generated by the matchers** should be studied. We cannot think of solving this problem in general, but we can analyze the problem in specific domains (like the comparison of specific aspects of schema objects) and make some experimental evaluations on the impact of different choices on the results of the integration process. Similarly, it must be studied how to represent **dependencies between matchers**, to perform a proper combination of their outcomes. Another point that has not been studied yet is the **comparison of uncertain data**. In addition to its theoretical interest, this is necessary to define a closed integration process, enabling the definition of an **algebra of model management operators** [16] and thus the iterative execution of data integration activities on several local data sources. Another complex problem is the management of **dependencies between different relationships**. In Section 3 we have shown some results to reduce the impact of these dependencies. However, the computation of exact probabilities is still intractable, and it is not even clear how to redistribute the probability mass incorrectly assigned to unsatisfiable mappings. Luckily, this case only applies to the usage of complex semantic relationships. However, it seems possible to tackle this problem adding some uncertainty also to the probability of the mappings, or estimating possible errors. Finally, it is necessary to develop mature **systems for** the management of **uncertain data**. In particular, in addition to the studies on uncertain data models and physical implementations of uncertain data, it is important to develop **adequate user interfaces** to access this complex data. Otherwise, the result of an uncertain integration process would not be of any practical use.

5 State of the art

As we have already mentioned, uncertainty is present in all (semi-)automatic data integration processes and methods. In this survey we focus on those works that concern directly the management of uncertainty.

Probability theory has been used for many years in data integration works, with the limitations already discussed in previous sections. For example, the system described in [17] tries to assign probabilities to alternative relationships between pairs of schema objects. The critical points described in Section 2.1 affect also this work: the algorithm used to generate probabilities is arbitrary, while their combination does not admit arbitrary dependencies. However, the characterizing feature of this and other early works using uncertainty theories

in data integration is that after probabilities have been evaluated a threshold is used to select matching and non matching objects. Therefore, the uncertainty generated during the integration process is lost. Probability theory has also been used in instance integration (entity reconciliation, or record linkage), and also in this case probabilities are used together with a decision model to choose exact mappings [18]. Another application of probability theory to the field of data integration is described in [19]. However, in this case probabilities do not characterize the uncertainty in the matching process and in the integrated schema, but are used to rank the local data sources with the aim of improving query processing. Data is not uncertain, and mappings between schema objects are well known, although not explicitly mentioned in the paper.

In 2005, three works focused on the management of uncertainty in data integration using probability theory or its extensions [20–22]. Interestingly, these works studied different and complementary aspects of the problem. [20] presents a method of uncertain schema integration with a multi-matcher architecture, and keeps the uncertainty modeled during the matching phase up to the merging step. Dempster-Shafer's theory, an extension of probability theory, is used as the formalism to represent uncertainty, and the authors define how uncertainty is represented and manipulated at each step of the process. In this work the result of the integration process is not explicitly represented with a data model for uncertain data, and the paper tackles the aforementioned critical points with independence assumptions that should be relaxed in real applications. Another topic not covered by this paper is the implementation of the method, that presents many complexity issues. A paper which from some points of view complements this work is [21], where a data model for uncertain data is provided to represent the result of a data integration process. In this paper the authors assume to already have a method that performs the integration and evaluates its uncertainty (probability), and focus on its representation. In addition, this work concerns the integration of instances, and not schemata, that are assumed to be equal. This can therefore be thought of as part of a method that matches schemata, and subsequently focuses on data mappings — one of the topics not covered in [20]. Finally, [22] and its extended version [23] focus on probabilistic schema matching. This papers do not provide a complete view of uncertain data integration processes, like in the two aforementioned works, but focus on the implementation of probabilistic classifiers (matchers), that have not been covered in detail in [20, 21]. As we have already mentioned, this is still an open issue, and also in this papers the probability assignment process is reasonable and supported by experiments, but arbitrary, and the aggregation of probabilities does not consider dependencies, but only the confidence we have in each classifier — it is therefore a way to weight the matchers. In the same year, a different approach has been presented to represent uncertainty using fuzzy logic [24]. This tries to formalize the concept of *similarity* between schema objects, used in many data integration approaches. At the same time, the application of probability theory to the representation and computation of mappings was object of studies in the semantic web community [25].

Recently, the importance of managing uncertain information in data integration has become well recognized [11, 26]. In [27] another approach to merge uncertain information has been proposed. The content of this work is analogous to [22], but more focused on the formalisms used to represent uncertainty — this kind of activity is also known as *information fusion*. A language to represent the result of an uncertain data integration process has been proposed in [28].

Another topic that has been discussed in many works on schema matching is the evaluation of top-K mappings. In this paper we have provided an experimental analysis of probabilistic top-K mappings to point out the relevance of uncertain information. The most recent work on this topic extends the results of [24] and proposes a method to compute K mappings and choose one mapping among them [29]. However, also in this case uncertainty is manipulated and represented only until a choice is made to keep only exact information.

6 Conclusion

In this paper we have described the status of the research about uncertainty in data integration. From some preliminary investigations, it seems already clear that the information we lose not considering uncertainty is relevant, and that the explicit management of uncertainty may also increase the quality of the integrated data sources. However, there are still many open problems, that we have briefly listed, and for which we may consider this field at its beginning. In particular, it is fundamental to have advances in related fields, from the interpretation of the mathematical theories of uncertainty to the modeling, implementation and external representation of uncertain data.

References

1. Batini, C., Lenzerini, M., Navathe, S.: A comparative analysis of methodologies for database schema integration. ACM Computing Surveys **18**(4) (1986) 323–364
2. Buneman, P., Davidson, S., Kosky, A.: Theoretical aspects of schema merging. In Pirotte, A., Delobel, C., Gottlob, G., eds.: Proceedings of Advances in Database Technology (EDBT '92). Volume 580 of LNCS., Berlin, Germany, Springer (1992) 152–167
3. McBrien, P., Poulovassilis, A.: A formalisation of semantic schema integration. Information Systems **23**(5) (1998) 307–334
4. Doan, A., Domingos, P., Halevy, A.Y.: Reconciling schemas of disparate data sources: A machine-learning approach. In: ACM SIGMOD Conference. (2001)
5. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. The VLDB Journal **10**(4) (2001) 334–350
6. Madhavan, J., Bernstein, P., Rahm, E.: Generic schema matching with Cupid. In: Proc. 27th VLDB Conference. (2001) 49–58
7. Lenzerini, M.: Data integration: a theoretical perspective. In: PODS Conference. (2002)
8. S. Mehiik, H.G.M., Rahm, E.: Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In: Proceedings of ICDE'02. (2002)

9. Melnik, S., Rahm, E., Bernstein, P.A.: Rondo: a programming platform for generic model management. In: Proc. SIGMOD 2003, ACM Press (2003) 193–204
10. Doan, A., Halevy, A.Y.: Semantic integration research in the database community: A brief survey. AI Magazine **26**(1) (2005) 83–94
11. Halevy, A., Rajaraman, A., Ordille, J.: Data integration: the teenage years. In: VLDB Conference, VLDB Endowment (2006) 9–16
12. Bernstein, P.A., Melnik, S., Churchill, J.E.: Incremental schema matching. In: VLDB Conference, VLDB Endowment (2006) 1167–1170
13. Do, H.H., Rahm, E.: Matching large schemas: Approaches and evaluation. Information Systems (2007) to appear.
14. Halevy, A.Y.: Data integration: A status report. In: BTW. Volume 26 of LNI., GI (2003) 24–29
15. May, W.: Information extraction and integration with Florid: The Mondial case study. Technical Report 131, Universität Freiburg, Institut für Informatik (1999) Available from http://dbis.informatik.uni-goettingen.de/Mondial.
16. Bernstein, P.A.: Applying model management to classical meta data problems. In: CIDR. (2003)
17. Hayne, S., Ram, S.: Multi-user view integration system (muvis): An expert system for view integration. In: Proceedings of the Sixth International Conference on Data Engineering, Washington, DC, USA, IEEE Computer Society (1990) 402–409
18. Dey, D., Sarkar, S.: Generalized normal forms for probabilistic relational data. IEEE Transactions on Knowledge and Data Engineering **14**(3) (2002) 485–497
19. Florescu, D., Koller, D., Levy, A.Y.: Using probabilistic information in data integration. In: VLDB Conference. (1997) 216–225
20. Magnani, M., Rizopoulos, N., McBrien, P., Montesi, D.: Schema integration based on uncertain semantic mappings. In Delcambre, L.M.L., Kop, C., Mayr, H.C., Mylopoulos, J., Pastor, O., eds.: ER. Volume 3716 of Lecture Notes in Computer Science., Springer (2005) 31–46
21. van Keulen, M., de Keijzer, A., Alink, W.: A probabilistic XML approach to data integration. In: ICDE. (2005)
22. Nottelmann, H., Straccia, U.: splmap: A probabilistic approach to schema matching. In: ECIR. (2005) 81–95
23. Nottelmann, H., Straccia, U.: Information retrieval and machine learning for probabilistic schema matching. Inf. Process. Manage. **43**(3) (2007) 552–576
24. Gal, A., Anaby-Tavor, A., Trombetta, A., Montesi, D.: A framework for modeling and evaluating automatic semantic reconciliation. VLDB Journal **14**(1) (2005) 50–67
25. Pan, R., Ding, Z., Yu, Y., Peng, Y.: A bayesian network approach to ontology mapping. In: proceedings of ISWC conference. (2005)
26. Gal, A.: Why is schema matching tough and what can we do about it? SIGMOD Rec. **35**(4) (2006) 2–5
27. Hunter, A., Liu, W.: Fusion rules for merging uncertain information. Information Fusion **7**(1) (2006)
28. Cali, A., Lukasiewicz, T.: An approach to probabilistic data integration for the semantic web. In: Proceedings of ISWC-URSW conference. (2006)
29. Gal, A.: Managing uncertainty in schema matching with top-k schema mappings. Journal on Data Semantics (2006)

A New Language and Architecture to Obtain Fuzzy Global Dependencies

Ramón Alberto Carrasco[1], Maria Amparo Vila[2], Mari Ángeles Aguilar[3]

[2] Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada, Spain
racg@ugr.es
[2] Dpto. Ciencias de la Computación e IA
Universidad de Granada, Spain
[3] Universidad de Granada, Spain

Abstract. At present, we have proceeded to extend SQL into a new language called dmFSQL (data mining Fuzzy Structured Query Language) which can be used to solve real problems of Data Mining. Besides, we have an architecture that permits us to use this language dmFSQL for Oracle© Database. This enables us to evaluate the process of Data Mining at both a theoretical and a practical level. Now, we extend this language and architecture to obtain fuzzy global dependencies (GDs) as a common framework to integrate fuzzy and gradual functional dependencies on any type of data. We consider that this model satisfies the requirements of Data Mining systems.

Keywords: Fuzzy Functional Dependencies, Gradual Functional Dependencies, Flexible Queries, Data Mining, Fuzzy Databases.

1 Introduction

We can define Data Mining (DM) as the process of extraction of interesting information from the data in databases. According to [6] a discovered knowledge is interesting when it is novel, potentially useful and non-trivial to compute. A series of new functionalities exist in DM, which reaffirms that it is an independent area [6]: high-level language on the discovered knowledge and for showing the results of the user's requests for information (e.g. queries); efficiency on large amounts of data; handling of different types of data; etc.

We considerer, that SQL does not satisfy the minimum requirements (above explained) to be a true DM language. To solve this, we have proceeded to extend the SQL language into a new language [3]: dmFSQL (data mining Fuzzy Structured Query Language). This new language integrates flexible queries, clustering and fuzzy classification techniques [2]. We have developed an architecture that permits us to use this language dmFSQL for one of the commercial DataBase Management System (DBMS) most frequently used: Oracle©. The core of this architecture is a server programmed mainly in PL/SQL language. This enables us to evaluate the process of DM at both a theoretical and a practical level.

Interest in functional dependencies (FDs) has been motivated by the fact that FDs can capture some forms of redundancy. Therefore the use of FDs has come about as a result of their usefulness in database design. Fuzzy functional dependencies (FFDs) arise in the framework of fuzzy relational databases. Various definitions of FFDs have been proposed. FFDs have not often been closely connected with database design. However, FFDs seem very appropriate to discover properties which exist in the current manifestation of the data, i.e. in a Data Mining (DM) process. We can make the same observation about gradual functional dependencies (GFDs) which are a special type of fuzzy dependencies that reflect monotonicity in the data. We have defined a new type of dependencies [1] fuzzy global dependencies (GDs) as a common framework to integrate fuzzy and gradual functional dependencies but only for a type of data: trapezoidal possibility distributions.

The objective of this paper is to redefine the fuzzy global dependencies for any type of data, on then to extend the dmFSQL language and architecture to obtain theses dependencies.

This paper is organized as follows: in Section 2 we introduce an explanation about the dmFSQL language. In Section 3 we introduce an explanation about the architecture of dmFSQL. In Section 4 we define fuzzy global dependencies (GDs) as a common framework to integrate fuzzy and gradual functional dependencies on any type of data. In Section 5 we extend the dmFSQL language and architecture to obtain theses new dependencies. In Section 6 are presented some experimental results on stock-market sceneries. Finally, we suggest some conclusions.

2 dmFSQL

We considerer, that SQL does not satisfy the minimum requirements to be a true DM language. To solve this, we proceed to extend the SQL language into a new language [3]: dmFSQL (data mining Fuzzy Structured Query Language). We define the new language with a series of desirable properties. We can say that these properties are not found jointly in the definition of similar languages [7, 8, 9]. The properties are: similar syntax to SQL; possibility of DM iterative, this is, the language should comply the property of closure, that is the possibility to apply the language to a prior result of this same language; an approach toward the techniques of DM that are considered useful in several sectors: flexible queries, clustering and classification.

2.1 dmFSQL for Flexible Queries

dmFSQL includes a language [3] that extends the SQL language to allow flexible queries. Thus, the language can manage fuzzy attributes which are classified by the system in 4 types:

- **Type 1:** These attributes are totally crisp, but they have some linguistic trapezoidal labels defined on them.
- **Type 2**: These attributes admit crisp data as well as possibility distributions over an ordered underlying domain.

- **Type 3**: On these attributes, some labels are defined and on these labels, a similarity relation has yet to be defined. These attributes have no relation of order.
- **Type 4**: It is a generic type (fuzzy or crisp), which admits some fuzzy treatment. We permitted this attribute is formed by more than a column of the table (complex attributes).

We show an abstract with the main extensions added to DML (Data Manipulation Language) of SQL:

- **Linguistic Labels:** They represent a concrete value of the fuzzy attribute. dmFSQL works with any kind of attributes therefore, by example, a label can have associated: a trapezoidal possibility, a text, a XML document, etc.
- **Fuzzy Comparators:** In addition to common comparators (=, >, etc.), dmFSQL includes fuzzy comparators in Table 1.
- **Fulfillment Thresholds γ:** For each simple condition a Fulfillment threshold may be established with the format <*condition*> *THOLD* γ, indicating that the condition must be satisfied with a minimum degree γ in [0,1].
- **CDEG(<attribute>) function**: This function shows a column with the fulfillment degree of the condition of the query for a specific attribute.

Table 1. Fuzzy Comparators for dmFSQL

Fuzzy Comparator		**Significance**
Possibility	**Necessity**	
FEQ	NFEQ	Fuzzy EQual
FGT	NFGT	Fuzzy Greater Than
FGEQ	NFGEQ	Fuzzy Greater or Equal
FLT	NFLT	Fuzzy Less Than
FLEQ	NFLEQ	Fuzzy Less or Equal
MGT	NMGT	Much Greater Than
MLT	NMLT	Much Less Than

Besides, dmFSQL includes an extension of the DDL (Data Definition Language) of SQL to specify these objects: fuzzy types, linguistic labels, fuzzy comparators…

2.2 dmFSQL for Clustering and Classification

We define a new type of object called *project* that does not exist in SQL. This object has mainly the following task: It is the backup to keep the initial conditions, intermediate and ends results of the DM process to carry out. These intermediate results are to improve the performance of the iterative DM process. The DDL of dmFSQL for Data Mining consists of a series of operations on the project object (create, alter, drop…). The DML of dmFSQL executes the true DM process. Continued we explain briefly the semantics of the commands of this language, a more detailed description can be found in [3]:

2.2.1 DDL of dmFSQL for Clustering and Classification

CREATE_MINING
With this sentence a new project can be created. In this project the conditions for the process are set to carry out DM. The simplified syntax is the following:

CREATE_MINING PROJECT *id_project* [ON OWNER *id_owner*] ON TABLE *id_table_orig_project*
WITH COLUMNS FOR [CLUSTERING '(' *list_columns_clu* | *list_columns_clu_cen* ')']
[CLASSIFICATION '(' *list_columns_cla* ')'] ';' where:

- *id_project*: project name.
- *id_owner*: owner of the project.
- *id_table_orig_project*: table name with the original data for the DM process.
- *list_columns_clu*: specifications on the table columns *id_table_orig_project* prominent for the clustering process.
- *list_columns_clu_cen*: if we want to characterize each one of the clusters obtained (with a row centroid), specifications on the table columns *id_table_orig_project* prominent for this characterization process,
- *list_columns_cla*: specifications on the table columns *id_table_orig_project* prominent for the classification process.

dmFSQL include others DDL sentences: to modify, drop, grant and revoke permission for the management of the project [3].

2.2.2 DML of dmFSQL for Clustering and Classification

We have explained the form to define a project by means of the DDL of dmFSQL. Now we can carry out the true DM process, by means of the use of the Data Manipulation Language (DML) of dmFSQL. The DML has a unique sentence **SELECT_MINING** by means of which, we will carry out the different processes of DM previously described:

SELECT_MINING CLUSTERING

This sentence is the interface to carry out the clustering process, and optionally, to characterize each one of the groups arisen of such process by means of an only row (centroid). The syntax is the following:

SELECT_MINING CLUSTERING *id_project* INTO TABLE_CLUSTERING *id_table_result_clu*
[',' TABLE_CENTROIDS *id_table_result_cen*]
OBTAINING {*n_clusters*|OPTIMAL_ABS|OPTIMAL_H3|OPTIMAL_MED} CLUSTERS where:

- *id_project*: project name.
- *id_table_result_clu:* table name to be created as result of the clustering process.
- *id_table_result_cen:* table name to be created as result of the characterization process of each cluster.
- *n*_clusters: this value is the number of groups to obtain after the result of the execution of the sentence.
- OPTIMAL_ABS, OPTIMAL_H3, OPTIMAL_MED: we can leave that the system determine the number of clusters to obtain more adequate with three methods [2]: OPTIMAL_ABS: optimal absolute partition; OPTIMAL_H3: optimal partition from the measure H_3; OPTIMAL_MED: optimal average partition.

SELECT_MINING CLASSIFICATION

This sentence is the interface to carry out the classification process. It uses as criterion of classification one of the two possible results of the clustering process: the table with the centroids, or the complete table turned out. The syntax is the following:

SELECT_MINING CLASSIFICATION *id_project* FROM *id_table_orig_cla* [TO *id_table_result_cla*]
ACCORDING_TO {TABLE_CENTROIDS {*id_table_result_cen* | LAST}
TABLE_CLUSTERING {*id_table_result_clu* | LAST} WITH *n_neighboard* NEIGHBOARD}
[WHERE CLUSTER_ID IS {*id_cluster* | THE_BEST}] THOLD *threshold* where:

- *id_project*: project name.
- *id_table_orig_cla*: table with the original data for the classification process.
- *id_table_result_cla:* table to be created as result of the classification process.
- *id_table_result_cen*: if we want to apply the method of fuzzy classification based on centroids [2], table with the centroids to use as criterion of classification.
- *id_table_result_clu*: if we want to apply the method of fuzzy classification based on *n*-nearest neighboard [2], the table with the result of the clustering process to use as criterion of classification.
- *n_neighboard*: this number indicates the number *n* of rows of the table *id_table_result_clu* that should be used for the criterion of classification based on *n*-nearest neighboard.
- *id_cluster*: it indicates whether we want to restrict the classification to a single group *id_cluster*, obtained before the clustering process. If we use the clause CLUSTER_ID IS THE_BEST then each row will be assigned to the cluster with greater degree of membership.
- *threshold*: minimum degree of membership to the different clusters (results of the clustering) that should satisfy the rows result of the classification.

3 Architecture of dmFSQL

We have developed an architecture that permits to use dmFSQL language available for Oracle© Databases. The architecture (Figure 1) is made up by: Data, dmFSQL Server and dmFSQL Clients. Following, we explain these elements:

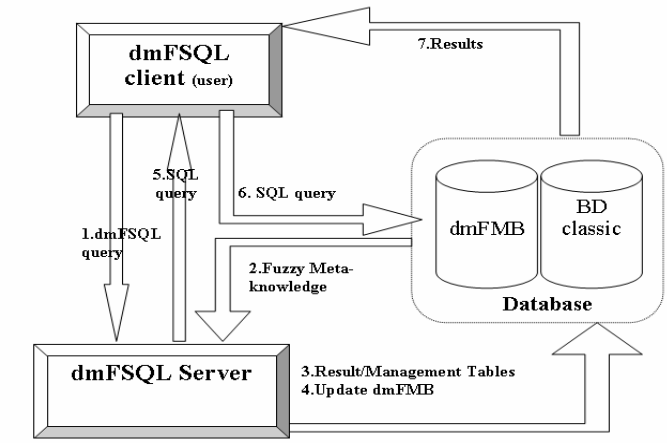

Figure 1. Architecture of dmFSQL

3.1 Data: Traditional Database and dmFMB

The data can be classified in two categories:

- **Traditional Database**: They are data from our relations with a special format to store the 4 types of fuzzy attributes.
- **data mining Fuzzy Meta-knowledge Base** (dmFMB): It is the support to the project object and it stores information about the Fuzzy Relational Database in a relational format. It stores attributes which admit fuzzy treatment for Data Mining and it will store different information for each one of them, depending on their type:
  - Fuzzy Attributes **Type 1**: In order to use crisp attributes in flexible queries we will only have to declare them as being a fuzzy attribute Type 1 and store the following data in the dmFMB: Trapezoidal linguistic labels; value for the margin of the approximate values (used en FEQ and NFEQ); etc.
  - Fuzzy Attributes **Type 2**: As well, as declare them as being a fuzzy attribute Type 2, these attributes have to store the same data in the dmFMB as the fuzzy attributes Type 1.
  - Fuzzy Attributes **Type 3**: They store in the dmFMB their linguistic labels, the similarity degree amongst themselves and the compatibility between attributes of this type, i.e., the attributes that use the same labels and that can be compared amongst them.
  - Attributes **Type 4**: The dmFMB stores information for the fuzzy treatment of the attributes Type 4:
    - **Fuzzy Comparison Functions**: The user can define the functions of comparison (Table 1) for the treatment of each attribute of Type 4. The format is: *CDEG (A fcomp B) -> [0,1]* with *CDEG* the compatibility degrees, *A*, *B* two attributes or linguistic labels Type 4 and *fcomp* any fuzzy comparator in Table 1. The user can associate each attribute functions already defined in the dmFMB.
    - **Representation Functions**: The user can optionally define it to show the attributes in a more comprehensible way. Of course, the user can associate each attribute functions already defined in the dmFMB.
    - **Linguistic labels**: They represent a concrete value of the attribute.
    - **Complex attributes**: We permitted this attribute is formed by more than a column of the table. Therefore, the dmFMB stores information on structure of the attributes Type 4.

3.2 dmFQL Server

It has been programmed mainly in PL/SQL and it carries out a lexical, syntactic and semantic analysis of the dmFSQL query. If errors, of any kind whatsoever, are found,

it will generate a table with all the found errors. If there are no errors, the dmFSQL Sever always generates a standard SQL sentence. The Server works (semantic treatment) depending on the kind of dmFSQL sentence:

- **DDL** sentences: Normally, the treatment consists either to insert information in the dmFMB or to execute grant sentences on the SGBD host.
- **DML of fuzzy sentences** (fuzzy SELECT command): The dmFSQL query is translated into a standard SQL sentence. The resulting SQL sentence includes reference to the following kinds for attributes Type 1, Type 2 and Type 3 (as we have seen, these functions are included in the dmFMB for the attributes Type 4):
  - Representation Functions: These functions are used to show the fuzzy attributes in a comprehensible way for the user and not in the internally used format.
  - Fuzzy Comparison Functions: They are utilized to compare the fuzzy values and to calculate the compatibility degrees (CDEG function).
- **DML of Data Mining** sentences (SELECT_MINING command): The treatment consists to call the Data Mining process: clustering, characterization or classification [2, 3]. These processes obtain a table and the Server returns a conventional SELECT command on this table.

4 Fuzzy Global Dependencies in Databases

We have defined a new type of dependencies [1] fuzzy global dependencies (GDs) as a common framework to integrate fuzzy and gradual functional dependencies but only for a type of data: trapezoidal possibility distributions. In this section we extend this definition on any type of data.

There have been several approaches to the problem of defining the concept of FFD but unlike classical FDs one single approach has not dominated. We begin by briefly describing the concept of classical FD, later we give a general definition of FFD and GFD based on fuzzy functions and then, we shall introduce a more relaxed definition of FFD and GFD in order to manage exceptions.

Definition 1. Functional Dependency (FD).

The relation R with attribute sets $X=(col\_ant_1, col\_ant_2,\ldots, col\_ant_l)$ and $Y=(col\_con_1, col\_con_2,\ldots, col\_con_q)$ in its scheme verifies the FD $X \rightarrow Y$ if and only if, for every instance r of R it is verified:

$\forall t_1, t_2 \in r, t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$

The concept of FFD given by Cubero and Vila in [4] is a smoothed version of the classical FD. The basic idea consists in replacing the equality used in the FD definition by fuzzy resemblance relations, in such a way that:

Definition 2. α -β Fuzzy Functional Dependency (α -β FFD).

The relation R verifies an α -β FFD $X \rightarrow_{FT} Y$ if and only if, for every instance r of R it is verified:

$\forall t_1, t_2 \in r$, $F(t_1[X], t_2[X]) \geq \alpha \Rightarrow T(t_1[Y], t_2[Y]) \geq \beta$ where F and T are fuzzy ressemblance relations.

The flexibility provided by the combined use of the parameters α and ß and the different kinds of resemblance relation should be noted. If F is a weak resemblance measure and T is a strong one, we get interesting properties for database design (decomposition of relations). A more detailed description of these concepts can be found in [4].

Often just a few tuples in a database can prevent the FFD from being completed. To avoid this, we can relax the FFD definition in such a way that all the tuples of the relationship are not forced to fulfill the above condition, therefore we define:

Definition 3. Confidence of a FFD.

For a instance *r* of *R* is verified an α -β FFD $X \rightarrow_{FT} Y$ with confidence *c*, where $c \in [0,1]$ is defined as:

$$\begin{cases} c = 0 \text{ if } Card\{(t_1, t_2)\ \ t_1, t_2 \in r / F(t_1[X], t_2[X]) \geq \alpha\} = 0 \\ c = \dfrac{Card\{(t_1, t_2)\ \ t_1, t_2 \in r / F(t_1[X], t_2[X]) \geq \alpha \wedge T(t_1[Y], t_2[Y]) \geq \beta\}}{Card\{(t_1, t_2)\ \ t_1, t_2 \in r / F(t_1[X], t_2[X]) \geq \alpha\}} \text{ Otherwise} \end{cases}$$

where $\wedge$ is the logical operator *and*. The basic idea consists in computing the percentage of tuples which fulfill the antecedent and consequent together with respect to those which only fulfill the consequent.

Definition 4. Support of a FFD.

For a instance *r* of *R* is verified an α -β FFD $X \rightarrow_{FT} Y$ with support *s*, where $s \in [0,1]$ s defined as:

$$\begin{cases} s = 0 \text{ if } n = 0 \\ s = \dfrac{Card\{(t_1, t_2)\ \ t_1, t_2 \in r / F(t_1[X], t_2[X]) \geq \alpha \wedge T(t_1[Y], t_2[Y]) \geq \beta\}}{n} \text{ otherwise} \end{cases}$$

where *n* is the number of tuples of the *r* instance of the relation R.

The idea is to find the percentage of tuples which fulfill the antecedent and consequent together with respect to the total rows of the relation.

Another way of considering the connections between data in databases is to specify a relationship between objects in a dataset and reflect monotonicity in the data by means of that we have called gradual fuzzy dependencies (GFDs). It is closely related to the idea of gradual rules introduced by Dubois and Prade [5]. An intuitive example of a GFD is "the bigger businesses are the higher earnings they have" and we assume that the concept of GFD can be considered, in this way, as similar to the FFD one. Therefore we define:

Definition 5. α–ß Gradual Functional Dependency (α -β GFD).

The relation *R* verifies an α -β GFD $X \rightarrow_{F'T'} Y$ if and only if, for every instance *r* of *R* it is verified:

$\forall t_1, t_2 \in r$, $F'(t_1[X], t_2[X]) \geq \alpha \Rightarrow T'(t_1[Y], t_2[Y]) \geq \beta$ where F' and T' are fuzzy relations of the type: *fuzzy greater than, fuzzy greater than or equal to, fuzzy less*

than, fuzzy less than or equal to, fuzzy not equal, etc. We can define an α -β GFD with confidence *c* and support *s* in the same way that we have made it for FFD (see Definition 3 and 4).

Now, it is necessary to relate the dmFSQL environment to our definitions. To do so, we first introduce a general definition of fuzzy global dependencies (GD) based on FSQL operators and FSQL CDEG function, later we will show how GD can be calculated with FSQL.

Definition 6. (α_i)-(β_j) Fuzzy Global Dependency ((α_i)-(β_j) GD)

The relation R with attribute sets X=(col_ant_1, col_ant_2,…, col_ant_l) and Y=(col_con_1, col_con_2,…, col_con_q) where the attributes are of fuzzy type 1, 2, 3 or 4 verifies a (α_i)-(β_j) GD $X \rightarrow_{F*T*} Y$ with (α_i)=(α_1,α_2,…,α_l) / $\alpha_i \in [0,1]$ $\forall$ i=1,…,l and (β_j)=(β_1,β_2,…,β_q) / $\beta_j \in [0,1]$ $\forall$ j=1,…,q, if and only if, for every instance r of R it is verified:

$$\forall\, t_1,\ t_2 \in r,\quad \wedge_{i=1,2...,l}[F^*_i(t_1[col\_ant_i],t_2[col\_ant_i]) \geq \alpha_i] \Rightarrow \wedge_{j=1,2...,q}[T^*_j(t_1[col\_con_j],t_2[col\_con_j]) \geq \beta_j]$$ where

$F^*_i: U \times U \rightarrow [0,1] / F^*_i(A,B)$=CDEG($A$ fuzzy_comp_ant$_i$ B)

$T^*_j: U \times U \rightarrow [0,1] / T^*_j(A,B)$=CDEG($A$ fuzzy_comp_con$_j$ B)

$\forall$ A, $B \in U$ where U is a instance of the dmFMB.

fuzzy_comp_ant$_i$, fuzzy_comp_con$_j$ defined as any fuzzy comparator in dmFSQL (see Table 1) defined on the fuzzy attributes A and B in the dmFMB.

Definition 7. α -β Fuzzy Global Dependency (α -β GD)

The relation R with attribute sets X=(col_ant_1, col_ant_2,…, col_ant_l) and Y=(col_con_1, col_con_2,…, col_con_q) where the attributes are of fuzzy type 1, 2, 3 or 4 verifies a α -β GD $X \rightarrow_{F*T*} Y$ with $\alpha \in [0,1]$ AND $\beta \in [0,1]$, if and only if, for every instance r of R it is verified:

$$\forall\, t_1,\ t_2 \in r,\quad \wedge_{i=1,2...,l}[F^*_i(t_1[col\_ant_i],t_2[col\_ant_i])] \geq \alpha \Rightarrow \wedge_{j=1,2...,q}[T^*_j(t_1[col\_con_j],t_2[col\_con_j])] \geq \beta \ \forall\ i=1,\ldots,l \text{ y } \forall\ j=1,\ldots,q$$

Now, we can make a new definition of FFDs and GFDs as a particular case of GDs:

Definition 8. (α_i)-(β_j) Fuzzy Functional Dependency ((α_i)-(β_j) FFD)

In Definition 6, if fuzzy_comp_ant$_i$, fuzzy_comp_con$_j \in$ {FEQ,NFEQ} then we say that R verifies an (α_i)-(β_j) FFD $X \rightarrow_{F*T*} Y$.

Definition 9. α -β Fuzzy Functional Dependency (α -β FFD).

In Definition 7, if fuzzy_comp_ant$_i$, fuzzy_comp_con$_j \in$ {FEQ,NFEQ} then we say that R verifies an α -β FFD $X \rightarrow_{F*T*} Y$.

Definition 10. (α $_i$)-(β $_j$) Gradual Functional Dependency ((α $_i$)-(β $_j$) GFD)

In Definition 6, if $\exists\ a\ /\ a \in \{1,2,\ldots,l\}$ which fulfils that fuzzy_comp_ant$_a \notin$ {FEQ,NFEQ} or $\exists\ b\ /\ b \in \{1,2,\ldots,q\}$ which fulfils that fuzzy_comp_con$_b \notin$ {FEQ,NFEQ} then we say that *R* verifies an (α $_i$)-(β $_j$) GFD $X \rightarrow_{F*T*} Y$.

Definition 11. α -β Gradual Functional Dependency (α -β GFD)

In Definition 7, if $\exists\ a\ /\ a \in \{1,2,\ldots,l\}$ which fulfils that fuzzy_comp_ant$_a \notin$ {FEQ,NFEQ} or $\exists\ b\ /\ b \in \{1,2,\ldots,q\}$ which fulfils that fuzzy_comp_con$_b \notin$ {FEQ,NFEQ} then we say that *R* verifies an a α -β GFD $X \rightarrow_{F*T*} Y$.

Of course, we can define GD with confidence *c* and support *s* in the same sense that we have made it for FFD. To simplify notation, in (α $_i$)-(β $_j$) DGD $X \rightarrow_{F*T*} Y$ we will denote F* as (fuzzy_comp_ant$_i$)* ∀ i=1,…,*l*, and similar notation for T*.

5 dmFSQL for Fuzzy Global Dependencies in Databases

In this section we process to extend the dmFSQL language (DDL and DML sentences) and architecture to obtain the fuzzy global dependencies above explained:

5.1 DDL of dmFSQL for Fuzzy Global Dependencies

CREATE_MINING

With this extension of CREATE_MINING sentence a new project to obtain GDs can be created. The simplified syntax is the following:

CREATE_MINING PROJECT *id_project* [ON OWNER *id_owner*] ON TABLE *id_table_orig_project*
WITH COLUMNS FOR [FGLOBAL_DEPENDENCIES '('
{ANTECEDENT *list_columns_gd* CONSEQUENT *list_columns_gd* |
ANTECEDENT *list_columns_gd* THOLD_ANT *threshold*
CONSEQUENT *list_columns_gd* THOLD_CON *threshold* |
ANTECEDENT *list_columns_gd_thold* CONSEQUENT *list_columns_gd_thold* } ')'] ';' where:

- *id_project*: project name.
- *id_owner*: owner of the project.
- *id_table_orig_project*: table name with the original data for the DM process in this project.
- *list_columns_gd*: They are the specifications on the table columns *id_table_orig_project* prominent for the process to obtain (α $_i$)-(β $_j$) Fuzzy Global

Dependencies (see Definition 6) or α -β Fuzzy Global Dependencies (see Definition 7). It is necessary to do the specifications for antecedent columns col_ant_i with i=1..l (ANTECEDENT clausule) and for the consequent columns col_con_j with j=1..q (CONSEQUENT clausule). Such specifications are the following:

- *id_column*: column name the antecedent i.e. col_ant_i or the consequent i.e. col_con_j. This column should be defined as a fuzzy type 1, 2, 3 or 4.
- FCOMP_FGLOBAL_DEPENDENCIES *fuzzy_comp*: is a fuzzy comparator in dmFSQL (see Table 1) defined on the fuzzy attribute *id_column* in the dmFMB. It indicates the fuzzy_comp_ant$_i$ for the antecedent and the fuzzy_comp_con$_j$ for consequent specify in Definition 6 and 7. If we want to obtain a α -β GD with a α and β value prefixed for the user, then we must to specify for the antecedent and consequent columns two clausules THOLD_ANT and THOLD_CON indicating for each one:
  - *threshold*: is a real number in [0,1] that indicates the minimum degree that must satisfy the T-norm of the fuzzy comparation of the antecedent and consequent, i.e. it is the α value for the antecedent and β value for the consequent.

  It is an optional value because we can obtain it automatically with the DML sentence.
- *list_columns_gd_thold*: They are the specifications on the table columns *id_table_orig_project* prominent for the process to obtain (α_i)-(β_j) Fuzzy Global Dependencies (see Definition 6) with the (α_i) and (β_j) values prefixed for the user. It is necessary to do the specifications for antecedent columns col_ant_i with i=1..l (ANTECEDENT clausule) and for the consequent columns col_con_j with j=1..q (CONSEQUENT clausule). Such specifications are the same that for *list_columns_gd* above explained including for each column *id_column* the following:
  - *threshold*: is a real number in [0,1] that indicates the minimum degree that must satisfy the fuzzy comparation (using *fuzzy_comp*) of the antecedent and consequent, i.e. it is the α_i values for the antecedent and the β_j values for the consequent.

5.2 DML of dmFSQL for Fuzzy Global Dependencies

SELECT_MINING FGLOBAL_DEPENDENCIES

This sentence is the interface to obtain the GDs. The syntax is the following:

SELECT_MINING FGLOBAL_DEPENDENCIES *id_project*
USING {T_NORM | SINGLE} THOLD_ANT_CON[WITH CONFIDENCE HIGHEST] where

- *id_project*: project name. Previously, for this project a create (or alter) sentence must have been carried out including the clause WITH COLUMNS FOR […] FGLOBAL_DEPENDENCIES.

With this sentence we can obtain:

- α -β GD $X \rightarrow_{F*T*} Y$: if we use the clausule USING T_NORM THOLD_ANT_CON.
- (α_i)-(β_j) GD $X \rightarrow_{F*T*} Y$: if we use the clausule USING SINGLE THOLD_ANT_CON.

Verifying if the prefixed (in the DDL sentence) α -β or (α_i)-(β_j) values are fulfilled for the GD, or computing theses values for the confidence highest possible, if we specify the WITH CONFIDENCE HIGHEST clausule. Always, the confidence, support and α -β or (α_i)-(β_j) values are return by the sentence in form a SQL query on a default table.

5.3 Extension of the dmFSQL Architecture to obtain GDs

We have extended the architecture to permits us use dmFSQL language to obtain GDs. The modifications of the architecture (showed in Figure 1) are been on the dmFSQL server:

- Lexical and syntactic analysis to incorporate the new DDL and DML sentences.
- Semantic analysis to incorporate the process to obtain GDs above explained.

6 Experimental results

In this section we are going to apply the previously outlined process to find out behaviour patterns on stock-market sceneries. The goal is to identify those patterns which imply earnings of a specific stock. To do so, we are going to apply the previously outlined process which use DM techniques implemented through dmFSQL. Let SHARES_ENTERPRISE be a relation defined as:

SHARES_ENTERPRISE (*value_name, date, williams, MA, value*)

which contains data for a specific enterprise, Telefonica S.A., in the Spanish Stock Market. The data corresponds with a period of time between 01/01/2005 and 08/20/2006. The meaning of the attributes is:

- value_name: identifies the enterprise which the stock value belongs to. In this case, all rows shown in Table 2 belong to the same enterprise: Telefonica S.A.
- date: date of the data.
- williams: 14-day Williams's oscillator value.
- MA: 20-day moving average value.
- value: value in euros for the enterprise at the end of the session.

The step needed to solve the problem are:

- Identification of the ideal earnings scenarios, taking into account some of the previously detailed indicators used in technical analysis.
- Expert's theory formulation related to earning scenarios.
- Expert's theory validation for earning scenarios given the historical data stored.

Earning scenarios identification and expert's theory formulation

By means of graphical representations of the data (Figure 2) of the table SHARES_ENTERPRISE and the indicators outlined in first section the expert identifies as ideal earning scenery:

- Period: from 06/27/2006 to 07/09/2006.
- Situation: starting from date 06/27/2006, a change in the trend value has been produced with a value of 81.22 (cross with horizontal line 80). There is no significant change until 07/09/2006, when the value line cross the MA line and %R value with horizontal line 20.

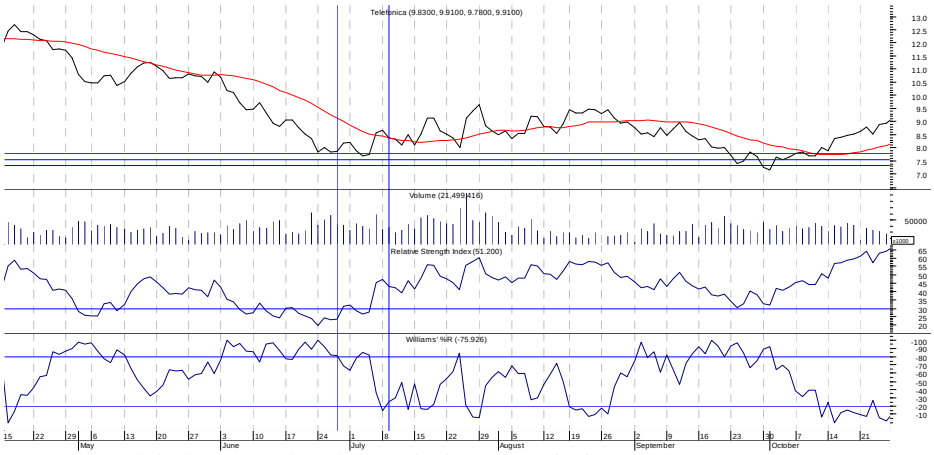

Figure 2. Telefonica S.A. values from 06/27/2006 to 07/09/2006.

So the expert formulates the theory about earning sceneries shown bellow: "Greater Williams index and roughly equal moving average implies a greater value for a specific enterprise"

Expert's theory validation through GDs using FSQL

First step consist in define the different attributes of table SHARES_ENTERPRISE in dmFMB:

- *williams*: although it is a crisp value, we decide to define it as Type 1. The value of the margin for approximate values has been determined as 10.
- *MA*: this is a crisp attribute but we decide define this as Type 1 in the dmFMB. The value of the margin for approximate values has been determined as 2.
- *value:* as well as MA attribute, this is a crisp attribute but we decide define this as Type 1 in the dmFMB. The value of the margin for approximate values has been determined as 2.

Table 2. Earning sceneries identified by the expert.

Value_name	Date	Williams	Expert's Interpretation	MA(20)	Value
TELEFONICA	27-jun-06	81,22	Purchase signal	9,64	8,19
TELEFONICA	28-jun-06	68,57		9,50	8,50
TELEFONICA	01-jul-06	63,01		9,36	8,54
TELEFONICA	02-jul-06	78,54		9,23	8,20
TELEFONICA	03-jul-06	84,95	Purchase signal	9,11	8,01
TELEFONICA	04-jul-06	82,26	Purchase signal	8,98	8,06
TELEFONICA	05-jul-06	36,02		8,88	8,92
TELEFONICA	08-jul-06	14,57	Sell signal	8,83	9,02
TELEFONICA	09-jul-06	25,37		8,79	8,73

Now, we define a project called SHARES by means a DDL sentence of dmFSQL (see Figure 3) and then we execute the Data Mining process by means a DML sentence (see Figure 4). The results of this sentence are showed in Figure 5. Thus, we can say that SHARES_TELEFONICA verifies:

(1, 1)–(1) GD (Williams, MA) → (FGT, FEQ)*(FGT)* (Value)

with confidence c=0.9. We can conclude that this GD is fulfilled with a sufficient degree and therefore he will be useful for future earnings sceneries identification.

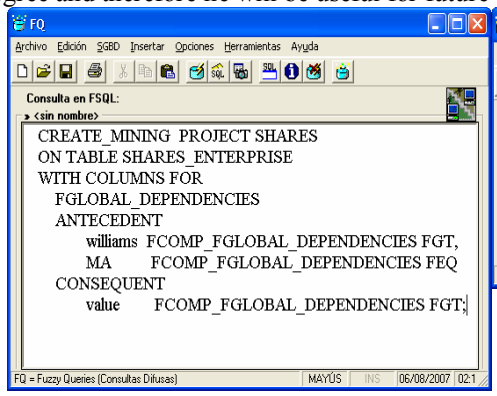

Figure 3. DDL sentence of dmFSQL

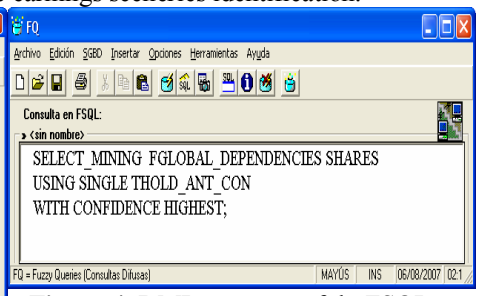

Figure 4. DML sentence of dmFSQL

Figure 5. Results of the DML sentence showed in Figure 4

7 Conclusion

We have extended the dmFSQL language and its architecture to obtain fuzzy global dependencies (GDs) as a common framework to integrate fuzzy and gradual functional dependencies [1] on any type of data. Now, this language [3] integrates flexible queries, clustering, characterization, fuzzy classification and fuzzy global dependencies techniques explained in [2]. This architecture has been designed and extended considering the desirable functionalities of DM systems [6]:

- Handling of Different Types of Data: dmFSQL permit to do mining on any type of data.
- Interactive Mining Knowledge: dmFSQL comply the property of closure, that is, the possibility to apply the language with a prior result of this same language. This allows the user to refine a DM request on line.
- Efficiency: dmFSQL has been designed to give the answer in real time. This is possible because a new object introduced in the language called *project*. It is the backup to keep the initial conditions, intermediate and ends results of the DM process to carry out. These intermediate results permit improve the performance of the iterative DM process.
- Friendly Interface: actually we have several client programs that work on dmFSQL architecture.

References

1. Carrasco, R.A., Vila, M.A., Galindo,J., Cubero, J.C.: "FSQL: a Tool for Obtaining Fuzzy Dependencies". 8th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, IPMU'2000, pp. 1916-1919. Madrid (Spain), July 2000.
2. Carrasco, R.A., Vila, M.A., Galindo, J.: "FSQL: a Flexible Query Language for Data Mining". In Enterprise Information Systems IV, eds. M. Piattini, J. Filipe and J. Braz, pp. 68-74. Ed. Kluwer Academic Publishers, 2002, ISBN: 1-4020-1086-9.
3. Carrasco, R.A., Vila, M.A., Araque F.:"dmFSQL: a Language for Data Mining" DEXA'2006, pp. 440-444, 2006 ISBN:0-7695-2641-1
4. Cubero, J.C., Vila, M.A. "A new definition of fuzzy functional dependency in fuzzy relational databases" International J. Intelligent Systems, Vol 9, pp. 441-449, 1994.
5. Dubois, D. and Prade, H., "Gradual rules in approximate reasoning". Information Sciences, Vol 61, pp. 103-122, 1992.
6. Frawley, W.J., Piatetsky-Shapiro, G., Matheus, C.J.: Knowledge Discovery in Databases: An Overview. In G. Piatetsky-Shapiro, W.J. Frawley eds. Knowledge Discovery in Databases pp. 1-31, The AAAI Press 1991.
7. Imielinski, T., Virmani, A.: MSQL: "A Query Language for Database Mining". In Data Mining and Kowledge Discovery III, eds. Fayyad, Mannila and Ramakrishnan, pp. 373-408. Ed. Kluwer Academic Publishers, 1999
8. Meier, A., Savary, C., Schindler, G., Veryha, Y.: "Database Schema with Fuzzy Classification and Classification Query Language". Technical report fCQL. Information Systems Research Group. University of Fribourg. Switzerland (2001).
9. Meo, R., Psaila, G., Ceri, S.: "A New SQL-like Operator for Mining Association Rules". 22nd International Conference on Very Large DataBases, VLDB'96, pp.122-133. Bombay (India), 1996.

About the Processing of Division Queries Addressed to Possibilistic Databases

Patrick Bosc, Nadia Iben Hssaien, Olivier Pivert

IRISA/ENSSAT, Technopole Anticipa, BP 80518,
22305 Lannion Cedex, France
{bosc, ibenhssaien, pivert}@enssat.fr

Abstract. This paper is situated in the area of possibilistic databases. Any possibilistic database has a canonical interpretation as a set of more or less possible regular databases, also called worlds. In order to manipulate such databases in a safe and efficient way, a constrained framework has been previously proposed, where a restricted number of querying operations are permitted (selection, union, projection and foreign key join which can handle attributes taking imprecise values). The key for efficiency resides in the fact that these operators do not require to make computations explicitly over all the more or less possible worlds. The division operation is dealt with in this paper.

Keywords: Ill-known values, possibilistic databases, database querying, strong representation system, world-based interpretation.

1 Introduction

In different application domains, one can notice a growing need for information systems capable of dealing with ill-known data. It is the case, for example, of data warehouses for which the gathering of information coming from different sources may induce imprecision (up to now, many such systems deal with this issue by means of usual "data cleaning" methods, at the expense of a loss of information). Different formalisms can be used to represent imprecise information (see for instance [7] and [11]), and the possibilistic setting is assumed in the rest of the paper. We thus consider databases where ill-known values are represented by means of possibility distributions. Let us recall that from a semantic point of view, a possibilistic database D can be interpreted as a weighted disjunctive set of regular databases (also called worlds) denoted by rep(D) (see e.g. [1] for a general presentation of the world-based semantics of imprecise databases). Each database (or world W) of rep(D) is obtained by choosing a unique candidate value in each imprecise attribute value in D. It implies that independence of imprecise values is assumed (i.e., any candidate can be taken whatever the other choices are). Each world W is associated with a degree of possibility corresponding to the minimum of the degrees tied to the candidates appearing in it (due to the property $\Pi(A \cap B) = \min(\Pi(A), \Pi(B))$, which applies when A and B are non-interactive events).

Example 1. Let us take a relation emp whose scheme is EMP(#id, years, job) where #id is an identifying number assigned to each employee, years stands for the number of years spent abroad and job is the last job he held. The two attributes years and job are assumed to be imprecise. The following extension:

emp

#id	years	job
7	{1/5 + 0.8/4 + 0.3/3}	cashier
23	{1/8 + 0.5/9}	{1/manager + 0.6/cashier}
2	2	salesman

can be associated with 12 (3 choices in the first tuple and 2 * 2 in the second) more or less possible worlds among which the world:

#id	years	job
7	4	cashier
23	8	manager
2	2	salesman

which is 0.8 (min(0.8, 1, 1)) possible. ♦

Of course, it would be appealing to be able to address queries as general as possible against such databases (according to the intuitive idea of extending the relational operations allowed in the context of regular databases), but it turns out that this is not obvious to do in the presence of imprecise attribute values. Indeed, the only general way to evaluate a query containing any relational operation would be to process it against each world. This approach is sound but raises the issue of tractability due to the huge number of worlds (as well as the issue of representing the result). However, it remains the semantic reference, i.e., it constitutes the basis that enables to found the validity of any other approach. In [5], we have devised a data model which is a strong representation system (see property (1) in the following section) for four operations: selection, union, projection and fk-join.

The key point is to evaluate queries in a "compact" way, i.e. directly on possibilistic relations, without computing the more or less possible worlds, with the guarantee that the result of a query Q is the same as that of Q over the set of the regular databases issued from the initial possibilistic database D.

In the present paper, we deal with the division operation in the framework of that model. We consider the division of relation r of schema R(A, X) by relation s of schema S(B) where A and B are compatible attributes (i.e. are defined on the same domains) and we focus on two particular cases: 1) attributes X and A are precise and B is imprecise and 2) attributes X and B are precise and A is imprecise. This will enable to enrich the set of the operators that can be used within the possibilistic database model previously defined and to express a wider range of queries.

The remainder of the paper is organized as follows. Section 2 is devoted to a brief presentation of the relational possibilistic model that will be used later. The operation of division in the possibilistic context, which is the heart of this contribution, is introduced and discussed in section 3, and an algorithm for a compact evaluation for

each case is proposed and justified. Section 4 is devoted to some related works. Some conclusions are given in section 5, as well as some lines for future works.

2 An Extended possibilistic Model

2.1 Objective

As mentioned before, a calculus based on the processing of a query Q against worlds is intractable and a compact approach to the calculus of the answer to Q must be found out. It is then necessary to be provided with both a data model and operations which have good properties: i) the data model must be closed for the considered operations, and ii) it must be possible to process any query (applying to the possibilistic database D) in a compact and sound way, i.e., the result must be a compact representation of the results that would be obtained if the query were applied to all the interpretations (worlds) drawn from D, i.e.:

$$rep(Qc(D)) = Q(rep(D)) . \quad (1)$$

where rep(D) denotes the set of worlds associated with D and Qc stands for the query obtained by replacing the operators of Q by their compact versions. This property characterizes data models called strong representation systems.

It turns out that the relational possibilistic model illustrated in example 1 cannot comply with this property (see [5] for a detailed justification of this assertion). An adapted data model, which has been introduced in [5], is briefly described hereafter.

2.2 Representing Possibly Missing Tuples

Because some operations (e.g. selection) filter candidate values, there is a need at the compact level for expressing that some tuples can have no representative in some worlds. A simple solution is to introduce a new attribute, denoted by N (valued in [0, 1]), which states whether or not it is legal to build worlds where no representative of the corresponding tuple is present, and, if so, the influence of this choice in terms of degree of possibility. The value of N associated with a tuple t expresses the certainty (as defined in the possibility theory) of the presence of a representative of t in any world. A tuple is denoted by a pair N/t where N equals 1 for tuples of initial possibilistic relations as well as when no candidate value has been discarded.

Example 2. Let us consider the following extension of the possibilistic relation im:

im

#i	ap	date	place
i_1	B-727	d_1	p_1
i_2	ATR-72	d_1	p_2
i_3	{1/B-727 + 0.7/ATR-42}	d_2	p_4
i_4	{1/B-727 + 1/B-747}	d_2	p_2

which is assumed to describe satellite images of aircrafts. Each image is supposed to represent a single aicraft (whose type ap may be ill-known due to the imprecise nature of the recognition process) and has been taken in a certain place on a certain date. The selection based on the condition "ap = B-727" leads to discard the candidates which are different from this desired value. Thanks to the introduction of attribute N, the result of the selection is:

res

#i	ap	date	place	N
i_1	B-727	d_1	p_1	1
i_3	B-727	d_2	p_4	0.3
i_4	B-727	d_2	p_2	0

In the second tuple N is equal to 0.3, i.e, 1 minus the possibility degree attached to the most possible alternative that has been discarded. From this result, it is possible to derive the world made of the single tuple $<i_1$, B-727, d_1, $p_1>$ whose degree of possibility is: min(1, 1 – 0.3, 1 – 0) = 0.7. ♦

2.3 Multiple Attribute Possibility Distributions

Another aspect of the model is related to the fact that it is sometimes necessary to express dependencies between candidate values coming from different attributes in a same tuple. This requires that the model incorporates attribute values defined as possibility distributions over several domains. This is feasible in the relational framework thanks to the concept of a nested relation. In such relations, exclusive candidates are represented as weighted tuples. Therefore, level-one relations keep their conjunctive meaning, whereas nested relations have a disjunctive interpretation.

Example 3. Let us consider the following intermediate relation int-r involving the nested attribute X(date, place):

int-r

#i	ap	X		N
		date	place	
i_1	B-727	$\{1</d_1, p_1> + 0.7/<d_1, p_2> + 0.4/<d_3, p_2>\}$		1
i_3	B-727	$<d_1, p_2>$		0.3
i_4	{0.4/B-737}	$\{0.3/<d_3, p_2>\}$		0

This relation is associated with 12 worlds since the first tuple admits 3 interpretations, the second and third ones have two interpretations among which Ø (no representative). ♦

In order to meet the objective of a compact processing of algebraic queries, the operators must be adapted so as to accept compact relations both as inputs and outputs. It turns out that only operations such that an input tuple participates in the production of at most one element of the result, can be expected to admit a compact version (see [4] for a justification). As a consequence, the intersection, the difference and the Cartesian product (then the join in the general case) are discarded and the four acceptable operators are: the selection, the projection, the fk-join (a specific join) and

the union. In this paper, we show that we can also deal with the division operation (with some restrictions).

2.4 A Brief Survey of the Four Operations

Due to space limitations, we limit ourselves to a brief introduction of the operators and their behaviour is then illustrated by an example. See [3, 5] for more details about the definition of these operators.

Unary operations

The three aspects of the selection are: the removal of unsatisfactory candidate values, the computation of the degree of certainty attached to each output tuple and the introduction of appropriate nested relations in the output relation if needed.

The role of the projection in the regular case is to remove undesired attributes. Here, the projection must: 1) keep the duplicates in level-one relations (this is justified in [5]), 2) suppress nested relations if necessary, 3) update the possibility degrees.

Binary operations

Beyond selections and projections, two binary operations can be processed in a compact fashion: fk-join and union. The fk-join allows for the composition of a possibilistic relation r of schema R(W, Z), where W and Z may take imprecise values, and a regular relation s whose schema is S(W, Y) where the functional dependency W → Y holds. It consists in completing tuples of r by adding the image of the W-component. By definition, this leads to a resulting relation involving the nested relation X(W, Y), which "connects" the pairs of candidates over W and Y.

Last, the union of two independent relations whose schemas are compatible keeps all the tuples issued from the two input relations without any duplicate removal (see [3] for more details and a justification).

Example 4. Let us consider the possibilistic database composed of the relations im1(IM), im2(IM) and pl(PL) whose respective schemas are IM(#i, ap, date, place) and PL(ap, lg, msp). The relations im1 and im2 are assumed to contain images of airplanes taken by two distinct satellites and each image, identified by a number (#i), taken on a certain location (place) a given day (date) is supposed to include a single (possibly ill known) airplane (ap). Relation pl gives the length (lg) and maximal speed (msp) of each airplane and is a regular relation.

pl

ap	lg	msp
a_1	20	1000
a_2	25	800
a_4	20	1200
a_5	20	1000

im1	#i	ap	date	place	N
	i_1	a_3	$\{1/d_1 + 0.7/d_3\}$	c_1	1
	i_2	$\{1/a_2 + 0.7/a_1\}$	d_1	c_2	1

im2	#i	ap	date	place	N
	i_3	$\{1/a_4 + 1/a_5\}$	$\{0.6/d_4 + 1/d_1\}$	c_3	1

Let us consider the query looking for the existence of images of airplanes whose maximal speed is over 900 km/h and taken by either of the two satellites at a date different from d_3 and d_4, which corresponds to the algebraic query Q: fk-join(union(select(im1, date $\notin \{d_3, d_4\}$), select(im2, date $\notin \{d_3, d_4\}$)), select(pl, msp > 900), {ap}, {ap}). With the extensions above, we obtain the resulting relation res hereafter:

res	#i	X Ap lg msp	date	place	N
	i_2	$\{0.7/<a_1, 20, 1000>\}$	d_1	c_2	0
	i_3	$\{1/<a_4, 20, 1200> + 1/<a_5, 20, 1000>\}$	$\{1/d_1\}$	c_3	0.4

which is associated with 6 worlds, among which the empty one.♦

3 Division of Possibilistic Relations

3.1 Introduction and example

The relational division, i.e., the division of relation r of schema R(A, X) by relation s of schema S(B) where A and B are sets of compatible attributes is defined as:

$$\text{div}(r, s, A, B) = \{x \mid \forall a \in s, <a, x> \in r\}. \quad (2)$$

In other words, the division of r by s returns the elements x which are associated in r with at least all the values a appearing in s.

In the presence of imprecise data (represented as possibility distributions), we conjecture that the division in the general case (division of a relation r of schema R(A, X) by a relation s of schema S(B), A and B being compatible attributes and the atributes A, B and X being possibly imprecise) cannot be processed without computing all of the interpretations of the pair of relations involved, which would lead to an exponential complexity. The following example illustrates the world-based processing method. For the sake of readability, attribute X is considered to be precise.

Example 5. Let us consider a possibilistic database B involving the two relations r and s represented below. Relation r has the following two interpretations:

$r_1 = \{<x_1, a_1>, <x_1, a_2>, <x_2, a_1>, <x_2, a_3>\}$ with possibility degree $\Pi_1 = 1$
$r_2 = \{<x_1, a_3>, <x_1, a_2>, <x_2, a_1>, <x_2, a_3>\}$ with possibility degree $\Pi_2 = 0.6$

while s has the following four interpretations:

$s_1 = \{<a_1>, <a_2>\}$ with possibility degree $\Pi'_1 = 0.4$
$s_2 = \{<a_1>, <a_3>\}$ with possibility degree $\Pi'_2 = 0.2$
$s_3 = \{<a_2>\}$ with possibility degree $\Pi'_3 = 1$
$s_4 = \{<a_2, a_3>\}$ with possibility degree $\Pi'_4 = 0.2$.

r

X	A
x_1	$\{1/a_1 + 0.6/a_3\}$
x_1	a_2
x_2	a_1
x_2	a_3

s

B
$\{0.4/a_1 + 1/a_2\}$
$\{1/a_2 + 0.2/a_3\}$

The value x_1 belongs to the result of the division of r by s in the worlds (r_1, s_1), (r_1, s_3), (r_2, s_3), (r_2, s_4). Thus, the possibility degree attached to x_1 in the result of the division equals:

$\max(\min(\Pi_1, \Pi'_1), \min(\Pi_1, \Pi'_3), \min(\Pi_2, \Pi'_3), \min(\Pi_2, \Pi'_4)) = 1.$

On the other hand, x_1 does *not* belong to the result of the division of r by s in the worlds (r_1, s_2), (r_1, s_4), (r_2, s_1), (r_2, s_2). Thus, according to the axioms of possibility theory, the necessity (certainty) degree attached to x in the result of the division is equal to:

$1 - \max(\min(\Pi_1, \Pi'_2), \min(\Pi_1, \Pi'_4), \min(\Pi_2, \Pi'_1), \min(\Pi_2, \Pi'_2)) = 1 - 0.4 = 0.6.$

As to x_2, according to the same type of calculus, its possibility degree is equal to 0.2. According to the axioms of possibility theory, one has $\Pi(E) < 1 \Rightarrow N(E) = 0$ for a Boolean event E, consequently the necessity degree attached to x_2 in the result of the division is zero.♦

Let us notice incidentally that a processing strategy based on the algebraic rewriting of the division operation, i.e.,

$$\text{div}(r, s, A, B) = \text{project}(r, X) - \text{project}((\text{project}(r, X) \times s) - r, X)$$

is not exploitable, since the Cartesian product (×) and the difference (–) are not operators for which the database model is a strong representation system.

In this paper, as a first step towards a tractable treatment of the division in an imprecise database context, we limit ourselves to the two particular cases where X is a precise attribute and either A or B is an imprecise one, and we point out a way to deal with this operation which does not compute the different worlds and complies with the characteristic property of a strong representation system. The result of the division computed thanks to this approach is the same as that delivered by the world-based approach. Indeed, it is a relation containing the elements x associated with two degrees Π and N (possibility and necessity) expressing the extent to which they possibly (resp. certainly) belong to the result of the division. The elements x appearing in the result are the ones such that $\Pi \neq 0$, and the degrees Π and N computed are, respectively, equal to the possibility of the most possible world where x belongs to the result of the division and 1 minus that of the most possible world

where x does not belong to the result of the division. Both cases (X is precise and either A or B is imprecise) and the underlying approaches are detailed in the following subsections.

3.2 Case 1: only attribute B is imprecise

This is the most simple of both cases. To deal with the division operation in this case, we propose a linear algorithm aiming at computing, for every element x of r, the degree of possibility Π that it belongs to the result of the division of r by s. The principle of this algorithm consists, for a given x of r, in scanning the different B-values in s. Let us denote by $scv(b_i)$ the set of (precise) candidate values corresponding to the interpretations of an imprecise B-value b_i in s, and by A(x) the set of all the A-values associated with x in r (A(x) can be seen as the result of the query "select A from r where X = x"). The candidate values $b_{i,j}$ from $scv(b_i)$ are assumed to be ranked in decreasing order on their possibility degrees $\pi_{i,j}$. For each imprecise b_i from s, the algorithm checks whether x is associated in r with at least one $b_{i,j}$ from $scv(b_i)$ and if so, it memorizes the highest possibility degree attached to such a value. Ultimately the final possibility degree attached to x in the result of the division is given by:

$$\Pi(x) = \min_{b_i \in s} \max_{b_{i,j} \in scv(b_i) \cap A(x)} \pi_{i,j}. \quad (3)$$

In fact, for formula (3) to be correct, an extra "virtual" candidate value with degree 1 – N, where N denotes the necessity degree associated with the tuple containing the B-value b_i in s, must be added to $scv(b_i)$. Indeed, one must take into account the case where the considered tuple of the divisor may have no representative, case whose possibility is equal to 1 – N. In the algorithm below, this situation is dealt with by means of the instruction $\Pi \leftarrow \max(\Pi, 1 - t.N)$.

As to the necessity degree, it is zero if the degree Π obtained is less than 1, otherwise, it is equal to 1 minus the possibility degree of the most possible candidate value, appearing in s, that is not associated with x in r.

$$N(x) = 1 - \max_{b_i \in s} \max_{b_{i,j} \in scv(b_i) - A(x)} \pi_{i,j}. \quad (4)$$

The algorithm given hereafter has to be performed for every element x of r.
Input: relation r and s whose generic tuples are respectively <x, a_i> and <b_j>, with $b_j = \{\pi_{j,1}/a_1 + \ldots + \pi_{j,n}/a_n\}$ and $\pi_{j,1} \geq \ldots \geq \pi_{j,n}$;
Output: degree of possibility Π that x belongs to the result of the division of r by s;
An imprecise value b_j is seen as a structured type composed of i) nb the number of its candidate values, and ii) the arrays val and poss containing, respectively, these candidate values and the associated possibility degrees. Besides, for each tuple t of s, the associated necessity degree is stored in t.N.

Body of the algorithm:

```
Π ← 1; PossMiss ← 0;
while not end(s) and Π ≠ 0 do
    read next tuple t of s;
    i ← 1; found1 ← false; found2 ← false; Πlocal ← 1;
```

```
    while i ≤ t.B.nb and not (found1 and found2) do
        present ← (<x, t.B.val[i]> ∈ r);
        if not found1 and present then
            Πlocal ← t.B.poss[i];
            found1 ← true
        else
            if not found2 and not present then
                PossMiss ← max(PossMiss, t.B.poss[i]);
                found2 ← true
            endif;
        endif;
        i ← i + 1;
    enddo;
    if not found1 and t.N = 1 then Π ← 0
    else
       Πlocal ← max(Πlocal, 1 - t.N);
       Π ← min(Π, Πlocal);
    endif;
enddo;
N ← 1 - PossMiss endif;
```

In order to prove the validity of the method, we have to show that the algorithm above provides the same result as a processing approach based on the computation of worlds, i.e., that the degrees Π and N computed by this algorithm are, respectively, those of the most possible world where x belongs to the result of the division and 1 minus the possibility degree of the most possible world where x does not belong to the result of the division. This is quite straightforward, by construction. Indeed, the most possible world where x belongs to the result of the division is obtained by choosing, in each tuple of s, the candidate value (if it exists) which is associated with x in r and which has the highest possibility (taking also into account the possibility of no representative for the tuple in s). The possibility of this world is the minimum of the degrees tied to the candidate values retained for its construction, and this is indeed what the algorithm computes. As to the most possible world where x does not belong to the result of the division, it is obtained by choosing i) the most possible candidate value $b_{i,j}$ appearing in s for which no association ($b_{i,j}$, x) exists in r (let us call t_j the tuple where $b_{i,j}$ appears), and ii) candidate values (including ∅ if the tuple may have no representative) with possibility 1 in all tuples other than t_j. This world is $\pi_{i,j}$ possible, $\pi_{i,j}$ being the possibility degree attached to the value $b_{i,j}$). Again, it is straightforward to see that the algorithm computes the correct value of the necessity, since it stores in the variable PossMiss the degree attached to the most possible B-value that is not associated with x in r, and finally delivers 1 – PossMiss.

Example 6. Let us consider the following relations r and s.

r

X	A
x_1	a_2
x_1	a_3
x_2	a_2
x_2	a_3
x_2	a_4

s

B	N
$\{1/a_2 + 0.7/a_3 + 0.4/a_1\}$	0.2
$\{1/a_2 + 0.2/a_4\}$	1
a_3	1

One gets the following results. For x_1: $\Pi = \min(\max(1, 0.7, 1 - 0.2), 1, 1) = 1$ and $N = 1 - \max(0.4, 0.2) = 0.6$. As a matter of fact, the most possible world where x_1 belongs to the result of the division is made of both r and the interpretation of s containing the tuples $<a_2>$ and $<a_3>$ whose possibility degree equals $\min(1, 1, 1) = 1$. As to the most possible world where x_1 does not belong to the result of the division, it is made of both r and the interpretation of s containing the tuples $<a_1>$, $<a_2>$ and $<a_3>$ whose possibility degree equals $\min(0.4, 1, 1) = 0.4$. Similarly, for x_2, one gets the degrees $\Pi = \min(\max(1, 0.7), \max(1, 0.2), 1) = 1$ and $N = 1 - 0.4 = 0.6$.♦

3.3 Case 2: only attribute A is imprecise

In this second case, for the calculus of the possibility degree Π, the algorithm we propose carries out a treatment for every element x of r, which consists in the following steps:

1) one performs a selection on relation r by means of the selection: "t.X = x and t.A ∈ s[B]", which is intended for retaining only the value x and the A-values appearing in s; let us call the relation we obtain r_x. If relation r_x contains a number of tuples which is less than the cardinality of relation s, the possibility and the necessity that x belongs to the result of the division of r by s are obviously zero.

2) one numbers the tuples of relation r_x (r_x is provided with an additional attribute num).

3) one "develops" every tuple of relation r_x. Every imprecise tuple of r_x is replaced by its different interpretations. For instance, a tuple $t_i = <\{a_1/\pi_1 + \ldots + a_n/\pi_n\}, x, num_i>$ of r_x will give birth to the precise tuples $<a_1, x, num_i, \pi_1>$, $<a_2, x, num_i, \pi_2>$, …, $<a_n, x, num_i, \pi_n>$.
The information num is used to solve the problem raised by the existence of dependencies between the interpretations of a same imprecise tuple. Indeed, two tuples having the same number cannot coexist in a same world.

Remark. Attribute N is not taken into account since it is not needed for the calculus of the final possibility degree Π in this case.

4) The tuples of r_x are ranked in decreasing order on the possibility degrees:

$$(val_1, x, num_1, \pi_1), (val_2, x, num_2, \pi_2), \ldots \text{ with } \pi_i < \pi_{i+1}.$$

5) Let us denote by k the cardinality of s. The degree of possibility Π attached to an element x can be deduced from r_x the following way. We compute the most possible set D_0 containing k tuples of r_x and satisfying the property "D_0 contains all the values present in relation s with different numbers".

The algorithm for the calculus of the degree of possibility is given hereafter.

Input: relation r_x whose tuples are of the form $t = <val_i, x, num_i, \pi_i>$;

Output: degree of possibility Π that the element x belongs to the result of the division of r by s.

Let us denote by n_x the cardinality of relation r_x, i.e., the number of interpretations of x, and define the relation $\Re$ which expresses "to be different":

$$(val, x, num, \pi)\ \Re\ (val', x, num', \pi') \text{ iff } val \neq val' \text{ and } num \neq num'. \quad (5)$$

This algorithm maintains sets of tuples, each of them, denoted by A_i, associated with an incomplete world (in the sense that a choice for each tuple of r has not been made yet). It guarantees that the set found first (thanks to step 4) containing k tuples which are all pairwise different in the sense of relation $\Re$ is the most possible one and then its possibility is that of the event "x belongs to div (r, x, A, B)";

Body of the algorithm:

```
Result ← false;
for each tuple t_i (i from 1 to n_x) of r_x do
  A_i ← {t_i};
  for each set A_j (j from 1 to i - 1) do;
    if t_i is in relation ℜ with the elements of A_j then
      add t_i to A_j endif;
    if card(A_j) = k then
      result ← true;
      Π ← t_i.π;
      exit;
    endif;
  endfor;
endfor;
if not result then Π = 0 endif;
```

As to the necessity degree, it is computed the following way:

- in the case where the possibility degree computed thanks to the previous algorithm is less than 1, the necessity degree will be 0;

- otherwise, the necessity that a given x belongs to div(r, s, A, B) is that of the event "the tuples of s are included in $r_x[A]$" where r_x is obtained from step 1 and $r_x[A]$ is the projection of r_x on attribute A.

Let us denote by $\{s_1, s_2, \ldots, s_k\}$ the tuples present in relation s. According to the axioms of possibility theory, this necessity degree is equal to:

$$\begin{aligned} N(s_1 \in r_x[A] \text{ and } \ldots \text{ and } s_k \in r_x[A]) &= 1 - \Pi(s_1 \notin r_x[A] \text{ or } \ldots \text{ or } s_k \notin r_x[A]) \\ &= 1 - \max(\Pi(s_1 \notin r_x[A]), \ldots, \Pi(s_k \notin r_x[A])) \\ &= 1 - \max(1 - N(s_1 \in r_x[A]), \ldots, 1 - N(s_k \in r_x[A])). \end{aligned}$$

In [6], we have proposed a linear complexity algorithm in order to compute the necessity that a given tuple t belongs to the result of a query Q ($N(t \in res(Q))$, here $r_x[A]$ plays the role of res(Q). Thus, the computation of N in the case considered here amounts to running k times this algorithm.

In order to demonstrate the validity of the proposed approach, we have to compare the result (possibility and necessity degrees) obtained thanks to the algorithms above with that obtained by means of the processing technique based on worlds.

The final possibility degree Π is found by considering the first k tuples of r_x which are all pairwise different in the sense of relation $\Re$ ($\Pi = \pi_k$). Let us call E the set made of these k tuples. One can build a world made of the tuples from E with an implicit completely possible choice ($\pi = 1$) for the other tuples (including the choice Ø). This world is the most possible one where x belongs to the division. Indeed, the k^{th} tuple (val_k, x, num_k, π_k) makes the property "E contains k tuples that are all different in the sense of relation $\Re$" become true (it was not true at the previous step). By construction (due to the order which generates the most possible worlds first), this world is the most possible one which satisfies the property.
See [6] for the proof related to the computation of the necessity.

Given that the division operation is considered, one must make sure that the result obtained is a quotient. Let us recall that the term "division" given to this operation rests on the fact that the relation t resulting from the division of r by s has the double characteristic of a quotient, namely:

$$1) \quad s \times t \subseteq r \text{ and } 2) \ \forall t', (t \subset t') \Rightarrow ((s \times t') \not\subseteq r).$$

The approach we propose here is equivalent to the one based on worlds. This equivalence allows us to state that the result delivered by our division is also a quotient since the property of quotient of the result of a division holds in each world.

As for the complexity of the proposed approach, it is of course much more reasonable than that of the approach based on the computation of worlds. For instance, if relation r_x contains p tuples and nc candidate values per possibility distribution for attribute A, the world-based approach will necessitate nc^p usual divisions of an interpretation of r by s, versus only $(p*nc)^2$ (at worst) tests of insertion into a set A_i (cf. the algorithm above) for the approach presented in this paper.

Example 7. Let us consider a company which owns warehouses in different French towns. These warehouses can store products from different departments. We have the following possibilistic database composed of the relations warehouse W and department D whose respective schemas are W(town, counter) and D(counter).

W

town	counter
Paris	{0.3/drink + 1/canned food}
Marseille	{1/home appliance + 0.4/fresh food}
Grenoble	{1/drink + 0.2/fresh food}
Paris	{1/canned food + 0.3/home appliance}
Paris	fresh food
Lyon	Home appliance

D

counter
fresh food
canned food
home appliance

Let us also consider the following query involving the division operator: "to which extent is it possible and certain that the warehouse located in Paris is able to supply all the counters?"

We perform, on relation W, the selection based on the condition "town = "Paris" and counter in D", and then we number each tuple, which leads to:

W_P

town	counter	num	N
Paris	{1/canned food}	1	0.7
Paris	{1/canned food + 0.3/home appliance}	2	1
Paris	fresh food	3	1

Each tuple in relation W_P is developed and the tuples obtained are ranked in decreasing order on the possibility degrees:

town	counter	num	Π
Paris	canned food	1	1
Paris	canned food	2	1
Paris	fresh food	3	1
Paris	home appliance	2	0.3

The algorithm runs as follows:

Once the first two rows have been accessed, we have:
A_1 = {(Paris, canned food, 1, 1)}
A_2 = {(Paris, canned food, 2, 1)}
The tuple <Paris, canned food, 2, 1> is not in relation $\Re$ with the element of A1.

The tuple <Paris, fresh food, 3, 1> is accessed. We have:
A_3 = {(Paris, fresh food, 3, 1)}
The tuple <Paris, fresh food, 3, 1> can be added to A_1 and A_2 since it is in relation $\Re$ with their elements, we thus have:
A_1 = {(Paris, canned food, 1, 1), (Paris, fresh food, 3, 1)}
A_2 = {(Paris, canned food, 2, 1), (Paris, fresh food, 3, 1)}

Then we access <Paris, home appliance, 2, 0.3> and we get:
A_4 = {(Paris, home appliance, 2, 0.3)}
This tuple can be added to A_1 (but not to A_2):
A_1 = {(Paris, canned food, 1, 1), (Paris, fresh food, 3, 1),
(Paris, home appliance, 2, 0.3)}
The cardinality of A_1 is 3 (the cardinality of relation D), the possibility that Paris belongs to the division is thus 0.3. It can be checked that it corresponds to the world containing the following interpretation of relation W:

town	counter
Paris	canned food
Paris	home appliance
Paris	fresh food

which is possible at degree 0.3, and which is the most possible one where Paris belongs to the result of the division. The necessity degree is zero since the possibility degree is less than 1. ♦

4 Related Works

The division operator in the context of fuzzy relations has been studied by several researchers, mainly in the case of fuzzy relations (associated with a gradual concept) involving only crisp values (see for instance [7, 8, 9, 12, 14]). However, a few authors have studied the division operator in the context considered here, i.e., that of relations with imprecise attribute values described by possibility distributions. Both [10] and [13] consider the division of r of schema R(A, X) by relation s of schema S(B) where A and B are imprecise attributes and X is a precise one. Unfortunately, none of these approaches is consistant with the world-based semantics of imprecise databases, in other terms it does not satisfy property (1). Let us consider the following example:

r

X	A
x	{1/a + 1/b}

s

B	N
a	1
b	1

According to both approaches – described in [10] and [13] –, the possibility that x belongs to the result of the division equals 1 whereas the result of the division is empty in the two worlds associated with relations r and s. The reason why these approaches are not sound (with respect to the world-based semantics) is that they ignore the disjunctive interpretation underlying possibility distributions. More precisely, the calculus only takes into account the fact that a value x is possibly associated in relation r with values a and b but ignores the (crucial) fact that it can not be associated with both values at the same time.

5 Conclusion

In this paper, we have considered the issue of querying possibilistic databases by means of a set of appropriate algebraic operators. First, we have recalled the main characteristics of an imprecise database model enabling to process such algebraic queries in a "compact" way, and which is a strong representation system for four querying operations: selection, union, projection and foreign-key join. Then, we have proposed a compact approach – which complies with the characteristic property of a strong representation system – to the processing of division queries in the framework of that model. The strategy proposed makes it possible to evaluate in a tractable way the division of a relation r of schema R(A, X) by a relation s of schema S(B) in the particular cases where X is a precise attribute and either A, or B is an imprecise one. Even though it is implied by the property of a strong representation system, it might be worth mentioning that in both cases, the division operator obtained can be composed with the other algebraic operators of the model since its result is represented by a table of the database model considered (every tuple of the result is of the form $<N_i/<\pi_i/x_i>>$).

As to future works, it is of interest to pursue this study in order to check whether the general case, i.e., where both relations r and s are imprecise, could be dealt with in a "compact" way at least in some particular situations. We also plan on implementing

the division operator defined in this paper so as to perform some experimental measures aimed at assessing the extra cost induced by the presence of imprecise values, with respect to a division query processed on a classical database. Another direction of research concerns the influence of the model of uncertainty chosen to represent ill-known information. More precisely, one may wonder if the approach proposed in this paper would still be valid in a probabilistic database framework.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of databases, Addison-Wesley, 1995.
2. Bosc, P., Dubois, D., Pivert, O., Prade, H.: Flexible queries in relational databases – The example of the division operator. Theoretical Computer Science, vol. 171 (1997), 281-301
3. Bosc, P., Pivert, O.: Towards an algebraic query language for possibilistic relations. 12th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'03). (2003) 671-676
4. Bosc, P., Pivert, O.: On a strong representation system for imprecise relational databases. 10th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU'04), (2004)1759-1766
5. Bosc, P., Pivert, O.: About projection-selection-join queries addressed to possibilistic relational databases. IEEE Transactions on Fuzzy Systems, vol. 13 (2005), 124-139
6. Bosc, P., Pivert, O.: About Yes/No Queries against Possibilistic Databases. International Journal of Intelligent Systems, vol. 22 (2007), 691-722
7. Bosc, P., Prade, H.: An Introduction to Fuzzy Set and Possibility Theory-Based approaches to the Treatment of Uncertainty and Imprecision in Database Management Systems. In: Uncertainty Management in Information Systems – From needs to solutions. Motro A. and Smets P. Eds Kluwer Academic Publishers, (1997) 285–324
8. Cubero, J.C., Medina, J.M., Pons, O., Vila, M.A.: The generalized selection: an alternative way for the quotient operations in fuzzy relational databases. Int. Conference on Information Processing and Management of Uncertainty in Knowledge-based Systems (IPMU'94), (1994) 23-30
9. Dubois, D., Prade, H.: Semantics of quotient operators in fuzzy relational databases. Fuzzy Sets and Systems, vol. 78 (1996) 89-94
10. Galindo, J., Medina, J.M., Aranda Garrido, M.C.: Fuzzy Division in Fuzzy Relational Databases: An Approach. Fuzzy Sets and Systems, vol. 121 (2001) 471–490
11. Imielinski, T., Lipski, W.: Incomplete Information in Relational Databases. Journal of ACM, vol. 31 (1984) 115–143
12. Mouaddib, N.: The nuanced relational division. 2nd IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'93), (1993) 1419-1424
13. Nakata, M.: Formulation of Division Operators in Fuzzy Relational Databases. In: Knowledge Management in Fuzzy Databases. Pons O. and Vila M.A. and Kacprzyk J. Eds. Physica Verlag, (2000) 144–15
14. Yager, R.R. (1991). Fuzzy quotient operators for fuzzy relational databases. International Fuzzy Engineering Symposium. (1991) 289-296

Consistent Joins Under Primary Key Constraints

Jef Wijsen

Université de Mons-Hainaut, Mons, Belgium,
jef.wijsen@umh.ac.be,
WWW home page: http://staff.umh.ac.be/Wijsen.Jef/

Abstract. Consider a database consisting of relations, each of which may be inconsistent with respect to its primary key. Repairing such database means selecting a maximal number of tuples from each relation without ever selecting two distinct tuples that agree on the primary key. We are interested in the following decision problem: will the natural join of the repaired relations always be nonempty, no matter which tuples are selected? We take a new, game-theoretic perspective on this problem and show its advantages.

1 Introduction

Database schemas consist of both data structures (like relational tables or XML trees) and integrity constraints. Usually, the integrity constraints are checked each time a piece of data is inserted, deleted, or modified. This guarantees that at all times, the data will be consistent with respect to the integrity constraints.

However, such "piecewise" incremental integrity maintenance is untenable in certain situations. Think of the following scenario. Two banks have merged and want to integrate client data. The two banks store a different marital status for the same person, identified by her national identification number (NN). That is, we may have two records:

$$\{\underline{\mathsf{NN}} : \mathit{999},\ \mathsf{Sex} : \mathit{Female},\ \mathsf{Status} : \mathit{Married}, \ldots\}\ \text{, and}$$
$$\{\underline{\mathsf{NN}} : \mathit{999},\ \mathsf{Sex} : \mathit{Female},\ \mathsf{Status} : \mathit{Divorced}, \ldots\}\ .$$

If we first insert either record in the integrated database, then the other record will be refused (NN is the primary key). This may be undesirable. A more elegant solution would be to register two "possible worlds:" one where client 999 is married, and one where she is divorced. A database that stores multiple possible worlds is usually called an *incomplete database*. If the possible worlds originate form the different ways of restoring consistency, they are commonly called *database repairs*.

We say that a piece of information is *certain (or consistent)* if it evaluates to true in all possible worlds. For example, *"Client 999 is female"* is certain. On the other hand, *"Client 999 is married"* is uncertain, as there is a possible world where this client is divorced.

Consistent query answers [1] is the problem of determining whether a given piece of information is certain. In technical terms, the repairs of an inconsistent database **db** are the maximal (under set inclusion) consistent subsets of **db**. Given a Boolean query q (representing the piece of information), the problem is to decide whether q evaluates to true on every repair of **db**.

Consistent query answering has gained considerable interest in recent years; see for example the invited talk by Chomicki [2]. In this article, we focus exclusively on primary key constraints and join queries of the form $r_1 \bowtie \ldots \bowtie r_m$. Primary keys are fundamental in the relational model and are often cited as a major source of inconsistency in data integration. Join queries have always enjoyed a lot of attention in the database research community.

Thus, we are given a relational database, i.e. a set of relational tables, each of which has a primary key. A repair of a relational table is obtained by selecting a maximum number of tuples from the table without ever selecting two tuples that agree on the primary key. We are interested in the following decision problem: Will the (natural) join of the repaired tables always be nonempty, no matter which tuples are selected?

For example, consider the database $\{r_1, r_2\}$ of Fig. 1. The schema of r_1 is AB with primary key A (which will be denoted as (AB, A) or $\underline{A}B$ later on). The schema of r_2 is BC with primary key B. Clearly, both r_1 and r_2 are inconsistent. To repair r_1, we must select $\{A : e, B : 3\}$ and either $\{A : a, B : 1\}$ or $\{A : a, B : 2\}$. To repair r_2, we must select $\{B : 2, C : c\}$ and either $\{B : 1, C : c\}$ or $\{B : 1, C : d\}$. This yields four possible repairs. It is easy to see that no matter which tuples are selected, the join of the repaired tables will always contain a tuple of the form $\{A : a, B : \beta, C : \gamma\}$ (where β and γ are placeholders for constants). One such repair $\{s_1, s_2\}$ is shown in Fig. 2, together with its join.

r_1

$\underline{A}$	B
a	1
a	2
e	3

r_2

$\underline{B}$	C
1	c
1	d
2	c

Fig. 1. Example database $\{r_1, r_2\}$.

s_1

$\underline{A}$	B
a	2
e	3

s_2

$\underline{B}$	C
1	d
2	c

$s_1 \bowtie s_2$

A	B	C
a	2	c

Fig. 2. Repair $\{s_1, s_2\}$ with nonempty join.

On the other hand, the database shown next (over a different schema) allows a repair with an empty join: choose the first tuple from r_1, and the second from

r_2; since these tuples disagree on the common attribute C, the join is empty.

r_1	$\underline{A}$	C
	a	1
	a	2

r_2	$\underline{B}$	C
	b	1
	b	2

The article is organized as follows. The following section gives a formal definition of the decision problem we are interested in. Section 3 discusses related work. In Section 4, we define a game played by two players, called Player K and Player S. We show that there exists a first-order formula that checks whether Player K has a winning strategy. Section 5 exhibits relationships between decision problem and game. Finally, Section 6 contains concluding remarks and raises fundamental questions for further research.

2 Preliminaries

A *(database) schema* is a finite set $\mathbf{R} = \{(R_1, K_1), \ldots, (R_m, K_m)\}$ where

- $m \geq 1$;
- for every $i \in \{1, \ldots, m\}$, R_i is a finite set of attributes and the *primary key* K_i is a nonempty subset of R_i;
- for every $i, j \in \{1, \ldots, m\}$, $i \neq j$ implies $R_i \neq R_j$.

It is common to underline the primary key: $\underline{K}X$ is a shorthand for (KX, K). We call $\mathbf{R}$ an *ordered* schema if the order in which the elements of $\mathbf{R}$ are listed is significant.

A *database* over $\mathbf{R}$ is a set $\mathbf{r} = \{r_1, \ldots, r_m\}$ where each r_i is a relation over R_i. The database $\mathbf{r}$ is *consistent* if for each $i \in \{1, \ldots, m\}$, r_i satisfies the key dependency $K_i \to R_i$. That is, in a consistent database, no two distinct tuples agree on their primary key.

We write $\mathsf{dbs}(\mathbf{R})$ for the set of all (not necessarily consistent) databases over $\mathbf{R}$. A *repair* of $\mathbf{r}$ is a consistent database $\mathbf{s} = \{s_1, \ldots, s_m\}$ such that for each $i \in \{1, \ldots, m\}$,

1. $s_i \subseteq r_i$; and
2. *Maximality*: if $s_i \subsetneq s'_i \subseteq r_i$, then s'_i violates $K_i \to R_i$.

We write $\bowtie \mathbf{r}$ as a shorthand for $r_1 \bowtie \ldots \bowtie r_m$, where $\bowtie$ is the natural join operator of the relational algebra.

For a fixed schema $\mathbf{R}$, we are interested in the complexity of checking membership of the following set:

$$\mathsf{CJOIN}_{\mathbf{R}} = \{\mathbf{r} \in \mathsf{dbs}(\mathbf{R}) \mid \text{for } \textit{each} \text{ repair } \mathbf{s} \text{ of } \mathbf{r}, \bowtie \mathbf{s} \neq \{\}\} \ .$$

That is, for a fixed database schema, decide whether all repairs of a given database have a nonempty join. Notice that the database schema is fixed and the complexity is measured in the number of tuples in $\mathbf{r}$, also known as data complexity. This formulation is fairly standard; it is a special case of the *consistent query answers* problem studied in [1].

3 Related Work

The repairs defined above are maximal consistent subsets of the original database. In the case of primary keys, it does not matter whether maximality is expressed relative to set inclusion (as in [3]) or cardinality (as in [4]). Inserting new tuples is useless for restoring primary key violations. Tuple modifications, as proposed in [5], are not considered in this article.

There is a straight relationship between $\mathsf{CJOIN}_{\mathbf{R}}$ and computing consistent query answers to conjunctive queries. For example, if $\mathbf{r} = \{r_1, r_2\}$ is a database over $\mathbf{R} = \{\underline{A}B, \underline{B}C\}$, then $\mathbf{r} \in \mathsf{CJOIN}_{\mathbf{R}}$ if and only if every repair $\mathbf{s} = \{s_1, s_2\}$ of $\mathbf{r}$ satisfies $\exists x \exists y \exists z (P_1(\underline{x}, y) \wedge P_2(\underline{y}, z))$, where the predicate symbol P_1 is interpreted by the relation over $\underline{A}B$, and P_2 by the relation over $\underline{B}C$. Fuxman and Miller [6] have made a number of breakthroughs in consistent answering to conjunctive queries under primary key constraints. Their work has inspired further work in that area, for example [7, 8], including the current article. Our aim is to exhibit a number of new ideas that may further foster this research topic.

Fuxman and Miller [9] give an algorithm for first-order rewriting of a subclass of Boolean conjunctive queries under primary key constraints. (A query is called Boolean if all its variables are quantified; the answer to such a query is either "true" or "false.") Given a query q of that subclass and a set Σ of primary key constraints, the algorithm computes a first-order formula $\varphi(q, \Sigma)$ such that for each database DB, the following statements are equivalent:

1. Every repair of DB satisfies q.
2. DB satisfies $\varphi(q, \Sigma)$.

The interest of this algorithm should be clear: to decide whether *every* repair satisfies the query (statement 1), we can simply evaluate the rewritten query on the original, not-necessarily-consistent database (statement 2), which takes only polynomial time. Although Fuxman and Miller have extended their results to non-Boolean queries later on [6], we will limit the discussion to Boolean queries here.

The above equivalence holds for a subclass of conjunctive queries that is characterized in terms of the Fuxman-Miller join graph (FM join graph). For a query $\exists^*(P_1(\boldsymbol{x}_1) \wedge \ldots \wedge P_m(\boldsymbol{x}_m))$, each atom $P_i(\boldsymbol{x}_i)$ is a vertex in the FM join graph. There is an oriented edge from $P_i(\underline{s_1, \ldots, s_k}, s_{k+1}, \ldots, s_n)$ to a distinct atom $P_j(\boldsymbol{x}_j)$ if some $s_{k+1}, \ldots, s_n$ is a variable that also occurs in $\boldsymbol{x}_j$. In this representation, the underlined coordinates constitute the primary key of the relation. The query rewriting algorithm of Fuxman and Miller [9] is correct for queries where:

- $1 \leq i < j \leq m$ implies $P_i \neq P_j$;
- the FM join graph is a tree; and
- if there is an edge from $P_i(\underline{s_1, \ldots, s_k}, s_{k+1}, \ldots, s_n)$ to $P_j(\boldsymbol{x}_j)$, then $\{s_{k+1}, \ldots, s_n\}$ includes every variable that occurs in the primary key of $P_j(\boldsymbol{x}_j)$.

The latter condition was relaxed later on by Grieco et al. [7].

FM join graphs naturally apply to schemas $\mathbf{R} = \{(R_1, K_1), \ldots, (R_m, K_m)\}$: draw an edge from (R_i, K_i) to (R_j, K_j) if $(R_i \setminus K_i) \cap R_j \neq \{\}$. In Fig. 3, for example, there is an edge from $\underline{A}B$ to $\underline{BC}D$ because B is a non-key attribute of $\underline{A}B$ and B also occurs in $\underline{BC}D$. The FM join graph is not a tree, because $\underline{BC}D$ has two incoming edges. The FM join graph of Fig. 4 is cyclic, because B occurs in both $\underline{A}BC$ and $\underline{A}BD$ in a non-key position.

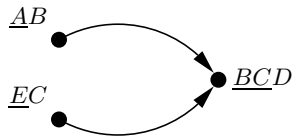

Fig. 3. FM join graph of $\{\underline{A}B, \underline{E}C, \underline{BC}D\}$.

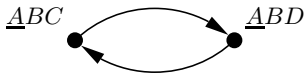

Fig. 4. FM join graph of $\{\underline{A}BC, \underline{A}BD\}$.

4 The Consistent Join Game

We present a game-theoretic approach to database repairing.

4.1 The Game

Let $\mathbf{r} = \{r_1, \ldots, r_m\}$ be a database over the ordered database schema $\mathbf{R} = \{(R_1, K_1), \ldots, (R_m, K_m)\}$. The game associated with $\mathbf{r}$ is played by two players called K and S. The game consists of m successive *turns* numbered $1, \ldots, m$. At the ith turn, Player K starts by picking a tuple $k_i \in r_i$, and Player S reacts by picking a tuple $t_i \in r_i$ such that $t_i[K_i] = k_i[K_i]$. That is, Player S picks a tuple that has the same primary key value as the tuple chosen by Player K. Since the database may be inconsistent, Player S may have the choice between multiple

tuples. Player K *wins the game* if the tuples $t_1, \ldots, t_m$, chosen by Player S, join together (i.e. if $\bowtie_{i=1}^{m} \{t_i\} \neq \{\}$). We say that Player K has a *winning strategy* on $\mathbf{r}$ if he or she can always win any game associated with $\mathbf{r}$, no matter how Player S plays.

Note that for the tuples picked by Player K, it is only the primary key value that matters to Player S. That is, if $k_i, k_i' \in r_i$ with $k_i[K_i] = k_i'[K_i]$ and Player K picked k_i, then he could as well have chosen k_i'. For this reason, we will often indicate only the primary key value chosen by Player K. Intuitively, K stands for Key: Player K repeatedly fixes a primary key value. S stands for Spoiler: Player S must "spoil" the construction of a nonempty join by Player K.

For example, Player K has a winning strategy on the database $\mathbf{r} = \{r_1, r_2\}$ shown in Fig. 1:

Turn 1. Player K starts by picking $\{A : a\}$. Then, Player S must pick either $\{A : a, B : 1\}$ or $\{A : a, B : 2\}$ form r_1.

Turn 2. If Player S picked $\{A : a, B : 1\}$ from r_1, then Player K now picks primary key value $\{B : 1\}$ from r_2. Player S can then choose between tuples $\{B : 1, C : c\}$ and $\{B : 1, C : d\}$ in r_2, each of which joins with $\{A : a, B : 1\}$. In the other case, if Player S picked $\{A : a, B : 2\}$ from r_1, then Player K now picks primary key value $\{B : 2\}$ from r_2. Player S must then pick $\{B : 2, C : c\}$, which joins with $\{A : a, B : 2\}$.

In the description of the game, we assumed some order on the relations in $\mathbf{r}$: r_1, r_2, r_3,... are considered in that order. It is easy to see that the existence of a winning strategy may depend on such order. For example, in Fig. 1, Player K would have no winning strategy if the game started with r_2 in turn 1, and continued with r_1 in turn 2.

4.2 First-Order Formula for Checking Winning Strategy

We build a first-order formula that allows checking whether Player K has a winning strategy on a given database $\mathbf{r}$. For example, for the database schema $\mathbf{R} = \{\underline{A}B, \underline{B}C\}$, the formula is (using tuple-calculus style):

$$\begin{array}{l} \exists k_1 \in r_1 (\forall t_1 \in r_1 (t_1[A] = k_1[A] \rightarrow \\ \quad \exists k_2 \in r_2 (\forall t_2 \in r_2 (t_2[B] = k_2[B] \rightarrow \\ \quad\quad \{t_1\} \bowtie \{t_2\} \neq \{\}))))) \end{array}$$

The first line corresponds to turn 1: Player K chooses k_1 and Player S reacts by picking any tuple t_1 such that $t_1[A] = k_1[A]$. The second line corresponds to turn 2: Player K chooses k_2 and Player S reacts by picking any tuple t_2 such that $t_2[B] = k_2[B]$. The last line tests the outcome: do the tuples selected by Player S have a nonempty join?

Let $\mathbf{r} = \{r_1, \ldots, r_m\}$ be a database over $\mathbf{R} = \{(R_1, K_1), \ldots, (R_m, K_m)\}$. Assume a first-order language, denoted $\mathcal{L}_{\mathbf{R}}$, that associates a predicate symbol P_i of arity $|R_i|$ to each $i \in \{1, \ldots, m\}$. If an $\mathcal{L}_{\mathbf{R}}$ formula is interpreted relative

to $\mathbf{r}$, then P_1 is interpreted by the relation over R_1, P_2 by the relation over R_2, and so on. Then, the foregoing formula is:

$$\begin{array}{l}\exists x, y(P_1(x,y) \wedge \forall y'(P_1(x,y') \rightarrow \\ \quad \exists v, w(P_2(v,w) \wedge (\forall w'(P_2(v,w') \rightarrow \\ \quad\quad y' = v))))\end{array}$$

The following theorem states that there exists a first-order formula that checks the existence of a winning strategy.

Theorem 1. *For each ordered database schema* $\mathbf{R}$*, there exists a computable* $\mathcal{L}_{\mathbf{R}}$ *sentence* $\varphi(\mathbf{R})$ *with the following property:*

$$\{\mathbf{r} \in \mathsf{dbs}(\mathbf{R}) \mid K \textit{ has a winning strategy on } \mathbf{r}\} = \{\mathbf{r} \in \mathsf{dbs}(\mathbf{R}) \mid \mathbf{r} \models \varphi(\mathbf{R})\} \ .$$

Proof. Crux. Let $\mathbf{R} = \{(R_1, K_1), \ldots, (R_m, K_m)\}$. Abusing syntax, the formula $\varphi(\mathbf{R})$ looks as follows:

$$\begin{array}{l}\exists k_1 \in r_1, \forall t_1 \in r_1 \text{ such that } t_1[K_1] = k_1[K_1], \\ \exists k_2 \in r_2, \forall t_2 \in r_2 \text{ such that } t_2[K_2] = k_2[K_2], \\ \qquad\qquad \vdots \\ \exists k_m \in r_m, \forall t_m \in r_m \text{ such that } t_m[K_m] = k_m[K_m], \\ \{t_1\} \bowtie \{t_2\} \bowtie \ldots \bowtie \{t_m\} \neq \{\} \ .\end{array}$$

Each existential quantifier corresponds to the choice of a primary key value by Player K; the subsequent universal quantifier corresponds to the choice of Player S, who can pick any tuple of the same relation with the same primary key value. The last line tests whether the tuples picked by Player S join together. □

5 Relating the Game to $\mathsf{CJOIN}_{\mathbf{R}}$

We now study the relationships between the decision problem $\mathsf{CJOIN}_{\mathbf{R}}$ and our game. The following result states that the existence of a winning strategy is a sufficient condition for membership of $\mathsf{CJOIN}_{\mathbf{R}}$.

Lemma 1. *Let* $\mathbf{r}$ *be a database over ordered database schema* $\mathbf{R}$*. If Player* K *has a winning strategy on* $\mathbf{r}$*, then* $\mathbf{r} \in \mathsf{CJOIN}_{\mathbf{R}}$*.*

Proof. Let $\mathbf{r} = \{r_1, \ldots, r_m\}$ and $\mathbf{R} = \{(R_1, K_1), \ldots, (R_m, K_m)\}$. Consider the ith turn. Each primary key value $k_i \in \pi_{K_i}(r_i)$ picked by Player K is present in every repair. Assume tuples $t_{i1}, \ldots, t_{il} \in r_i$ such that $t_{i1}[K_i] = \ldots = t_{il}[K_i] = k_i$, violating the key dependency $K_i \rightarrow R_i$. Player S chooses among all possible ways to repair this violation. The desired result follows because Player K can find m tuples that join together, no matter how the database is repaired. □

The opposite of Lemma 1 is generally not true. That is, the existence of a winning strategy is not necessary for membership of $\mathsf{CJOIN}_{\mathbf{R}}$ Consider the following database $\mathbf{r} = \{r_1, r_2\}$ with schema $\mathbf{R} = \{\underline{A}C, \underline{B}C\}$.

r_1

$\underline{A}$	C	
a	1	(t_1)
a	2	(t_2)
d	3	(t_3)

r_2

$\underline{B}$	C	
e	1	(u_1)
b	2	(u_2)
b	3	(u_3)

Player K has no winning strategy; consider what can happen:

1. In turn 1, if K picks $\{A : a\}$, then S picks t_2. Next, in turn 2, if K picks $\{B : b\}$ (picking $\{B : e\}$ would immediately lead to defeat), then S picks u_3. K does not win, because t_2 and u_3 do not join.
2. In turn 1, if K picks $\{A : d\}$, then S must pick t_3. Next, in turn 2, if K picks $\{B : b\}$, then S picks u_2. Again, K does not win, because t_3 and u_2 do not join.

On the other hand, it is easy to verify $\mathbf{r} \in \mathsf{CJOIN}_{\mathbf{R}}$ in this example. Consequently, the opposite of Lemma 1 does not hold in general. Next, we study database schemas for which the opposite of Lemma 1 is true.

Keys Comparable by Set Inclusion

We show that the inverse of Lemma 1 is true for two schemas that are not covered by the results in [6, 7]; these schemas are $\{\underline{A}BC, \underline{A}BD\}$ and $\{\underline{A}B, \underline{A}C, \underline{BC}D\}$. The FM join graph of the first schema is cyclic (see Fig. 4). The FM join graph of the second schema is not a tree (replace E by A in Fig. 3).

Proposition 1. *Let* $\mathbf{R} = \{\underline{A}BC, \underline{A}BD\}$ *or* $\mathbf{R} = \{\underline{A}B, \underline{A}C, \underline{BC}D\}$. *Let* $\mathbf{r}$ *be a database over* $\mathbf{R}$. *If* $\mathbf{r} \in \mathsf{CJOIN}_{\mathbf{R}}$, *then Player* K *has a winning strategy on* $\mathbf{r}$.

Proof. We prove the case $\mathbf{R} = \{\underline{A}BC, \underline{A}BD\}$ (the other case is similar). Assume the existence of $\mathbf{r} \in \mathsf{CJOIN}_{\mathbf{R}}$ on which Player K has no winning strategy. Assume that a number of tuples t with $t(A) = 1$ join together:

r_1

$\underline{A}$	B	C
1	b	c_1
		$\vdots$
1	b	c_l

r_2

$\underline{A}$	B	D
1	b	d_1
		$\vdots$
1	b	d_n

If these are all the tuples t satisfying $t(A) = 1$, then Player K has an obvious winning strategy, a contradiction. We conclude by contradiction that r_1 or r_2 must also contain a tuple u with $u(A) = 1$ and $u(B) \neq b$. Then, we can construct a repair $\{s_1, s_2\}$ such that $s_1 \bowtie s_2$ contains no tuple t satisfying $t(A) = 1$.

The same reasoning can be repeated for the tuples t satisfying $t(A) = 2$, $t(A) = 3$, and so on (we assume w.l.o.g. that the A-column contains only positive integers). It follows that $\mathbf{r} \notin \mathsf{CJOIN}_{\mathbf{R}}$, a contradiction. □

The result can easily be generalized to schemas where primary keys are comparable by set inclusion.

Corollary 1. *Let* $\mathbf{R} = \{\underline{A}BC, \underline{A}BD\}$. *There exists a computable* $\mathcal{L}_{\mathbf{R}}$ *sentence* $\varphi(\mathbf{R})$ *such that* $\mathsf{CJOIN}_{\mathbf{R}} = \{\mathbf{r} \in \mathsf{dbs}(\mathbf{R}) \mid \mathbf{r} \models \varphi(\mathbf{R})\}$. *An analogous result holds for* $\mathbf{R} = \{\underline{A}B, \underline{A}C, \underline{BC}D\}$.

Proof. The result follows from Theorem 1, Lemma 1, and Proposition 1.

Since evaluating $\varphi(\mathbf{R})$ is in polynomial time, it follows that for the two schemas considered, $\mathsf{CJOIN}_{\mathbf{R}}$ is in $\mathbf{P}$.

Acyclic Database Schemas

In 1983, Beeri, Fagin, Maier, and Yannakakis [10] introduced a construct of join graph and join tree, which are different from the join graphs introduced by Fuxman and Miller [6]. We will use the authors' initials to differentiate between both constructs (BFMY and FM).

Definition 1. *A* BFMY join tree *of schema* $\mathbf{R} = \{(R_1, K_1), \ldots, (R_m, K_m)\}$ *is an undirected tree such that:*[1]

1. *the set of vertices is* $\mathbf{R}$*;*
2. *each edge* $\{(R_i, K_i), (R_j, K_j)\}$ *is labeled by* $R_i \cap R_j$*; and*
3. *for every pair* (R_i, K_i), (R_j, K_j) *of distinct vertices, for each* $A \in R_i \cap R_j$, *each edge along the unique path between* (R_i, K_i) *and* (R_j, K_j) *includes label* A.

Note that primary keys, which were absent in [10], are not restricted in any way by the foregoing definition. They have been added by us to ease the notation in the results to follow. Fig. 5 shows a BFMY join tree for the schema $\{\underline{A}B, \underline{E}C, \underline{BC}D\}$. Note that the graph is undirected and that each edge is labeled by the attributes shared by its endpoints. Compare with the FM join graph of the same schema shown in Fig. 3

Fig. 6 shows a BFMY join tree for the schema $\{\underline{A}BC, \underline{A}BD\}$. Compare with the FM join graph of the same schema shown in Fig. 4.

Definition 2. *An ordered database schema* $\mathbf{R} = \{(R_1, K_1), \ldots, (R_m, K_m)\}$ *is* primary-key-acyclic *(or* PK-acyclic*) if it has a BFMY join tree with root* (R_1, K_1) *such that:*

1. *If* (R_j, K_j) *is the parent of* (R_i, K_i), *then* $j < i$. *Thus, the elements in* $\mathbf{R}$ *appear in increasing depth.*
2. *If* (R_j, K_j) *is the parent of* (R_i, K_i), *then* $R_j \cap R_i \subseteq K_i$. *Note that* $R_j \cap R_i$ *is also the label of the edge between* (R_j, K_j) *and* (R_i, K_i).

[1] An undirected graph is a tree if it contains no cycles. An undirected tree becomes directed (or oriented) by selecting one vertex as the root.

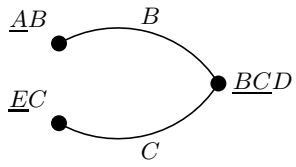

Fig. 5. BFMY join tree of $\{\underline{A}B, \underline{E}C, \underline{BC}D\}$.

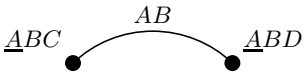

Fig. 6. BFMY join tree of $\{\underline{A}BC, \underline{A}BD\}$.

For example, the schema of Fig. 5 is not PK-acyclic, because:

- If $\underline{A}B$ or $\underline{BC}D$ is chosen as the root, then $\underline{BC}D$ becomes the parent of $\underline{E}C$, but $\{B, C, D\} \cap \{E, C\} \not\subseteq \{E\}$.
- If $\underline{E}C$ is chosen a the root, then $\underline{BC}D$ becomes the parent of $\underline{A}B$, but $\{B, C, D\} \cap \{A, B\} \not\subseteq \{A\}$.

Fig. 7 shows a PK-acyclic database schema. This can be verified by choosing either $\underline{A}F$ or $\underline{A}BD$ as the root of the tree.

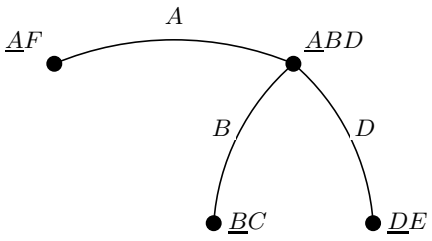

Fig. 7. BFMY join tree of $\{\underline{A}F, \underline{A}BD, \underline{B}C, \underline{D}E\}$. The schema is PK-acyclic.

Lemma 2. *Let* $\mathbf{R}$ *be a PK-acyclic database schema and* $\mathbf{r}$ *a database over* $\mathbf{R}$. *If* $\mathbf{r} \in \mathsf{CJOIN}_{\mathbf{R}}$, *then Player K has a winning strategy on* $\mathbf{r}$.

Proof. Let $\mathbf{R} = \{(R_1, K_1), \ldots, (R_m, K_m)\}$. The tree structure on $\mathbf{R}$ carries over to $\mathbf{r}$: if (R_j, K_j) is the parent of (R_i, K_i), then r_j is the parent of r_i, where r_j and r_i are the relations over R_j and R_i respectively.

We say that a tuple t joins with a tuple u if $\{t\} \bowtie \{u\} \neq \{\}$. For each $i \in \{1, \ldots, m\}$, we define the *spoiler tuples* of r_i. They are determined in decreasing depth, i.e. if r_j is the parent of r_i, then the spoiler tuples of r_i are determined before those of r_j.

1. If r_j is a leaf node, then r_j contains no spoiler tuples.
2. Assume that r_j is not a leaf node and that the spoiler tuples of its children have been determined. A tuple $t \in r_j$ is a spoiler tuple if for some child r_i of r_j, for each $u \in r_i$, if t joins with u, then there exists $u_s \in r_i$ such that $u_s[K_i] = u[K_i]$ and u_s is a spoiler tuple.

Notice that if r_j is the parent of r_i and $t \in r_j$ joins with no tuple of r_i, then t is a spoiler tuple. We outline the winning strategy of Player K:

Turn 1. Pick $k_1 \in \pi_{K_1}(r_1)$ such that there is no spoiler tuple $t \in r_1$ satisfying $t[K_1] = k_1$. If Player K has no such choice, then the spoiler tuples allow constructing a repair $\mathbf{s}$ of $\mathbf{r}$ such that $\bowtie \mathbf{s} = \{\}$, contradicting $\mathbf{r} \in \mathsf{CJOIN}_{\mathbf{R}}$.

Turn i > 1. Let $t_1, \ldots, t_{i-1}$ be the tuples selected by Player S in the preceding turns. Pick $k_i \in \pi_{K_i}(r_i)$ such that $t_1 \bowtie \ldots \bowtie t_{i-1} \bowtie k_i \neq \{\}$ and there is no spoiler tuple $t \in r_i$ satisfying $t[K_i] = k_i$.

Player S must react by picking a tuple $t_i \in r_i$ such that $t_i[K_i] = k_i$. Since the attributes common to $t_1 \bowtie \ldots \bowtie t_{i-1}$ and t_i are all contained in K_i, it follows $t_1 \bowtie \ldots \bowtie t_{i-1} \bowtie t_i \neq \{\}$.

Assume that Player K has no such choice, i.e. for all $k_i \in \pi_{K_i}(r_i)$ such that $t_1 \bowtie \ldots \bowtie t_{i-1} \bowtie k_i \neq \{\}$, there is a spoiler tuple $t \in r_i$ satisfying $t[K_i] = k_i$. Then, Player K has deviated from the strategy in a previous turn, a contradiction.

This concludes the proof. □

Corollary 2. *For each PK-acyclic database schema* $\mathbf{R}$*, there exists a computable* $\mathcal{L}_{\mathbf{R}}$ *sentence* $\varphi(\mathbf{R})$ *such that* $\mathsf{CJOIN}_{\mathbf{R}} = \{\mathbf{r} \in \mathsf{dbs}(\mathbf{R}) \mid \mathbf{r} \models \varphi(\mathbf{R})\}$*. Hence,* $\mathsf{CJOIN}_{\mathbf{R}}$ *is in* $\mathbf{P}$*.*

Proof. The result follows from Theorem 1, Lemma 1, and Lemma 2.

6 Conclusion

The problem $\mathsf{CJOIN}_{\mathbf{R}}$ is a restricted case of the problem known as *consistent query answers* [1]: decide whether every repair of a given database satisfies a Boolean query. The restrictions in $\mathsf{CJOIN}_{\mathbf{R}}$ are the following:

- the constraints are primary key constraints;
- the query is a constant-free query of the form $\exists^*(P_1(\boldsymbol{x}_1) \wedge \ldots \wedge P_m(\boldsymbol{x}_m))$, where $i \neq j$ implies $P_i \neq P_j$.

Consistent answering to conjunctive queries under primary key constraints has been the subject of a number of other articles [6–8]. The goal of this article was not to identify new tractable instances of this problem. The results of Corollary 1 may not have appeared previously, but are still subject to generalization. The results identified by Corollary 2 are covered by [7].

On the other hand, this article is the first one (to the best of our knowledge) to look at the problem from a game-theoretic perspective. We believe that the game-based approach is practical and intuitive. For example, it explains well the quantifiers in first-order formulas that check membership of $\mathsf{CJOIN}_{\mathbf{R}}$; see Theorem 1. It may lead to easier proofs of existing results (like the proof of Lemma 2) and can be helpful in proving new results. Another, more marginal, contribution of the current article lies in the use of BFMY join graphs as an alternative for FM join graphs.

Finally, we raise a fundamental open question about the strength of our games. The problem is whether the following three conditions are equivalent for all database schemas $\mathbf{R}$:

1. $\mathsf{CJOIN}_{\mathbf{R}} = \{\mathbf{r} \in \mathsf{dbs}(\mathbf{R}) \mid K \text{ has a winning strategy on } \mathbf{r}\}$.
2. There exists a computable $\mathcal{L}_{\mathbf{R}}$ sentence $\varphi(\mathbf{R})$ such that
$$\mathsf{CJOIN}_{\mathbf{R}} = \{\mathbf{r} \in \mathsf{dbs}(\mathbf{R}) \mid \mathbf{r} \models \varphi(\mathbf{R})\} \ .$$
3. $\mathsf{CJOIN}_{\mathbf{R}}$ is known to be in $\mathbf{P}$.

From Theorem 1, it follows $1 \Rightarrow 2$. Next, it is easy to see that $2 \Rightarrow 3$. This is because the first-order formula $\varphi(\mathbf{R})$ does not depend on data (i.e. its computation has constant time data complexity) and can be evaluated in polynomial time data complexity. The opposite implications ($3 \stackrel{?}{\Rightarrow} 2$ and $2 \stackrel{?}{\Rightarrow} 1$) are open:

$3 \stackrel{?}{\Rightarrow} 2$**.** If $\mathsf{CJOIN}_{\mathbf{R}}$ is known to be in $\mathbf{P}$, is there always a first-order formula for checking membership of $\mathsf{CJOIN}_{\mathbf{R}}$?

$2 \stackrel{?}{\Rightarrow} 1$**.** If there is a first-order formula for checking membership of $\mathsf{CJOIN}_{\mathbf{R}}$, is there always such a formula that is the encoding of a winning strategy (as in Theorem 1)?

Note incidentally that for the schema $\mathbf{R} = \{\underline{A}C, \underline{B}C\}$, statement 1 is false (see the paragraph following Lemma 1), statement 2 is false (we have a proof using Ehrenfeucht-Fraïssé games), and since $\mathsf{CJOIN}_{\mathbf{R}}$ is known to be **coNP**-complete, it is not known to be in $\mathbf{P}$. It follows that for this schema, the three statements are equivalent (they are all false). For PK-acyclic database schemas, on the other hand, the three statements are each true.

References

1. Chomicki, J., Marcinkowski, J.: Minimal-change integrity maintenance using tuple deletions. Information and Computation **197**(1-2) (2005) 90–121

2. Chomicki, J.: Consistent query answering: Five easy pieces. In: Int. Conf. on Database Theory (ICDT 2007). Volume 4353 of LNCS., Springer (2007) 1–17
3. Arenas, M., Bertossi, L.E., Chomicki, J.: Consistent query answers in inconsistent databases. In: Proc. 18th ACM Symp. on Principles of Database Systems, ACM Press (1999) 68–79
4. Lin, J., Mendelzon, A.O.: Merging databases under constraints. Int. J. Cooperative Inf. Syst. **7**(1) (1998) 55–76
5. Wijsen, J.: Database repairing using updates. ACM Trans. Database Syst. **30**(3) (2005) 722–768
6. Fuxman, A., Miller, R.J.: First-order query rewriting for inconsistent databases. J. Comput. Syst. Sci. **73**(4) (2007) 610–635
7. Grieco, L., Lembo, D., Rosati, R., Ruzzi, M.: Consistent query answering under key and exclusion dependencies: Algorithms and experiments. In: Proc. 14th ACM Int. Conf. on Information and Knowledge Management (CIKM '05), ACM (2005) 792–799
8. Lembo, D., Rosati, R., Ruzzi, M.: On the first-order reducibility of unions of conjunctive queries over inconsistent databases. In: EDBT Workshops. Volume 4254 of LNCS., Springer (2006) 358–374
9. Fuxman, A.D., Miller, R.J.: First-order rewriting for inconsistent databases. In: Proc. 10th Int. Conf. on Database Theory (ICDT 2005). Volume 3363 of LNCS., Springer (2005) 337–351
10. Beeri, C., Fagin, R., Maier, D., Yannakakis, M.: On the desirability of acyclic database schemes. J. ACM **30**(3) (1983) 479–513

Making Aggregation Work in Uncertain and Probabilistic Databases*

Raghotham Murthy and Jennifer Widom
Stanford University
{rsm,widom}@cs.stanford.edu

Abstract. We describe how aggregation is handled in the *Trio* system for uncertain and probabilistic data. Because "exact" aggregation in uncertain databases can produce exponentially-sized results, we provide three alternatives: a *low* bound on the aggregate value, a *high* bound on the value, and the *expected* value. These variants return a single result instead of a set of possible results, and they are generally very efficient to compute for both full-table and grouped aggregation queries. We provide formal definitions and semantics, a description of our implementation, and some preliminary analytical and experimental results for the one aggregate (*expected-average*) for which we compute an approximation.

1 Introduction

Trio is a prototype database management system under development at Stanford, designed specifically for storing and querying data with *uncertainty* and *lineage* [13, 23]. Trio's query language, *TriQL*, is an adaptation and extension of SQL [21]. Previous papers and the Trio system have focused so far on *select-project-join* queries and some set operations [2, 13]. In this paper we begin tackling the problem of TriQL queries with aggregation [11].

In a typical semantics for uncertain or probabilistic data based on *possible-instances* (also called *possible-worlds*), it is well known that the result size for a query with aggregation can grow exponentially with data size [3, 6, 15]: there can be an exponential number of possible-instances, with potentially different aggregation results in each one. For example, if an uncertain relation has 10 tuples, each of which exists with probability 0.9, then an aggregate function like `SUM` returns 2^{10} values in its result (modulo duplicates). It has been shown that aggregation in this setting is a #P-hard problem [8].

To make computation feasible, and for general usability, in Trio we decided to offer several variants to exhaustive aggregation. Specifically, we support variants for each aggregation function that return a single value over uncertain data, instead of a set of possible values: a function returning the lowest possible value of the aggregate result (*low*), the highest possible value (*high*), or the expected value (*expected*), the latter of which takes confidences or probabilities into account. It turns out that almost all of these functions can be computed efficiently in the Trio system, and the one exception (*expected-average*) can be approximated effectively.

* This work was supported by the National Science Foundation under grants IIS-0324431 and IIS-0414762, and by grants from the Boeing and Hewlett-Packard Corporations. We also thank Jeff Ullman and the rest of the Trio group for many useful discussions.

The paper proceeds as follows. In Section 2 we review the Trio data model and query semantics, and we introduce a running example. Then we cover the paper's main contributions:

- We provide formal definitions for our aggregate function variants *low*, *high*, and *expected*. Their semantics is defined in terms of *exhaustive* aggregation, which itself follows TriQL's formal query semantics (Section 3).
- We briefly review our implementation of Trio's data model and query language, which is built on top of a conventional DBMS [13]. We then show how Trio's data encoding scheme made it easy for us to implement nearly all of our aggregate function variants (Section 4).
- Our one difficult function, *expected-average*, is implemented using the approximation of *expected-sum* divided by *expected-count*. We explore the error introduced by this approximation, and we identify an interesting special case where we obtain the correct *expected-average* (Section 5).

Related work is discussed in Section 6. In this paper we focus on aggregation queries posed over a single base table, and we do not consider lineage. Extending our techniques to Trio's full *ULDB* model including derived tables with lineage [2], and to the full TriQL query language [21], is the subject of future work; see Section 7.

2 Data Model and Running Example

In this section we review Trio's model for uncertain data, which subsumes typical probabilistic databases. Our overview is brief, and it does not include aspects of Trio's *ULDB* data model (notably lineage) not relevant to this paper. For a full treatment see [2].

An uncertain relation is a multiset of *x-tuples*. Each x-tuple is comprised of one or more mutually-exclusive *alternatives*. Each alternative is a regular tuple and has an associated probabilistic confidence value in $[0, 1]$. In a single x-tuple, if Σ is the sum of the confidence values of all alternatives, then $0 \leq \Sigma \leq 1$. If $\Sigma < 1$, then the entire x-tuple may not exist; we represent this case as an additional special alternative denoted ϕ, whose confidence is $(1 - \Sigma)$.[1] Trio also supports uncertain relations with alternatives but no confidence values [2]; restricting the definitions and algorithms in this paper to ignore confidence values is straightforward and not further discussed.

As is typical, an uncertain relation in Trio represents a set of *possible-instances*. The following observations have been developed formally in [2] and elsewhere:

- Each possible-instance is a regular relation selecting one alternative tuple from each x-tuple, or no tuple if alternative ϕ is selected. The total number of possible-instances is the product of the number of alternatives in the x-tuples.
- Each possible-instance has an associated *probability*, which is the product of the confidence values for the selected alternatives in the instance. The possible-instance probabilities sum to 1.

[1] We are using a slightly different notation from [2], for presentation purposes only. In [2], *maybe-tuples* denoted by "?" annotations represented x-tuples that may not exist, instead of ϕ alternatives as we use here. The two representations are isomorphic.

time	(color, length)
1	(gray, 20) *.5* || (black, 20) *.4* || ϕ *.1*
2	(black, 18) *.8* || (brown, 16) *.2*
2	(brown, 20) *1.0*

Fig. 1. Relation `Sightings(time,color,length)`

- Each alternative a is selected in some subset of the possible-instances. The probabilities of these instances sum to the confidence of alternative a. As a special case, if an alternative has confidence 1 (and therefore is the only alternative in its x-tuple), then it appears in all possible-instances.
- If a possible-instance has only ϕ alternatives, then it is empty, denoted $\{\}$. There can be at most one such instance.

2.1 Running Example: Squirrel Sightings

We present a fabricated, highly-simplified eco-monitoring application for examples throughout the paper. (This application was inspired partially by the *Christmas Bird Count* [5], an original motivating example for Trio [23].) Human volunteers observe squirrels on the Stanford campus and record their observations. Volunteers record the *color* (species) and approximate *length* (rounded to the nearest centimeter) of each squirrel sighting, along with the time of the observation.

Figure 1 shows a Trio table `Sightings(time,color,length)`. Each x-tuple in `Sightings` represents one observation. We follow the new Trio convention of denoting the *certain* attributes separately from the alternatives: An attribute is certain if, in each x-tuple, it contains the same value in all alternatives. In `Sightings`, attribute `time` is certain, i.e., we assume the time of each observation (denoted as an integer) is accurate. In an observation, a volunteer may be uncertain about either the color or the length of the squirrel, or both, and may assign relative confidence values to the different possibilities. The symbol || separates the alternatives. For example, in the first listed observation, the volunteer was not sure if the 20-centimeter squirrel was black (confidence 0.5) or gray (confidence 0.4). Furthermore, he has only 0.9 confidence that the animal was a squirrel at all, resulting in a ϕ alternative with confidence 0.1. In the second listed observation, the volunteer saw either an 18-centimeter black squirrel (confidence 0.8) or a 16-centimeter brown squirrel (confidence 0.2). In the third listed observation, the volunteer was certain that he saw a 20-centimeter brown squirrel (confidence 1.0).

Relation `Sightings` has six possible-instances, labeled I_1 through I_6 and shown in Figure 2. Each instance I_i has a probability P_i associated with it, as described above and formalized in [2]. Note that we do not explicitly represent ϕ alternatives in the instances.

3 Aggregation

In this section, we first discuss aggregate functions applied to the entire table, which we refer to as *full-table* aggregation. We extend in Sections 3.2–3.3 to *grouped* aggregation, i.e., queries with a `GROUP BY` clause. Recall that we are restricting ourselves to

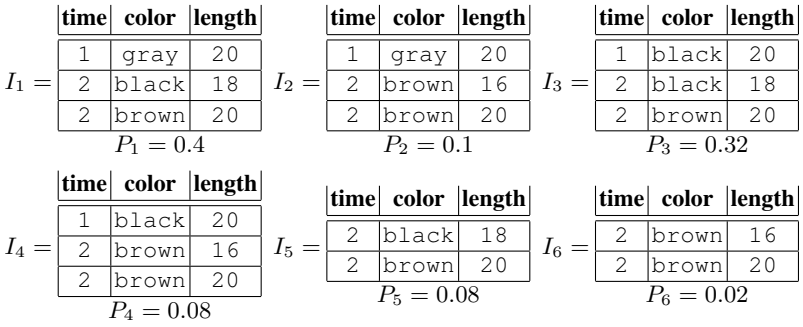

Fig. 2. Possible-Instances of `Sightings` relation

aggregation queries over a single base table. Filtering predicates in the `WHERE` clause are permitted—they are applied before aggregation and do not affect our definitions or techniques. Likewise, `HAVING` predicates can be incorporated easily into our definitions and techniques.

Recall the semantics of relational queries over uncertain databases as described in, e.g., [2]. Consider an uncertain relation U representing n possible-instances $I_1, \ldots, I_n$. A query Q on U produces a result relation R that represents the set of n possible answers: $Q(I_1), \ldots, Q(I_n)$. Aggregation follows the same semantics: An aggregation query on an uncertain relation produces a result whose possible-instances are the result of the aggregation applied to the possible-instances of the input relation.

More concretely, consider an aggregation query A with `COUNT`, `SUM`, `AVG`, `MIN`, or `MAX` applied to an uncertain relation U. (`DISTINCT` aggregates are not covered in this paper; they will be incorporated as future work.) The result contains exactly one x-tuple, which has one alternative a_j corresponding to each possible-instance I_j of U. Alternative a_j contains the result of aggregation query A over instance I_j. Note that over an empty relation (i.e., over the empty possible-instance, if it exists for U), SQL semantics dictates that the result of `COUNT` is 0, while the result of `SUM`, `AVG`, `MIN`, or `MAX` is `NULL`. The confidence associated with alternative a_j is the probability of instance I_j. Although we are not addressing lineage in this paper, here we can easily see that lineage of each alternative is the set of alternatives in its corresponding possible-instance.

In our running example, the TriQL query to find the average length of obverved squirrels looks just like its SQL counterpart:

```
SELECT AVG(length) as avgLength FROM Sightings
```

The answer returned is:

`(19.33)` *.4* || `(18.67)` *.1* || `(19.33)` *.32* || `(18.67)` *.08* || `(19)` *.08* || `(18)` *.02*

This query result has one alternative for each of the six possible-instances in Figure 2. As can be seen from the example, the result of this *exhaustive* aggregation is exponential in the size of the input relation. Even if we merge duplicates (using Trio's `MERGED` option [21]), reference [8] shows that the aggregation computation is #P-hard. Hence, we define a set of practical variants for aggregation over uncertain data.

3.1 Practical Variants to Exhaustive Aggregation

We provide three variants for each of the five aggregate functions. (Thus, including exhaustive aggregation, Trio supports a total of 20 aggregate functions.) We define the variants in terms of the result of the corresponding exhaustive aggregate function, which we denote $\mathcal{A}$.

Let us begin with the *low* variant. A *low* aggregate returns one x-tuple having one alternative that is the lowest non-`NULL` alternative in $\mathcal{A}$. Note that there can be at most one `NULL` in $\mathcal{A}$, corresponding to the empty possible-instance, and it can only occur for aggregates `SUM`, `AVG`, `MIN`, and `MAX`; on the empty possible-instance `COUNT` returns 0. If the only alternative in $\mathcal{A}$ is `NULL`, then *low* returns `NULL` as well. TriQL supports five *low* aggregate functions: `LCOUNT`, `LSUM`, `LAVG`, `LMIN`, and `LMAX`. For example, we can retrieve the lowest possible average squirrel length with the TriQL query:

```
SELECT LAVG(length) FROM Sightings
```

The result is a single x-tuple having one alternative with value 18.

Similarly, TriQL supports *high* aggregate functions `HCOUNT`, `HSUM`, `HAVG`, `HMIN`, and `HMAX`, returning the highest possible aggregate value. In our running example, if we replace `LAVG` in the query above with `HAVG`, we get one x-tuple having one alternative with value 19.33.

The third and most interesting variant is the set of *expected* aggregate functions: `ECOUNT`, `ESUM`, `EAVG`, `EMIN`, and `EMAX`. An *expected* aggregate is the weighted average of all the non-`NULL` alternatives in the corresponding exhaustive result $\mathcal{A}$, where the weights are based on the alternative's confidence values as defined earlier. (Recall that confidences on result alternatives correspond to probabilities on possible-instances.) One subtlety is that we ignore the empty possible-instance for all expected aggregates, except for `ECOUNT`. For example, for `ESUM` we compute the weighted average of all possible sums, whenever a sum exists. (To do so, we must scale the confidence values of the non-`NULL` alternatives in $\mathcal{A}$ so they add up to 1, but implementing this computation turns out to be easy; see Section 4.) On the other hand, for `ECOUNT` we do consider the empty possible-instance, since it contributes a count of 0 to the expected result. Note that *expected* aggregate functions typically return values that are not an alternative of the exhaustive result $\mathcal{A}$.

Considering our running example once again, the expected average length of observed squirrels can be computed from the exhaustive `avgLength` result shown earlier. We get:

$$19.33 \cdot 0.4 + 18.67 \cdot 0.1 + 19.33 \cdot 0.32 + 18.67 \cdot 0.08 + 19 \cdot 0.08 + 18 \cdot 0.02 = 19.16$$

Thus, the query:

```
SELECT EAVG(length) FROM Sightings
```

returns one x-tuple having one alternative with value 19.16. Note the following two points about our new aggregate functions:

- In all of the variants, the result has exactly one alternative. We set the confidence of that alternative to 1.0.
- For all five aggregate functions applied to any data, the variants satisfy the desirable constraint $low \leq expected \leq high$.

3.2 Grouped Aggregation

Now consider queries with `GROUP BY` clauses. We first briefly review grouped aggregation queries in SQL [11]. Then we define their meaning over uncertain relations, again following TriQL semantics. Finally we extend our *low*, *high*, and *expected* variants for grouped aggregation.

In SQL, the `GROUP BY` clause contains a list of *grouping attributes*. Call this list G, and without loss of generality assume it is a single attribute. The result of a query with grouped aggregation has one tuple for each value g of G appearing in the query result prior to aggregation. Let F be the query's aggregate function. (Again without loss of generality, assume there is only one aggregate function in the `SELECT` clause, along with the grouping attribute.) The tuple in the query result for value $G = g$ contains the result of aggregate function F applied to $\sigma_{G=g}(R)$.

Now consider an uncertain relation U with possible-instances $I_1, \ldots I_n$. The result of a grouped aggregation query Q, as usual, represents the possible answers $Q(I_1)$, $\ldots, Q(I_n)$. Trio implements this semantics by producing one x-tuple for each value g of G that appears in at least one possible-instance. Logically, this x-tuple contains the result of full-table aggregation on the uncertain relation $\sigma_{G=g}(U)$, followed by a correction for the empty possible-instance. As noted earlier, full-table aggregation produces a 0 alternative (for `COUNT`) or a `NULL` alternative (for the other aggregate functions) if the input relation has $\{\}$ as one of its possible-instances. In grouped aggregation, when this alternative would be produced by full-table aggregation over $\sigma_{G=g}(U)$, it is replaced by the special ϕ alternative, representing the fact that a group for $G = g$ does not appear in all possible-instances. The confidence for an alternative, as usual, represents the probability of the corresponding possible-instance(s).

In our running example, to find the average length for each squirrel color, we write:

```
SELECT color, AVG(length) AS avgLength FROM Sightings
GROUP BY color
```

The result follows. Notice that in the result `color` is a certain attribute.

color	**avgLength**
black	(18) *.4* || (19) *.32* || (20) *.08* || (20) *.08* || ϕ *.12*
brown	(20) *.4* || (18) *.1* || (20) *.32* || (18) *.08* || (20) *.08* || (18) *.02*
gray	(20) *.4* || (20) *.1* || ϕ *.5*

For each group, there is one alternative for each possible-instance in which the group exists. For example, "black" has 4 alternatives since it appears in 4 of the 6 possible-instances, while "brown" has 6 alternatives since it appears in all 6 of the possible-instances. When a group does not appear in all possible-instances, it also has a ϕ alternative.

Note that with Trio's `MERGED` option [21], where we merge duplicate alternative values, we get:

color	**avgLength**
black	(18) *.4* || (19) *.32* || (20) *.16* || ϕ *.12*
brown	(20) *.8* || (18) *.2*
gray	(20) *.5* || ϕ *.5*

3.3 Variants for Grouped Aggregation

The *low* and *high* aggregates in a grouped aggregation are the obvious extension of the full-table case: *low* returns the lowest value of the aggregate for every group appearing in the exhaustive answer, and *high* returns the highest possible value. Suppose we want low and high bounds for average squirrel lengths based on color. The TriQL query is:

```
SELECT color, LAVG(length) AS lowAvgLength,
HAVG(length) as highAvgLength FROM Sightings GROUP BY color
```

The result is:

color	lowAvgLength	highAvgLength
black	18	20
brown	18	20
gray	20	20

Grouped aggregation queries with *expected* aggregates are also an extension of the full-table case: For every group, *expected* returns the weighted-by-confidences average of the non-ϕ alternatives in its corresponding exhaustive answer, after scaling the confidences of the non-ϕ alternatives so that they add up to 1. Note that here, unlike in the full-table case, for ECOUNT we do not factor in a count of 0 for possible-instances in which a group does not exist. As with the other aggregate functions and to correctly follow TriQL possible-instance semantics, a ϕ alternative appears in the exhaustive COUNT result, not a 0, when a group does not appear in some possible-instances.

In our running example, the expected average length of squirrels based on color is given by:

```
SELECT color, EAVG(length) AS expectedAvgLength FROM Sightings
```

and the result is:

color	expectedAvgLength
black	18.73
brown	19.6
gray	20

4 Implementation

We have implemented all 20 aggregate functions in the Trio system: *exhaustive*, *low*, *high*, and *expected* for each of COUNT, SUM, AVG, MIN, and MAX. Each function is supported in both full-table and grouped form—typically the implementation for grouped is a fairly direct extension of the full-table version. We first briefly review how the Trio system is built on top of a conventional relational DBMS; for details see [13]. We then describe how the encoding we use for uncertain relations facilitates simple query translation and stored procedures to efficiently compute all but one (EAVG) of the aggregate functions.

Consider a Trio relation $\mathtt{T}(\mathtt{A}_1, \ldots, \mathtt{A}_n)$. Relation T is stored in a conventional relational table with four additional attributes: $\mathtt{T\_enc}(\mathtt{xid}, \mathtt{aid}, \mathtt{conf}, \mathtt{certain}, \mathtt{A}_1, \ldots, \mathtt{A}_n)$. Each alternative of each x-tuple in T is stored as its own tuple in T_enc. The additional attributes in T_enc are as follows:

- `xid` identifies the x-tuple
- `aid` identifies an alternative within the x-tuple
- `conf` contains the confidence of the alternative
- `certain` is a flag to indicate whether the x-tuple has a ϕ alternative

For example, the Trio relation `Sightings(time,color,length)` from Figure 1 is encoded in the regular relational table `Sightings_enc` shown in Figure 3.

xid	aid	conf	certain	time	color	length
101	1	0.5	0	1	gray	20
101	2	0.4	0	1	black	20
102	3	0.8	1	2	black	18
102	4	0.2	1	2	brown	16
103	5	1.0	1	2	brown	20

Fig. 3. `Sightings_enc` is the Trio encoding for the table in Figure 1

In reality, Trio implements the table in Figure 3 as a virtual view over two tables, `Sightings_c` and `Sightings_u`, joined on `xid`: Table `Sightings_c` has one row per x-tuple, containing the certain attributes and the special column `certain`. Table `Sightings_u` has one row per alternative, containing the uncertain attributes and the special column `conf`.

Note that ϕ alternatives are not explicitly represented. They are encoded in the `certain` attribute of each tuple: `certain` = 0 if there is a ϕ alternative and `certain` = 1 if not. Implicitly, if there is a ϕ alternative, its confidence is $1-\Sigma$, where Σ is the sum of the confidences of the other alternatives. For example, x-tuple 101, which corresponds to the first observation in the `Sightings` table, has two tuples in `Sightings_enc`, one for each non-ϕ alternative, and `certain` is 0.

Table 1 summarizes the methods we used to implement the 20 different aggregate functions. In the table we have combined *low* and *high*, as well as `MIN` and `MAX`, since they are always symmetric. Recall that we are considering aggregation queries over one base table, possibly with filtering predicates. A "translation" entry in Table 1 indicates that we are able to implement that aggregate by a simple translation from TriQL queries to queries on the encoded table; this approach works for 10 of the 20 cases. Except for `EAVG`, which is approximated, the remaining 9 are implemented using efficient stored procedures. (Since Trio is currently built on top of the *Postgres* DBMS, we use `PL/pgSQL` [14] and `SPI` [19] for stored procedures.)

For all 20 aggregate functions, the full-table and grouped versions are implemented in a similar fashion. In general we focus our discussion on the full-table version, although we do illustrate some grouped cases. Note that in no cases do we actually perform the $\sigma_{G=g}(R)$ selections used in the definition of grouped aggregation.

The remainder of this section first discusses exhaustive aggregation (Section 4.1). We then cover the aggregates implemented through translation (Section 4.2), and finally the remaining aggregates implemented as stored procedures (Section 4.3). Aggregate `EAVG` is an important special case—an efficient algorithm to compute the exact `EAVG`

	COUNT	SUM	AVG	MIN/MAX
exhaustive	stored procedure	stored procedure	stored procedure	stored procedure
low/high	translation	translation	stored procedure	translation
expected	translation	translation	approximation	stored procedure

Table 1. Summary of different implementation methods for the 20 aggregate functions

remains an open problem. There have been several proposals for computing approximate answers, e.g., [3, 10]. In Trio, we approximate EAVG as ESUM divided by ECOUNT, with compensation for the empty possible-instance. Section 5 discusses this approximation, and presents some analytical and experimental results for the error.

4.1 Exhaustive Aggregates

In this section we describe our implementation of the exhaustive aggregates. As seen in the first row of Table 1, all of them are implemented using stored procedures. As shown in [6], SUM and AVG can have different values in each possible instance and therefore have an exponentially-sized result. Aggregates COUNT, MIN, and MAX have a polynomial number of different values, although the number may still be too high to be usable in practice.

We use algorithms similar to the ones described in [6] to implement COUNT, MIN, and MAX, and will not describe them further. We did implement the exponential algorithms for SUM and AVG, primarily for experimental purposes—they usually cannot be used except for very small relations. Our "user-friendly" query processor first counts the number of possible-instances, which can be done very efficiently. If the number of possible-instances is too high (say $> 2^{15}$), we return an error message and do not permit exhaustive SUM and AVG in these cases.

When the possible-instances are few enough that we permit exhaustive SUM and AVG, our algorithm first computes and materializes, into a temporary table, all possible combinations of the alternatives in the input relation (identified by their aid's), i.e., it enumerates all possible-instances. This table is then joined back with the input table to fetch and aggregate the actual data. For example, the stored procedure to compute exhaustive full-table AVG on Sightings first populates temporary table tmp_combos_Sightings with the possible combinations of alternatives (each with a combo_id), then computes the following query:[2]

```
SELECT AVG(length) AS avgLength, PRODUCT(trio_conf) AS conf
FROM Sightings S INNER JOIN tmp_combos_Sightings T USING (aid)
GROUP BY combo_id
```

This query returns AVG values for all possible-instances along with their corresponding confidences. Simple post-processing generates the single x-tuple comprising the final result. The grouped version (by color for example) is similar to the query above, except the GROUP BY also includes the grouping attributes, and there is a final ORDER BY on the grouping attributes:

[2] Aggregate function PRODUCT is similar to SUM, except it multiplies instead of adds.

```
SELECT color, AVG(length) AS avgLength, PRODUCT(trio_conf) AS conf
FROM Sightings S INNER JOIN tmp_combos_Sightings T USING (aid)
GROUP BY combo_id, color ORDER BY color
```

Post-processing on the result of this query generates one x-tuple for each group.

4.2 Translation-Based Aggregates

For the aggregates implemented by translation, we automatically rewrite the TriQL query with aggregation into a SQL aggregation query over our encoded data. For these aggregates, the computation is about as efficient as aggregating over a conventional relation. We show a few translations here that demonstrate the principle techniques; additional translations are included in the extended technical report version of this paper [12]. Note that `WHERE` predicates could be added easily to our example queries.

For each translation, we show the example TriQL query over `Sightings` and the corresponding translation to a SQL query, along with a brief explanation. Clearly, these translations are much more efficient than computing the corresponding *exhaustive* aggregate, as are the translation-based aggregates not included, which are very similar. The most expensive translations perform a `GROUP BY` on `xid`, but none of the translations have complexity more than $O(n \log n)$ for n tuples in the relation.

`LCOUNT`

TriQL: `SELECT LCOUNT(*) FROM Sightings`
Description: Get the count of all certain x-tuples.
SQL: `SELECT COUNT(DISTINCT xid) FROM Sightings_enc WHERE certain = 1`

`ESUM`

TriQL: `SELECT ESUM(length) FROM Sightings`
Description: Get the weighted average of all the lengths and compensate for the empty possible-instance. Recall `PRODUCT` is analogous to `SUM` but multiplies instead of adds. `1-PRODUCT(qconf)` yields the probability of the non-empty possible-instances.
SQL:
```
SELECT SUM(weightedlength)/(1-PRODUCT(qconf)) AS ESUM
FROM (SELECT SUM(length*conf) AS weightedlength, 1-SUM(conf)
AS qconf FROM Sightings_enc GROUP BY xid) qconfs
```

`ECOUNT`

TriQL: `SELECT ECOUNT(*) FROM Sightings`
Description: Get the weighted average of all non-ϕ alternatives.
SQL: `SELECT SUM(conf) AS ECOUNT FROM Sightings_enc`

Grouped `HSUM`

TriQL: `SELECT color,HSUM(length) FROM Sightings GROUP BY color`
Description: Get the high sum of the lengths by color.
SQL:
```
SELECT color, SUM(CASE certain = 1 or maxlength > 0 THEN
maxlength ELSE 0 END) AS HSUM FROM (SELECT color, certain,
MAX(length) AS maxlength FROM Sightings_enc GROUP BY color,
certain, xid) maxlengths GROUP BY color
```

4.3 Stored-Procedure Aggregates

From Table 1 we see that the remaining implementations to discuss, aside from EAVG covered in the next section, are the stored procedures for LAVG, HAVG, EMIN, and EMAX, in their full-table and grouped versions. Here we describe intuitively what the stored procedures do. Complete pseudocode is provided in the extended technical report [12].

Full-table LAVG (and HAVG):

> We first compute the "certain" average, i.e., the average of all x-tuples with exactly one alternative with confidence 1.0. This value provides an upper bound on the low average. Then, from the rest of the x-tuples, we identify alternatives that can decrease the average by considering the minimum alternative from each x-tuple. We consider these values in ascending order so we can stop when the average stops decreasing. The procedure for HAVG is nearly identical—it looks for alternatives that increase the average, in descending order.

Full-table EMIN (and EMAX):

> We consider all alternative values for the aggregated attribute, in ascending order, until we have "covered" at least one x-tuple that does not have a ϕ alternative, or we have considered all values. This stopping condition guarantees that at least one of the considered values exists in every possible-instance. For each considered value v, we compute the total probability of the possible-instances that contain any alternative with value v, but no lower value. We compute the weighted average of the considered values to get our expected MIN. This algorithm can be implemented with a linear scan over the result of an ORDER BY query, with extra space required to accumulate confidences until a "covering" x-tuple is found. Note that in the worst case, we require O(n) extra space. The procedure for EMAX is nearly identical—it enumerates the alternative values in descending instead of ascending order.

Grouped aggregation queries with LAVG, HAVG, EMIN, and EMAX are implemented using very similar algorithms, except the computation is performed per-group instead of across the entire table.

5 Expected-Average

The only aggregate remaining to be discussed is *expected-average* (EAVG). Expected-average over uncertain data is known to be difficult to compute; [3] and [10] provide approximate algorithms. Like [3], we decided to use the simple approximation based on ESUM/ECOUNT, because it is efficient to compute and, as we will see, the error is usually low and in an interesting special case, zero. However, unlike [10], we have not been able to provide a theoretical bound for the error in the general case.

We first explain how we compute EAVG, and then discuss the error. A small issue arises because of the way we treat empty possible-instances for the different aggregates in their all-table and grouped forms. Table 2 summarizes this treatment, based on the definitions and justification in Section 3.2. Specifically, since ESUM and ECOUNT are

	COUNT	SUM, AVG, MIN, MAX	ECOUNT	other *expected* aggs
all-table	empty possible-instance creates 0 alternative	empty possible-instance creates NULL alternative	incorporates 0 alternative from *exhaustive*	do not incorporate NULL alternative from *exhaustive*
grouped	uncertain group has ϕ alternative		do not incorporate ϕ alternative from *exhaustive*	

Table 2. Treatment of empty possible-instance and uncertain groups

R
(2) *1.0*
(100) *.01* $\|\|$ ϕ *.99*

S
(2) *.01* $\|\|$ ϕ *.99*
(100) *1.0*

	ESUM	ECOUNT	EAVG*(exact)*	EAVG*(approx)*	*absolute error*	*relative error*
R	3	1.01	2.49	2.97	.48	.19
S	100.02	1.01	99.51	99.03	.48	.0048

Fig. 4. Extreme example for error due to varying confidence/value distributions

treated the same way for grouped aggregates, we can simply use ESUM / ECOUNT as our approximation. However, for all-table aggregates, ECOUNT incorporates the empty possible-instance, while ESUM does not. Fortunately we can easily compensate by further dividing ECOUNT by the probability of the non-empty possible-instances, which recall from Section 4 can be computed efficiently. In the remainder of this section we discuss all-table aggregation; the generalization to grouped is straightforward.

Let us consider the error in our approximation. We consider the relative error between the exact expected-average, denoted *eavg*, and our approximation, denoted *approx*: $error = \left|\frac{eavg - approx}{eavg}\right|$. We first identify a special case in which there is no error.

Theorem 1. Let U be any uncertain relation such that the confidences of all non-ϕ alternatives in each x-tuple sum to the same value. In this case our approximation of *eavg* based on ESUM divided by ECOUNT produces the correct value. □

A proof appears in the extended technical report [12]. Note that as a special case of the theorem, there is no error when all x-tuples have confidences summing to 1.0, i.e., there are no ϕ alternatives in U.

When Theorem 1 doesn't apply, intuitively the error in the approximation is caused by the varying combinations of confidences and values contributing to the average. Consider the extreme example in Figure 4. We show two uncertain relations, and analyze how our EAVG approximation behaves on them. We see that the relative error in S is only 0.0048 whereas the relative error in R is 0.19 even though the absolute error is the same in both cases (0.48). Intuitively, the relative error increases when the higher-valued alternative has lower confidence and the lower-valued alternative has higher confidence.

We ran an experiment to further explore this effect. In our experiment, to compute the relative error in the approximation we needed to compute the exact expected-average by enumerating all possible-instances. Thus, we were restricted to very small

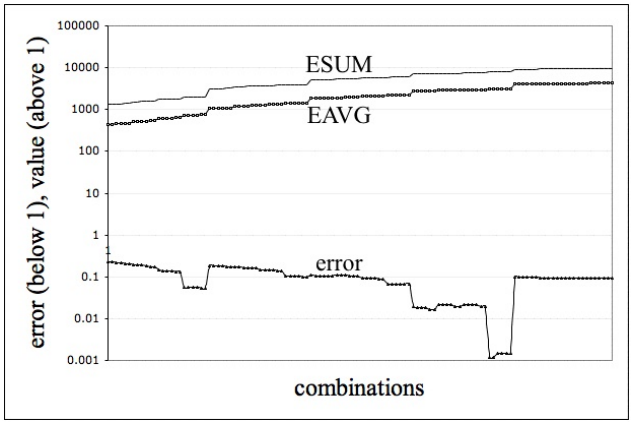

Fig. 5. Relative error for combinations of confidences and alternative values

data sizes. Nevertheless, the effect we were looking for is visible. We fixed a set of N distinct alternative values and a set of N distinct confidence values, and considered all $N!$ combinations of alternative-confidence pairings. Each combination is a new Trio table over which we aggregate to obtain one set of data points – *expected-sum*, exact *expected-average*, approximate *expected-average*, absolute error, and relative error. Figure 5 shows the results for $N = 5$. On the x-axis we enumerate the combinations, ordered by increasing *expected-sum* (topmost line) and therefore also increasing exact *expected-average* (second-to-top) since the count is fixed. Thus, on the left end we have the highest confidence assigned to the lowest value, the second-highest confidence to the second-lowest value, and so on. The right end is the converse. The relative error in our *expected-average* approximation is plotted on the bottom line. Note that the y-axis is logarithmic. We did not see any correlation between *expected-average* and absolute error (not shown in the graph). However, we do see a clear trend where the relative error decreases with increased *expected-average*, but the curve is not smooth. The lack of a smooth curve may be due in part to the small data size required for error computation, however completely understanding the relationship between our approximation and data/confidence characteristics remains an open problem.

6 Related Work

Non-aggregation queries on probabilistic and uncertain databases have been studied extensively for several years, e.g., [1, 2, 4, 7–9, 20]. Aggregation queries have been studied more recently, e.g., [3, 6, 10, 15–17, 22]. Reference [3] considers computing expected aggregates for hierarchical domains, using an approximation similar to the one we have chosen for `EAVG`. Reference [6] gives algorithms for exhaustive aggregation over uncertain data without confidences. It also shows that `COUNT`, `MIN`, and `MAX` have polynomial-sized results whereas `SUM` and `AVG` have exponentially-sized results. The work is extended in [22] to uncertain data with probabilities. We use similar techniques for exhaustive aggregation, while providing more practical variants of aggregate functions.

The most recent work in the area of approximate algorithms for expected aggregates [10] improves upon previous algorithms for expected-average. It uses a probabilistic stream model and provides a generating-function based algorithm to compute very close approximations. However, it requires multiple passes over the data, and may not improve much upon the error of our approximation in practical applications. Reference [17] applies aggregation queries to resolve inconsistencies in the data, but does not focus on the aggregate computation itself. Reference [15] uses a linear programming approach to define aggregations, and reference [16] uses a fuzzy sets approach.

None of the previous work we are aware of considers a full suite of aggregate variants for uncertain and probabilistic databases, including `low`, `high`, and `expected` for all five aggregate functions in their full-table and grouped forms. In addition, we have implemented all of our algorithms as part of a complete prototype for managing and querying uncertain data.

7 Conclusions and Future Work

We introduced aggregate function variants for uncertain data that are much more efficient to compute than exact (exhaustive) aggregates, and are likely to be more practical from a usability perspective. Based on the data encoding scheme in the Trio system, we were able to implement many of the aggregate variants, in their full-table and grouped forms, through simple query translation. The remaining aggregates are implemented as stored procedures, but they are still quite efficient—generally relying on one or two aggregate queries over regular relations, with an additional order-by in the worst case. For the one problematic aggregate, *expected-average*, we compute an approximation based on *expected-sum* over *expected-count*. We have found that the error is low in general, and is guaranteed to be zero in certain cases.

This paper reports our initial progress in making aggregation work in the Trio system. There are many important avenues of future work:

- `DISTINCT` aggregates are nontrivial to add, although we do not expect major obstacles.
- Much more challenging is handling queries that include joins, as well as queries involving Trio tables with lineage. In both cases, the data values to be aggregated may be *nonindependent*, which complicates several of our aggregates but the *expected* ones in particular. Ideas from [18] on dealing with correlated tuples in uncertain relations may be helpful.
- We have not yet characterized the error in our *expected-average* approximation. We expect similar approximations may have been used in statistics, where we hope to find some insights.
- Finally, we have performed only preliminary experiments, and they used very small data sets so that exhaustive aggregates could be computed for comparison purposes. A more thorough experimental study can be conducted that considers performance as well as error characteristics.

References

1. D. Barbara, H. Garcia-Molina, and D. Porter. The Management of Probabilistic Data. *IEEE Transactions on Knowledge and Data Engineering*, 4(5):487–502, 1992.

2. O. Benjelloun, A. Das Sarma, A. Y. Halevy, and J. Widom. ULDBs: Databases with Uncertainty and Lineage. In *Proc. of Intl. Conference on Very Large Databases (VLDB)*, pages 953–964, Seoul, Korea, 2006.
3. D. Burdick, P. Deshpande, T. S. Jayram, R. Ramakrishnan, and S. Vaithyanathan. OLAP Over Uncertain and Imprecise Data. In *Proc. of Intl. Conference on Very Large Databases (VLDB)*, pages 970–981, Trondhiem, Norway, 2005.
4. R. Cavallo and M. Pittarelli. The Theory of Probabilistic Databases. In *Proc. of Intl. Conference on Very Large Databases (VLDB)*, pages 71–81, Brighton, England, 1987.
5. Christmas Bird Count Home Page. `http://www.audubon.org/bird/cbc`.
6. A.L.P. Chen, J. Chiu, and F.S.C. Tseng. Evaluating Aggregate Operations Over Imprecise Data. *IEEE Transactions on Knowledge and Data Engineering*, 08(2):273–284, 1996.
7. R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating Probabilistic Queries Over Imprecise Data. In *Proc. of ACM-SIGMOD International Conference on Management of Data*, pages 551–562, San Diego, California, 2003.
8. N.N. Dalvi and D. Suciu. Efficient Query Evaluation on Probabilistic Databases. In *Proc. of Intl. Conference on Very Large Databases (VLDB)*, pages 864–875, Toronto, Canada, 2004.
9. D. Dey and S. Sarkar. A Probabilistic Relational Model and Algebra. *ACM Transactions on Database Systems*, 21(3):339–369, 1996.
10. T.S. Jayram, S. Kale, and E. Vee. Efficient Aggregation Algorithms for Probabilistic Data. In *Proc. of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2007.
11. A. Klug. Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions. *Journal of the ACM*, 29(3):699–717, 1982.
12. R. Murthy and J. Widom. Making Aggregation Work in Uncertain and Probabilistic Databases. Technical report, Stanford InfoLab, June 2007. Available at `http://dbpubs.stanford.edu/pub/2007-7`.
13. M. Mutsuzaki, M. Theobald, A. de Keijzer, J. Widom, P. Agrawal, O. Benjelloun, A. Das Sarma, R. Murthy, and T. Sugihara. Trio-One: Layering Uncertainty and Lineage on a Conventional DBMS (Demo). In *Proc. of Conference on Innovative Data Systems Research (CIDR)*, pages 269–274, Pacific Grove, California, 2007.
14. PL/pgSQL - SQL Procedural Language. Manual at `http://www.postgresql.org/docs/7.4/interactive/plpgsql.html`.
15. R. Ross, V. S. Subrahmanian, and J. Grant. Aggregate Operators in Probabilistic Databases. *J. ACM*, 52(1):54–101, 2005.
16. E. A. Rundensteiner and L. Bic. Evaluating Aggregates in Possibilistic Relational Databases. *Data Knowl. Eng.*, 7(3):239–267, 1992.
17. B. Scotney and S. McClean. Database Aggregation of Imprecise and Uncertain Evidence. *Inf. Sci. Inf. Comput. Sci.*, 155(3-4):245–263, 2003.
18. P. Sen and A. Deshpande. Representing and Querying Correlated Tuples in Probabilistic Databases. In *Proc. of Intl. Conference on Data Engineering (ICDE)*, April 2007.
19. Postgres Server Programming Interface. `http://www.postgresql.org/docs/8.1/interactive/spi.html`.
20. D. Suciu and N. Dalvi. Foundations of Probabilistic Answers to Queries. In *Proc. of ACM-SIGMOD International Conference on Management of Data*, pages 963–963, Baltimore, Maryland, 2005. ACM Press.
21. TriQL: The Trio Query Language. Available from Stanford Trio Home Page at `http://www.infolab.stanford.edu/trio`.
22. F.S.C Tseng, A.L.P Chen, and W-P. Yang. Answering Heterogeneous Database Queries with Degrees of Uncertainty. *Distributed and Parallel Databases*, 1(3):281–302, 1993.
23. J. Widom. Trio: A System for Integrated Management of Data, Accuracy, and Lineage. In *Proc. of Conference on Innovative Data Systems Research (CIDR)*, pages 262–276, Pacific Grove, California, 2005.