Incorporating Domain Knowledge and User Expertise in Probabilistic Tuple Merging

Fabian Panse and Norbert Ritter

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

Abstract. Today, probabilistic databases (PDB) become helpful in several application areas. In the context of cleaning a single PDB or integrating multiple PDBs, duplicate tuples need to be merged. A basic approach for merging probabilistic tuples is simply to build the union of their sets of possible instances. In a merging process, however, often additional domain knowledge or user expertise is available. For that reason, in this paper we extend the basic approach with aggregation functions, knowledge rules, and instance weights for incorporating external knowledge in the merging process.

Keywords: probabilistic data, tuple merging, external knowledge.

1 Introduction

In recent time, the need for probabilistic databases grows in many real-world applications [17,18,8,15]. In general, for certain databases as well as for probabilistic databases duplicates are pervasive problems of data quality [7]. To solve this problem duplicates have to be identified and merged. Strategies for resolving data conflicts in a merge of certain tuples is extensively discussed in the literature [3,11]. However, there is only a low attention on the merge of probabilistic tuples, so far. Nevertheless, if probabilistic source data are given, the degree of uncertainty which has to be resolved during the merging process is higher than in the merge of certain tuples. On the other hand, probabilistic data models provide new capabilities for handling conflicts in the merging process. Thus, tuple merging becomes also more powerful [6,16]. In [12] we introduce a basic approach for merging the instance data of probabilistic tuples which is conceptually based on the set union operator. In real duplicate elimination scenarios, however, often a lot of domain knowledge or user expertise is available. This knowledge cannot be included in our simple merging approach. For that reason, we extend this approach by enabling the user to define aggregation functions for single attributes, and instance weights as well as knowledge rules for whole instances. The incorporation of external knowledge is an important property, because in several scenarios a simple union of all possible instances does not correspond with the semantics of some attributes (see motivating example below), or the set of possible instances can be evidently reduced.

S. Benferhat and J. Grant (Eds.): SUM 2011, LNAI 6929, pp. 289–302, 2011.

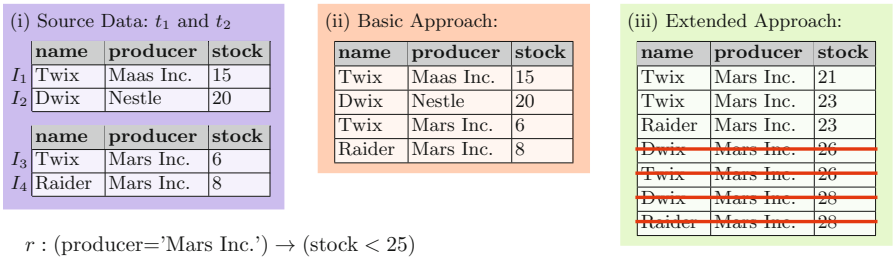


Fig. 1. The possible instances of t_1 (I_1 and I_2) and t_2 (I_3 and I_4) (i), the instances resulting from merging $\{t_1, t_2\}$ with the basic approach (ii), and the instances resulting from merging $\{t_1, t_2\}$ whilst taking external knowledge into account (iii)

As a motivating example, we consider a merge of the two base-tuples t_1 and t_2 as shown in Figure 1. Both tuples have two possible instances ($\{I_1, I_2\}$ for t_1 and $\{I_3, I_4\}$ for t_2) and are defined on a schema *inventory* with the three attributes *name*, *producer* and *stock*. Both tuples represent the same product (and hence are duplicates), but the stock information of each tuple belongs to different orders. Therefore, in this scenario neither 15, 20, 6 nor 8 items of this product, but rather 21, 23, 26 or 28 items are available. As a consequence, the true stock value of this product results from the sum of the stocks of both base-tuples instead of being the stock of one of them. Moreover, the responsible user knows that the producer name of the second tuple ('Mars Inc.') is the correct one. Thus, this value is chosen for all possible instances of the merged tuple. Finally, it is known that the company never bought more than 25 items of an article produced by 'Mars Inc.'. Hence some of the resultant instances can be excluded for sure (see knowledge rule r). In conclusion, the result of the extended approach is much more accurate than the result of the basic approach. Moreover, by using the extended approach the final values of all attributes are nearly known for sure ('Mars Inc.' with certainty 1, 'Twix' and '23' with certainty 2/3).

The main contributions of this paper are:

- a discussion about different kinds of external knowledge and in which way these can be incorporated into the merging process.
- a detailed description of a merging approach extended by aggregation functions, knowledge rules, and instance weights. Moreover, we show that this approach is a generalization of existing methods for merging certain tuples and is a generalization of our basic approach based on the set union operator.
- a discussion about the characteristics of the extended merging approach.

The outline of the paper is as follows: First we present some basics on probabilistic data and duplicate elimination including our basic approach for probabilistic tuple merging (Section 2). Then we present the types of external knowledge we handle in this work (Section 3). In Section 4, we discuss aggregation functions, knowledge rules, and instance weights in more detail, before introducing the extended version of our probabilistic tuple merging approach in Section 5. Finally, we present related work in Section 6 and conclude in Section 7.

2 Basics

In this section, we introduce a definition of probabilistic tuples, before we present the concept of duplicate elimination including our basic approach for merging probabilistic tuples [12].

2.1 Probabilistic Tuples

In this paper, we primarily focus on the merge of tuples in relational tuple-independent probabilistic databases defined on an arbitrary probability measure P. This class of databases includes BID-tables [4], x-relations [2] without lineage (e.g. base x-relations without external lineage), and U-relations [9] where tuples with different TIDs do not share same variables.

Due to duplicates are most often detected in base relations (data cleaning) or in combining multiple independent source (data integration) duplicate elimination in tuple-independent PDBs already covers a wide space of real-world scenarios. To be independent from the used representation system (BID, MayBMS, etc.), we consider a probabilistic tuple within the *possible world semantics* [4]. Thus, similar to the ULDB model [2], we define a probabilistic tuple as a set of possible mutually exclusive instances, also denoted as tuple alternatives [2] and define a probabilistic relation (referred to as $\mathcal{R}^p$) as a set of probabilistic tuples.

Definition 1 (Probabilistic Tuple): *Let $sch(\mathcal{R})$ be a relation schema with the domain $dom(\mathcal{R})$. A probabilistic tuple t defined on $sch(\mathcal{R})$ is a set of possible instances $pI(t) = \{I_1, \ldots, I_k\}$ where each instance is an ordinary tuple $I_i \in dom(\mathcal{R})$. Moreover, each instance $I \in pI(t)$ is assigned with a probability $P(t[I]) > 0$ where $t[I]$ is the event that I is the true instance of t. Trivially, all possible instances of t are mutually exclusive: $(\forall I_1, I_2 \in pI(t)) : P(t[I_1] \mid t[I_2]) = 0$. The probability that t exist is: $P(t) = \sum_{I \in pI(t)} P(t[I]) \leq 1$.*

Since all tuples are independent of each other, the true instantiation of one tuple does not depend on the true instantiation of another tuple:

$$(\forall t_1, t_2 \in \mathcal{R}^p, t_1 \neq t_2) : (\forall I_1 \in pI(t_1), I_2 \in pI(t_2)) : P(t_1[I_1] \mid t_2[I_2]) = P(t_1[I_1])$$

To make some of the considerations of this paper easier, we introduce the null-instance $I_\perp$ which represents the case a probabilistic tuple does not exist ($P(t[I_\perp]) = 1 - P(t)$). The null-instance is schemaless, i.e. $\pi_A(I_\perp) = I_\perp$ for every valid set of attributes A and $I_\perp \times S = I_\perp$ for every relation S. For simplification, we also define the set $pI_\perp(t)$:

$$pI_\perp(t) = \begin{cases} pI(t) \cup \{I_\perp\}, & \text{iff } P(t) < 1 \\ pI(t), & \text{else} \end{cases} \quad (1)$$

Note that the null instance $I_\perp$ and hence the set $pI_\perp(t)$ are only virtual and not stored in the database.

In the rest of the paper, we represent the set of possible instances of a probabilistic tuple by an own table (one row per instance). Figure 2.1 shows an example of a tuple modeling the movie 'Crash' with two possible instances I_1 and I_2.

	title	year	studio	total	P
I_1	Crash	2004	WB	12k	0.5
I_2	Cash	2005	US	15k	0.2
$I_\perp$					0.3

Fig. 2. A probabilistic tuple t with $pI_\perp(t) = \{I_1, I_2, I_\perp\}$, $I_1 =$ ("Crash",2004,"WB",12k), $I_2 =$ ("Cash",2005,"US",15k), $P(t[I_1]) = 0.5$, $P(t[I_2]) = 0.2$, and $P(t[I_\perp]) = 0.3$

2.2 Duplicate Elimination

In the first duplicate elimination step multiple representations of same real-world entities are detected [7]. The result of this step is a partitioning of the set of input tuples into duplicate cluster (one cluster for each real-world entity).

In the second step, all tuples of one duplicate cluster are merged to a single one. This step is usually denoted as tuple merging [19] or data fusion [3]. In [12] we gave a first discussion on tuple merging in probabilistic data. We split probabilistic tuple merging into two steps: (a) a merging of instance data and (b) a merging of tuple membership. In this paper, we consider only merging of tuples defined in same contexts and hence we simply include membership merging in instance merging by using the null-instance $I_\perp$. In the following, we always consider a single cluster and hence denote the merged tuple as t_μ.

Basic Merging Approach. The basic approach for probabilistic tuple merging we present in [12] is based on the set union operator. This means that an instance is possible for the merged tuple, if it was possible for at least one base-tuple. Since the merging is not associative, if the resultant probability is simply computed by the probabilities of the base-tuples, we assign a weight $w(t)$ to each base-tuple t and define that the weight of a merged tuple $t_\mu = \mu(\{t_1, \ldots, t_k\})$ results from the sum of the weights of its base-tuples ($w(t_\mu) = w(t_1) + \ldots + w(t_k)$). If tuple merging is considered within the context of data integration, the reliabilities of the corresponding sources can be used as tuple weights. Probabilities are computed by a weighted average. Let t_μ be the tuple merged from the base-tuples of cluster C, our basic approach of probabilistic tuple merging can be formalized as:

$$pI_\perp(t_\mu) = \bigcup_{t \in C} pI_\perp(t) \quad (2) \qquad \forall I \in pI_\perp(t_\mu), \quad P(t_\mu[I]) = \sum_{t \in C} \frac{w(t)}{w(t_\mu)} P(t[I]) \quad (3)$$

3 Domain Knowledge and User Expertise

In this paper, we consider two different kinds of external knowledge: (i) domain knowledge which is generally applicable for a specific domain, as for example the information that stock values have to be summed up (see motivating example) and (ii) user expertise which is only applicable for individual items or groups of items (sources, tuples, values, etc.), as for example the information that the studio 'WB' does not produce movies for adults. Domain knowledge usually concerns metadata like the correct semantics of relational schemas or the correct

scope of attribute domains. Therefore, domain knowledge does not change over time very frequently and hence acquired once, it can be used for multiple merging processes. In contrast, user expertise often concerns only the given instance data (see example above). Thus, a user can be very competent for the data of one merging process and incompetent for the data of another, even if both processes work on equivalent schemas. For example, a user can be an expert for horror movies and a non-expert for romantic movies.

With respect to its effects on the merge result, external knowledge can be furthermore classified into the following four types:

1. Knowledge about specific semantics of individual attributes or sets of attributes. For example, the knowledge that the numbers of sold tickets stored in duplicate entries in a box office list belong to different points of time and hence the maximum value has to be chosen. Knowledge of this type is usually domain knowledge.
2. Knowledge about the true instance of one attribute value or a set of attribute values. For example, the user knows that one of the given values is the true one, or the user knows that the true value is missing and introduces it himself. Knowledge of this type usually results from user expertise.
3. Knowledge required for excluding some combinations of attribute values for sure. For example, such knowledge can be based on physical rules (e.g. a studio cannot have produced movies before it was founded), economical rules (e.g. a salary cap), or private guidelines (e.g. a specific company never buys more than 100 items from a single article per month). Knowledge of this type can be domain knowledge, user expertise or a combination of both.
4. Knowledge about new evidence or further evidence, or a user's own degree of belief. For example, at merging time a person is known to life rather in Italy than in France. Knowledge of this type is most often user expertise.

A modeling of knowledge about tuple correlations (e.g. a specific attribute has to be unique) implies a definition of new tuple dependencies. Since we restrict to tuple-independent PDBs, this is out of the scope of this paper.

4 Methods for Incorporating External Knowledge

For incorporating domain knowledge and user expertise, we resort to three classical concepts which have already been partially used in the merge of certain data: user-defined aggregation functions, user-defined knowledge rules, and user-defined weights of possible instances.

Aggregation functions can be used to assign specific semantics to concrete attributes (knowledge of Type 1) or to define the true value for a concrete attribute by hand (knowledge of Type 2). In contrast, knowledge rules are excellently suited for excluding instances which violate a given pattern of regulations (knowledge of Type 3). Instance weights can be used to accommodate new evidence (knowledge of Type 4).

Aggregation functions are already used for resolving conflicts in the fusion of certain data by Bleiholder et al. [3]. Note that we use the concept of aggregation

for another purpose. In the merge of certain data for each attribute only a single value can be stored. Thus, conflicting values need to be resolved and the usage of aggregation functions is often mandatory, even if the user does not know how to aggregate these values at best. In contrast, such a conflict resolution is not required by using a probabilistic data model as the target model because all possible values can be stored simultaneously. We use these functions only for incorporating available context information. Therefore, in our approach these functions should be only used, if this information is given for sure (or at least very likely). This in turn implies that a lot of aggregation functions listed in [3] are not suitable for our purpose (e.g. First, Last, Random, etc.), because they do not express a certain kind of knowledge.

Knowledge rules are already used by Whang et al. [19] for preventing invalid merging results and hence for detecting the best merging easier. In general, we use these rules for same purposes, because we use them to avoid invalid instances.

In the rest of this section, we take a closer look at these three concepts and how they can be used to express a certain kind of knowledge.

4.1 Aggregation Functions

Aggregation functions are a simple and adequate method for incorporating external knowledge into a tuple merging process. For aggregation we consider functions as defined for conflict resolution in certain data [3]. This set of functions can be classified into deciding functions which choose one of the given values and mediating functions which create a new value.

Moreover, aggregation functions can be of a simple or complex nature. A simple aggregation function only takes the values of the considered attribute into account or is a constant function which does not need any input at all. In contrast, complex aggregation functions also consider the values of other attributes (from input as well as output) as own input. Thus, they aggregate a set of given input values depending from the result of aggregating other attributes or from the initial values of other attributes. As an example, consider a deciding function with takes the value for attribute a_i that occurs as most often with the value already chosen for attribute a_j.

Simple Aggregation Functions. A simple aggregation function aggregates only the values of the attribute it is defined for or returns a constant. Let f_i be a simple aggregation function which aggregates the input values $A_i^I = \{v_1, \ldots, v_m\}$ of an attribute a_i to the single output value $v_{f_i} \in dom(a_i)$, $f_i(A_i^I)$ is defined as:

$$f_i : dom(a_i)^m \rightarrow dom(a_i) \qquad f_i : \{v_1, \ldots, v_m\} \mapsto v_{f_i}$$

If for aggregating a set of instances M only simple functions are used (each for another attribute), each function can be applied independently and the output instance I_O results by the cross product of all the functions' output values. Let $A = \{a_1, \ldots, a_n\}$ be a set of attributes an aggregation function is defined for (f_i for a_i), the result from aggregating the input set $\pi_A(M)$ with $f_1 - f_n$ is:

$$I_O = f_1(\pi_{a_1}(M)) \times f_2(\pi_{a_2}(M)) \times \ldots \times f_n(\pi_{a_n}(M)) = (v_{f_1}, v_{f_2}, \ldots, v_{f_n})$$

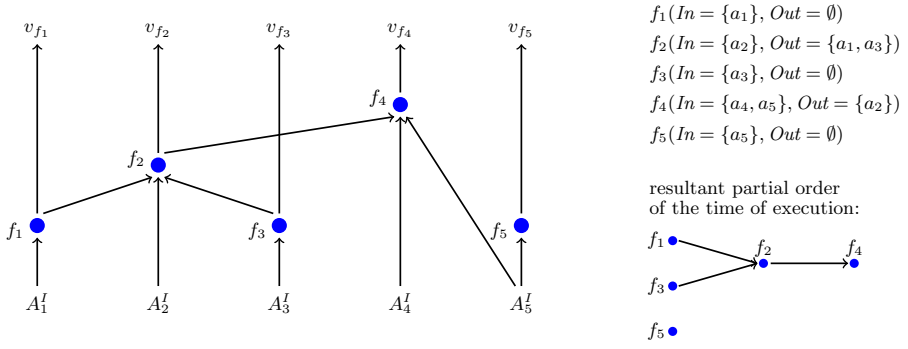


Fig. 3. Set of five sample aggregation functions $\{f_1, \ldots, f_5\}$ along with the dependencies between their input $A_i^I = \pi_{a_i}(M)$ and output $v_{f_i} = f_i(In, Out)$ (left side) and the resultant partial order of their time of execution (right side, below)

Complex Aggregation Functions. A complex aggregation function also consider the input values and/or output values of other functions for producing its own output. Thus, we need a more general definition of aggregation functions. In the following, a function f_i aggregating the values of attribute a_i is described by a set of attributes In which values are used from the input data[1] and by a set of attributes Out which values are used from the output data. Note, for each $a_j \in In$ a set of input values is used, i.e. A_j^I, but for each $a_j \in Out$ only a single output value is processed, i.e. v_{f_j}. Thus, $f_i(In, Out)$ is defined as:

$$f_i : \{A_j^I \mid a_j \in In\} \times \{v_{f_j} \mid a_j \in Out\} \mapsto v_{f_i}$$

Note, by using this description, a function f_i is simple, if the set In contains at most the attribute a_i and the set Out is empty: $f_i(In = \{a_i\}, Out = \emptyset)$.

The combined execution of aggregation functions becomes much more complicated, if complex functions are involved. Certainly, an output value of one function has to be produced before it can be serve as the input for another function. In general, a given set of complex aggregation functions has to be executed according to the partial order of the dependencies between their input values and output values. This fact is illustrated in Figure 3 where the dependencies between the input values and output values of five aggregation functions (left side) as well as their resultant partial order (right side) are depict.

The combined execution of a set of complex aggregation functions F for aggregating the set of instances M to the output instance I_O using the set of attributes A as input is in the following denoted as $I_O = \mathfrak{F}(F, \pi_A(M))$.

4.2 Knowledge Rules

Compared to aggregation functions, knowledge rules enhance the capability to incorporate knowledge further on in two ways: First, whereas aggregation

[1] The input of a function can also contain metadata like the age of a value [3].

functions only influence the set of attributes $\{a_l, \ldots, a_n\}$ for which such a function is defined, by the usage of knowledge rules conditions for whole instances can be specified. Second, it is in the nature of aggregation functions that they choose a single value and exclude all other ones. Nevertheless, such a restrictive knowledge is often not available (knowledge of Type 3), but instead we can only exclude a single value (or few values) to be the true one. Such restrictions of the set of possible instances, however, cannot be realized by aggregation functions.

Knowledge rules are logical rules of inference (*premises* $\rightarrow$ *conclusions*) which take premises and return conclusions. A rule is violated by an instance, if for this instance all premises are valid, but the conclusions are not. In this way, impossible instances can be excluded from the merging result.

As an example we consider a combination of the general domain knowledge that studios cannot have produced movies before they were founded and the specific user expertise that the studio 'WB' (Warner Bros. Entertainment) was founded in 1923. Thus, we can conclude that each instance having the value 'WB' as studio name and having a value year lower than 1923 cannot be true:

$$\text{rule } r_1 : \text{studio='WB'} \rightarrow \text{year} \geq 1923 \tag{4}$$

A knowledge rule can use values from the output (the instances of the merged tuple) as well as values from the input (the instances of the base-tuples). One meaningful example is the condition that a combination of values for an attribute set A is only valid for the merged tuple, if it was valid for at least one base-tuple:

$$\text{rule } r_2 : I \in \pi_A(pI(t_\mu)) \rightarrow (\exists t \in C) : I \in \pi_A(pI(t)) \tag{5}$$

Knowledge rules are applied to each possible instance individually. Instances violating one or more of the defined rules can be excluded to be the true one and hence are removed from the merging result.

4.3 Instance Weights

In our basic approach (Section 2.2), we use tuple weights for (a) making the merging process associative and (b) allowing an assignment of different degrees of trust to individual sources. To make the merging process more adaptable to further evidence known at merging time, we also allow a definition of weights on instance level: $w(t, I)$ is the weight of instance I for tuple t. Thus, the user can prefer a base-instance I_1 to another base-instance I_2 ($w(t, I_1) > w(t, I_2)$) or can exclude a base-instance I from the merging process for sure ($w(t, I) = 0$) without manipulating the original probabilities. Typically, weights are assigned for each instance individually and hence represent user expertise. Nevertheless, weights can be also assigned by a given pattern (e.g. a weight w is assigned to instances satisfying a specific condition derived from the semantics of the considered universe of discourse) and hence also can be used to express domain knowledge.

We define the weight of a tuple t as the expected weight of its instances: $w(t) = \sum_{I \in pI(t)} w(t, I)P(t[I])$. For ensuring associativity, the weight of a merged tuple is still the sum of the tuple it is merged from. Moreover, all instances of the merged tuple are weighted equally, i.e. the new evidence is already incorporated in the resultant probabilities.

5 Extended Approach for Probabilistic Tuple Merging

For incorporating external knowledge according to the possible world semantics, the aggregation functions have to be applied to each possible combination (so called *merging lists*) of the base-tuples' instances (one instance per base-tuple) individually. Each merging list $M = \{I_1, \ldots, I_k\}$ contains as many instances as base-tuples to be merged (in this case k). We consider the instances in a merging list to be sorted by their corresponding base-tuples, meaning that I_i originates from tuple t_i. Performing aggregation on a single merging list is denoted as fusion. Knowledge rules are applied to the merging lists' fused instances.

Let $C = \{t_1, \ldots, t_k\}$ be a set of base-tuples to be merged. Let $w(t, I)$ be the weight defined for instance $I \in pI(t), t \in C$ and let $w(t_\mu) = \sum_{t \in C} w(t)$ be the total weight of all base-tuples. Moreover, let $N = \{r_1, \ldots, r_q\}$ be a set of knowledge rules and let $A = \{a_1, \ldots, a_{l-1}, a_l, \ldots, a_n\}$ be the attributes of the considered schema, where for each of the attributes $A_2 = \{a_l, \ldots, a_n\}$ an aggregation function is defined for (f_i for a_i). Our tuple merging approach extended with aggregation functions, knowledge rules, and instance weights is performed by the following steps (the first two steps are illustrated in Figure 4):

1. **Divide the Input.** First, all merging lists are built:

$$\mathcal{M}(\{t_1, t_2, \ldots, t_k\}) = \{\{I_1, I_2 \ldots, I_k\} \mid I_i \in pI_\perp(t_i), t_i \in C\} \tag{6}$$

2. **Apply Aggregation.** Then, each merging list $M \in \mathcal{M}(C)$ is fused by applying the set of aggregation functions $F = \{f_l, \ldots, f_n\}$ (defined for attributes $\{a_l, \ldots, a_n\}$) having the attribute set $A_I \subseteq A$ as input. For the attributes without any aggregation function (attributes $A_1 = \{a_1, \ldots, a_{l-1}\}$) all possible values are taken into account (recall $I_\perp \times S = I_\perp$ and $\pi_A(I_\perp) = I_\perp$):

$$\forall M \in \mathcal{M}(C), \quad \mu(M) = \pi_{A_1}(M) \times \mathfrak{F}(F, \pi_{A_I}(M)) \tag{7}$$

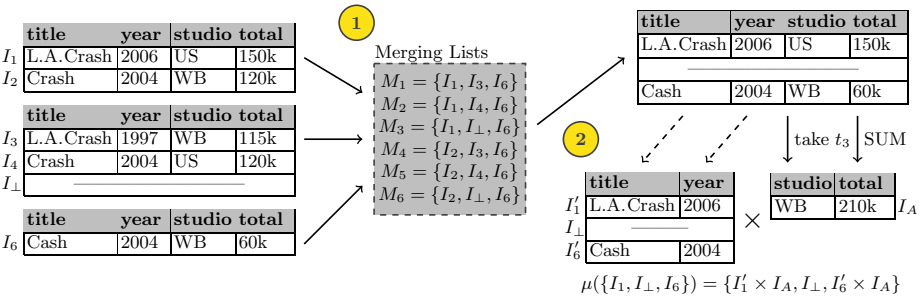
   During this step the probabilities of the resultant instances can be directly computed. Let $P(M) = \prod_{I \in M} P(I)$ be the probability[2] of the merging list M. The probability of an instance I_μ dependent on M results in:

$$P(I_\mu \mid M) = \frac{1}{w(t_\mu)} \times \sum\nolimits_{I_i \in M, \pi_{A_1}(I_i) = \pi_{A_1}(I_\mu)} w(t_i, I_i) \tag{8}$$

   Note, the probabilities of duplicate instances eliminated by the relational projection operator in Formula 7 are added up. Duplicate instances resulting from the fusion of different merging lists are handled in Step 4.

3. **Apply Rules.** Third, the set of knowledge rules N is checked for each instance resulting from fusing the merging list M. If an instance is invalid for at least one rule, this instance is removed from the fusion result $\mu(M)$.

[2] Due to all tuples are independent of each other, the probability of M is equal to the product of the probabilities of all its instances $I \in M$.



	title	year	studio	total
I_1	L.A.Crash	2006	US	150k
I_2	Crash	2004	WB	120k

	title	year	studio	total
I_3	L.A.Crash	1997	WB	115k
I_4	Crash	2004	US	120k
$I_\perp$				

	title	year	studio	total
I_6	Cash	2004	WB	60k

	title	year	studio	total
	L.A.Crash	2006	US	150k
	Cash	2004	WB	60k

	title	year
I'_1	L.A.Crash	2006
$I_\perp$		
I'_6	Cash	2004

$\times$

studio	total	
WB	210k	I_A

Fig. 4. Building of all merging lists (Step 1) and fusing the merging list $M_3 = \{I_1, I_\perp, I_6\}$ (Step 2). The total values are added up (mediating function). For the studio name the value from the instance of tuple t_3 is chosen (deciding function). For the movie title and the production all values are taken into account.

4. **Combine Results.** Finally, the fusion results of all merging lists are combined to the final set of possible instances $pI_\perp(t_\mu)$ and all probabilities $P(t_\mu[I]), \forall I \in pI_\perp(t_\mu)$ are computed. If instances are eliminated by knowledge rules, a final normalization need to be applied.

$$pI_\perp(t_\mu) = \bigcup_{M \in \mathcal{M}(C)} \mu(M) \qquad (9) \qquad\qquad P(t_\mu[I]) = \sum_{M \in \mathcal{M}(C)} P(I \mid M)\, P(M) \qquad (10)$$

Note, if for all attributes an aggregation function is defined ($l = 1$), for each merging list a single possible instance results (we denote this setting a full-aggregation). Otherwise, for each merging list at most as many instances as base-tuples can result (one for each of the merged instances).

If the instance data of each base-tuple is certain (each tuple has exact one possible instance), only one merging list is built. If in addition a full-aggregation is applied, from tuple merging a single possible instance results. Thus, this approach is a generalization of conflict resolution used for tuple merging in certain databases. In contrast, if no aggregation function is defined, the result contains each instance possible for at least one base-tuple. Therefore, this approach is also a generalization of our basic approach for probabilistic tuple merging.

In the example of Figure 4, three tuples of a relation *box office* are merged. Two aggregation functions are defined. The total amount of sold tickets results in the sum of all values of attribute *total* (mediating function). Moreover, the user knows that the *studio* name of the third tuple is correct. For that reason, this value is chosen for all instances (deciding function). For the attributes *title* and *year* no functions are specified. Thus, all possible values are taken into account.

5.1 Scalability

Given a duplicate cluster of size k. Assume that each tuple has averagely l possible instances (including $I_\perp$). From the basic approach at most $k \times l$ instances result. In contrast, in the extended approach l^k merging lists are built. Thus, if aggregation functions are used, at most $k \times l^k$ instances can be result. This is an

increase to the basic approach of l^{k-1} times. Experience has shown that cluster rarely have more than 5 tuples and uncertainty often can be adequate modeled by around 10 possible instances. Certainly, this increase is still tremendous ($5{\cdot}10^5$ instead of 50), but can be flexibly reduced to a desired amount of data by only taking the most likely (resultant or base) instances into account. An important reflection of future work is to compute the most probable resultant instances more efficiently by pruning of irrelevant merging lists first.

5.2 Characteristics of Merging Approaches

In [12], we introduce a set of characteristics of merging approaches which are useful in many merging scenarios. Some of these characteristics are:

- **Independence of the Merging Order:** The tuple resulting from merging multiple base-tuples should be independent from the merging order. This requirement is important, if tuple merging is considered as a part of a data integration process and the integration is performed in a pairwise fashion instead of integrating all sources at one time. This independence is given, if tuple merging is associative:

$$\mu(T) = \mu(\{\mu(T \setminus T_i), \mu(T_i)\}) \text{ for all } T_i \subset T \tag{11}$$

- **Stability:** We denote a merging to be stable, if from deduplicating a duplicate free relation the relation itself results. This property is given, if tuple merging is idempotent ($\mu(\{t\}) = t$).

- **SP-Query Consistency:** Query consistency means that the result of querying a merged tuple should be equal to merging the tuples resulting from querying the individual base-tuples. Since we only consider queries on single tuples, joins and set-based operators are not taken into account. Moreover, aggregation functions change the tuples' schema. Thus, a consistency w.r.t. queries with aggregations is generally not possible and we restrict to the probabilistic equivalences of the algebraic operations selection and projection (we use the definition of the world-set algebra [9]). In the following, we define this class of queries as SP-Queries. Let $\mathcal{R}^p$ be a probabilistic relation and let Q be the set of all possible SP-Queries which can be formulated on $\mathcal{R}^p$, the requirement of SP-Query consistency is formalized as:

$$(\forall T \subseteq \mathcal{R}^p, \forall q \in Q) : \mu(\{q(t) \mid t \in T\}) = q(\mu(T)) \tag{12}$$

  SP-Query consistency is very useful to reduce the dataflow between source databases and target database, because irrelevant data (tuples in case of selection, attributes in case of projection) can be already excluded at the source databases' sides. This is especially important, if data are paid by amount. However, for doing that there must be a certain attribute serving as real-world identifier. Otherwise, the duplicate detection result can be influenced by an early performing of SP-Queries.

Now, we discuss the influences on these characteristics by using aggregation functions, knowledge rules, and instance weights:

Basic Approach: The basic approach is independent of the merging order, SP-Query consistent and stable.

Ext. Approach + Instance Weights: Instance weights only affect stability, but non-stability is even the goal of taking further evidence into account.

Ext. Approach + Aggregation Functions: The approach extended with aggregation functions (simple or complex) is associative, if all these functions are associative and stable, if all these functions are idempotent. Both properties are satisfied by aggregation functions usually used for modeling attribute semantics (e.g. sum, max). The extended approach is not consistent with selections, because a removing of tuples and instances definitely influences the set of values to be aggregated. This approach, however, is consistent with projections, if either only simple aggregation functions are used or none of the excluded attributes is required as input for one of the aggregation functions defined for the remaining attributes.

Ext. Approach + Knowledge Rules: The approach extended with knowledge rules is stable, if none of the base-instances violates a rule, i.e. the source data is consistent with the set of given rules. Independence of merging order is not satisfied, if an intermediate result violates a rule which would not be violated by the final result. This can only happen, if new values arise and hence only, if knowledge rules are used in combination with aggregation functions. Nevertheless, the most useful aggregation functions are monotonically increasing (sum, max) or monotonically decreasing (min). Thus, rules restricting an attribute domain to a range with an lower and an upper bound (e.g. the stock value is between 10 and 100) do not pose a problem. Finally, this approach is consistent with selection, but not consistent with projection, in cases one of the excluded attributes is used in a rule.

In summary, the merging approach extended with aggregation functions, knowledge rules, and instance weights guarantees independence of the merging order in the most useful scenarios (associative aggregation functions and meaningful rules). Stability can be ensured, if wanted by using only tuple-uniformly weighted instances and cleaning data first. Consistency with selection cannot be ensured in general, if aggregation functions are used. In contrast, if attributes are projected carefully, consistency with projection can be achieved easily.

6 Related Work

Tuple merging in certain data is considered in different works [5,3,11,19]. Since in certain data only single values can be stored, conflicts always have to be resolved by applying aggregation functions. In contrast, because we process probabilistic data, in our approach such functions are not mandatory, but a helpful capability to incorporate external knowledge into the merging process.

Robertson et al. [14] consider tuple merging within a transposition of certain data. Merging of two tuples with contrary instance data is not provided (in such cases both tuple are denoted to be *non mergeable*).

DeMichiel [6] and Tseng et al. [16] use *partial values* (resp. *probabilistic values*) to resolve conflicts between certain values by taking multiple possible instances into account. Consequently, these approaches already produce uncertain data as result data. This is similar to our basic approach for instance merging if each base-tuple is considered to be certain. Nevertheless, both approaches consider conflict resolution on an attribute by attribute basis. Dependencies between possible attribute values are not considered.

Andritsos et al. [1] define queries on multiple conflicting duplicates. Thus instead of merging the tuples of each cluster into a single one, query results are derived from sets of mutual exclusive base-tuples. Since to each cluster's tuple a probability can be assigned, this approach is mostly identical to our basic approach applied to certain base-tuples. However, concepts for incorporating external knowledge are not provided.

Van Keulen et al. [17] store conflicting duplicates in a probabilistic database and use user feedback to resolve these conflicts at query time. Thus, they do not directly incorporate the user expertise into the merging process.

A merging of tuples representing uncertain information is proposed by Lim et al. [10], but instead of probability theory this approach is based on the *Dempster-Shafer theory of evidence* and hence is not applicable for probabilistic data.

None of these studies, however, allows probabilistic data as source data.

7 Conclusion

Many applications naturally produce probabilistic data. For integrating probabilistic data from multiple sources in a consistent way or to clean a single database duplicate tuples need to be identified and merged. We consider duplicate detection in probabilistic data in [13] and introduce a basic approach for merging the instance data of probabilistic tuples in [12]. In this paper, we focus on the incorporation of external knowledge (domain knowledge or user expertise) in the merging process making the merging result more accurate. For that purpose, we generalize our basic approach by incorporating aggregation functions, knowledge rules, and instance weights. Finally, we analyze in which way these extensions influence several important characteristics of merging processes.

In future research we aim to make the merging process more adaptable for individual needs. The merging approach based on the set union operator produces data correct as possible. Moreover, this approach is associative and SP-Query consistent. Nevertheless, the more base-tuples are merged, the more possible instances result. Thus, the merged tuple becomes more and more uncertain. For that reason, we need new methods enabling the user to make a trade-off between correctness and certainty being best for him. Finally, we are working on techniques making the extended merging approach more efficient.

References

1. Andritsos, P., Fuxman, A., Miller, R.J.: Clean Answers over Dirty Databases: A Probabilistic Approach. In: ICDE, p. 30–41 (2006)
2. Benjelloun, O., Sarma, A.D., Halevy, A., Widom, J.: ULDBs: Databases with Uncertainty and Lineage. In: VLDB, pp. 953–964 (2006)
3. Bleiholder, J., Naumann, F.: Data fusion. ACM Comput. Surv. 41(1) (2008)
4. Dalvi, N.N., Suciu, D.: Efficient query evaluation on probabilistic databases. VLDB J. 16(4), 523–544 (2007)
5. Dayal, U.: Processing Queries Over Generalization Hierarchies in a Multidatabase System. In: VLDB, pp. 342–353 (1983)
6. DeMichiel, L.G.: Resolving Database Incompatibility: An Approach to Performing Relational Operations over Mismatched Domains. IEEE Trans. Knowl. Data Eng. 1(4), 485–493 (1989)
7. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate Record Detection: A Survey. IEEE Trans. Knowl. Data Eng. 19(1), 1–16 (2007)
8. Khoussainova, N., Balazinska, M., Suciu, D.: Probabilistic event extraction from rfid data. In: ICDE, pp. 1480–1482 (2008)
9. Koch, C.: MayBMS: A System for Managing Large Uncertain and Probabilistic Databases. In: Managing and Mining Uncertain Data. Springer, Heidelberg (2009)
10. Lim, E.-P., Srivastava, J., Shekhar, S.: An Evidential Reasoning Approach to Attribute Value Conflict Resolution in Database Integration. IEEE Trans. Knowl. Data Eng. 8(5), 707–723 (1996)
11. Motro, A., Anokhin, P.: Fusionplex: Resolution of Data Inconsistencies in the Integration of Heterogeneous Information Sources. Information Fusion 7(2), 176–196 (2006)
12. Panse, F., Ritter, N.: Tuple Merging in Probabilistic Databases. In: MUD, pp. 113–127 (2010)
13. Panse, F., van Keulen, M., de Keijzer, A., Ritter, N.: Duplicate Detection in Probabilistic Data. In: NTII, pp. 179–182 (2010)
14. Robertson, E., Wyss, C.M.: Optimal Tuple Merge is NP-Complete. Technical Report TR599, IUCS (2004)
15. Suciu, D., Connolly, A., Howe, B.: Embracing Uncertainty in Large-Scale Computational Astrophysics. In: MUD, pp. 63–77 (2009)
16. Tseng, F.S.-C., Chen, A.L.P., Yang, W.-P.: Answering Heterogeneous Database Queries with Degrees of Uncertainty. Distributed and Parallel Databases 1(3), 281–302 (1993)
17. van Keulen, M., de Keijzer, A.: Qualitative effects of knowledge rules and user feedback in probabilistic data integration. VLDB J. 18(5), 1191–1217 (2009)
18. Wang, D.Z., Michelakis, E., Franklin, M.J., Garofalakis, M., Hellerstein, J.M.: Probabilistic declarative information extraction. In: ICDE, pp. 173–176 (2010)
19. Whang, S.E., Benjelloun, O., Garcia-Molina, H.: Generic entity resolution with negative rules. VLDB J. 18(6), 1261–1277 (2009)