VLDBJ manuscript No.
(will be inserted by the editor)

10^(10^6) Worlds and Beyond: Efficient Representation and Processing of Incomplete Information

Lyublena Antova · Christoph Koch · Dan Olteanu

the date of receipt and acceptance should be inserted later

Abstract We present a decomposition-based approach to managing probabilistic information. We introduce *world-set decompositions (WSDs)*, a space-efficient and complete representation system for finite sets of worlds. We study the problem of efficiently evaluating relational algebra queries on world-sets represented by WSDs. We also evaluate our technique experimentally in a large census data scenario and show that it is both scalable and efficient.

1 Introduction

Incomplete information is commonplace in real-world databases. Classical examples can be found in data integration and wrapping applications, linguistic collections, or whenever information is manually entered and is therefore prone to inaccuracy or incompleteness.

As a motivation, consider two manually completed forms that may originate from a census and which allow for more than one interpretation (Figure 1). For simplicity we assume that social security numbers consist of only three digits. For instance, Smith's social security number can be read either as "185" or as "785". We can represent the available information using a relation with or-sets:

(TID)	S	N	M
t_1	{ 185, 785 }	Smith	{ 1, 2 }
t_2	{ 185, 186 }	Brown	{ 1, 2, 3, 4 }

This relation represents $2 \cdot 2 \cdot 2 \cdot 4 = 32$ *possible worlds*.

This article is an extended version of the paper with the same name that appeared in the Proceedings of the International Conference on Data Engineering (ICDE) 2007 [7].

L. Antova (lantova@cs.cornell.edu)
Cornell University

C. Koch (koch@cs.cornell.edu)
Cornell University

D. Olteanu (dan.olteanu@comlab.ox.ac.uk)
Oxford University

Given such an incompletely specified database, it must of course be possible to access and process the data. Two data management tasks shall be pointed out as particularly important, query evaluation and *data cleaning* [25, 16, 26], by which certain worlds can be shown to be impossible and can be excluded. The results of both types of operation turn out not to be representable by or-set relations in general. Consider for example the integrity constraint that all social security numbers be unique. For our example database, this constraint excludes 8 of the 32 worlds, namely those in which both tuples have the value 185 as social security number. It is impossible to represent the remaining 24 worlds using or-set relations. This is an example of a constraint that can be used for data cleaning; similar problems are observed with queries, e.g., the query asking for pairs of persons with differing social security numbers.

What we could do is store each world explicitly using a table called a *world-set relation* of a given set of worlds. Each tuple in this table represents one world and is the concatenation of all tuples in that world (see Figure 2).

The most striking problem of world-set relations is their size. If we conduct a survey of 50 questions on a population of 200 million and we assume that one in 10^4 answers can be read in just two different ways, we get 2^{10^6} worlds. Each such world is a substantial table of 50 columns and $2 \cdot 10^8$ rows. We cannot store all these worlds explicitly in a world-set relation (which would have 10^{10} columns and 2^{10^6} rows). Data cleaning will often eliminate only some of these worlds, so a DBMS should manage those that remain.

This article aims at dealing with this complexity and proposes the new notion of *world-set decompositions (WSDs)*. These are decompositions of a world-set relation into several relations such that their product (using the product operation of relational algebra) is again the world-set relation.

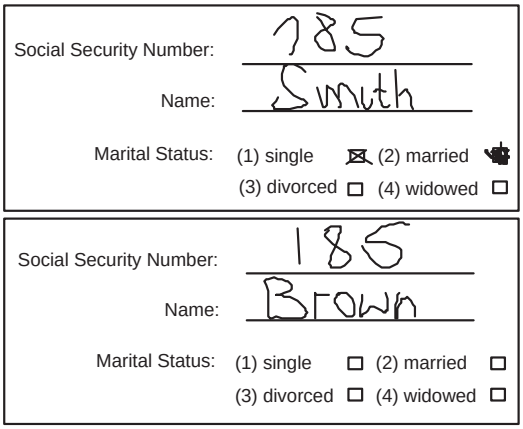


Fig. 1 Two completed survey forms.

t_1.S	t_1.N	t_1.M	t_2.S	t_2.N	t_2.M
185	Smith	1	186	Brown	1
185	Smith	1	186	Brown	2
185	Smith	1	186	Brown	3
185	Smith	1	186	Brown	4
185	Smith	2	186	Brown	1
		⋮			
785	Smith	2	186	Brown	4

Fig. 2 World-set relation containing only worlds with unique social security numbers.

Example 1 The world-set represented by our initial or-set relation can also be represented by the product

t_1.S
185
785

×

t_1.N
Smith

×

t_1.M
1
2

×

t_2.S
185
186

×

t_2.N
Brown

×

t_2.M
1
2
3
4

The WSD representation of an or-set relation requires in general the same amount of space as the or-set relation itself.□

Example 2 In the same way we can represent the result of data cleaning with the uniqueness constraint enforced on the social security numbers as the product of Figure 3.

t_1.S	t_2.S
185	186
785	185
785	186

×

t_1.N
Smith

×

t_1.M
1
2

×

t_2.N
Brown

×

t_2.M
1
2
3
4

Fig. 3 WSD of the relation in Figure 2.

The above product is exactly the world-set relation in Figure 2. The presented decomposition is based on the *independence* between sets of fields, subsequently called *components*. Only fields that depend on each other, for example $t_1.S$ and $t_2.S$, belong to the same component. Since $\{t_1.S, t_2.S\}$ and $\{t_1.M\}$ are independent, they are put into different components. □

Often, one can quantify the uncertainty of a dependency of possible values using probabilities. For example, an automatic extraction tool that extracts structured data from text can produce a ranked list of possible extractions, each associated with a probability of being the correct one [19]. WSDs can elegantly represent such uncertain data by using a new column P for each component. This column then defines the probabilities of the dependencies of the values in each component tuple.

t_1.S	t_2.S	P
185	186	0.2
785	185	0.4
785	186	0.4

×

t_1.N	P
Smith	1

×

t_1.M	P
1	0.7
2	0.3

×

t_2.N	P
Brown	1

×

t_2.M	P
1	0.25
2	0.25
3	0.25
4	0.25

Fig. 4 Probabilistic version of the WSD of Figure 3.

Example 3 Figure 4 gives a probabilistic version of the WSD of Figure 3. The probabilities in the last component state that the possible values for the marital status in tuple t_2 are equally likely. In case of t_1, it is more likely to be single (value 1) than married. The probabilities for the name values for t_1 and t_2 equal one, as this information is certain. □

Given a probabilistic WSD $\{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$, we obtain a possible world by choosing one tuple w_i out of each component relation $\mathcal{C}_i$. The probability of this world is then computed as $\prod_{i=1}^{m} w_i.P$. For example, in Figure 4, choosing the first, the second, and the third tuple from the first, the third, and the fifth component, respectively, results in the world

R	SSN	Name	MS
t_1	185	Smith	2
t_2	186	Brown	2

This world's probability is $0.2 \cdot 1 \cdot 0.3 \cdot 1 \cdot 0.25 = 0.015$.

In practice, it is often the case that fields or even tuples carry the same values in all worlds. For instance, in the census data scenario discussed above, we assumed that only one field in 10000 has several possible values. Such a world-set decomposes into a WSD in which most fields in component relations are certain, i.e., have precisely one tuple.

We will also consider a refinement of WSDs, *WSDTs*, which store information that is the same in all possible worlds once and for all in so-called *template relations*.

Example 4 The world-set of the previous examples can be represented by the WSDT of Figure 5. □

WSDTs combine the advantages of c-tables [20] and WSDs. In particular, WSDTs can be naturally viewed as c-tables where the body of the c-table corresponds to the template relation, and whose formulas have been put into a *normal form* represented by the component relations, and null

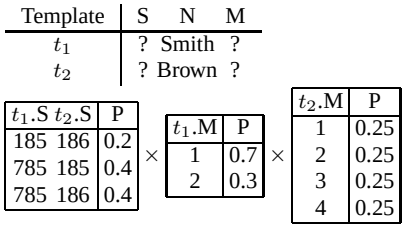


Fig. 5 Probabilistic WSD with a template relation.

values '?' in the template relations represent fields on which the worlds disagree. Indeed, each tuple in the product of the component relations is a possible value assignment for the variables in the template relation. While query evaluation needs to access both the template relation and the components, this brings advantages that are best justified in cases where most data is certain. The following c-table with global condition Φ is equivalent to the WSDT in Figure 5 (without the probabilistic weights):

T	S	N	M
	x	Smith	y
	z	Brown	w

$$\Phi = ((x = 185 \wedge z = 186) \vee (x = 785 \wedge z = 185) \vee (x = 785 \wedge z = 186)) \wedge (y = 1 \vee y = 2) \wedge (w = 1 \vee w = 2 \vee w = 3 \vee w = 4)$$

The technical contributions of this article are as follows.

- We formally introduce WSDs and WSDTs and study some of their properties. Our notion is a refinement of the one presented above and allows to represent worlds over multi-relation schemas which contain relations with varying numbers of tuples. WSD(T)s can represent any finite set of possible worlds over relational databases and are therefore a strong representation system for *any relational query language*.
- A practical problem with WSDs and WSDTs is that a DBMS that manages such representations has to support relations of arbitrary arity: the schemata of the component relations of a decomposition depend on the data. Unfortunately, DBMSs (e.g. PostgreSQL) in practice often do not support relations beyond a fixed arity.
  For that reason we present refinements of the notion of WSDs, the *uniform WSDs (UWSDs)*, and their extension by template relations, the *UWSDTs*, and study their properties as representation systems.
- We show how to process relational algebra queries over world-sets represented by UWSDTs. For illustration purposes, we discuss query evaluation in the context of the more visual WSDs.
  We also develop a number of optimizations and techniques for normalizing the data representations obtained by queries to support scalable query processing even on very large world-sets.
- We develop data cleaning techniques in the context of WSDs. We focus on two kinds of dependencies, functional dependencies and a class of equality-generating dependencies, and adapt the *Chase procedure* (cf. [24, 3, 18]) for incomplete information to the framework of WSDs.
- We describe a prototype implementation built on top of the PostgreSQL RDBMS. Our system is called MayBMS[1] and supports the management of incomplete information using UWSDTs.
- We report on our experimental evaluation of UWSDTs as a representation system for large finite sets of possible worlds. Our experiments show that UWSDTs yield scalable techniques for managing incomplete information. We found that the size of UWSDTs obtained as query answers or data cleaning results remains close to that of a single world. Furthermore, the query processing time is also comparable to processing just a single world and thus a classical relational database.

WSDs are designed to cope with large sets of worlds which exhibit local dependencies and large commonalities. This data pattern can be found in many applications. Besides the census scenario, Section 10 describes two further applications: managing inconsistent databases using minimal repairs [10, 12] and medical data.

A fundamental assumption of this work is that one wants to manage *finite sets of possible worlds*. This is justified by previous work on representation systems starting with Imielinski and Lipski [20], by recent work [15, 4, 11], and by current application requirements. Our approach can deal with databases with unresolved uncertainties. Such databases are still valuable. It should be possible to do data transformations that preserve as much information as possible, thus necessarily mapping between sets of possible worlds. In this sense, WSDs represent a *compositional framework* for querying and data cleaning. A different approach is followed in, e.g., [10, 13], where the focus is on finding *certain answers* of queries on incomplete and inconsistent databases.

Related Work. Early work on managing incomplete information in the relational setting was presented in [20] which introduced *v-tables* and *c-tables*. In v-tables the tuples can contain both constants and variables, and each combination of possible values for the variables yields a possible world. Relations with or-sets [21] can be viewed as v-tables, where each variable occurs only at a single position in the table and can only take values from a fixed finite set, the or-set of the field occupied by the variable. The so-called *c-tables* [20] extend v-tables with conditions specified by logical formulas over the variables, thus constraining the possible values,

[1] The version of the MayBMS system released in early 2009, available at http://maybms.sourceforge.net, uses a representation system other than WSDs.

and form a strong representation system for relational query languages.

The probabilistic databases of [15, 14] and the dirty relations of [4] are examples of practical representation systems that are not strong for relational algebra. As query answers in general cannot be represented as a set of possible worlds in the same formalism, query evaluation is focused on computing the certain answers to a query, or the probability of a tuple being in the result. Such formalisms close the possible worlds semantics using clean answers [4] and probabilistic-ranked retrieval [15]. As we will see in this article, our approach subsumes the aforementioned two and is strictly more expressive than them.

In parallel to our approach, [28, 11] propose ULDBs that combine uncertainty and a low-level form of lineage to model any finite world-set. Like the dirty relations of [4], ULDBs represent a set of independent tuples with alternatives. Lineage is then used to represent dependencies among alternatives of different tuples and thus is essential for the expressive power of the formalism. Note that lineage corresponds to local conditions in c-tables [20].

As both ULDBs and WSDs can model any finite world-set, they inherently share some similarities, yet differ in important aspects. WSDs support efficient algorithms for finding a minimal data representation based on relational factorization. Differently from ULDBs, WSDs allow representing uncertainty at the level of tuple fields, not only of tuples. This causes, for instance, or-set relations to have linear representations as WSDs, but (in general) exponential representations as ULDBs. As reported in [11], resolving tuple dependencies, i.e., tracking which alternatives of different tuples belong to the same world, often requires the computation of lineage closure. Additionally, query operations on ULDBs can produce inconsistencies and anomalies, such as erroneous dependencies and inexistent tuples. In contrast, WSDs avoid both pitfalls.

In [29] probabilistic databases are modeled using graphical models. Each tuple has an associated random variable, specifying the existence of the tuple in the database, and correlations between tuples are given by links between the corresponding nodes in the graphical model. Querying the graphical model is done by introducing new factors, and computing confidence of tuples is reduced to probabilistic inference in graphical models. Note that world-set decompositions correspond to flat graphical models, where the conditional independence between variables is made explicit. Indeed, WSDs are based on the idea of independence between variables (attribute values), which is a special kind of conditional independence. In some cases, graphical models can be more succinct than WSDs. However, evaluating relational algebra queries on top of graphical models tends to produce flat models with high treewidth, which makes confidence computation hard on graphical models.

The model used by the Orion system [30] can be seen as an extension of the world-set-decomposition model for continuous distributions. There, correlated attributes are grouped together and represented by a single joint distribution. Similarly, in a WSD each component represents the joint (discrete) distribution of a set of correlated attributes.

In [9] we provide complexity results for different decision problems on WSDs, such as query possibility and certainty, and present a polynomial algorithm for relational decomposition. Finally, the more recent work [5] builds upon WSDs to create a representation system where correlations can be represented in a more intensional but still relational way, which ensures more compact representation and efficient query processing. [17] shows how to manage interval probabilities, either because the exact probabilities are not known, or because they were introduced by evaluating selections on top of approximated confidence values.

2 Preliminaries

We use the named perspective of the relational model with the operations selection σ, projection π, product $\times$, union $\cup$, difference $-$, and attribute renaming δ (cf. e.g. [2]). A *relational schema* is a tuple $\Sigma = (R_1[U_1], \ldots, R_k[U_k])$, where each R_i is a relation name and U_i is a set of attribute names. Let $\mathbf{D}$ be a set of domain elements. A *relation* over schema $R[A_1, \ldots, A_k]$ is a set of tuples $(A_1 : a_1, \ldots, A_k : a_k)$ where $a_1, \ldots, a_k \in \mathbf{D}$. A *relational database* $\mathcal{A}$ over schema Σ is a set of relations $R^{\mathcal{A}}$, one for each relation schema $R[U]$ from Σ. Sometimes, when no confusion of database may occur, we will use R rather than $R^{\mathcal{A}}$ to denote one particular relation over schema $R[U]$. By the size of a relation R, denoted $|R|$, we refer to the number of tuples in R. For a relation R over schema $R[U]$, let $sch(R)$ denote the set U of its attributes and let $ar(R)$ denote the arity of R.

A *product m-decomposition* of a relation R is a set of non-nullary relations $\{C_1, \ldots, C_m\}$ such that $C_1 \times \cdots \times C_m = R$. The relations $C_1, \ldots, C_m$ are called *components*. A product m-decomposition of R is *maximal* if there is no product n-decomposition of R with $n > m$.

A set of *possible worlds* (or *world-set*) over schema Σ is a set of databases over schema Σ. Let $\mathbf{W}$ be a set of structures, rep be a function that maps from $\mathbf{W}$ to world-sets of the same schema. Then $(\mathbf{W}, rep)$ is a *strong representation system* for a query language if, for each query Q of that language and each $\mathcal{W} \in \mathbf{W}$ such that Q is applicable to the worlds in $rep(\mathcal{W})$, there is a structure $\mathcal{W}' \in \mathbf{W}$ such that $rep(\mathcal{W}') = \{Q(\mathcal{A}) \mid \mathcal{A} \in rep(\mathcal{W})\}$. Obviously,

Lemma 1 *If rep is a function from a set of structures $\mathbf{W}$ to the set of all finite world-sets, then $(\mathbf{W}, rep)$ is a strong representation system for any relational query language.*

3 World-Set Decompositions

In order to use classical database techniques for storing and querying incomplete data, we develop a scheme for representing a world-set $\mathbf{A}$ by a single relational database.

Let $\mathbf{A}$ be a finite world-set over schema $\Sigma = (R_1, \ldots, R_k)$. For each R in Σ, let $|R|_{\max} = \max\{|R^{\mathcal{A}}| : \mathcal{A} \in \mathbf{A}\}$ denote the maximum cardinality of relation R in any world of $\mathbf{A}$. Given a world $\mathcal{A}$ with $R^{\mathcal{A}} = \{t_1, \ldots, t_{|R^{\mathcal{A}}|}\}$, let $inline(R^{\mathcal{A}})$ be the tuple obtained as the concatenation (denoted $\circ$) of the tuples of $R^{\mathcal{A}}$ in an arbitrary order padded with a special tuple $t_\perp = \underbrace{(\perp, \ldots, \perp)}_{ar(R)}$ up to arity $|R|_{\max}$:

$$inline(R^{\mathcal{A}}) := t_1 \circ \cdots \circ t_{|R^{\mathcal{A}}|} \circ (\underbrace{t_\perp, \ldots\ldots\ldots\ldots, t_\perp}_{|R|_{\max} - |R^{\mathcal{A}}|})$$

Then tuple

$$inline(\mathcal{A}) := inline(R_1^{\mathcal{A}}) \circ \cdots \circ inline(R_k^{\mathcal{A}})$$

encodes all the information in world $\mathcal{A}$. The "dummy" tuples with $\perp$-values are only used to ensure that the relation R has the same number of tuples in all worlds in $\mathbf{A}$. We extend this interpretation and generally define as $t_\perp$ any tuple that has at least one symbol $\perp$, i.e., $(A_1 : a_1, ..., A_n : a_n)$, where at least one a_i is $\perp$, is a $t_\perp$ tuple. This allows for several different inlinings of the same world-set.

By a *world-set relation* of a world-set $\mathbf{A}$, we denote the relation $\{inline(\mathcal{A}) \mid \mathcal{A} \in \mathbf{A}\}$. This world-set relation has schema $\{R.t_i.A_j \mid R[U] \in \Sigma, 1 \leq i \leq |R|_{\max}, A_j \in U\}$. Note that in defining this schema we use t_i to denote the position (or identifier) of tuple t_i in $inline(R^{\mathcal{A}})$ and not its value.

Given the above definition that turned every world in a tuple of a world-set relation, computing the initial world-set is an easy exercise. In order to have every world-set relation define a world-set, let a tuple extracted from some $t_{R^{\mathcal{A}}} = inline(R^{\mathcal{A}})$ be in $R^{\mathcal{A}}$ iff it does not contain any occurrence of the special symbol $\perp$. That is, we map $t_{R^{\mathcal{A}}} = (a_1, \ldots, a_{ar(R)\cdot|R|_{\max}})$ to $R^{\mathcal{A}}$ as

$$inline^{-1}\ (t_{R^{\mathcal{A}}}) := \{(a_{ar(R)\cdot k+1}, \ldots, a_{ar(R)\cdot(k+1)}) \mid 0 \leq k < |R|_{\max}, a_{ar(R)\cdot k+1} \neq \perp, \ldots, a_{ar(R)\cdot(k+1)} \neq \perp\}.$$

If $t_{\mathcal{A}} = t_{R_1^{\mathcal{A}}} \circ \ldots \circ \ldots t_{R_k^{\mathcal{A}}}$ is the inlining for world $\mathcal{A}$, we can restore $\mathcal{A}$ in the following way:

$$inline^{-1}(t_{\mathcal{A}}) = (inline^{-1}(t_{R_1^{\mathcal{A}}}), \ldots, inline^{-1}(t_{R_k^{\mathcal{A}}}))$$

Observe that although world-set relations are not unique as we have left open the ordering in which the tuples of a given world are concatenated, all world-set relations of a world-set $\mathbf{A}$ are equally good for encoding the world-set because they can be mapped invariantly back to $\mathbf{A}$. However different orderings of the tuples might have implications on the compactness of the decomposition. Note that for each world-set relation a maximal decomposition exists, is unique, and can be efficiently computed [9].

Definition 1 Let $\mathbf{A}$ be a world-set and W a world-set relation representing $\mathbf{A}$. Then a *world-set m-decomposition (m-WSD)* of $\mathbf{A}$ is a product m-decomposition of W.

We will refer to each of the m elements of a world-set m-decomposition as *components*, and to the component tuples as *local worlds*. Somewhat simplified examples of world-set relations and WSDs over a single relation R (thus "R" was omitted from the attribute names of the world-set relations) were given in Section 1. Further examples can be found in Section 4. It should be emphasized that with WSDs we can also represent multiple relational schemata and even arbitrary correlations of fields across relations (by having components with fields from different relations).

Definition 2 Let $\mathcal{W} = \{C_1, \ldots, C_m\}$ be an m-WSD. Then the function rep that maps $\mathcal{W}$ to a set of possible worlds is defined as

$$rep(\mathcal{W}) = \bigcup\{inline^{-1}(t) \mid t \in C_1 \times \ldots \times C_m\}$$

It immediately follows from our definitions that

Proposition 1 *Any finite set of possible worlds can be represented as a world-set relation and as a* 1*-WSD.*

Corollary 1 (Lemma 1) *WSDs are a strong representation system for any relational query language.*

As pointed out in Section 1, this is not true for or-set relations. For the relatively small class of world-sets that can be represented as or-set relations, the size of our representation system is linear in the size of the or-set relations. As seen in the examples, our representation is *much more space-efficient than world-set relations*.

Modeling Probabilistic Information. We can quantify the uncertainty of the data by means of probabilities using a natural extension of WSDs. A *probabilistic world-set m-decomposition* (probabilistic m-WSD) is an m-WSD $\{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$, where each component relation $\mathcal{C}$ has a special attribute P in its schema defining the probability for the local worlds, that is, for each combination of values defined by the component. For a component tuple t_C we have $t_C.P \in (0, 1]$. To ensure valid probability distribution, we require that the probabilities in a component sum up to one, i.e. $\sum_{t_C \in \mathcal{C}} t_C.P = 1$.

Probabilistic WSDs[2] generalize the probabilistic tuple-independent model of [15], as we show next.

[2] Like most recent work we assume that probabilities are given as input, for example by an expert, or learned. Follow-up work of the authors discusses how probabilities can be introduced using queries [8].

Example 5 Figure 6 (a) is an example taken from [15]. It shows a tuple-independent probabilistic database with two relations S and T. Each tuple is assigned a confidence value, which represents the probability of the tuple being in the database, and the tuples are assumed independent. A possible world is obtained by choosing a subset of the tuples in the tuple-independent probabilistic database, and its probability is computed by multiplying the probabilities for selecting a tuple or not, depending on whether that tuple is in the world. The set of possible worlds for D is given in Figure 6 (b). For example, the probability of the world D_3 can be computed as $(1 - 0.2) \cdot 0.5 \cdot 0.6 = 0.06$. □

S	A	B	P
s_1	m	1	0.8
s_2	n	1	0.5

T	C	D	P
t_1	1	p	0.6

(a)

world	P
$D_1 = \{s_1, s_2, t_1\}$	0.24
$D_2 = \{s_1, t_1\}$	0.24
$D_3 = \{s_2, t_1\}$	0.06
$D_4 = \{t_1\}$	0.06
$D_5 = \{s_1, s_2\}$	0.16
$D_6 = \{s_1\}$	0.16
$D_7 = \{s_2\}$	0.04
$D_8 = \emptyset$	0.04

(b)

Fig. 6 A tuple-independent probabilistic database for relations S and T (a), and the represented set of possible worlds (b).

We obtain a probabilistic WSD in the following way. Each tuple t with confidence c in a tuple-independent probabilistic database induces a WSD component representing two local worlds: the local world with tuple t and probability c, and the empty world with probability $1 - c$. Figure 7 gives the WSD encoding of the tuple-independent probabilistic database of Figure 6. Of course, in probabilistic WSDs we can assign probabilities not only to individual tuples, but also to combinations of values for fields of different tuples or relations.

C_1	s_1.A	s_1.B	P
1	m	1	0.8
2	⊥	⊥	0.2

×

C_2	s_2.A	s_2.B	P
1	n	1	0.5
2	⊥	⊥	0.5

×

C_3	t_1.C	t_1.D	P
1	1	p	0.6
2	⊥	⊥	0.4

Fig. 7 WSD equivalent to the probabilistic database in Figure 6 (a).

Adding Template Relations. We now present our refinement of WSDs with so-called *template relations*. A template stores information that is the same in all possible worlds and contains special values '?' $\notin$ **D** in fields at which different worlds disagree.

Let $\Sigma = (R_1, \ldots, R_k)$ be a schema and **A** a finite set of possible worlds over Σ. Then, the database $(R_1^0, \ldots, R_k^0, \{C_1, \ldots, C_m\})$ is called an *m-WSD with template relations (m-WSDT)* of **A** iff there is a WSD $\{C_1, \ldots, C_m, D_1, \ldots, D_n\}$ of **A** such that $|D_i| = 1$ for all i and if relation D_i has attribute $R_j.t.A$ and value v in its unique $R_j.t.A$-field, then the template relation R_j^0 has a tuple with identifier t whose A-field has value v.

Of course WSDTs again can represent any finite world-set and are thus a strong representation system for any relational query language. Example 4 shows a WSDT for the running example of the introduction.

Uniform World-Set Decompositions. In practice, database systems often do not support relations of arbitrary arity (e.g., WSD components). For that reason we introduce next a modified representation of WSDs called *uniform WSDs*. Instead of having a variable number of component relations, possibly with different arities, we store all values in a single relation C that has a fixed schema. We use the fixed schema consisting of the three relation schemata

$$C[FID, LWID, VAL], F[FID, CID], W[CID, LWID, PR]$$

where FID is a triple[3] $(Rel, TupleID, Attr)$ denoting the $Attr$-field of tuple $TupleID$ in database relation Rel.

In this representation we need a restricted flavor of world-ids called *local world-ids* (LWIDs). The local world-ids refer only to the possible worlds within one component. LWIDs avoid the drawbacks of "global" world IDs for the individual worlds. This is important, since the size of global world IDs can exceed the size of the decomposition itself, thus making it difficult or even impossible to represent the world-sets in a space-efficient way. If any world-set over a given schema and a fixed active domain is permitted, one can verify that global world-ids cannot be smaller than the largest possible world over the schema and the active domain.

Given a WSD $\{C_1, \ldots, C_m\}$ with schemata $C_i[U_i]$, we populate the corresponding UWSD as follows.

- $((R, t, A), s, v) \in C$ iff, for some (unique) i, $R.t.A \in U_i$ and the field of column $R.t.A$ in the tuple with id s of C_i has value v.
- $F := \{((R, t, A), C_i) \mid 1 \leq i \leq m,\ R.t.A \in U_i\}$,
- $(C_i, s, p) \in W$ iff there is a tuple with identifier s in C_i, whose probability is p.

Intuitively, the relation C stores each value from a component together with its corresponding field identifier and the identifier of the component-tuple in the initial WSD (column $LWID$ of C). The relation F contains the mapping between tuple fields and component identifiers, and W keeps track of the worlds present for a given component.

In general, the VAL column in the component relation C must store values for fields of different type. One possibility is to store all values as strings and use casts when required. Alternatively, one could have one component relation for each data type. In both cases the schema remains fixed.

[3] FID really takes three columns, but for readability we keep them together under a common name in this section.

R^0	S	N	M
t_1	?	Smith	?
t_2	?	Brown	3

C	FID	LWID	VAL
	(R, t_1, S)	1	185
	(R, t_2, S)	1	186
	(R, t_1, S)	2	785
	(R, t_2, S)	2	185
	(R, t_1, S)	3	785
	(R, t_2, S)	3	186
	(R, t_1, M)	1	1
	(R, t_1, M)	2	2

F	FID	CID
	(R, t_1, S)	C_1
	(R, t_1, M)	C_2
	(R, t_2, S)	C_1

W	CID	LWID	PR
	C_1	1	0.2
	C_1	2	0.4
	C_1	3	0.4
	C_2	1	0.7
	C_2	2	0.3

Fig. 8 A UWSDT corresponding to the WSDT of Figure 5.

Finally, we add template relations to UWSDs in complete analogy with WSDTs, thus obtaining the UWSDTs.

Example 6 We modify the world-set represented in Figure 4 such that the marital status in t_2 can only have the value 3. Figure 8 is then the uniform version of the WSDT of Figure 4. Here R^0 contains the values that are the same in all worlds. For each field that can have more than one possible value, R^0 contains a special placeholder, denoted by '?'. The possible values for the placeholders are defined in the component table C. In practice, we can expect that the majority of the data fields can take only one value across all worlds, and can be stored in the template relation. □

Proposition 2 *Any finite set of possible worlds can be represented as a* 1*-UWSD and as a* 1*-UWSDT.*

It follows again that UWSD(T)s are a strong representation system for *any relational query language*.

Remark 1 In theory and as presented in this section, WSDs can be obtained by decomposing the world-set relation, and an efficient algorithm for achieving this is described in [9]. However, we consider this infeasible in practice as the number of possible worlds (which determines the size of the world-set relation) can be exponential. Instead, we assume that in practice WSDs will be constructed by starting off with a "dirty" relation describing the possible values, and then repairing the database to satisfy given constraints. The decomposition algorithm will be then used to optimize the representation. □

4 Queries on World-set Decompositions

In this section we study the query evaluation problem for WSDs. As pointed out before, UWSDTs are a better representation system than WSDs; nevertheless WSDs are simpler to explain and visualize and the main issues regarding query evaluation are the same for both systems.

The goal of this section is to provide, for each relational algebra query Q, a query $\hat{Q}$ such that for a WSD $\mathcal{W}$,

$$rep(\hat{Q}(\mathcal{W})) = \{Q(\mathcal{A}) \mid \mathcal{A} \in rep(\mathcal{W})\}.$$

Of course we want to evaluate queries directly on WSDs using $\hat{Q}$ rather than process the individual worlds using the original query Q.

The algorithms for processing relational algebra queries presented next are orthogonal to whether or not the WSD stores probabilities. According to our semantics, a query is conceptually evaluated in each world and extends the world with the result of the query in that world. In Section 6, we also consider queries that look across worlds and compute the *confidence* of tuples in query results.

When compared to traditional query evaluation, the evaluation of relational queries on WSDs poses new challenges. First, since decompositions in general consist of several components, a query $\hat{Q}$ that maps from one WSD to another must be expressed as a set of queries, each of which defines a different component of the output WSD. Second, as certain query operations may cause new dependencies between components to develop, some components may have to be merged (i.e., part of the decomposition undone using the product operation $\times$). Third, the answer to a (sub)query Q_0 must be represented within the same decomposition as the input relations to correctly represent the correlations between the input and the result of the subquery; indeed, we want to compute a decomposition of world set $\{(\mathcal{A}, Q_0(\mathcal{A})) \mid \mathcal{A} \in rep(\mathcal{W})\}$ in order to be able to resort to the input relations as well as the result of Q_0 within each world. Consider for example a query $\sigma_{A=1}(R) \cup \sigma_{B=2}(R)$. If we first compute $\sigma_{A=1}(R)$, we must store it in the same WSD as the input relation, otherwise the connection between worlds of R and the selection $\sigma_{A=1}$ is lost and we cannot compute $\sigma_{A=1}(R) \cup \sigma_{B=2}(R)$ correctly.

We say that a relation P is a copy of another relation R in a WSD if R and P have the same tuples in every world represented by the WSD. For a component C, an attribute $R.t.A_i$ of C and a new attribute $P.t.B$, the function ext extends C by a new column $P.t.B$ that is a copy of $R.t.A_i$:

$$\mathsf{ext}(C, A_i, B) := \{(A_1 : a_1, \ldots, A_n : a_n, B : a_i) \mid (A_1 : a_1, \ldots, A_n : a_n) \in C\}$$

Then $\mathsf{copy}(R, P)$ executes $C := \mathsf{ext}(C, R.t_i.A, P.t_i.A)$ for each component C and each $R.t_i.A \in sch(C)$.

The implementation of some operations requires the composition of components. Let C_1 and C_2 be two components with schemata $(A_1, \ldots, A_k, P)$, and $(B_1, \ldots, B_l, P)$, respectively. Then the composition of C_1

algorithm select[$A\theta c$] // compute $P := \sigma_{A\theta c}R$
begin
  copy(R, P);
  for each $1 \le i \le |P|_{max}$ **do begin**
    let C be the component of $P.t_i.A$;
    for each $t_C \in C$ **do**
      if not $(t_C.(P.t_i.A)\,\theta\,c)$ **then**
        $t_C.(P.t_i.A) := \bot$
    propagate-$\bot(C)$;
  end
end

algorithm select[$A\theta B$] // compute $P := \sigma_{A\theta B}R$
begin
  copy(R, P);
  for each $1 \le i \le |P|_{max}$ **do begin**
    let C be the component of $P.t_i.A$;
    let C' be the component of $P.t_i.B$;
    if $(C \neq C')$ **then**
      replace components C, C' by $C :=$ compose(C, C');
    for each $t_C \in C$ **do**
      if not $(t_C.(P.t_i.A)\,\theta\,t_C.(P.t_i.B))$ **then**
        $t_C.(P.t_i.A) := \bot$
    propagate-$\bot(C)$;
  end
end

algorithm product // compute $T := R \times S$
begin
  for each $1 \le j \le |S|_{max}$ and $R.t_i.A \in sch(R)$ **do begin**
    let C be the component of $R.t_i.A$;
    $C :=$ ext$(C, R.t_i.A, T.t_{ij}.A)$;
  end;
  for each $1 \le i \le |R|_{max}$ and $S.t_j.A \in sch(S)$ **do begin**
    let C' be the component of $S.t_j.A$;
    $C' :=$ ext$(C', S.t_j.A, T.t_{ij}.A)$;
  end
end

algorithm union // compute $T := R \cup S$
begin
  for each $1 \le i \le |R|_{max}$ and $A \in sch(R)$ **do begin**
    let C be the component of $R.t_i.A$;
    $C :=$ ext$(C, R.t_i.A, T.(R.t_i).A)$;
  end;
  for each $1 \le j \le |S|_{max}$ and $A \in sch(S)$ **do begin**
    let C' be the component of $S.t_j.A$;
    $C' :=$ ext$(C', S.t_j.A, T.(S.t_j).A)$;
  end
end

algorithm project[U] // compute $P := \pi_U(R)$
begin
  copy(R, P);
  for each $1 \le i \le |P|_{max}$**do**
    while no fixpoint is reached **do begin**
      let C be the component of $P.t_i.A$, where $A \in U$;
      let $C' \neq C$ be the component of $P.t_i.B$, where
        $B \notin U$ and $(\forall A' \in U : P.t_i.A' \notin sch(C'))$ and
        $(\exists t_{C'} \in C' : t_{C'}.B = \bot)$;
      replace components C, C' by $C :=$ compose(C, C');
      propagate-$\bot(C)$;
      project away $P.t_j.B$ from C where $B \notin U$ and $j \le i$;
    end
  for each $1 \le i \le |P|_{max}$ and $B \notin U$ **do begin**
    let C be the component of $P.t_i.B$;
    project away $P.t_i.B$ from C;
  end
end

algorithm rename // compute $\delta_{A \to A'}(R)$
begin
  for each $1 \le i \le |R|_{max}$ **do begin**
    let C be the component of $R.t_i.A$;
    $C := \delta_{R.t_i.A \to R.t_i.A'}(C)$;
  end;
end

algorithm difference // compute $P := R - S$
begin
  copy(R, P);
  for each $1 \le i \le |P|_{max}$**do**
    for each $1 \le j \le |S|_{max}$**do**
      let $C_1, \ldots, C_k$ be the components for the fields of $P.t_i$ and $S.t_j$;
      replace $C_1, \ldots, C_k$ by $C :=$ compose$(C_1, \ldots, C_k)$;
      for each $t_C \in C$ **do begin**
        if $t_C.(P.t_i.A) = t_C.(S.t_j.A)$ for all $A \in sch(R)$ **then**
          $t_C.(P.t_i.A) := \bot$;
      end
    end
end

Fig. 9 Evaluating relational algebra operations on WSDs.

and C_2 is defined as:

$$\begin{aligned}&\text{compose}(C_1, C_2) :=\\ &\quad \{(A_1 : a_1, \ldots, A_k : a_k, B_1 : b_1, \ldots, B_l : b_l,\\ &\qquad P : p_1 \cdot p_2) \mid\\ &\qquad (A_1 : a_1, \ldots, A_k : a_k, P : p_1) \in C_1,\\ &\qquad (B_1 : b_1, \ldots, B_l : b_l, P : p_2) \in C_2\}\end{aligned}$$

In the non-probabilistic case the composition of some components is simply their relational product. In the probabilistic case, the probability of a tuple in the resulting component is the product of the probabilities of the corresponding tuples from the input components.

Figure 9 presents implementations of the relational algebra operations selection (of the form $\sigma_{A\theta c}$ or $\sigma_{A\theta B}$, where

A and B are attributes, c is a constant, and θ is a comparison operation, $=$, $\neq$, $<$, $\leq$, $>$, or $\geq$), projection, relational product and union on WSDs. In each case, the input WSD is *extended* by the result of the operation.

Given a relational algebra query Q, let $\hat{Q}$ denote the query processor on WSDs we obtain by replacing each operation of Q by its corresponding operation on WSDs.

Theorem 1 (Correctness) *Let $\mathcal{W}$ be a WSD and let $\mathcal{W}'$ be the WSD obtained from $\hat{Q}(\mathcal{W})$ by dropping all relations but the result relation of $\hat{Q}$. Then,*

$$rep(\mathcal{W}') = \{Q(\mathcal{A}) \mid \mathcal{A} \in rep(\mathcal{W})\}.$$

Proof The proof of correctness of the translation is by induction on the structure of query Q.
Base case: Let $Q = R$. Then the result of the query is $\{R^{\mathcal{A}} \mid \mathcal{A} \in rep(\mathcal{W})\} = rep(\mathcal{W}')$.
Induction step: Let $Q = \sigma_{A\theta c}(Q')$ and suppose

$$rep(\mathcal{W}') \neq \{Q(\mathcal{A}) \mid \mathcal{A} \in rep(\mathcal{W})\}$$

Suppose first that $\mathcal{W} = \{C\}$, that is $\mathcal{W}$ is a 1-WSD. Let $t_{\mathcal{A}} \in C$ be a tuple in C that corresponds to a world $\mathcal{A}$. The translation $\hat{Q}$ replaces with $\perp$ the values for all tuples in $t_{\mathcal{A}}$ that do not satisfy the selection condition, and let $t'_{\mathcal{A}}$ be the result of this operation. Thus by definition $inline^{-1}(t'_{\mathcal{A}}) = Q(\mathcal{A})$ and $rep(\mathcal{W}) = \{Q(\mathcal{A}) \mid \mathcal{A} \in rep(\mathcal{W})\}$.

Consider now an m-WSD $\mathcal{W} = \{C_1, \ldots, C_m\}$ and let $t_{\mathcal{A}} = t_{C_1} \circ \ldots \circ t_{C_m}$ be the tuple for world $\mathcal{A}$, where $t_{C_i} \in C_i$. Let $t \in Q'$ and let $t.A$ be defined in component C_i. If $t_{C_i}.(t.A)\theta c$, $\hat{Q}$ leaves the values for t unchanged in t_{C_i}, otherwise $t_{C_i}.(t.A)$ is replaced by $\perp$. But then by our semantics $inline^{-1}(t_{\mathcal{A}})$ does not contain tuple t, as $t_{\mathcal{A}}.(t.A) = \perp$. Thus $inline^{-1}(t_{\mathcal{A}})$ contains exactly the tuples in $Q(\mathcal{A})$.

The correctness for the remaining operators follows along the same lines. □

Let us now have a closer look at the evaluation of relational algebra operations on WSDs. For this, we use as running example the set of eight worlds over the relation R of Figure 10 (a) and its maximal 7-WSD of Figure 10 (b). The second component (from the left) of the WSD spans over several tuples and attributes and each of the remaining six components refer to one tuple and one attribute. The first tuple of the second component of the WSD of Figure 10 contains the values for $R.t_1.B$, $R.t_1.C$, and $R.t_2.B$, i.e. some but not all of the attributes of the first and second tuple of $R^{\mathcal{A}}$, for all worlds $\mathcal{A}$. In our attempt to keep the WSDs readable, we consistently show in the following examples only the WSDs of the result relations.

Selection with condition $A\theta c$. In order to compute a selection $P := \sigma_{A\theta c}(R)$, we first compute a copy P of relation R and subsequently drop tuples of P that do not match the selection condition.

Dropping tuples is a fairly subtle operation, since tuples can spread over several components and a component can define values for more than one tuple.

Thus a selection must not delete tuples from component relations, but should mark fields as belonging to deleted tuples using the special value $\perp$. To evaluate $\sigma_{A\theta c}(R)$, our selection algorithm of Figure 9 checks for each tuple t_i in the relation P and t_C in component C with attribute $P.t_i.A$ whether $t_C.(P.t_i.A)\theta c$. In the negative case the tuple $P.t_i$ is marked as deleted in all worlds that take values from t_C. For that, $t_C.(P.t_i.A)$ is assigned value $\perp$, and all other attributes $P.t_i.A'$ of C referring to the same tuple t_i of P are assigned value $\perp$ in t_C, (cf. the algorithm propagate-$\perp$ of Figure 12). This assures that if we later project away the attribute A of P, we do not erroneously "reintroduce" tuple $P.t_i$ into worlds that take values from t_C.

```
algorithm propagate-⊥(C: component)
begin
  for each t_C ∈ C and P.t_i.A ∈ sch(C) do
    if t_C.(P.t_i.A) = ⊥ then
        for each A' such that P.t_i.A' ∈ sch(C) do
          t_C.(P.t_i.A') := ⊥;
end
```

Fig. 12 Propagating $\perp$-values.

Example 7 Figure 11 shows the answers to $\sigma_{C=7}(R)$ and $\sigma_{B=1}(R)$. Note that the resulting WSDs should contain both the query answer P and the original relation R, but due to space limitations we only show the representation of P. One can observe that for both results in Figure 11 we obtain worlds of different sizes. For example the worlds that take values from the first tuple of the second component relation in Figure 11 (a) do not have a tuple t_1, while the worlds that take values from the second tuple of that component relation contain t_1. □

Selection with condition $A\theta B$. The main added difficulty of selections with conditions $A\theta B$ as compared to selections with conditions $A\theta c$ is that it creates dependencies between two attributes of a tuple, which do not necessarily reside in the same component.

As the current decomposition may not capture exactly the combinations of values satisfying the join condition, components that have values for A and B of the same tuple are composed. After the composition phase, the selection algorithm follows the pattern of the selection with constant.

Example 8 Consider the query $\sigma_{A=B}(R)$, where R is represented by the 7-WSD of Figure 10. Figure 13 shows the query answer, which is a 4-WSD that represents five worlds,

A	B	C
1	1	0
4	3	0
6	6	7

A	B	C
2	1	0
4	3	0
6	6	7

A	B	C
1	1	0
5	3	0
6	6	7

A	B	C
2	1	0
5	3	0
6	6	7

A	B	C
1	2	7
4	4	0
6	6	7

A	B	C
2	2	7
4	4	0
6	6	7

A	B	C
1	2	7
5	4	0
6	6	7

A	B	C
2	2	7
5	4	0
6	6	7

(a) Set of eight worlds of the relation R.

R.t_1.A
1
2

$\times$

R.t_1.B	R.t_1.C	R.t_2.B
1	0	3
2	7	4

$\times$

R.t_2.A
4
5

$\times$

R.t_2.C
0

$\times$

R.t_3.A
6

$\times$

R.t_3.B
6

$\times$

R.t_3.C
7

(b) 7-WSD of the world-set of (a).

Fig. 10 World-set and its decomposition.

P.t_1.A
1
2

$\times$

P.t_1.B	P.t_1.C	P.t_2.B
$\perp$	$\perp$	3
2	7	4

$\times$

P.t_2.A
4
5

$\times$

P.t_2.C
$\perp$

$\times$

P.t_3.A
6

$\times$

P.t_3.B
6

$\times$

P.t_3.C
7

(a) $P := \sigma_{C=7}(R)$ applied to the WSD of Figure 10.

P.t_1.A
1
2

$\times$

P.t_1.B	P.t_1.C	P.t_2.B
1	0	$\perp$
$\perp$	$\perp$	$\perp$

$\times$

P.t_2.A
4
5

$\times$

P.t_2.C
0

$\times$

P.t_3.A
6

$\times$

P.t_3.B
$\perp$

$\times$

P.t_3.C
7

(b) $P := \sigma_{B=1}(R)$ applied to the WSD of Figure 10.

Fig. 11 Selections $P := \sigma_{C=7}(R)$ and $P := \sigma_{B=1}(R)$ with R from Figure 10.

P.t_1.A	P.t_1.B	P.t_1.C	P.t_2.A	P.t_2.B
1	1	0	$\perp$	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
$\perp$	$\perp$	$\perp$	4	4
2	2	7	4	4
2	2	7	$\perp$	$\perp$

$\times$

P.t_2.C
0

$\times$

P.t_3.A	P.t_3.B
6	6

$\times$

P.t_3.C
7

Fig. 13 $P = \sigma_{A=B}(R)$ with R from Figure 10.

R.t_1.A
1
2

$\times$

R.t_1.B	R.t_2.A
3	5
4	6

$\times$

R.t_2.B
7
8

$\times$

S.t_1.C
a
b

$\times$

S.t_1.D	S.t_2.C
c	e
d	f

$\times$

S.t_2.D
g
h

(a) WSD of two relations R and S.

t_{11}.A	t_{12}.A
1	1
2	2

$\times$

t_{11}.B	t_{12}.B	t_{21}.A	t_{22}.A
3	3	5	5
4	4	6	6

$\times$

t_{21}.B	t_{22}.B
7	7
8	8

$\times$

t_{11}.C	t_{21}.C
a	a
b	b

$\times$

t_{11}.D	t_{21}.D	t_{12}.C	t_{22}.C
c	c	e	e
d	d	f	f

$\times$

t_{12}.D	t_{22}.D
g	g
h	h

(b) WSD of their product $R \times S$.

Fig. 14 The product operation $R \times S$.

where one world has three tuples, three worlds have two tuples each, and one world has one tuple. □

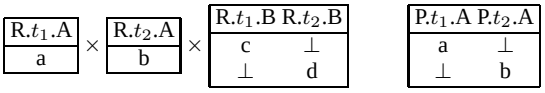
Product. The product $T := R \times S$ of two relations R and S, which have disjunct attribute sets and are represented by a WSD requires that the product relation T extends a component C with $|S|_{max}$ (respectively $|R|_{max}$) copies of each column of C with values of R (respectively S). Additionally, the ith (jth) copy is named $T.t_{ij}.A$ if the original has name $R.t_i.A$ or $S.t_j.A$.

Example 9 Figure 14 (b) shows the WSD for the product of relations R and S represented by the WSD of Figure 14 (a). To save space, the relations R and S have been removed from Figure 14 (b), and attribute names do not show the relation name "T". □

Projection. A projection $P = \pi_U(R)$ on an attribute set U of a relation R represented by the WSD $\mathcal{C}$ is translated into (1) the extension of $\mathcal{C}$ with the copy P of R, and (2) projections on the components of $\mathcal{C}$, where all component attributes that do not refer to attributes of P in U are discarded. Before removing attributes, however, we need to propagate $\perp$-values, as discussed in the following example.

Example 10 Consider the 3-WSD of Figure 15 (a) representing a set of two worlds for R, where one world contains only the tuple t_1 and the other contains only the tuple t_2. Let P' represent the first two components of R, which contain all values for the attribute A in both tuples. The relation P' is not the answer to $\pi_A(R)$, because it encodes one world with *both* tuples, and the information from the third component of R that only one tuple appears in each world is lost. To

compute the correct answer, we progressively (1) compose the components referring to the same tuple (in this case all three components), (2) propagate $\bot$-values within the same tuple, and (3) project away the irrelevant attributes. The correct answer P is given in Figure 15 (b). □



(a) WSD for R. (b) WSD for P.

Fig. 15 Projection $P := \pi_A(R)$.

The algorithm for projection is given in Figure 9. For each tuple t_i, attribute A in the projection list, and attribute B not in the projection list, the algorithm first propagates the $\bot$-values of $P.t_i.B$ of component C' to $P.t_i.A$ of component C. If C and C' are the same, the propagation is done locally within the component. Otherwise, C and C' are merged before the propagation. Note that the propagation is only needed if some tuples of C' have at $\bot$-value for t_i.B. This procedure is performed until no other components C and C' exist that satisfy the above criteria. After the propagation phase, the attributes not in the projection list are dropped from all remaining components.

Union. The algorithm for computing the union $T := R \cup S$ of two relations R and S works similarly to that for the product. Each component C containing values of R or S is extended such that in each world of C all values of R and S become also values of T.

Renaming. The operation $\delta_{A \to A'}(R)$ renames attribute A of relation R to A' by renaming all attributes $R.t.A$ in a component C to $R.t.A'$.

Difference. To compute the difference operation $P := R - S$ we scan and compose components of the two relations R and S. For the worlds where a tuple t from R matches some tuple from S, we place $\bot$-values to denote that t is not in these worlds of P; otherwise t becomes a tuple of P. The difference is by far the least efficient operation to implement not only on WSDs, where it can lead to the composition of all components, but also on the other succinct representation systems. However, if we want to close the possible world semantics and compute the confidence of tuples in the answer to a difference query, we can often avoid computing the representation of the result. [22] makes the observation that computing the confidence of tuples in the answer to a difference can be done by computing confidence of the negated positive query which in turn can be efficiently approximated. This is a special case of computing the conditional probability $P(\phi \mid \psi)$ of a positive query ϕ given a universal constraint ψ. The formula ψ can express for example a functional dependency, or another type of equality-generating dependency. The answer to the original query can be obtained by

$$P(\phi \mid \psi) = P(\phi)P(\phi \wedge \psi) = P(\phi)\left(P(\phi) - P(\phi \wedge \neg\psi)\right)$$

where $\neg\psi$ is existential.

Discussion. The operators selection, product and union can be implemented in polynomial time on WSDs. The implementation of the join selection, projection and difference can require the composition of components and can potentially lead to an exponential blow-up in the representation. Composing components in the projection operation can be avoided by introducing an additional "exists" column to replace columns with $\bot$-values that are projected away. With this addition, the projection can also be implemented in polynomial time. As for join and difference, the exponential blow-up can be avoided by encoding correlations in a more intensional way than the one offered by WSDs. This is the case of U-relations [5], for instance, which generalize WSDs.

Remark 2 The evaluation of relational algebra queries does not depend on the probabilities of the worlds, since it is conceptually performed in each possible world. Evaluating projection and join selection modifies the WSD by composing components; in that case we recompute the probabilities of the tuples in the new component. With the exception of those two operators, all other (positive) relational algebra operators do not need access to the probabilities stored with the data. The confidence computation operator presented in Section 6 makes use of the probability information. □

5 Efficient Query Evaluation on UWSDTs

The algorithms for computing the relational operations on WSDs presented in Section 4 can be easily adapted to UWSDTs. To do this, we follow closely the mapping of WSDs, represented as sets of components $\mathcal{C}$, to equivalent UWSDTs, represented by a triple (F,C,W) and at least one template relation R^0:

- Consider a component K of WSD $\mathcal{C}$ having an attribute $R.t.A$ with a value v. In the equivalent UWSDT, this value can be stored in the template relation R^0 if v is the only value of $R.t.A$, or in the component C otherwise. In the latter case, the template R^0 contains the placeholder $R.t.A$ in the tuple t. In addition, in the mapping relation F there is an entry with the placeholder $R.t.A$ and a component identifier c, and C contains a tuple formed by $R.t.A$, the value v and a world identifier w.
- Worlds of different sizes are represented in WSDs by allowing $\bot$ values in components, and in UWSDTs by allowing for a same placeholder different amount of values in different worlds.

Any relational query is rewritten in our framework to a sequence of SQL queries, except for the projection and selection with join conditions, where the fixpoint computations are encoded as recursive PL/SQL programs. In all cases, the size of the rewriting is linear in the size of the input query. For the operators that require pure SQL only this essentially means that the complexity of querying is preserved and remains polynomial. Figure 16 shows the implementation of the selection with constant on UWSDTs.

algorithm select[$A\theta c$] // compute $P := \sigma_{A\theta c}R$
begin
1. $P^0 := \sigma_{A\theta c \vee A=?}R^0$;
2. $F := F \cup \{(P.t.B, k) \mid (R.t.B, k) \in F, t \in P^0\}$;
3. $C := C \cup \{(P.t.B, w, v) \mid (R.t.B, w, v) \in C, t \in P^0,$
$(B = A \Rightarrow v\theta c)\}$;
// Remove incomplete world tuples
4. $C := C - \{(P.t.X, w, v) \in C \mid (P.t.X, k), (P.t.Y, k) \in F,$
$t \in P^0, X \neq Y, \nexists v' : (P.t.Y, w, v') \in C\}$;
5. $F := F - \{(P.t.B, k) \mid (P.t.B, k) \in F,$
$\nexists w, v : (P.t.B, w, v) \in C\}$;
6. $P^0 := P^0 - \{t \mid t \in P^0, \nexists B, a : (P.t.B, a) \in F\}$;
end

Fig. 16 Evaluating $P := \sigma_{A\theta c}(R)$ on UWSDTs.

In contrast to some algorithms of Figure 9, for UWSDTs we do not create a copy P of R at the beginning, but rather compute directly P from R using standard relational algebra operators. The template P^0 is initially the set of tuples of R^0 that satisfy the selection condition, or have a placeholder '?' for the attribute A (line 1). We extend the mapping relation F with the placeholders of P^0 (line 2), and the component relation C with the values of these placeholders, where the values of placeholders $P.t.A$ for the attribute A must satisfy the selection condition (line 3). If a placeholder $P.t.A$ has no value satisfying the selection condition, then t is removed from P^0 (line 6) and all placeholders of t are removed from F (line 5) together with their values from C (line 4).

Many of the standard query optimization techniques are also applicable in our context. For our experiments reported in Section 9, we performed the following optimizations on the sequences of SQL statements obtained as rewritings. For the evaluation of a query involving join, we merge the product and the selections with join conditions and distribute projections and selections to the operands. When evaluating a query involving several selections and projections on the same relation, we again merge these operators and perform the steps of the algorithm of Figure 16 only once. We further tuned the query evaluation by employing indices and materializing often used temporary results.

6 Confidence Computation in Probabilistic WSDs

Section 4 discusses query evaluation algorithms for relational algebra on top of WSDs. Since we consider queries that are semantically evaluated within each world, these algorithms do not need to explicitly take into account probability distributions over the possible worlds.

In this section, we also consider queries that look across worlds and compute confidence of tuples. The *confidence* of a tuple t in the result of a query Q is defined as the sum of the probabilities of the worlds that contain t in the answer to Q. Clearly, iterating over all possible worlds is infeasible. We therefore adopt an approach where we only iterate over the local worlds of the relevant components.

// compute $c := conf(t)$
algorithm conf (tuple t over schema $(A_1, \ldots, A_m)$)
begin
let $t_1, \ldots, t_n$ be the ids of tuples defined by input WSD $\mathcal{W}$;
// Keep only columns and rows of components of $\mathcal{W}$
// that define possible fields of t
for each $C_i \in \mathcal{W}$ **do** $C_i' := \pi_{\lambda_i}(\sigma_{\phi_i}(C_i))$ where
$\lambda_i := sch(C_i) \cap \{t_l.A_j \mid 1 \leq l \leq n, 1 \leq j \leq m\}$ and
$\phi_i := \bigvee_{t_l.A_j \in \lambda_i} (t_l.A_j = t.A_j)$;
compute equivalent tuple-level WSD $\mathcal{W}'$ of the above set of C_i';
(i.e., compose components defining fields of the same tuple)
$c := 0$; // initially, confidence of t is 0
for each $C \in \mathcal{W}'$ **do**
begin
$conf_C := 0$; // probability that C defines tuples that equal t
for each $t_C \in C$ **do**
if $t = (t_C.(t_i.A_1), \ldots, t_C.(t_i.A_m))$ for some t_i
then $conf_C := conf_C + t_C.P$;
// matches in C are independent from those in other components
$c := 1 - (1 - c) \cdot (1 - conf_C)$;
end
end

Fig. 17 Computing confidence of possible tuples.

Figure 17 gives our confidence computation algorithm for a tuple t over schema $(A_1, \ldots, A_m)$. It first computes a pruned version of the input WSD, where we keep for each component only columns that define fields for attributes $t_l.A_1, \ldots, t_l.A_m$ of any tuple id t_l, and only rows that define fields, whose values equal the corresponding ones in t.

Next, a tuple-level representation of the pruned WSD is computed. This representation enforces that all fields of any tuple encoded by the WSD are defined in the same component. Confidence computation can be performed efficiently on tuple-level WSDs. This tuple-level normalization can however lead to a exponential blowup in the representation size. This is necessary, since there may be exponentially many possible tuples encoded by a WSD. Moreover, special cases of confidence computation, such as deciding whether a tuple

is certain, i.e., it occurs in all worlds represented by a WSD, are known to be NP-hard [9].

Confidence computation on tuple-level WSDs is based on the observation that worlds containing t are extensions of local worlds from a component C where some tuples equal t. Since the local worlds of a component define non-overlapping sets of worlds, to compute the probability that a component C defines tuples that equal t, we only need to sum up the probabilities of the local worlds of C that define t. Furthermore, since any two components of a WSD are independent of each other, the events stating that a given component defines tuples that equal t are pairwise independent.

We next consider the operator possible that computes the tuples appearing in at least one world of the world-set. Formally, for a relation name R and a world-set $\mathbf{A}$, the operator possible is defined as:

$$\mathsf{possible}(R)(\mathbf{A}) := \{t \mid \mathcal{A} \in \mathbf{A}, t \in R^{\mathcal{A}}\}$$

```
// compute P := possible(R)
algorithm possible (relation R over schema (A_1, ..., A_m))
begin
  let t_1, ..., t_n be the ids of tuples in R defined by input WSD W;
  // Keep only columns of components of the input WSD W
  // that define possible tuples in R
  for each C_i ∈ W do C'_i := π_{λ_i}(C_i) where
    λ_i := sch(C_i) ∩ {R.t.A | t ∈ {t_1, ..., t_n}, A ∈ {A_1, ..., A_m}}
  compute equivalent tuple-level WSD W' of the above set of C'_i;
  (i.e., compose components defining fields of the same tuple)
  P := ∅; // initially, no tuple is possible
  for each tuple id t ∈ {t_1, ..., t_n} do
    for each C ∈ W' do
      add π_{R.t.A_1,...,R.t.A_m}(σ_{⋀_{1≤j≤m} R.t.A_j ≠ ⊥}(C)) to P;
end
```

Fig. 18 Computing possible tuples.

Figure 18 gives an algorithm for computing the set of possible tuples of a relation R in the non-probabilistic case. The algorithm first discards all columns of components in the input WSD that do not define possible fields for tuples of R. It then computes an equivalent tuple-level WSD representation of the set of components of the previous step. As for confidence computation, this tuple-level normalization can lead to an exponential blowup. Also here, this is unavoidable, since a WSD can represent exponentially many possible tuples (similar to or-sets). In case the input WSD is already tuple-level, it then encodes polynomially many possible tuples and our algorithm would only need polynomially many computation steps.

In the probabilistic case, the operator possible can be extended to also compute the confidence of the possible tuples, see Figure 19. Confidences of tuples in query results can

```
// compute P := possible^p(R)
algorithm possible^p
begin
  P := ∅;
  for each distinct t in possible(R) do
    add (t, conf(t)) to P;
end
```

Fig. 19 Computing possible tuples together with their confidences.

then be computed in two steps: First computing the query result, and then computing the possible tuples and their confidences.

Example 11 Consider the probabilistic WSD of Figure 4, query $Q = \pi_S(R)$, and tuple $t = (185)$. Let C_1 denote the first component. This component represents the answer to the projection query. There are two tuple ids whose values match the given tuple t, and they are already defined in the same component C_1. To compute the confidence of t we therefore need to sum up the probabilities of the first and second local world, obtaining $0.2+0.4 = 0.6$. The following table contains the possible tuples in the answer to Q together with their confidences:

Q	S	conf
	185	0.6
	186	0.6
	785	0.8

□

7 Normalizing probabilistic WSDs

The normalization of a WSD is the process of finding an equivalent probabilistic WSD that takes the least space among all its equivalents. Examples of not normalized WSDs are non-maximal WSDs (with respect to product decomposition) or WSDs defining invalid tuples (i.e., tuples that do not appear in any world). Note that removing invalid tuples and maximizing world-set decompositions can be performed in polynomial time [9].

Figure 20 gives three algorithms that address these normalization problems. The second algorithm decomposes a component into a set of components whose product is equal to the original component. A polynomial-time algorithm for finding the prime factorization of a relation, i.e. for maximally decomposing a relation is presented in [9]. The third algorithm scans for identical tuples in a component and compresses them into one by summing up their probabilities.

Example 12 The WSD of Figure 11 (a) has only $\perp$-values for $P.t_2.C$. This means that the tuple t_2 of P is absent (or invalid) in all worlds and can be removed. The equivalent WSD of Figure 21 shows the result of this operation. Similar

```
algorithm remove_invalid_tuples
begin
  for each 1 ≤ i ≤ |P|_max and A ∈ sch(P) do begin
    let C be the component of P.t_i.A;
    if π_{P.t_i.A} = {⊥} then
      for each B ∈ sch(P) do begin
        let C' be the component of P.t_i.B;
        project away P.t_i.B from C';
      end
  end
end

algorithm decompose
begin
  while no fixpoint is reached do begin
    let C be a component such that
      C = C_1 × ... × C_n;
    replace C by C_1, ..., C_n;
  end
end

algorithm compress
begin
  while no fixpoint is reached do begin
    let C be a component, w_1, w_2 ∈ C such that
      w_1.A = w_2.A for all A ∈ sch(C), A ≠ P;
    let w be a tuple such that w.P := w_1.P + w_2.P,
      w.A := w_1.A for all A ∈ sch(C), A ≠ P;
    replace w_1, w_2 in C by w;
  end
end
```

Fig. 20 Algorithms for WSD normalization.

simplifications apply to the WSD of Figure 11 (b), where tuples t_2 and t_3 are invalid. □

P.t_1.A
1
2

×

P.t_1.B	P.t_1.C
⊥	⊥
2	7

×

P.t_3.A
6

×

P.t_3.B
6

×

P.t_3.C
7

Fig. 21 Normalization of WSD of Figure 11 (a).

Example 13 The 4-WSD of Figure 13 admits the equivalent 5-WSD, where the third component is decomposed into two components. This non-maximality case cannot appear for UWSDTs, because all but the first component contain only one tuple and are stored in the template relation, where no component merging occurs. □

8 Chasing Dependencies

In this section we address the problem of removing inconsistent worlds from a probabilistic database. We present a method called *Chase* [3,24,2] in the spirit of the work of [18] for data cleaning on a world-set decomposition of a relation R, given a set of dependencies Φ.

We consider the following types of dependencies over a relation R:

– *functional dependencies* denoted by

$$A_1, \ldots, A_m \rightarrow A_0, \text{ where } A_i \in sch(R), 0 \leq i \leq m$$

– *single-tuple equality-generating dependencies* of the form

$$\phi_1 \wedge \phi_2 \wedge ... \wedge \phi_m \Rightarrow \phi_0$$

where each $\phi_i(A_i) = A_i \theta_i c_i, 0 \leq i \leq m$ is a binary operation comparing the value of an attribute $A_i \in sch(R)$ with a constant c_i. Relation R satisfies a single-tuple equality-generating dependency[4] egd (denoted by $R \models egd$) if for each tuple $t \in R$

$$t.A_1\ \theta_1\ c_1 \wedge \ldots \wedge t.A_m\ \theta_k\ c_m \Rightarrow t.A_0\ \theta_0\ c_0$$

To remove worlds inconsistent with an integrity constraints from a set of possible worlds represented as a WSD we need to exclude combinations of values from the components that cause the constraint to be violated. For that we may need to compose components to be able to enforce the dependencies. Recall Example 2 from the introduction. The uniqueness constraint for the social security number is a functional dependency $S \rightarrow N, M$, equivalent to the two functional dependencies $S \rightarrow N$ and $S \rightarrow M$. To enforce this constraint we combined the two S fields ($t_1.S$ and $t_2.S$) in the same component and removed the worlds in which both have the same value (see Figure 4).

Assume now that from a reliable source we have the information that the person with social security number 785 is married. The current decomposition allows invalid combinations of values: those worlds in which $t_1.S = 785$ and $t_1.M \neq 1$ (1 is the code for married). To remove inconsistencies, we must compose the first and the third components and remove from the new component all tuples that do not satisfy the given dependency. When removing tuples of a component we must also renormalize the probabilities of the remaining tuples so that they sum up to one again. This is easily done in the following way: if a tuple with probability x is removed from a component, and y is the original probability of a tuple that remains, then the new probability y' of the second tuple is recomputed as $y' = y/(1-x)$. In our example, as a result of the data-cleaning step we obtain the 4-WSD in Figure 22.

Enforcing a dependency on a WSD resembles the selection operation with condition $A\theta B$ presented in Chapter 4. In both cases we identify dependencies across components and compose dependent components. Nevertheless there is

[4] Subsequently, whenever we refer to edgs, we mean single-tuple edgs.

t_1.S	t_2.S	$t_1.M$	P
185	186	1	0.1842
185	186	2	0.0790
785	185	1	0.3684
785	186	1	0.3684

×

t_1.N	P
Smith	1

×

t_2.N	P
Brown	1

×

t_2.M	P
1	0.25
2	0.25
3	0.25
4	0.25

Fig. 22 Result of chasing $S = 785 \Rightarrow M = 1$ on the WSD in Figure 4.

an important difference between the two operations. In the selection operation we are interested in finding, for each world, *the subset of tuples* valid in it. On the other hand, when enforcing dependencies on a WSD, we want to get *the maximal subset of the possible worlds* such that the dependencies hold for *all tuples*. If a tuple has no valid values in any of the worlds, this automatically means that the database is inconsistent with respect to the given set of dependencies.

As seen in the previous examples, cleaning inconsistent worlds involves two basic steps: (1) composing dependent components into one and (2) removing inconsistent tuples from the resulting component, and normalizing the probabilities of the remaining tuples so that they sum up to one. Executing these two steps for each dependency and each (pair of) tuple(s) in the input WSD results in a WSD satisfying all constraints.

Before proceeding to the formal algorithm for chasing dependencies, we introduce the following notations.
If $fd = A_1, \ldots, A_m \rightarrow A_0$ is a functional dependency for relation R, s, t are tuples ids in R and all attributes $s.A_i$, $t.A_i$ with $0 \leq i \leq m$ are defined in a component C, and t_C is a tuple of C, we will use $t_C \models fd(s,t)$ to express the condition that the dependency fd is satisfied for s and t in the worlds t_C:

$$t_C \models fd(s,t) \Leftrightarrow$$
$$\bigwedge_i (t_C.(s.A_i) = t_C.(t.A_i)) \Rightarrow t_C.(s.A_0) = t_C.(t.A_0)$$

Similarly, if t is a tuple id for relation R,

$$egd = A_1\ \theta_1\ c_1 \wedge \ldots \wedge A_m\ \theta_m\ c_m \Rightarrow A_0\ \theta_0\ c_0$$

is an equality-generating dependency over R and all attributes $t.A_i, 0 \leq i \leq m$ are defined in a component C, and t_C is a tuple of C, $t_C \models egd(t)$ is true if and only if the dependency egd is satisfied for t in the worlds t_C:

$$t_C \models egd(t) \Leftrightarrow \bigwedge_i (t_C.(t.A_i)\ \theta_i\ c_i) \Rightarrow t_C.(t.A_0)\ \theta_0\ c_0$$

The algorithm of Figure 24 implements the data cleaning for a given world-set decomposition and a set of dependencies Φ. Note that as opposed to the traditional chase on tableaux ([24]), here we do not need a fixpoint computation but a single pass over all dependencies and tuples in the WSD. The reason for this is that enforcing a functional or equality-generating dependency on a WSD cannot induce further inconsistencies in the data.

We can further refine the data cleaning rules and avoid redundant operations if we make the following observations. For a functional dependency

$$fd = A_1, \ldots, A_m \rightarrow A_0$$

and tuples s and t, if for an attribute $A_i, 1 \leq i \leq m$ it holds that $s.A_i = t.A_i$ in all worlds, we do not need to join the components defining $s.A_i$ and $t.A_i$. Alternatively, if in all worlds $s.A_0 \neq t.A_0$, we can leave the components for $s.A_0$ and $t.A_0$ unmerged. The same idea can be applied for an equality-generating dependency

$$egd = A_1\ \theta_1\ c_1 \wedge \ldots \wedge A_m\ \theta_m\ c_m \Rightarrow A_0\ \theta_0\ c_0$$

tuple s and an attribute $A_i, 1 \leq i \leq m$, such that $\phi_i(t.A_i) = true$ in all worlds, or $\phi_0(t.A_0)$ is always $false$, we do not need to compose the corresponding component.

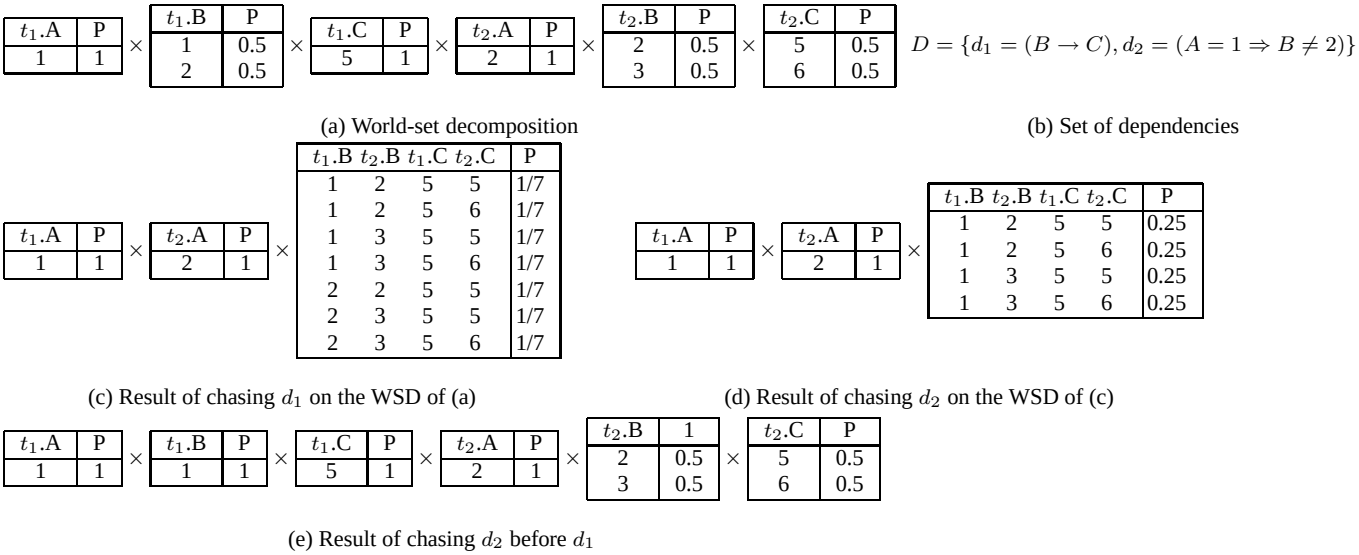
The chase procedure is not affected by the order in which dependencies are chased, as it always produces the set of possible worlds consistent with the given dependencies. However, order may have an impact on the size of the resulting decomposition. This means that the world-set decomposition produced by the Chase algorithm may be non-maximal, which was also the case with querying. Consider for example the WSD in Figure 23 (a) and the set of two dependencies

$$D = \{d_1 = (B \rightarrow C), d_2 = (A = 1 \Rightarrow B \neq 2)\}$$

Chasing $d_1 = B \rightarrow C$ requires the compositions of the components for t_1.B, t_2.B, t_1.C and t_2.C to remove the worlds in which $t_1.B = t_2.B$ and $t_1.C \neq t_2.C$ (see Figure 23 (c)), and enforcing d_2 deletes tuples from the resulting component (see Figure 23 (d)). However, if we start with d_2, in the resulting WSD d_1 will also be satisfied and no merging of components will be necessary (Figure 23 (e)). Note that although the two world-set decompositions are different, they are equivalent with respect to the set of possible worlds they represent. Indeed, the WSD in Figure 23 (d) can be reduced to the one in Figure 23 (e) using the normalization techniques from Section 7.

As in the case of querying, the chase might need to merge an arbitrary number of components. However, if constraints are local and do not span over numerous tuples, the chase will also behave nicely.

The following theorems prove the correctness of the Chase algorithm.



(a) World-set decomposition

(b) Set of dependencies

(c) Result of chasing d_1 on the WSD of (a)

(d) Result of chasing d_2 on the WSD of (c)

(e) Result of chasing d_2 before d_1

Fig. 23 Impact of order on chasing.

```
// chase a set of dependencies Φ
algorithm chase
begin
  for each d for relation R in Φ do
    if d = A_1, ..., A_m → A_0 then // d is a fd
      for each s,t ∈ R : {s,t} ⊭ d in some world do begin
        let C_{j_i}, C_{k_i} be the component of s.A_i, t.A_i,
          respectively, for each 0 ≤ i ≤ m;
        replace C_{j_0}, ..., C_{j_m}, C_{k_0}, ..., C_{k_m} in W
          by their product C;
        for each t_C ∈ C do
          if t_C ⊭ d(s,t) then
            remove t_C from C;
            for each t'_C ∈ C do
              // normalize probabilities
              t'_C.P = t'_C.P/(1 − t_C.P);
            end if
        if C = ∅ then error("World-set is inconsistent");
      end
    else if d = φ_1 ∧ ... ∧ φ_m → φ_0 // d is an egd
      for each t ∈ R : {t} ⊭ d in some world do begin
        let C_i be the component of t.A_i, for 0 ≤ i ≤ m;
        replace C_0, ..., C_m in W by their product C;
        for each t_C ∈ C do
          if t_C ⊭ d(t) then
            remove t_C from C;
            for each t'_C ∈ C do
              // normalize probabilities
              t'_C.P = t'_C.P/(1 − t_C.P);
            end if
        if C = ∅ then error("World-set is inconsistent.");
      end;
end.
```

Fig. 24 Algorithm for chasing integrity constraints on probabilistic WSDs.

Theorem 2 *The algorithm of Figure 24 terminates on all inputs.*

Theorem 3 (Correctness) *For a WSD $\mathcal{W}$ and a set of dependencies Φ, the algorithm of Figure 24 exits with an error message if no world is consistent with the given set of dependencies, or computes a WSD $\mathcal{W}'$ s.t. $rep(\mathcal{W}') \subseteq rep(\mathcal{W})$ and for each $\mathcal{A} \in rep(\mathcal{W})$:*

$\mathcal{A} \in rep(\mathcal{W}') \Leftrightarrow \mathcal{A} \models \Phi$.

9 Experimental Evaluation

The literature knows a number of approaches to representing incomplete information databases, but little work has been done so far on expressive yet efficient representation systems. An ideal representation system would allow a large set of possible worlds to be managed using only a small overhead in storage space and query processing time when compared to a single world represented in a conventional way. In the previous sections we presented the first step towards this goal. This section reports on experiments with a large census database with noise represented as a UWSDT, where the focus is on representation sizes and processing times for relational algebra queries on world-set decompositions. We do not investigate here the confidence computation aspect of query processing. Followup work of the authors [23] reports on experiments using scalable confidence computation techniques.

Setting. The experiments were conducted on a Dual Intel Xeon 5335 processor machine[5] with 32 GB RAM, running

[5] The processor has 8 cores running at 2.0 Ghz. The experiments were run on a single core.

Red Hat Enterprise Linux 4 (Linux Kernel 2.6.18) and PostgreSQL 8.3 configured to use 256MB as buffer.

Datasets. The IPUMS 5% census data (Integrated Public Use Microdata Series, 1990) [27] used for the experiments is the publicly available 5% extract from the 1990 US census, consisting of 50 (exclusively) multiple-choice questions. It is a relation with 50 attributes and 12491667 tuples (approx. 12.5 million). The size of this relation stored in PostgreSQL is ca. 3 GB. We also used excerpts representing the first 0.1, 0.5, 0.75, 1, 5, 7.5, and 10 million tuples.

Adding Incompleteness. We added incompleteness as follows. First, we generated a large set of possible worlds by introducing noise. After that, we cleaned the data by removing worlds inconsistent with respect to a given set of dependencies. Both steps are detailed next.

We introduced noise by replacing some values with or-sets[6]. We experimented with different noise ratios: 0.005%, 0.01%, 0.05%, 0.1%. For example, in the 0.1% scenario one in 1000 fields is replaced by an or-set. The size of each or-set was randomly chosen in the range $[2, min(8, size)]$, where $size$ is the size of the domain of the respective attribute (with a measured average of 3.5 values per or-set). In one scenario we had far more than 2^{624449} worlds, where 624449 is the number of the introduced or-sets and 2 is the minimal size of each or-set (cf. Figure 27).

We then performed data cleaning using 12 equality generating dependencies, representing real-life constraints on the census data, shown in Figure 25. These represent real-life constraints on the census data. The first one for example says that citizens born in the USA are not immigrants, and the second one requires that citizens who served in the second world war have done their military service. Note that or-set relations are not expressive enough to represent the cleaned data with dependencies.

To remove inconsistent worlds with respect to given dependencies, we apply the chase algorithm from Section 8, see also [6]. The chase is implemented in Java as a layer on top of PostgreSQL. Figure 26 shows a log-log scale of the times obtained for chasing the 12 dependencies on datasets with different sizes and uncertainty ratios.

Figure 27 shows the effect of chasing our dependencies on the 12.5 million tuples and varying placeholder density. As a result of merging components, the number of components with more than one placeholder (#comp>1) grows linearly with the increase of placeholder density, reaching about 1.7% of the total number of components (#comp) in the 0.1% case. A linear increase is witnessed also by the chasing time when the number of tuples is also varied. Figure 28 breaks down the distribution of component size, that is the number of placeholders per component for some of our scenarios. One can see that the number of components with larger size drops down very quickly and most fields remain independent. Since we used an anonymized version

[6] We consider it infeasible to iterate over all worlds in secondary storage or to compute UWSDT decompositions by comparing the worlds.

1	CITIZEN	$= 0$	$\Rightarrow$	IMMIGR	$= 0$
2	FEB55	$= 1$	$\Rightarrow$	MILITARY	$! = 4$
3	KOREAN	$= 1$	$\Rightarrow$	MILITARY	$! = 4$
4	VIETNAM	$= 1$	$\Rightarrow$	MILITARY	$! = 4$
2	WWII	$= 1$	$\Rightarrow$	MILITARY	$! = 4$
6	MARITAL	$= 0$	$\Rightarrow$	RSPOUSE	$! = 6$
7	MARITAL	$= 0$	$\Rightarrow$	RSPOUSE	$! = 5$
8	LANG1	$= 2$	$\Rightarrow$	ENGLISH	$! = 4$
9	RPOB	$= 52$	$\Rightarrow$	CITIZEN	$! = 0$
10	SCHOOL	$= 0$	$\Rightarrow$	KOREAN	$! = 1$
11	SCHOOL	$= 0$	$\Rightarrow$	FEB55	$! = 1$
12	SCHOOL	$= 0$	$\Rightarrow$	WWII	$! = 1$

Fig. 25 Example dependencies for cleaning census data.

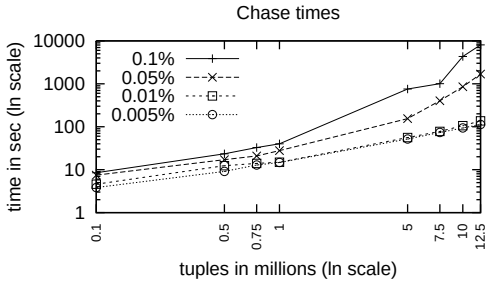


Fig. 26 Time for chasing the dependencies of Figure 25 on UWSDTs of various sizes and densities.

	Density	0.005%	0.01%	0.05%	0.1%
Initial	#comp	31095	62517	312699	624311
After chase	#comp	30820	61945	309788	618466
	#comp>1	268	547	2805	5612
	$\|C\|$	105150	211770	1061212	2117219
	$\|R\|$	12.5M	12.5M	12.5M	12.5M
After Q_1	#comp	674	1503	7333	14251
	#comp>1	4	7	51	76
	$\|C\|$	1773	4017	19225	37661
	$\|R\|$	46608	46827	48460	50466
After Q_2	#comp	21	46	256	459
	#comp>1	0	1	4	16
	$\|C\|$	92	245	1221	2361
	$\|R\|$	82996	83029	83275	83616
After Q_3	#comp	61	113	492	961
	#comp>1	0	0	0	0
	$\|C\|$	136	244	1075	2023
	$\|R\|$	17951	18019	18456	19054
After Q_4	#comp	1568	3150	15523	31379
	#comp>1	15	22	141	322
	$\|C\|$	4870	9117	46715	94747
	$\|R\|$	402349	402541	404031	405830
After Q_5	#comp	16	36	175	266
	#comp>1	16	36	175	266
	$\|C\|$	24451	40552	279295	561545
	$\|R\|$	158519	188790	378849	584207
After Q_6	#comp	94	193	950	1877
	#comp>1	0	0	0	0
	$\|C\|$	519	1077	5303	10304
	$\|R\|$	229592	230102	234195	239621

Fig. 27 UWSDTs characteristics for 12.5M tuples.

Size	Density	size 1	size 2	size 3	size 4 and more
5M	0.005%	12409	105	5	0
5M	0.01%	24454	213	7	0
5M	0.05%	122652	1065	38	2
5M	0.1%	245561	2142	93	1
10M	0.005%	24310	200	6	0
10M	0.01%	48943	430	16	1
10M	0.05%	245373	2164	83	0
10M	0.01%	497618	884	0	0
12.5M	0.005%	30552	261	7	0
12.5M	0.01%	61398	522	25	0
12.5M	0.05%	306983	2703	98	4
12.5M	0.1%	612854	5384	223	5

Fig. 28 Distribution of component size (number of placeholders per component) of the chased relations for different sizes and densities.

$$
\begin{aligned}
Q_1 &:= \sigma_{\mathsf{YEARSCH}=17\wedge\mathsf{CITIZEN}=0}(R)\\
Q_2 &:= \pi_{\mathsf{POWSTATE,CITIZEN,IMMIGR}}(\sigma_{\mathsf{CITIZEN}<>0\wedge\mathsf{ENGLISH}>3}(R))\\
Q_3 &:= \pi_{\mathsf{POWSTATE,MARITAL,FERTIL}}(\sigma_{\mathsf{POWSTATE}=\mathsf{POB}}\\
&\qquad (\sigma_{\mathsf{FERTIL}>4\wedge\mathsf{MARITAL}=1}(R)))\\
Q_4 &:= \sigma_{\mathsf{FERTIL}=1\wedge(\mathsf{RSPOUSE}=1\vee\mathsf{RSPOUSE}=2)}(R)\\
Q_5 &:= \delta_{\mathsf{POWSTATE}\to P_1}(\sigma_{\mathsf{POWSTATE}>50}(Q_2)) \bowtie_{P_1=P_2}\\
&\qquad \delta_{\mathsf{POWSTATE}\to P_2}(\sigma_{\mathsf{POWSTATE}>50}(Q_3))\\
Q_6 &:= \pi_{\mathsf{POWSTATE,POB}}(\sigma_{\mathsf{ENGLISH}=3}(R))
\end{aligned}
$$

Fig. 29 Queries on IPUMS census data.

of the census dataset, we did not perform the chase with key dependencies like the ones described in Section 1. Note that when chasing dependencies we only need to compose components if the possible values for the fields allow for a constraint to be violated, that is, if there is an invalid combination of values for the respective fields. Thus while chasing key constraints can in theory require the composition of all components for a given attribute, this is unlikely to happen in practice as it will require the existence of a chain of pairs of uncertain key fields that share at least one value.

Queries. Six queries were chosen to show the behavior of relational operators combinations under varying selectivities (cf. Figure 29). Query Q_1 returns the entries of US citizens with PhD degree. The less selective query Q_2 returns the place of birth of US citizens born outside the US that do not speak English well. Query Q_3 retrieves the entries of widows that have more than three children and live in the state where they were born. The very unselective query Q_4 returns all married persons having no children. Query Q_5 uses query Q_2 and Q_3 to find all possible couples of widows with many children and foreigners with limited English language proficiency in US states with IPUMS index greater than 50 (i.e., eight 'states', e.g., Washington, Wisconsin, Abroad). Finally, query Q_6 retrieves the places of birth and work of persons speaking English well.

Figure 27 describes some characteristics of the answers to these queries when applied on the cleaned 12.5M tuples of IPUMS data: the total number of components (#comp) and of components with more than one placeholder (#comp$>$1), the size of the component relation C, and the size of the template relation R. One can observe that the number of components increases linearly with the placeholder density and that compared to chasing, query evaluation leads to a much smaller amount of component merging.

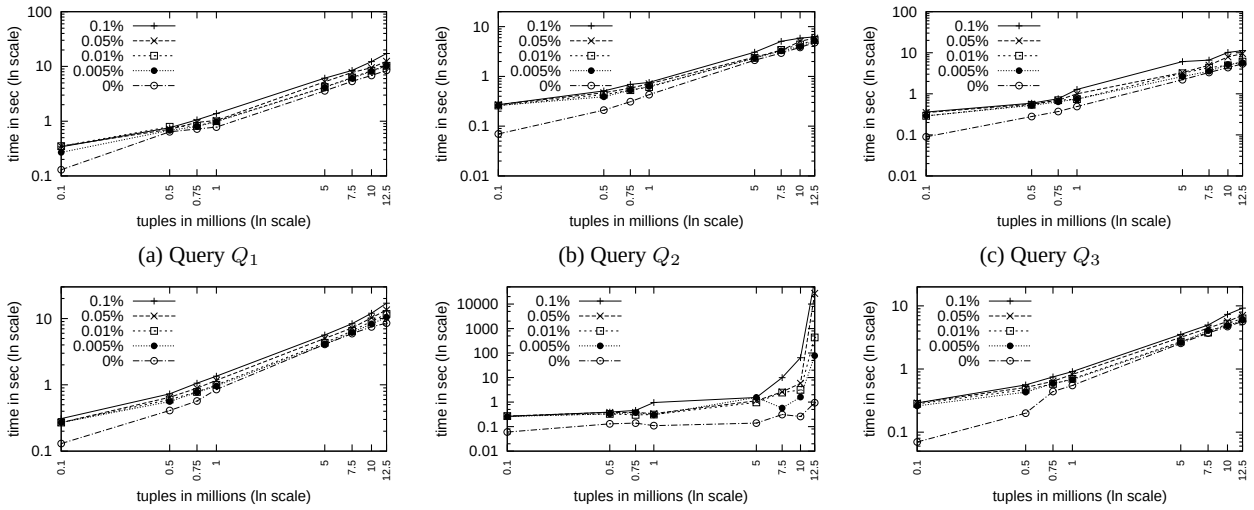
Figure 30 shows that all six queries admit efficient and scalable evaluation on UWSDTs of different sizes and placeholder densities. The Figure plots on a log-log scale the evaluation time versus the size of the relation, and each line corresponds to a different noise density. The evaluation time for all queries but Q_5 on UWSDTs follows very closely the evaluation time in the one-world case. The one-world case corresponds to density 0% in our diagrams, i.e., when no placeholders are created in the template relation and consequently there are no components. In this case, the original queries (that is, not the rewritten ones) of Figure 29 were evaluated only on the (complete) template relation.

Although the evaluation of join conditions on UWSDTs can require exponential time (due to the composition of arbitrarily many components), our experiments suggest that they behave well in practical cases, as illustrated in Figures 30 (c) and (e) for queries Q_3 and Q_5 respectively. The time reported for Q_5 does not include the time to evaluate its subqueries Q_2 and Q_3. In our largest scenarios (12.5M tuples and varying densities of uncertainty), the time to evaluate query Q_5 increases non-linearly, partly due to the change of query plans used by PostgreSQL and triggered by the increase in the input data size.

In summary, our experiments show that UWSDTs behave very well in practical cases. We found that the size of UWSDTs obtained as query answers remains close to that of one of their worlds. Furthermore, the processing time for queries on UWSDTs is comparable to processing one world. The explanation for this is that in practice there are rather few differences between the worlds. This keeps the mapping and component relations relatively small and the lion's share of the processing time is taken by the templates, whose sizes are about the same as of a single world.

10 Application Scenarios

Our approach is designed to cope with large sets of possible worlds, which exhibit local dependencies and large commonalities. This data pattern can be found in many applications. In addition to the census scenario used in Section 9, we next discuss two further application scenarios that can profit from our approach. As for the census scenario, we consider it infeasible both to iterate over all possible worlds



(a) Query Q_1 (b) Query Q_2 (c) Query Q_3

(d) Query Q_4 (e) Query Q_5 (f) Query Q_6

Fig. 30 The evaluation time for queries of Figure 29 on UWSDTs of various sizes and densities.

in secondary storage, or to compute UWSDT decompositions by comparing the worlds. Thus we also outline how our UWSDTs can be efficiently computed.

Inconsistent databases. A database is inconsistent if it does not satisfy given integrity constraints. Sometimes, enforcing the constraints is undesirable. One approach to manage such inconsistency is to consider so-called *minimal repairs*, i.e., consistent instances of the database obtained with a minimal number of changes [10]. A repair can therefore be viewed as a possible (consistent) world. The number of possible minimal repairs of an inconsistent database may in general be exponential; however, they substantially overlap. For that reason repairs can be easily modeled with UWSDTs, where the consistent part of the database is stored in template relations and the differences between the repairs in components. Current work on inconsistent databases [10] focuses on finding *consistent query answers*, i.e., answers appearing in all possible repairs (worlds). With our approach we can provide more than that, as the answer to a query represents a set of possible worlds. In this way, we preserve more information that can be further processed using querying or data cleaning techniques.

Medical data. Another application scenario is modeling information on medications, diseases, symptoms, and medical procedures, see, e.g., [1]. A particular characteristic of such data is that it contains a big number of clusters of interdependent data. For example, some medications can interact negatively and are not approved for patients with some diseases. Particular medical procedures can be prescribed for some diseases, while they are forbidden for others. In the large set of possible worlds created by the complex interaction of medications, diseases, procedures, and symptoms, a particular patient record can represent one or a few possible worlds. Our approach can keep interdependent data within components and independent data in separate components. One can ask then for possible patient diagnostics, given an incompletely specified medical history of the patient, or for commonly used medication for a given set of diseases.

In [1] interdependencies of medical data are modeled as links. A straightforward and efficient approach to wrap such data in UWSDTs is to follow the links and create one component for all interrelated values. Additionally, each different kind of information, like medications, diseases, is stored in a separate template relation.

11 Conclusion

This article presents one of the first database approaches to managing probabilistic data on a large scale. We describe world-set decompositions which can compactly store large sets of possible worlds by exploiting independence of uncertainty at the attribute level. WSDs form a strong representation system for any relational query language. This is an important property for implementing operations that transform world-sets such as data cleaning or evaluating expressive queries on top of the world-set; it also allows for decoupling confidence computation from relational algebra processing and using a preferred query plan for optimal performance. Our experimental evaluation shows that WSDs admit efficient query evaluation.

Acknowledgments

This work has been supported by grant KO 3491/1-1 of the German National Science Foundation (DFG) and later by grant IIS-0812272 of the US National Science Foundation.

References

1. http://www.medicinenet.com.
2. Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
3. Alfred V. Aho, Catriel Beeri, and Jeffrey D. Ullman. "The Theory of Joins in Relational Databases". *ACM Transactions on Database Systems*, **4**(3):297–314, 1979.
4. Periklis Andritsos, Ariel Fuxman, and Renee J. Miller. Clean answers over dirty databases: A probabilistic approach. In *Proc. ICDE*, 2006.
5. Lyublena Antova, Thomas Jansen, Christoph Koch, and Dan Olteanu. Fast and simple relational processing of uncertain data. In *Proc. ICDE*, 2008.
6. Lyublena Antova, Christoph Koch, and Dan Olteanu. $\mathbf{10^{10^6}}$ worlds and beyond: Efficient representation and processing of incomplete information. Technical Report cs.DB/0606075, ACM CORR, 2006. v1.
7. Lyublena Antova, Christoph Koch, and Dan Olteanu. $\mathbf{10^{10^6}}$ worlds and beyond: Efficient representation and processing of incomplete information. In *Proc. ICDE*, 2007.
8. Lyublena Antova, Christoph Koch, and Dan Olteanu. "From Complete to Incomplete Information and Back". In *Proc. SIGMOD*, 2007.
9. Lyublena Antova, Christoph Koch, and Dan Olteanu. World-set decompositions: Expressiveness and efficient algorithms. In *Proc. ICDT*, 2007.
10. Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *Proc. PODS*, pages 68–79, 1999.
11. Omar Benjelloun, Anish Das Sarma, Alon Halevy, and Jennifer Widom. ULDBs: Databases with uncertainty and lineage. In *Proc. VLDB*, 2006.
12. Philip Bohannon, Wenfei Fan, Michael Flaster, and Rajeev Rastogi. "A Cost-Based Model and Effective Heuristic for Repairing Constraints by Value Modification. In *Proc. SIGMOD*, 2005.
13. Andrea Calí, Domenico Lembo, and Riccardo Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proc. PODS*, pages 260–271, 2003.
14. Reynold Cheng, Sarvjeet Singh, and Sunil Prabhakar. U-DBMS: A database system for managing constantly-evolving data. In *Proc. VLDB*, pages 1271–1274, 2005.
15. Nilesh Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. In *Proc. VLDB*, pages 864–875, 2004.
16. H. Galhardas, D. Florescu, D. Shasha, and E Simon. "AJAX: An Extensible Data Cleaning Tool". In *Proc. SIGMOD*, 2000.
17. Michaela Götz and Christoph Koch. A compositional framework for complex queries over uncertain data. In *Proc. ICDT*, 2009.
18. Gösta Grahne. Dependency satisfaction in databases with incomplete information. In *Proc. VLDB*, pages 37–45, 1984.
19. Rahul Gupta and Sunita Sarawagi. Creating probabilistic databases from information extraction models. In *Proc. VLDB*, 2006.
20. T. Imielinski and W. Lipski. Incomplete information in relational databases. *Journal of ACM*, 31:761–791, 1984.
21. T. Imielinski, S. Naqvi, and K. Vadaparty. Incomplete objects — a data model for design and planning applications. In *Proc. SIGMOD*, pages 288–297, 1991.
22. Christoph Koch. Approximating predicates and expressive queries on probabilistic databases. In *Proc. PODS*, 2008.
23. Christoph Koch and Dan Olteanu. Conditioning probabilistic databases. In *Proc. VLDB*, 2008.
24. David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. "Testing Implications of Data Dependencies". *ACM Transactions on Database Systems*, **4**(4):455–469, 1979.
25. Erhard Rahm and Hong Hai Do. "Data Cleaning: Problems and Current Approaches". *IEEE Data Engineering Bulletin*, 2000.
26. V. Raman and J.M. Hellerstein. "Potter's Wheel: An Interactive Data Cleaning System". In *Proc. VLDB*, 2001.
27. Steven Ruggles, Matthew Sobek, Trent Alexander, Catherine A. Fitch, Ronald Goeken, Patricia Kelly Hall, Miriam King, and Chad Ronnander. Integrated public use microdata series: V3.0, 2004. http://www.ipums.org.
28. Anish Das Sarma, Omar Benjelloun, Alon Halevy, and Jennifer Widom. Working models for uncertain data. In *Proc. ICDE*, 2006.
29. Prithviraj Sen and Amol Deshpande. "Representing and Querying Correlated Tuples in Probabilistic Databases". In *Proc. ICDE*, 2007.
30. Sarvjeet Singh, Chris Mayfield, Rahul Shah, Sunil Prabhakar, Susanne Hambrusch, Jennifer Neville, and Reynold Cheng. Database support for probabilistic attributes and tuples. In *Proc. ICDE*, 2008.