10^{10^6} Worlds and Beyond: Efficient Representation and Processing of Incomplete Information

Lyublena Antova, Christoph Koch, and Dan Olteanu

Lehrstuhl für Informationssysteme
Universität des Saarlandes
Saarbrücken, Germany
{lublena,koch,olteanu}@infosys.uni-sb.de

ABSTRACT

Current systems and formalisms for representing incomplete information generally suffer from at least one of two weaknesses. Either they are not strong enough for representing results of simple queries, or the handling and processing of the data, e.g. for query evaluation, is intractable.

In this paper, we present a decomposition-based approach to addressing this problem. We introduce *world-set decompositions (WSDs)*, a space-efficient formalism for representing any finite set of possible worlds over relational databases. WSDs are therefore a strong representation system for any relational query language. We study the problem of efficiently evaluating relational algebra queries on sets of worlds 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

Real-world data collections tend to be incomplete rather than complete. Classical examples of incompleteness can be found in data integration and wrapping applications, linguistic collections, or whenever information is manually entered and is therefore prone to inaccuracy or partiality. Nevertheless, there has been little research so far into expressive *yet scalable* systems for representing incomplete information.

Current techniques for representing incomplete information can be classified into two groups. The first group includes representation systems such as *v-tables* [10] and *relations with or-sets* [11] which are not strong enough to represent all results of relational algebra queries within the same formalism. 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 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* (tables with conditions) [10] belong to the second group of formalisms. They extend v-tables by adding conditions specified by logical formulas over the variables, thus constraining the possible values. Although c-tables are a strong representation system, they have not found application in practice. The main reason for this is probably that they are awkward to handle; storing c-tables directly is rather inefficient, and it is known that already the *data complexity* [17], i.e. the complexity of the query evaluation problem under the assumption that the query size is fixed, of relational algebra over c-tables is NP-hard [9]. In fact, due to the great power of the condition formulas, processing queries on c-tables tends to be intractable already for very small databases.

As a motivation, let us consider the following example. Figure 2 shows two manually completed forms that may originate from a census or some other kind of survey and which allow for more than one interpretation. For simplicity we assume that social security numbers consist only of 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, as the one in Figure 1.

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

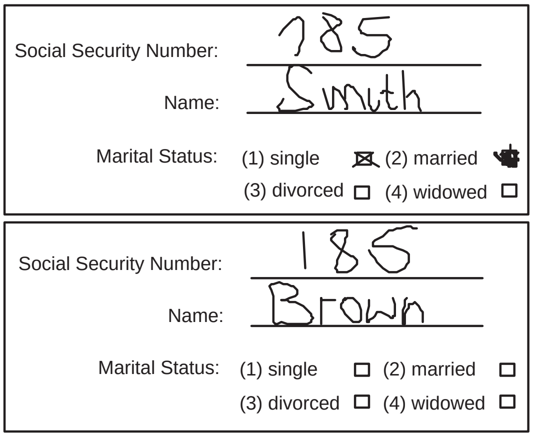
Figure 1: Or-set relation.

It is easy to see that this or-set relation represents $2 * 2 * 2 * 4 = 32$ possible relations, also called *worlds*.

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, the evaluation of queries on the data, and the use of *data cleaning* procedures by which certain unlikely or invalid worlds can be excluded. However, the results of both types of operation turn out not to be representable by or-set relations in general. Consider for example the following query: "Find all pairs of people that have a different marital status." Each tuple in the result contains two marital status fields, which cannot have the same value at the same time. This renders some combinations of values invalid, and the query result cannot be represented as an or-set relation.

An example integrity constraint for data cleaning that makes it impossible to store the data as an or-set relation is the requirement that all social security numbers should be unique. For our example database, this rule will eliminate 8 of the 32 worlds, namely the ones in which both tuples have the value 185 as social security number.

It is impossible to represent the 24 worlds that remain after applying the data cleaning rule from above using or-



Social Security Number: 185

Name: Smith

Marital Status: (1) single ☒ (2) married ■
(3) divorced □ (4) widowed □

Social Security Number: 185

Name: Brown

Marital Status: (1) single □ (2) married □
(3) divorced □ (4) widowed □

Figure 2: Two completed survey forms.

set relations. What we could do is store each possible world explicitly. One way to do this is using a table called a *world-set relation* of a given set of possible worlds. Each tuple in this table represents one world and is the concatenation of all tuples in that world (see Figure 3).

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 10000 answers can be read in just two different ways, we get 2^{10^6} possible worlds. Each such world is a substantial table of 50 columns and 200 million 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 be able to manage those that remain.

In this paper, we aim at dealing with this complexity. Our approach is based on 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.

Example 1.1. The world-set represented by our initial or-set relation can also be represented by the product in Figure 4.

Example 1.2. In the same way we can represent the result of data cleaning with the uniqueness constraint for the social security numbers as the product of Figure 5.

One can observe that the result of this product is exactly the world-set relation in Figure 3. 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 separate components. □

In practice, it is often the case that corresponding 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 are in component relations that have precisely one tuple.

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

Figure 3: World-set relation for the remaining 24 worlds after excluding the ones with duplicated social security numbers.

$t_1.S$
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

Figure 4: World-set decomposition for the initial or-set relation.

$t_1.S$	$t_2.S$
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

Figure 5: WSD of the world-set relation of Figure 3.

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

Example 1.3. The world-set of the previous examples can be represented by the WSDT of Figure 6. □

Template	S	N	M
t_1	?	Smith	?
t_2	?	Brown	?

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

×

t_1.M
1
2

×

t_2.M
1
2
3
4

Figure 6: WSD with a template relation.

Using WSDs and WSDTs we can represent sets of worlds that cannot be expressed with or-set relations; at the same time the decomposition is just slightly larger than the original or-set relation for those cases where world-sets are actually expressible as or-set relations and can be compared.

c-table	S	N	M	
				$(x = 185 \wedge z = 186) \vee$ $(x = 785 \wedge z = 185) \vee$ $(x = 785 \wedge z = 186)$
	x	Smith	y	$y = 1 \vee y = 2$
	z	Brown	w	$w = 1 \vee w = 2 \vee w = 3 \vee w = 4$

Figure 7: A c-table encoding the WSDT of Figure 6.

WSDTs combine the advantages of WSDs and c-tables. In fact, WSDTs can be naturally viewed as c-tables whose formulas have been put into a *normal form* represented by the component relations, and variables represent fields where the worlds disagree. Indeed, each tuple in the product of the component relations is one possible value assignment for the variables in the template relation.

The c-table in Figure 7 is equivalent to the WSDT in Figure 6. The WSDT components that encode dependencies spanning over several tuples become in the equivalent c-table global conditions, and each other component becomes local condition to its respective tuple.

The technical contributions of this paper 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 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, DBMS (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 first discuss the query evaluation problem in the context of the much more graphic 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 briefly describe a prototype implementation on top of the PostgreSQL RDBMS that supports the management of incomplete information using UWSDTs.

- We initiate a study of the data cleaning problem in the context of UWSDTs. We focus on two kinds of dependencies, functional dependencies and a class of (in)equality-generating dependencies, and adapt the *Chase procedure* (cf. [12, 3, 8]) for incomplete information to the framework of 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 allow highly scalable techniques for managing incomplete information. We found that the size of UWSDTs obtained as query answers or data cleaning remains close to that of a single world. Furthermore, the processing time for queries on UWSDTs is also comparable to processing just a single world and thus a classical relational database.

A fundamental assumption of this work is that one wants to store and manage *sets* of possible worlds. We believe that this is justified by previous work on representation systems, starting with Imielinski and Lipski [10], and by current application requirements. Data cleaning is often an incremental process that requires to store large intermediate results, world-sets, in databases. Our approach can deal with databases in which some uncertainties could not be resolved. Such databases are still valuable. It should be possible to do data transformation queries that preserve as much information as possible, thus necessarily mapping from sets of possible worlds to sets of possible worlds. A different approach is followed in work on finding *certain answers* of queries on incomplete-information databases (see e.g. [6]).

Differently from c-tables, WSDs cannot represent infinite world-sets. A second assumption of this work is that *finite* world-sets are relevant. We present several application scenarios in this paper (such as census) where this clearly is the case. Furthermore, as discussed above, (U)WSDTs can be seen as a normal form for the c-tables representing finite world-sets. (U)WSDTs support efficient representation and processing, with the promise of yet more expressive while still scalable representation systems to follow in the future.

The structure of this paper is as follows. We start by introducing some required notation in Section 2. In Section 3, we formally define the (U)WSDTs. Sections 4 and 5 discuss efficient query evaluation in our framework and Section 6 addresses the data cleaning problem on WSDTs. Section 7 presents our experimental evaluation of the approach. Finally, Sections 8 and 10 discuss further application scenarios of WSDs as well as future work.

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* Σ is a set of constructs of the form $R[U]$, where R is a relation name and U is a set of attribute names. The arity $|U|$ of R is denoted by $ar(R)$.

Let $\mathbf{D}$ be a finite 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 $\mathcal{S}(R)$ denote the set U of its attributes.

Let R be a relation with schema $R[U]$. Then a disjoint m-partition $\{U_1, \ldots, U_m\}$ of U is called a *(product) m-decomposition* of R iff $\pi_{U_1}(R) \times \cdots \times \pi_{U_m}(R) = R$. The projections $\pi_{U_1}(R), \ldots, \pi_{U_m}(R)$ are called *components*.

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 to world-sets of the same schema. Then *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 2.1. *If rep is a function whose image is the set of all finite world-sets, then rep is a strong representation system for any relational query language.*

For the remainder of the paper we consider that the arity of all our database relations is at least one and the projection operation does not project to the empty set of attributes.

3. WORLD-SET DECOMPOSITIONS

3.1 The Basic Notion

In order to use classical database techniques for storing and querying 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 Σ. For each $R \in \Sigma$, let $|R|_{\max} = \max\{|R^{\mathcal{A}}| : \mathcal{A} \in \mathbf{A}\}$ denote the maximum cardinality of R in any world of $\mathbf{A}$. Given a world $\mathcal{A}$ with $R^{\mathcal{A}} = \{t_1, \ldots, t_{|R^{\mathcal{A}}|}\}$, let $t_{R^{\mathcal{A}}}$ be the tuple obtained as the concatenation (denoted $\circ$) of the tuples of $R^{\mathcal{A}}$ padded with a special null value $\bot \notin \mathbf{D}$ up to arity $ar(R) \cdot |R|_{\max}$,

$$t_{R^{\mathcal{A}}} := t_1 \circ \cdots \circ t_{|R^{\mathcal{A}}|} \circ (\underbrace{\bot, \ldots\ldots\ldots\ldots, \bot}_{ar(R)\cdot(|R|_{\max} - |R^{\mathcal{A}}|)}).$$

Then tuple $t_{\mathcal{A}} := t_{R_1^{\mathcal{A}}} \circ \cdots \circ t_{R_k^{\mathcal{A}}}$ for $\Sigma = \{R_1, \ldots, R_k\}$ encodes all the information in world $\mathcal{A}$. Now, by the *world-set relation* $W(\mathbf{A})$ of world-set $\mathbf{A}$, we denote the relation $\{t_{\mathcal{A}} \mid \mathcal{A} \in \mathbf{A}\}$. Relation $W(\mathbf{A})$ has schema (attributes) $\{R.t_i.A_j \mid R[U] \in \Sigma, 1 \le i \le |R|_{\max}, A_j \in U\}$.

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

$$t_{R^{\mathcal{A}}} \mapsto \{(a_{ar(R)\cdot k+1}, \ldots, a_{ar(R)\cdot(k+1)}) \mid 0 \le k < |R|_{\max},$$
$$a_{ar(R)\cdot k+1} \neq \bot, \ldots, a_{ar(R)\cdot(k+1)} \neq \bot\}.$$

(So one can think of $\bot$ as a deletion marker for tuples.)

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 our purposes because rep_{WS} maps them invariantly back to $\mathbf{A}$.

Definition 3.1. Let $\mathbf{A}$ be a world-set. Then a *world-set m-decomposition (m-WSD)* of $\mathbf{A}$ is a tuple $\{C_1, \ldots, C_m\}$ such that $C_1 \times \cdots \times C_m = W(\mathbf{A})$, that is, the schemata of $\{C_1, \ldots, C_m\}$ constitute an m-decomposition of the world-set relation $W(\mathbf{A})$. The world-set represented by m-WSD $\{C_1, \ldots, C_m\}$, subsequently called $rep(\{C_1, \ldots, C_m\})$, is $rep_{WS}(C_1 \times \cdots \times C_m)$.

Remark 3.2. There is a fairly large literature on the universal relation assumption and relational decomposition, particularly on lossless join decomposition, cf. e.g. [16, 2]; however, previous work assumes that decompositions are defined intensionally using dependencies. Decompositions of extensionally given relations are less natural in the classical context because such decompositions may break on updates.

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 components with fields from different relations.

It immediately follows from our definitions that

Proposition 3.3. *Any finite set of possible worlds can be represented as a world-set relation and as a* 1*-WSD.*

Corollary 3.4 (Lemma 2.1). *WSDs are a strong representation system for any relational query language.*

As demonstrated 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.*

An m-WSD is called *maximal(ly decomposed)* if no n-WSD of R exists with $n > m$.

Proposition 3.5. *For each world-set relation a maximal decomposition exists and is unique.*

Proof. Existence is clear because a world-set relation is also a 1-WSD. Uniqueness is shown by contradiction: Given relation R of schema $R[U]$, assume that there are two different maximal m-decompositions $\{U_1, \ldots, U_m\}$ and $\{V_1, \ldots, V_m\}$ of R. Since the two decompositions are different, there are two sets U_i, V_j such that $U_i \neq V_j$ and $U_i \cap V_j \neq \emptyset$. But then, as of course $R = \pi_{U-V_j}(R) \times \pi_{V_j}(R)$, we have $\pi_{U_i}(R) = \pi_{U_i}\big(\pi_{U-V_j}(R) \times \pi_{V_j}(R)\big) = \pi_{U_i - V_j}(R) \times \pi_{U_i \cap V_j}(R)$. It follows that $\{U_1, \ldots, U_{i-1}, U_i - V_j, U_i \cap V_j, U_{i+1}, \ldots, U_m\}$ is an $(m+1)$-decomposition of R, and m-decompositions cannot be maximal. Contradiction. □

3.2 Adding Template Relations

We now present our refinement of WSDs that uses *template relations* to store information that is the same in all possible worlds. The template relations contain special values '?' $\notin \mathbf{D}$ in fields at which different worlds disagree.

We will assume that tuples t have unique ids $\hat{t}$. Let $\Sigma = \{R_1, \ldots, R_k\}$ be a schema and $\mathbf{A}$ a finite set of possible worlds over Σ. Then, a 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 $R_j.\hat{t}.A$ and value v in its unique $R_j.\hat{t}.A$-field, then R_j^0 has a tuple with id $\hat{t}$ whose A-field has value v.

Of course WSDTs again can represent any finite world-set and are thus a strong representation system for relation query languages. Note that in contrast to WSDs, WSDTs admit a unique maximal decomposition only if the template relation R^0 is fixed. Example 1.3 shows a WSDT for the running example of the introduction.

3.3 Uniform World-Set Decompositions

In practice database systems often do not support relations of arbitrary arity. For that reason we introduce next a modified representation of WSDs called *uniform WSDs*. We use the fixed schema consisting of the three relation schemata $F[\mathit{FID}, \mathit{CID}]$, $W[\mathit{CID}, \mathit{LWID}]$, $C[\mathit{FID}, \mathit{LWID}, \mathit{VAL}]$, where *FID* is a triple[1] (*Rel*, *TupleID*, *Column*) denoting the *Column*-field of tuple $TupleID$ in database relation *Rel*. Instead of having a variable number of component relations, possibly with different arities, we store all values in a single big relation C that has a fixed schema.

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 $\hat{C}_i[U_i]$ (we now distinguish between the relation and its name), we populate the corresponding UWSD as follows.

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

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.

Example 3.6. We modify the world-set represented in Figure 5 such that the marital status in t_2 can only have the value 3. We obtain then a set of six worlds that can be represented using our alternative representation with fixed relational schemata for F, W, and C as shown in Figure 8.□

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

Example 3.7. Consider the example of Figure 9, which is the uniform version of the WSDT of Figure 6. 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 '?'. Just as before, the possible values for the placeholders are defined in the component table C. In practice, for incomplete-information databases, 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. □

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

F	FID	CID
	(R, t_1, S)	C_1
	(R, t_1, N)	C_2
	(R, t_1, M)	C_3
	(R, t_2, S)	C_1
	(R, t_2, N)	C_4
	(R, t_2, M)	C_5

W	CID	LWID
	C_1	1
	C_1	2
	C_1	3
	C_2	1
	C_3	1
	C_3	2
	C_4	1
	C_5	1

Figure 8: A uniform WSD for our running example.

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
	C_1	1
	C_1	2
	C_1	3
	C_2	1
	C_2	2

Figure 9: A UWSDT corresponding to the WSDT of Figure 6.

It is easy to verify that

Proposition 3.8. *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*.

4. QUERIES ON 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 in both representation systems. For that reason we will first concentrate on query evaluation in the WSD framework. Query evaluation for UWSDTs is then discussed in Section 5.

4.1 Relational Algebra Operations on WSDs

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

```
algorithm select[$A\theta c$]   // compute $P := \sigma_{A\theta c} R$
begin
  copy($R, P$);
  for each $1 \leq i \leq |P|_{max}$ 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 product   // compute $T := R \times S$
begin
  for each $1 \leq j \leq |S|_{max}$ and $R.t_i.A \in \mathcal{S}(R)$ 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 $1 \leq i \leq |R|_{max}$ and $S.t_j.A \in \mathcal{S}(S)$ 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 $1 \leq i \leq |R|_{max}$ and $A \in \mathcal{S}(R)$ 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 $1 \leq j \leq |S|_{max}$ and $A \in \mathcal{S}(S)$ 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 select[$A\theta B$]   // compute $P := \sigma_{A\theta B} R$
begin
  copy($R, P$);
  for each $1 \leq i \leq |P|_{max}$ 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 $C := C \times C'$;
    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 project[$U$]   // compute $P := \pi_U(R)$
begin
  copy($R, P$);
  for each $1 \leq i \leq |P|_{max}$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 $(\forall A' \in U : P.t_i.A' \notin \mathcal{S}(C'))$ and
        $(\forall t_{C'} \in C' : \bot \in t_{C'})$;
      replace components $C$, $C'$ by $C := C \times C'$;
      propagate-$\bot(C)$;
      project away $P.t_j.B$ from $C$ where $B \notin U$ and $j \leq i$;
    end
  for each $1 \leq i \leq |P|_{max}$ 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
```

Figure 10: Evaluating relational algebra operations on WSDs.

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 Q.

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 ×). Third, the answer to a (sub)query Q_0 must be represented within the same decomposition as the input relations; 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.

EXAMPLE 4.1. Consider for example a query $\sigma_{A=1}(R) \cup \sigma_{B=2}(R)$. If we first compute $\sigma_{A=1}(R)$, we must not replace R by $\sigma_{A=1}(R)$, otherwise R will not be available for the computation of $\sigma_{B=2}(R)$. On the other hand, if $\sigma_{A=1}(R)$ is stored in a separate WSD, the connection between worlds of R and the selection $\sigma_{A=1}$ is lost and we can again not compute $\sigma_{A=1}(R) \cup \sigma_{B=2}(R)$. □

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 relation 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 column $R.t.A_i$:

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

Then copy(R, P) executes $C :=$ ext$(C, R.t_i.A, P.t_i.A)$ for each component C of our WSD and each $R.t_i.A \in \mathcal{S}(C)$.

Figure 10 presents implementations of the 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, =, ≠, <, ≤, >, or ≥), projection, relational product, and union of relational algebra on WSDs. In each case, the input WSD is *extended* by the result of the operation.

Remark 4.2. Note that most of these operations can be expressed as relational algebra queries themselves. However, the encoding in relational algebra is nonuniform in that each WSD may have a different schema; at the same time the encodings of these operations in relational algebra can be easily generated, but are not fixed. We do not provide algorithms for generating here because of lack of space; yet they can be easily derived from the algorithms of Figure 10.

An exception here is the selection with "join condition" $A\theta B$. A *fixed* such query can require the merging of an arbitrary number of component relations into a single component, causing an exponential blowup in the size of the WSD. Since relational algebra has polynomial-time data complexity [2], i.e., fixed queries can only produce results of size polynomial in the input data, selection with conditions $A\theta B$ *cannot* be expressible in relational algebra. For that matter, such selections on WSDs are also not expressible in more expressive query languages with polynomial-time data complexity, such as datalog with stratified negation. As it turns out, the ability to create new identifiers is necessary to form exponential sets of tuples. □

The algorithms of Figure 10 provide us with a machinery to execute any relational algebra query using the operations σ, π, $\times$, and $\cup$. Renaming and difference are discussed in Section 4.2. 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 4.3 (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,*

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

4.2 Discussion and Examples

Let us now have a closer look at the algorithms of Figure 10 for evaluating relational algebra operations on WSDs. For this, we use as running example the set of eight worlds over the relation R of Figure 11 and its maximal 7-WSD of Figure 11 (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 11 (b) 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 $\mathcal{A}$.

Because of space limitations and 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 of component relations may neither contain any world-tuple $t \in R^{\mathcal{A}}$ in its entirety nor will each component tuple in general represent (parts of) only one world-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 10 checks for each pair of tuples t_i in the relation P and t_C in the component C that has attribute $P.t_i.A$ whether $t_C.(P.t_i.A)$ satisfies the selection condition[2], i.e., $t_C.(P.t_i.A)\theta c$. If it does not, the tuple $P.t_i$ is marked as deleted in all worlds that take values from t_C. This means 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 13). 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-$\perp$($C$: component)
begin
  for each $t_C \in C$ and $P.t_i.A \in \mathcal{S}(C)$ do
    if $t_C.(P.t_i.A) = \perp$ then
        for each $A'$ such that $P.t_i.A' \in S(C)$ do
          $t_C.(P.t_i.A') := \perp$;
end
```

Figure 13: Propagating $\perp$-values.

Example 4.4. The answer P to $\sigma_{C=7}(R)$ is represented by the WSD of Figure 12 (a). Figure 12 (b) shows the result of query $\sigma_{B=1}(R)$. 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 12 we may obtain worlds of different sizes. For example the worlds that take values from the first tuple of the second component relation in Figure 12 (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 two attributes of a tuple are accessed, which do not necessarily reside in the same component.

The 7-WSD of Figure 11 (b) has no component containing values for both the attributes A and B of any tuple of the decomposed worlds. Therefore, the test of the join condition for each of these tuples has to span over several components containing values for A and B. Additionally, the join condition may exclude some possible combinations of the values from different components. Therefore, the current decomposition may not capture exactly the combinations of values satisfying the join condition. It may then be necessary to merge the components that have values for A and B within a same tuple of the decomposed worlds. After the composition phase, the selection algorithm follows the pattern of the selection with constant.

[2] Of course the tuples over P can be different – even their number may vary – in each of the possible worlds over P. However, since world-set relations, and therefore their decompositions, reserve a slot for the same $|P|_{max}$ tuples in each world, and just some worlds may have some of these tuples marked as invalid, we may just as well refer to a tuple of P as an object that has a different *value*, and in some cases $\perp$, in each possible world.

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

×

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

×

R.t_2.A
4
5

×

R.t_2.C
0

×

R.t_3.A
6

×

R.t_3.B
6

×

R.t_3.C
7

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

Figure 11: World-set and its decomposition.

P.t_1.A
1
2

×

P.t_1.B	P.t_1.C	P.t_2.B
⊥	⊥	3
2	7	4

×

P.t_2.A
4
5

×

P.t_2.C
⊥

×

P.t_3.A
6

×

P.t_3.B
6

×

P.t_3.C
7

(a) $P := \sigma_{C=7}(R)$ applied to the WSD of Figure 11 (b).

P.t_1.A
1
2

×

P.t_1.B	P.t_1.C	P.t_2.B
1	0	⊥
⊥	⊥	⊥

×

P.t_2.A
4
5

×

P.t_2.C
0

×

P.t_3.A
6

×

P.t_3.B
⊥

×

P.t_3.C
7

(b) $P := \sigma_{B=1}(R)$ applied to the WSD of Figure 11 (b).

Figure 12: Selections $P := \sigma_{C=7}(R)$ and $P := \sigma_{B=1}(R)$ with R from Figure 11 (b).

P.t_1.A	P.t_1.B	P.t_1.C	P.t_2.B
1	1	0	3
⊥	⊥	⊥	3
⊥	⊥	⊥	4
2	2	7	4

×

P.t_2.A
4
5

×

P.t_2.C
0

×

P.t_3.A
6

×

P.t_3.B
6

×

P.t_3.C
7

(a) 6-WSD after filtering tuple t_1 using condition $A = B$.

P.t_1.A	P.t_1.B	P.t_1.C	P.t_2.A	P.t_2.B
1	1	0	⊥	⊥
⊥	⊥	⊥	⊥	⊥
⊥	⊥	⊥	4	4
2	2	7	4	4
2	2	7	⊥	⊥

×

P.t_2.C
0

×

P.t_3.A
6

×

P.t_3.B
6

×

P.t_3.C
7

(b) 5-WSD after filtering tuples t_1, t_2 using condition $A = B$.

P.t_1.A	P.t_1.B	P.t_1.C	P.t_2.A	P.t_2.B
1	1	0	⊥	⊥
⊥	⊥	⊥	⊥	⊥
⊥	⊥	⊥	4	4
2	2	7	4	4
2	2	7	⊥	⊥

×

P.t_2.C
0

×

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

×

P.t_3.C
7

(c) 5-WSD after filtering all tuples using condition $A = B$

Figure 14: $P = \sigma_{A=B}(R)$ with R from Figure 11 (b).

R.t_1.A
1
2

×

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

×

R.t_2.B
7
8

×

S.t_1.C
a
b

×

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

×

S.t_2.D
g
h

(a) WSD of two relations R and S.

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

×

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

×

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

×

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

×

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

×

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

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

Figure 15: The product operation $R \times S$.

EXAMPLE 4.5. Consider the query $\sigma_{A=B}(R)$, where R is represented by the 7-WSD of Figure 11 (b). Figure 14 (a) shows the 6-WSD obtained from the one of Figure 11 (b) after tuple t_1 has been processed (requiring the composition of the first and second component of the WSD of Figure 11 (b)). Further applying the selection to tuple t_2 yields the 5-WSD of Figure 14 (b). Finally, processing tuple t_3 leads to the composition of the second and the third components, as shown in Figure 14 (c). This 4-WSD represents five worlds, where one world has three tuples, three worlds have two tuples each, and one world has one tuple. □

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 4.6. Consider the 3-WSD of Figure 16 (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 $\perp$-values within the same tuple, and (3) project away the irrelevant attributes. The correct answer P is given in Figure 16 (b).

Note that despite the component merging done by the projection, the size of the answer does not grow exponentially, because of attribute pruning and propagation of $\perp$-values.□

R.t_1.A
a

×

R.t_2.A
b

×

R.t_1.B	R.t_2.B
c	$\perp$
$\perp$	d

(a) WSD for relation R.

P.t_1.A	P.t_2.A
a	$\perp$
$\perp$	b

(b) P := $\pi_A(R)$.

Figure 16: Projection $P := \pi_A(R)$.

The algorithm for projection is given in Figure 10. For each tuple t_i, attribute A in the projection list, and attribute B not in the projection list, the algorithm first propagates the $\perp$-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 all tuples (i.e., worlds) of C' have at least one $\perp$-value, i.e., there is no world with values for all attributes of C'. 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.

Product. The product $T := R \times S$ of two relations R and S, which have disjunct attribute sets and are represented by a WSD $\mathcal{C}$ 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 4.7. Figure 15 (b) shows the WSD for the product of relations R and S represented by the WSD of Figure 15 (a). To save space, the relations R and S have been removed from Figure 15 (b), and attribute names do not show the relation name "T". □

Union, Difference, and Renaming. The algorithm of Figure 10 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.

Two operations of relational algebra remain to be discussed, operator renaming δ and difference "$-$". The operation $\delta_{A\rightarrow A'}(R)$, which renames attribute A or relation R to A', is very easy to implement. For each tuple t of R, let C be the component that has the attribute $R.t.A$. Then we rename this attribute to $R.t.A'$ as $C := \delta_{R.t.A\rightarrow R.t.A'}(C)$.

The difference operation $R - S$ may require to merge all components that handle fields of R or S and is thus by far the least efficient to implement. However, this is a known problem of strong representation systems for relational algebra. For example, it is known that evaluating the difference operation on c-tables can take exponential time [9].

4.3 Normalizing WSDs

The algorithms of Figure 10 can produce WSDs, which are not maximal, even if their input is. Recall from Section 3 that each WSD admits one maximal equivalent decomposition, which takes the least space among all its equivalents. Also, due to propagation of $\perp$-values, the components can encode tuples invalid in all worlds. Figure 17 gives two algorithms that address these normalization problems.

```
algorithm remove_invalid_tuples
begin
  for each $1 \le i \le |P|_{max}$ and $A \in \mathcal{S}(P)$ do begin
    let $C$ be the component of $P.t_i.A$;
    if $\pi_{P.t_i.A} = \{\perp\}$ then
      for each $B \in \mathcal{S}(P)$ 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 $C = C_1 \times C_2$;
    replace $C$ by $C_1$, $C_2$;
  end
end
```

Figure 17: Algorithms for WSD normalization.

Example 4.8. The WSD of Figure 12 (a) has only $\perp$-values for $P.t_2.C$. This can be interpreted as the absence of tuple t_2 of P from all worlds, and allows us to remove the entries of that tuple from all components. Applied to this WSD, the algorithm remove_invalid_tuples yields the equivalent WSD of Figure 18. Similar simplifications apply to the WSD of Figure 12 (b), where tuples t_2 and t_3 are invalid. □

Example 4.9. The 4-WSD of Figure 14 (c) admits the equivalent 5-WSD of Figure 14 (b), where the second component is decomposed into two components. Such a case of non-maximality is detected by the algorithm decompose. □

$$\begin{array}{|c|}\hline P.t_1.A \\ \hline 1 \\ 2 \\ \hline\end{array} \times \begin{array}{|cc|}\hline P.t_1.B & P.t_1.C \\ \hline \perp & \perp \\ 2 & 7 \\ \hline\end{array} \times \begin{array}{|c|}\hline P.t_3.A \\ \hline 6 \\ \hline\end{array} \times \begin{array}{|c|}\hline P.t_3.B \\ \hline 6 \\ \hline\end{array} \times \begin{array}{|c|}\hline P.t_3.C \\ \hline 7 \\ \hline\end{array}$$

Figure 18: Normalization for WSD of Figure 12 (a).

REMARK 4.10. Note that the non-maximality case of the WSD from Figure 14 (c) cannot appear for UWSDTs, because all but the first component of Figure 14 (c) contain only one tuple and are stored in the template relation, where no component merging occur.

Clearly, the removal of invalid tuples needs polynomial time in the size of the WSD. Less obvious, finding the maximal equivalent decomposition of a component with *arbitrary* arity can also be done in polynomial time [5]. □

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 $\perp$ 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. Due to space limitations, we only give our efficient implementation of the selection with constant in Figure 19.

In contrast to some algorithms of Figure 10, 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 7, 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 19 only once. We further tuned the query evaluation by employing indices and materializing often used temporary results.

```
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. $P^0 := P^0 - \{t \mid t \in P^0, \nexists B,a : (P.t.B,a) \in F\}$;
  7. $W := \pi_{cid,lwid}(F \bowtie C)$;
end
```

Figure 19: Evaluating $P := \sigma_{A\theta c}(R)$ on UWSDTs.

6. CHASING DEPENDENCIES

In this section we address the problem of removing inconsistent worlds in an incompletely specified database. We present a method called *Chase* [3, 12, 2] in the spirit of the work of [8] 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 $A_1, \ldots, A_m \rightarrow A_0$, where $A_i, \in S(R), 0 \leq i \leq m$, and *equality-generating dependencies* of the form $\phi_1 \wedge \phi_2 \wedge ... \wedge \phi_m \Rightarrow \phi_0$, where each $\phi_i(A_i) = A_i\theta_i c_i$, for $0 \leq i \leq m$, is a binary operator comparing the value of an attribute $A_i \in S(R)$ with a constant c_i. Relation R satisfies an equality-generating dependency egd (denoted by $R \models egd$) if for each tuple $t \in R : t.A_1\theta_1 c_1 \wedge \ldots \wedge t.A_m\theta_k c_m \Rightarrow t.A_0\theta_0 c_0$.

Recall Example 1.2 from the introduction. The uniqueness constraint for the social security number is a functional dependency $S \rightarrow N, M$, which is of course 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 5).

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. As a result of this data-cleaning step we obtain the 4-WSD in Figure 20.

One would probably observe that enforcing a dependency on a WSD resembles the selection operation with condition $A\theta B$ presented in Section 4. In both cases we identify depen-

$$\begin{array}{|ccc|}\hline t_1.\text{S} & t_2.\text{S} & t_1.M \\ \hline 185 & 186 & 1 \\ 185 & 186 & 2 \\ 785 & 185 & 1 \\ 785 & 186 & 1 \\ \hline \end{array} \times \begin{array}{|c|}\hline t_1.\text{N} \\ \hline \text{Smith} \\ \hline \end{array} \times \begin{array}{|c|}\hline t_2.\text{N} \\ \hline \text{Brown} \\ \hline \end{array} \times \begin{array}{|c|}\hline t_2.\text{M} \\ \hline 1 \\ 2 \\ 3 \\ 4 \\ \hline \end{array}$$

Figure 20: Result of chasing $S = 785 \Rightarrow M = 1$ on the WSD in Figure 5.

```
algorithm Chase
Input: $\Phi$: set of dependencies, $\mathcal{W}$: $WSD$
Output: a $WSD$ satisfying $\Phi$

begin
  for each $d \in \Phi$ do
    if $d = A_1, \ldots, A_m \to A_0$ then // d is a fd on $\mathcal{W}$
      for each $s, t \in \mathcal{W} : \{s, t\} \not\models d$ do begin
        let $C_{j_i}, C_{k_i}$ be the component of $s.A_i, t.A_i$,
          respectively, for each $0 \leq i \leq m$;
        replace $C_{j_0}, \ldots, C_{j_m}, C_{k_0}, \ldots, C_{k_m}$ in $\mathcal{W}$
          by their product $C$;
        $C := \sigma_d(C)$;
        if $C = \emptyset$ then return $\emptyset$;
      end
    else if $d = \phi_1 \wedge \ldots \wedge \phi_m \to \phi_0$ // d is a egd on $\mathcal{W}$
      for each $t \in \mathcal{W} : \{t\} \not\models d$ do begin
        let $C_i$ be the component of $t.A_i$, for $0 \leq i \leq m$;
        replace $C_0, \ldots, C_m$ in $\mathcal{W}$ by their product $C$;
        $C := \sigma_d(C)$;
        if $C = \emptyset$ then return $\emptyset$;
      end;
  return $\mathcal{W}$;
end.
```

Figure 21: Chase

dencies across components and compose dependent components. Nevertheless there is an important difference between the two operations. In the selection operation we are interested in finding *a subset of the tuples* valid in *some world*. If a tuple fails to satisfy the selection condition, it is simply dropped from the result. On the other hand, when enforcing dependencies on a WSD, we want to get *a 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. Repeating these two steps iteratively for each dependency and each tuple in the given WSD $\mathcal{W}$ would result in a WSD $\mathcal{W}$' satisfying all constraints.

Before proceeding to the formal algorithm for chasing dependencies, we introduce the following notations.

If $fd = A_1, \ldots, A_m \to A_0$ is a functional dependency, $s, t \in \mathcal{W}$ and all attributes $s.A_i, t.A_i, 0 \leq i \leq m$ are defined in a component C, the operation that retrieves all worlds in C satisfying fd can be expressed by

$\sigma_{C.(s.A_1) \neq C.(t.A_1) \vee \ldots \vee C.(s.A_m) \neq C.(t.A_m) \vee C.(s.A_0) = C.(t.A_0)}(C)$, where σ is the relational selection. For the sake of brevity we write $\sigma_{fd}(C)$. Similarly, if t is a tuple from $\mathcal{W}$, $egd = A_1\theta_1c_1 \wedge ... \wedge A_m\theta_mc_m \Rightarrow A_0\theta_0c_0$ is an equality-generating dependency and all attributes $t.A_i, 0 \leq i \leq m$ are defined in a component C, the operation

$\sigma_{\neg(C.(t.A_1)\theta_1c_1) \vee \ldots \vee \neg(C.(t.A_m)\theta_mc_m) \vee (C.(t.A_0)\theta_0c_0)}(C)$ retrieves all valid worlds (with respect to egd) from C. We write shortly $\sigma_{egd}(C)$.

The algorithm of Figure 21 implements the data cleaning for a given world-set decomposition and a set of dependencies Φ.

THEOREM 6.1. *The Chase algorithm of Figure 21 terminates on all inputs.*

THEOREM 6.2 (CORRECTNESS). *For a WSD $\mathcal{W}$ and a set of dependencies Φ, the algorithm of Figure 21 computes a WSD $\mathcal{W}'$ s.t. $rep(\mathcal{W}') \subseteq rep(\mathcal{W})$ and for each $\mathcal{A} \in rep(\mathcal{W})$,*

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

7. 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 store a large set of possible worlds 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. We introduced UWSDTs and studied the query processing problem in this context. This section reports on experiments with a large census database with noise.

Experimental Setting. The experiments were conducted on a 3GHz/2GB Pentium machine running Linux 2.6.8 and PostgreSQL 8.0.

Datasets. The IPUMS 5% census data (Integrated Public Use Microdata Series, 1990) [14] 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, 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. We experimented with different noise densities: 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). Note that in one scenario we had far more than 2^{624449} possible worlds, where 624449 is the number of the introduced or-sets and 2 is the minimal size of each or-set (cf. Figure 23).

Data Cleaning. We enhance our Chase algorithm of Section 6 in several ways. For a given dependency we retrieve all inconsistencies at once instead of doing that tuplewise, and we perform as many updates as possible at the same time, instead of having one update per inconsistent tuple.

This is in the spirit of the semi-naive algorithm for datalog evaluation [2].

The Chase uses the 12 equality-generating dependencies from Figure 22. These represent real-life constraints on the data. For example the first dependency states that citizens born in the USA are not immigrants, and dependencies two to five require that citizens who served in various wars have done their military service. Note that relations with or-sets are not expressive enough to represent the cleaned data with dependencies.

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

Figure 22: Dependencies for cleaning census data.

Figure 23 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.

	Density	0.005%	0.01%	0.05%	0.1%
Initial	#comp	31117	62331	312730	624449
After chase	#comp	30918	61791	309778	612956
	#comp>1	249	522	2843	10880
	$\|C\|$	108276	217013	1089359	2150935
	$\|R\|$	12.5M	12.5M	12.5M	12.5M
After Q_1	#comp	702	1354	7368	14244
	#comp>1	1	4	40	158
	$\|C\|$	1742	3625	19773	37870
	$\|R\|$	46600	46794	48465	50499
After Q_2	#comp	25	56	312	466
	#comp>1	0	1	8	9
	$\|C\|$	93	269	1682	2277
	$\|R\|$	82995	83052	83357	83610
After Q_3	#comp	38	76	370	742
	#comp>1	0	0	0	0
	$\|C\|$	89	202	1001	2009
	$\|R\|$	17912	17936	18161	18458
After Q_4	#comp	1574	3034	15776	30729
	#comp>1	11	28	127	557
	$\|C\|$	4689	9292	48183	94409
	$\|R\|$	402345	402524	404043	405869
After Q_5	#comp	3	10	53	93
	#comp>1	3	10	53	93
	$\|C\|$	1221	5263	33138	50780
	$\|R\|$	150604	173094	274116	393396
After Q_6	#comp	97	189	900	1888
	#comp>1	0	0	0	0
	$\|C\|$	516	1041	4993	10182
	$\|R\|$	229534	230113	234335	239488

Figure 23: UWSDTs characteristics before chasing and after chasing and querying for 12.5M tuples.

Queries. Six queries were chosen to show the behavior of relational operators combinations under varying selectivities (cf. Figure 24). 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 23 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 25 shows that all six queries admit efficient and scalable evaluation on UWSDTs of different sizes and placeholder densities. For accuracy, each query was run ten times, and the median time for computing and storing the answer is reported.

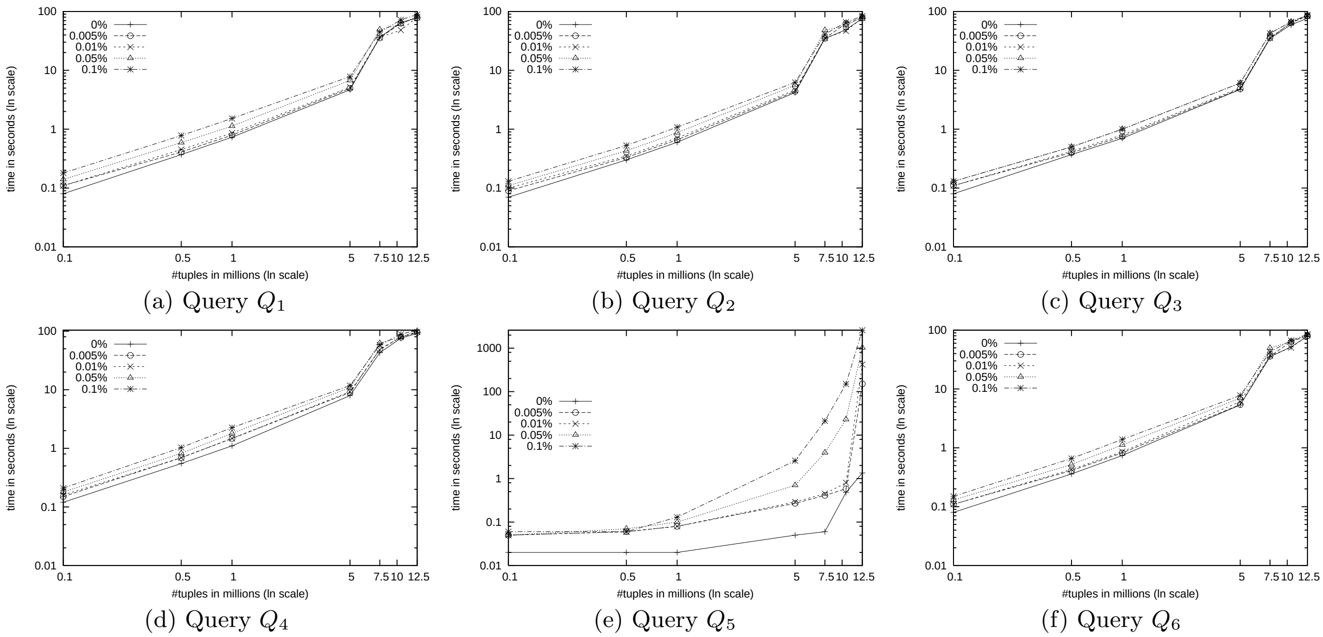
The evaluation time for all queries but Q_5 on UWSDTs follows very closely the evaluation time in the one-world case. The one-world case corresponds to density 0% in our diagrams, i.e., when no placeholders are created in the template relation and consequently there are no components. In this case, the queries of Figure 24 were evaluated only on the template relation.

An interesting issue is that all diagrams of Figure 25 show a substantial increase in the query evaluation time for the 7.5M case. As the jump appears also in the one-world case, it suggests poor memory management of Postgres in the case of large tables. We verified this statement by splitting the 12.5M table into chunks smaller than 5M and running query Q_1 on those chunks to get partial answers. The final answer is represented then by the union of each UWSDT relation from these partial answers.

Although the evaluation of join conditions on UWSDTs can require theoretically exponential time (due to the com-

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

Figure 24: Queries on IPUMS census data.



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

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

Figure 25: The evaluation time for queries of Figure 24 on UWSDTs of various sizes and densities.

position of some decomposed components), our experiments suggest that they behave well in practical cases, as illustrated in Figures 25 (c) and (e) for queries Q_3 and Q_5 respectively. Note that the time reported for Q_5 does not include the time to evaluate its subqueries Q_2 and Q_3.

In summary, our experiments show that UWSDTs behave very well in practice. 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.

8. 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 7, 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 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 [6]. 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 [6] 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.

9. RELATED WORK

Recent years have witnessed important contributions towards scalable methods for managing incompleteness in relational databases. This section compares expressiveness and processing aspects of [7, 4, 15] with the WSD approach.

Tuple independence. Dalvi and Suciu propose probabilistic databases to model data incompleteness [7]. Each tuple in a probabilistic databases is assigned a confidence value, which represents the probability of the tuple being in the database. Dalvi and Suciu make the strong assumption that each tuple is an independent probabilistic event, thus their model cannot represent any finite world-set as WSDs can. Rather, a possible world is a subset of the tuples in the database and its probability is computed as the product of probabilities of the tuples in the world and of the ones that are not.

S	A	B	
s_1	m	1	0.8
s_2	n	1	0.5

T	C	D	
t_1	1	p	0.6

Figure 26: A probabilistic database.

Figure 26 shows an example of a probabilistic database D used in [7] with two relations S and T. The possible worlds for D are given in Figure 27.

world	
$D_1 = \{s_1, s_2, t_1\}$	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

Figure 27: The possible worlds of the probabilistic database in Figure 26.

A natural probabilistic extension of WSDs can model the probabilistic databases of Dalvi and Suciu in the following way. Each tuple t with confidence c in a probabilistic database induces a WSD component representing two local worlds: the local world with tuple t and confidence c, and the empty world with confidence $1 - c$. Such a probabilistic extension would further allow to assign probabilities not only to individual tuples (as proposed by Dalvi and Suciu), but also to *dependencies* of values across tuples, as defined within WSD components (and not possible under the tuple independence assumption). Figure 28 gives the WSD encoding of the probabilistic database of Figure 26.

An important contribution of [7] is the evaluation of rank queries under the strong assumption of tuple independence. This is especially challenging because in general query evaluation produces dependencies between tuples and complicates rank computation. In contrast to this, WSDs can represent uniformly tuple dependency and independency. In a follow-up work [13] the probabilistic databases are extended to represent (intentionally) complex correlated events as SQL views.

Or-set independence. Or-set relations [11] are a simple formalism for representing incompleteness by allowing relation fields to contain finite sets of values, called or-sets. Each value in an or-set is a possible assignment for the respective field and each combination of assignments for the different fields yields a possible world. As or-set relations exhibit an independence of the contained or-sets, they are too weak to represent answers even to simple selections, cf. the discussion in Section 1. In contrast, WSDs can represent *any* world-set, while being as succinct as or-set relations for the case of world-sets representable as or-set relations. Also, in such cases, processing WSDs is as efficient as processing or-set relations.

Tuple-set independence. Miller et al. [4] study the problem of finding clean answers over dirty databases. A relation in a dirty database is defined here as a set of independent tuples, where each tuple has a set of (mutually exclusive, thus dependent) alternatives. Each alternative is assigned a probability of being in the clean database. Thus one possible instance of a dirty database contains exactly one alternative of each tuple.

In contrast to or-set relations, Miller's representation system contain alternatives for whole tuples rather than for single tuple fields only. Its salient weakness is the inability (1) to efficiently represent alternatives for single tuple fields (as or-set relations and WSDs do), and (2) to represent alternatives for sets of fields across several tuples (as WSDs do). To see the former point, consider an or-set relation with one tuple, which has m or-sets, each with n values. The number of alternatives of the same tuple is then n^m, i.e, the number of possible worlds. As a concrete example, Figure 29 gives the encoding of the or-set relation from Figure 1 in Miller's representation system.

(TID)	SSN	Name	Marital
t_1	185	Smith	1
t_1	185	Smith	2
t_1	785	Smith	1
t_1	785	Smith	2
t_2	185	Brown	1
t_2	185	Brown	2
t_2	185	Brown	3
t_2	185	Brown	4
t_2	186	Brown	1
t_2	186	Brown	2
t_2	186	Brown	3
t_2	186	Brown	4

Figure 29: The or-set relation of Figure 1 encoded as a dirty database.

The clean answer of a query is defined as the set of tuples that are possible answers of that query, together with their probabilities. Miller et al. investigate the query evaluation problem for the restricted class of queries with acyclic join graphs.

Tuple-set independence can be easily modeled in WSDs by having a component for each set of alternatives (which become component's tuples). For example, the above example can be represented compactly as the WSD in Figure 4.

Note that the or-set, tuple, and tuple-set independence assumptions discussed above drastically simplify the query evaluation on WSDs, because each tuple of a database relation becomes now a tuple in a component. Component merging, the most expensive operation on WSDs, is thus only needed when joining independent relations and not needed for relational algebra operations on a single relation.

Strong Representation Systems. Recent work by Widom

S.s_1.A	S.s_1.B			S.s_2.A	S.s_2.B			T.t_1.C	T.t_1.D	
m	1	0.8	×	n	1	0.5	×	1	p	0.6
$\perp$	$\perp$	0.2		$\perp$	$\perp$	0.5		$\perp$	$\perp$	0.4

Figure 28: WSD equivalent to the probabilistic database in Figure 26.

et al. on uncertain information and lineage [18, 15] proposes a system called Trio for managing incomplete data. Out of a plethora of so-called working models of varying expressiveness, Trio has a complete model (called TCM), i.e., TCM can represent any finite world-set. Because this is also the goal of WSDs, their work and ours inherently share some similarities, though they differ in important theoretical and practical aspects.

TCM lies at the intersection of the c-tables of [10] and the probabilistic databases of [13]. More precisely, in TCM the worlds of a relation are represented as a table containing all tuples of that relation as independent events. Additionally, TCM allows to express complex tuple events using propositional formulas over tuple identifiers. As discussed in the introduction, WSDs can be seen as a normal form for c-tables, and thus for TCM instances.

As shown in [15], various decision problems like tuple possibility and certainty are NP-hard for TCM, which makes TCM as unrealistic for practical purposes as c-tables are. In contrast, in a current work [5] we show that both these problems admit polynomial decision in the context of WSDs.

To date, the scalability of WSDs and Trio could not be compared in practice, for Trio is still being built (cf. [15]).

Infinite world-sets. So far work on representing infinite world-sets has been mainly theoretical. Formalisms include v-tables, where fields can be occupied by variables, and c-tables, where in addition to variables one can specify logical constraints on their values. In [5] we propose an extension of WSDs that adds support for infinite world-sets. This extension makes WSDs equivalent in expressive power to c-tables and bears the promise of more efficient data management and query processing.

10. FUTURE WORK

While first experiments suggest that WSDs are a very promising framework for representing and managing incomplete information, we are currently working on evaluating WSDs in applications other than survey data.

We have started the analysis of the data cleaning problem in the context of UWSDTs and our experiments make use of partially cleaned data. In the future, we will study the data cleaning problem within our framework in greater depth.

Although in many application scenarios, e.g. census, the data does not change over time, we intend to address the problem of making *updates* to WSDs in the future. Note that updates are easy to handle in the case when changes made are the same in all worlds. For instance, adding a tuple to each world of a UWSDT just means to add the tuple once to the template relation. However, updating world-sets with incomplete information – even defining meaningful notions of this – is an interesting problem for future research.

We are currently working on generalizing the framework of WSDs to support also infinite sets of possible worlds by taking an approach that integrates WSDs with c-tables. Our WSDs correspond to a certain CNF-like *normal form* for the condition formulas in c-tables. In the future, we would like to elaborate on this to define syntactic normal forms for c-tables which are at the same time powerful as representation systems (for finite as well as infinite world-sets) and allow for scalable representation, data cleaning, and query processing.

11. REFERENCES

[1] http://www.medicinenet.com.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[3] A. V. Aho, C. Beeri, and J. D. Ullman. “The Theory of Joins in Relational Databases”. *ACM Trans. Database Syst.*, **4**(3):297–314, 1979.

[4] P. Andritsos, A. Fuxman, and R. J. Miller. Clean answers over dirty databases: A probabilistic approach. In *Proc. ICDE*, page 30, 2006.

[5] L. Antova, C. Koch, and D. Olteanu. World-set decompositions: Expressiveness and efficient algorithms, 2006. Unpublished manuscript available from the authors.

[6] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *Proc. PODS*, pages 68–79, 1999.

[7] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *Proc. VLDB*, pages 864–875, 2004.

[8] G. Grahne. Dependency satisfaction in databases with incomplete information. In *Proc. VLDB*, pages 37–45, 1984.

[9] G. Grahne. *The Problem of Incomplete Information in Relational Databases*. Number 554 in LNCS. Springer-Verlag, 1991.

[10] T. Imielinski and W. Lipski. Incomplete information in relational databases. *Journal of ACM*, 31:761–791, 1984.

[11] T. Imielinski, S. Naqvi, and K. Vadaparty. Incomplete objects — a data model for design and planning applications. In *Proc. SIGMOD*, pages 288–297, 1991.

[12] D. Maier, A. O. Mendelzon, and Y. Sagiv. “Testing Implications of Data Dependencies”. *ACM Trans. Database Syst.*, **4**(4):455–469, 1979.

[13] C. Re, N. Dalvi, and D. Suciu. Probabilistic databases: Where and how. Technical report, University of Washington, 2005.

[14] S. Ruggles, M. Sobek, T. Alexander, C. A. Fitch, R. Goeken, P. K. Hall, M. King, and C. Ronnander. Integrated public use microdata series: V3.0, 2004.

[15] A. D. Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working models for uncertain data. In *Proc. ICDE*, 2006. initially published as Stanford Technical Report, Feb. 2005.

[16] J. D. Ullman. *Principles of Database & Knowledge-Base Systems Vol. 2: The New Technologies*. Computer Science Press, 1989.

[17] M. Y. Vardi. “The Complexity of Relational Query Languages”. In *Proc. STOC*, pages 137–146, 1982.

[18] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *Proc. CIDR*, 2005. initially published as Stanford Technical Report, Aug. 2004.