# of Uncertain Data

Lyublena Antova, Thomas Jansen, Christoph Koch, and Dan Olteanu

Saarland University Database Group
Saarbrücken, Germany

{lublena, jansen, koch, olteanu}@infosys.uni-sb.de

Abstract— **This paper introduces U-relations, a succinct and purely relational representation system for uncertain databases. U-relations support attribute-level uncertainty using vertical partitioning. If we consider positive relational algebra extended by an operation for computing possible answers, a query on the logical level can be translated into, and evaluated as, a single relational algebra query on the U-relational representation. The translation scheme essentially preserves the size of the query in terms of number of operations and, in particular, number of joins. Standard techniques employed in off-the-shelf relational database management systems are effective for optimizing and processing queries on U-relations. In our experiments we show that query evaluation on U-relations scales to large amounts of data with high degrees of uncertainty.**

I. INTRODUCTION

Several recent works [10], [9], [8], [2], [15], [4], [6] aim at developing scalable representation systems and query processing techniques for large collections of uncertain data as they arise in data cleaning, Web data management, and scientific databases. Most of them are based on a possible worlds semantics, and for all of them such a semantics can be conveniently defined.

Four desiderata for representation systems for incomplete information appear important.

1. Expressiveness. The representation should be closed under the application of (relational algebra) queries and data cleaning algorithms (which remove some possible worlds). That is, the results of applying such operations to the represented data should be again representable within the formalism.

2. Succinctness. It should be possible to represent large sets of alternative worlds using fairly little space.

3. Efficient query evaluation. A trade-off is required between the succinctness of a representation formalism and the complexity of evaluating interesting queries. This trade-off follows from established theoretical results [1], [11], [6]. However, while the formalisms in the literature tend to differ in succinctness, several have polynomial-time data complexity for (decision) problems such as tuple possibility under *positive* (but not full) relational algebra. This includes v-tables [12], [11], uncertainty-lineage databases (ULDBs) [8], and world-set decompositions (WSDs) [6].

4. Ease of use for developers and researchers in the sense that the representation system can be easily put on top of a relational DBMS. This in particular includes that queries on the logical schema level can be translated down to, ideally, relational algebra queries on the representation relations and that this translation is simple and easy to implement. This goal is motivated by the availability and maturity of existing relational database technology.

An important aspect of a representation system is whether it represents uncertainty at the *attribute-level* or at the *tuple-level*. Attribute-level representation refers to the succinct representation of relations in which two or more fields of the same tuple can independently take alternative values [6]. Attribute-level representation, as supported by c-tables [12] and WSDs, offers finer granularity of independence than tuple-level approaches like [8], [10], [2]. This is useful in applications like data cleaning, where the values of several fields of a single tuple can be independently uncertain. For instance, the US Census Bureau maintains relations with dozens of columns (> 50), most of which may require cleaning [4].

U-relations. In this paper, we develop and study *U-relations*, a representation system that we introduce with the following example.

Example I.1. Let us assume that an aerial photograph of a battlefield shows four vehicles at distinct positions 1 to 4. The resolution of the image does not allow for the identification of vehicle types, but we can draw certain conclusions from earlier reconnaissance and a calculation of the maximum distance each vehicle may have covered since. Say we know that vehicle 1 is (a) a friendly tank. Vehicles 2 and 3 are (b) a friendly transport and (c) an enemy tank, but we do not know which one is which. Nothing is known about vehicle 4. Fig. 1a shows a schematic drawing of how this scenario can arise. Only 1 is in the range of (a); 2 and 3 are in the ranges of (b) and (c); and position 4 is near the border of the photograph but outside the ranges of (a), (b), and (c), so this vehicle must have newly moved onto the map.

We want to model this by an uncertain database of schema R(Id, Coord, Type, Faction), representing the ids (1–4), coordinate positions, types, and factions of the vehicles on the map. Let us assume there are only two vehicle types (tank or transport) and two factions (friend or enemy). Then there are eight possible worlds. We obtain one by taking three choices –

W	Var	Rng
	x	1
	x	2
	y	1
	y	2
	z	1
	z	2

U_1	D	T_R	Id
		a	1
	$x \mapsto 1$	b	2
	$x \mapsto 2$	b	3
	$x \mapsto 1$	c	3
	$x \mapsto 2$	c	2
		d	4

U_2	D	T_R	Type
		a	Tank
		b	Transport
		c	Tank
	$y \mapsto 1$	d	Tank
	$y \mapsto 2$	d	Transport

U_3	D	T_R	Faction
		a	Friend
		b	Friend
		c	Enemy
	$z \mapsto 1$	d	Friend
	$z \mapsto 2$	d	Enemy

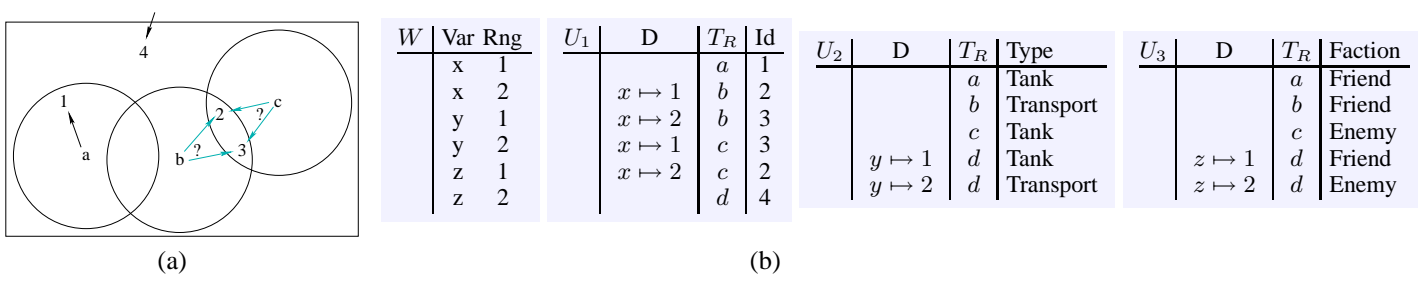(a)   (b)

Fig. 1.   Map with moving vehicles (a) and U-relational database representation of the possible worlds at the time the aerial photograph detecting vehicles 1,2,3,4 was taken (b).

answering the following questions: Has the friendly transport (b) now become vehicle 2 ($x \mapsto 1$) or 3 ($x \mapsto 2$)? Is vehicle 4 a tank ($y \mapsto 1$) or a transport ($y \mapsto 2$)? Is vehicle 4 friendly ($z \mapsto 1$) or an enemy ($z \mapsto 2$)? Thus the uncertainty can be naturally modelled using three variables x, y, z that each can independently take one of two values.

We model this scenario by the U-relational database shown in Fig. 1b. We use vertical partitioning (cf. e.g. [7], [16]) to achieve attribute-level representation. R is represented using four U-relations, one for each column of R. The U-relation for the coordinate positions (which are all certain) is not shown since we do not want to use it subsequently, but of course, conceptually, coordinate positions are an important feature of the example and have to be part of the schema. In addition there is a relation W which defines the possible values the three variables can take.

We can compute a vertical decomposition of one world given by a valuation θ of the variables x, y, z by (1) removing all the tuples from the U-relations whose D columns contain assignments that are inconsistent with θ (For example, if $\theta = \{x \mapsto 1, y \mapsto 1, z \mapsto 1\}$ then we remove the third and fifth tuples of U_1 and the fifth tuples of U_2 and U_3.) and then (2) projecting the D columns away. Of course we can resolve the vertical partitioning by joining the decomposed relations on the tuple id columns T_R. $\square$

U-relations have the following properties:

- **Expressiveness**: U-relations are *complete* for finite sets of possible worlds, that is, they allow for the representation of any finite world-set.
- **Succinctness**: U-relations represent uncertainty on the attribute level. Even though they allow for more efficient query evaluation, U-relations are, as we show, exponentially more succinct than ULDBs and WSDs. That is, there are (relevant) world-sets that necessarily take exponentially more space to represent by ULDBs or WSDs than by U-relations.
- **Leveraging RDBMS technology**: U-relations allow for a large class of queries (positive relational algebra extended by the operation "possible") to be processed *using relational algebra only*, and thus efficiently in the size of the data. Our approach is the first so far to achieve this for the above-named query language. Indeed, this not only settles that there is a succinct and complete *attribute-level* representation for which the so-called tuple

Q-possibility problem for positive relational algebra is in polynomial time (previously open [6]) but puts a rich body of research results and technology at our disposal for building uncertain database systems. This makes U-relations the most efficient and scalable approach to managing uncertain databases to date.

- **Parsimonious translation**: The translation from relational algebra expressions on the logical schema level to query plans on the physical representations replaces a selection by a selection, a projection by a projection, a join by a join (however, with a more intricate join condition), and a "possible" operation by a projection. We have observed that state-of-the-art RDBMS do well at finding efficient query plans for such physical-level queries.

Ease of use: A main strength of U-relations is their simplicity and low "cost of ownership":

- The representation system is purely relational and in close analogy with relational representation schemes for vertically decomposed data. Apart from the column store relations that represent the actual data, there is only a single auxiliary relation W (which we need for computing certain answers, but not for possible answers).
- Query evaluation can be fully expressed in relational algebra. The translation is quite simple and can even be done by hand, at least for moderately-sized queries.
- The query plans obtained by our translation scheme are usually handled well by the query optimizers of off-the-shelf relational DBMS, so the implementation of special operators and optimizer extensions is not strictly needed for acceptable performance.

Thus U-relations are not only suited as a representation system for dedicated uncertain database implementations such as MayBMS [4], but are also relevant to "casual users" of representation systems for uncertain data, such as researchers in data cleaning and data integration who want to store and query uncertain data without great effort.

Apart from those implicitly mentioned above, we make the following further contributions in this paper.

- We study algebraic query optimization and present equivalences that hold on vertically decomposed representations. We address query optimization using those in the context of managing uncertainty with U-relations.
- We present an algorithm for normalizing a U-relational representation obtained from a query. Normalized U-

relational databases yield a conceptually simple algorithm for computing the certain answers of queries. In particular, certain answer tuples on normalized tuple-level representations can be computed using relational algebra only, which is not true in general for previous representation systems.

- We provide experimental evidence for the efficiency and relevance of our approach.

The structure of the paper is as follows. Section II establishes U-relations formally. Section III presents our reduction from queries on the logical level to relational algebra on the level of U-relations and addresses algebraic query evaluation. Section IV presents the normalization algorithm. Section V discusses the relationship between U-relations, WSDs and ULDBs and argues that U-relations combine the advantages of the other two formalisms without sharing their drawbacks. Section VI describes how probabilistic information can be modelled using a natural extension of U-relations. In Section VII, we report on our experiments with U-relations. We conclude with Section VIII.

II. U-RELATIONAL DATABASES

We define *world-sets* in close analogy to the case of c-tables [12]. Consider a finite set of variables over finite domains. A *possible world* is represented by a total valuation (or assignment) f : Var $\mapsto$ Rng of variables to constants in their domains, and the world-set is represented by the finite set of all total valuations[1]. We represent relationally the variable set and the associated domains by a *world-table* over schema $W(\text{Var,Rng})$ such that W consists of all pairs (x, v) of variables x and values v in the domain of x.

Example II.1. The world-table W in Fig. 1 defines three variables x, y, z, whose common domain is $\{1, 2\}$. The number of worlds defined by W is $2 \cdot 2 \cdot 2 = 8$. □

Given a world-table W, a *world-set descriptor* over W, or ws-descriptor for short, is a valuation $\overline{d}$ such that its graph is a subset of W. If $\overline{d}$ is a *total* valuation, then it represents one world. In our examples, to represent the entire world-set we use an *empty* ws-descriptor, as a shortcut for a singleton ws-descriptor with a new variable with a singleton domain.

We are now ready to define databases of U-relations.

Definition II.2. A *U-relational database* for a world-set over schema $\Sigma = (R_1[\overline{A_1}], \ldots, R_k[\overline{A_k}])$ is a tuple

$$(U_{1,1}, \ldots, U_{1,m_1}, \ldots, U_{k,1}, \ldots, U_{k,m_k}, W),$$

where W is a world-table and each relation $U_{i,j}$ has schema $U_{i,j}[\overline{D_{i,j}}; \overline{T}_{R_i}; \overline{B_{i,j}}]$ such that $\overline{D_{i,j}}$ defines ws-descriptors over W, $\overline{T}_{R_i}$ defines tuple ids, and $\overline{B_{i,1}} \cup \cdots \cup \overline{B_{i,m_i}} = \overline{A_i}$.

A ws-descriptor $\{c_1 \mapsto l_1, \ldots, c_k \mapsto l_k\}$ is relationally encoded in $\pi_{\overline{D_{i,j}}}(U_{i,j})$ of arity $n \geq k$ as a tuple $(c_1 \mapsto$

l_1, \ldots, c_k \mapsto l_k, c_{k+1} \mapsto l_{k+1}, \ldots, c_n \mapsto l_n)$, where each $c_i \mapsto l_i$ is a $c_j \mapsto l_j$ for any j and all i with $1 \leq j \leq k < i \leq n$.

Although we speak of vertical partitioning, we do not require the value columns of $U_{i,j}$ to disjointly partition the columns of R_i. Indeed, overlap may be useful to speed up query evaluation, see e.g. [16].

We next define the semantics of a U-relational database. To obtain a possible world we first choose a total valuation f over W. We then process the U-relations tuple by tuple. If the function f extends[2] the ws-descriptor $\overline{d}$ of a tuple of the form $(\overline{d}, \overline{t}, \overline{a})$ from a U-relation of schema $(\overline{D}, \overline{T}, \overline{A})$, we insert in that world the values $\overline{a}$ into the $\overline{A}$-fields of the tuple with identifier $\overline{t}$. In general this may leave some tuples partial in the end (i.e., the values for some fields have not been provided). These tuples are removed from the world.

We require, for a U-relational database $(U_1, \ldots, U_n, W)$ to be considered valid, that the representation does not provide several contradictory values for a tuple field in the same world. Formally, we require, for all $1 \leq i, j \leq n$, and tuples $t_1 \in U_i[\overline{D_i}, \overline{T_i}, \overline{A_i}]$ and $t_2 \in U_j[\overline{D_j}, \overline{T_j}, \overline{A_j}]$ such that U_i and U_j are vertical partitions of the same relation, that if there is a world that extends both $t_1.\overline{D_i}$ and $t_2.\overline{D_j}$, then for all $A \in (\overline{A_i} \cap \overline{A_j})$, $t_1.A = t_2.A$ must hold.

Example II.3. Suppose there are two U-relations with schemata $U_1[\overline{D_1}; T_R; A, B]$ and $U_2[\overline{D_2}; T_R; B, C]$ that jointly represent columns A, B, and C of a relation R. Assume tuples $(c_1, 1, t_1, a, b) \in U_1$ and $(c_2, 2, t_1, b', c) \in U_2$, $b \neq b'$. Then U_1 and U_2 cannot form part of a valid U-relational database because there would be a world with $c_1 \mapsto 1, c_2 \mapsto 2$ in which the tuple from U_1 requires field $t_1.B$ to take value b while the tuple from U_2 requires the same field to take value b'. □

A salient property of U-relational databases is that they form a *complete representation system* for finite world-sets.

Theorem II.4. *Any finite set of worlds can be represented as a U-relational database.*

III. QUERY PROCESSING

The semantics of a query Q on a world-set is to evaluate Q in each world. For complete representation systems like U-relational databases, there is an equivalent, more efficient approach [12]: Translate Q into a query $\hat{Q}$ such that the evaluation of $\hat{Q}$ on a U-relational encoding of the world-set produces the U-relational encoding of the answer to Q.

Queries on vertical decompositions. U-relations rely essentially on vertical decomposition for succinct (attribute-level) representation of uncertainty. To evaluate a query, we first need to reconstruct relations from vertical decompositions by (1) joining two partitions on the common tuple id attributes and (2) discarding the combinations that yield inconsistent ws-descriptors. We call this operation *merge* and give its precise

---

[1]This is a generalization of world-set decompositions of [4], where component ids are variables and local world ids are domain values.

[2]That is, for all x on which $\overline{d}$ is defined, $\overline{d}(x) = f(x)$.

definition in Fig. 4, where the two above conditions are defined by α and ψ, respectively.

Example III.1. Consider the U-relational database of Fig. 1. The query $\sigma_{\text{Faction}='\text{Enemy}'\wedge\text{Type}='\text{Tank}'}(R)$ lists the enemy tanks on the map. To answer this query, we need to *merge* the necessary partitions of R and obtain a new query with $merge(\pi_{\text{Faction}}(R), \pi_{\text{Type}}(R))$ in the place of R. $\square$

Our query evaluation approach can take full advantage of query evaluation and optimization techniques on vertical partitions. First, it does not require to reconstruct the entire relations involved in the query, but rather only the necessary vertical partitions. Second, necessary partitions can be flexibly merged in during query evaluation. Thus early and late tuple materialization [16] carry over naturally to our framework. For this, our *merge* operator allows to merge two partitions not only if they are given in their original form, but also if they have been modified by queries.

The first advantage only holds for so-called *reduced* U-relational databases, which do not have tuples that cannot be completed in any world. That is, each tuple of a reduced U-relation can always be completed to an actual tuple in a world. The advantage becomes evident even for a simple projection query. Consider a reduced database containing a U-relation U defining the A attribute of R. To evaluate $\pi_A(R)$ we do not need to merge in all U-relations defining the attributes of R and later project on A. Instead, the answer is simply U. In the following, we assume that the input database is always reduced. As we will discuss next, our query evaluation technique always produces reduced U-relations for reduced input U-relational databases.

Example III.2. Consider the following non-reduced database of two U-relations:

U_1	D	T	A
	$c_1 \mapsto 1$	t_1	a_1
	$c_2 \mapsto 1$	t_2	a_2

U_2	D	T	B
	$c_1 \mapsto 1$	t_1	b_1
	$c_1 \mapsto 2$	t_1	b_2

In each U-relation the second tuple cannot find a partner in the other U-relation with which a complete tuple (with both attributes A and B) can be formed. If these second tuples are removed, the database is reduced. $\square$

We can always reduce a U-relational database as follows: We filter each U-relation using semijoins with each of the other U-relations representing data of the same relation R_i. The semijoin conditions are the α and ψ-conditions.

Proposition III.3. *Given a schema Σ, there is a relational algebra query that reduces a U-relational database over Σ.*

Algebraic equivalences. Fig. 2 gives algebraic equivalences of relational algebra expressions with merge operator on vertical decompositions: Merging is the reverse of vertical partitioning, it is commutative and associative, it commutes with selections, joins, and projections.

Standard heuristics known from classical query optimization for relational algebra apply here as well. Intuitively, we

$$merge(\pi_{\overline{X}}(R), \pi_{\overline{A}-\overline{X}}(R)) = R, \qquad \text{where } \overline{A} = \mathbf{sch}(R) \quad (1)$$
$$merge(R, S) = merge(S, R) \qquad (2)$$
$$merge(merge(R, S), T) = merge(R, merge(S, T)) \qquad (3)$$
$$\sigma_{\phi(\overline{X})}(merge(R, S)) = merge(\sigma_{\phi(\overline{X})}(R), S) \qquad (4)$$
$$\text{where } \overline{X} \subseteq \mathbf{sch}(R)$$
$$merge(R, S) \bowtie_{\phi(\overline{X}, \overline{Y})} T = merge(R \bowtie_{\phi(\overline{X}, \overline{Y})} T, S) \qquad (5)$$
$$\text{where } \overline{X} \cup \overline{Y} \subseteq \mathbf{sch}(R) \cup \mathbf{sch}(T)$$
$$\pi_{\overline{X}}(merge(R, S)) = merge(\pi_{\overline{X} \cap \overline{A}}(R), \pi_{\overline{X} \cap \overline{B}}(S)) \qquad (6)$$
$$\text{where } \mathbf{sch}(R) = \overline{A}, \mathbf{sch}(S) = \overline{B}$$

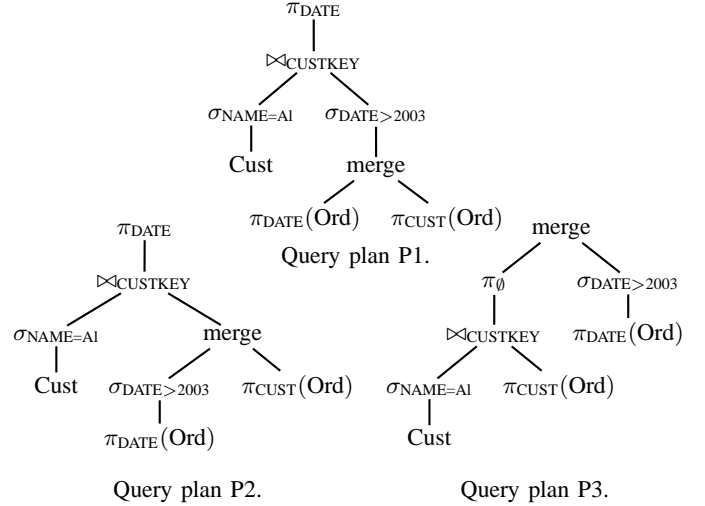Fig. 2. Algebraic equivalences for relational algebra queries with merge operator.

Fig. 3. Three equivalent query plans.

usually push down projections and selections and merge in U-relations as late as possible. An interesting new case is the decision on join ordering among an explicit join from the input query and a join due to merging: If the merge is executed before the explicit join, it may reduce the size of an input relation to join. We have seen in our experiments that the standard selectivity-based cost measures employed by relational database management systems do a good job, as long as the queries remain reasonably small.

Example III.4. Consider a U-relational database $\mathcal{U}$ that represents a set of possible worlds over two TPC-H relations Ord and Cust (short for Order and Customer, respectively) [17]. $\mathcal{U}$ has one U-relation for each attribute of the two relations, of which we only list DATE and CUSTKEY for Ord, and NAME and CUSTKEY for Cust. The following query finds all dates of orders placed by Al after 2003:

$$\pi_{\text{DATE}}(\sigma_{\text{NAME}='\text{Al}'}(\text{Cust}) \bowtie_{\text{CUSTKEY}} \sigma_{\text{DATE}>2003}(\text{Ord}))$$

Fig. 3 shows three possible plans P1, P2, and P3 using operators on vertical decompositions. The naïve plan P1 first reconstructs Ord from its two partitions then applies the selection and the join with Cust. In P2 and P3 the merge operator is pushed up in the plans, first immediately above the

$$\text{Let } U_1 := [\![Q_1]\!] \text{ with schema } [\overline{D}_1, \overline{T}_1, \overline{A}_1],$$
$$U_2 := [\![Q_2]\!] \text{ with schema } [\overline{D}_2, \overline{T}_2, \overline{A}_2],$$
$$\alpha := \bigwedge_{T \in \overline{T}_1 \cap \overline{T}_2} (U_1.T = U_2.T),$$
$$\psi := \bigwedge_{D' \in U_1.\overline{D}_1, D'' \in U_2.\overline{D}_2} (D'.\text{Var} = D''.\text{Var} \Rightarrow D'.\text{Rng} = D''.\text{Rng}).$$
$$[\![possible(Q_1)]\!] := \pi_{\overline{A}_1}(U_1)$$
$$[\![\pi_{\overline{X}}(Q_1)]\!] := \pi_{\overline{D}_1, \overline{T}_1, \overline{X}}(U_1), \quad \text{where } \overline{X} \subseteq \overline{A}_1$$
$$[\![\sigma_\phi(Q_1)]\!] := \sigma_\phi(U_1), \quad \text{where } \phi \text{ on } \overline{A}_1$$
$$[\![Q_1 \bowtie_\phi Q_2]\!] := \pi_{\overline{D}_1, \overline{D}_2, \overline{T}_1, \overline{T}_2, \overline{A}, \overline{B}}(U_1 \bowtie_{\phi \wedge \psi} U_2),$$
$$\text{where } \overline{T}_1 \cap \overline{T}_2 = \emptyset$$
$$[\![merge(Q_1, Q_2)]\!] := \pi_{\overline{D}_1, \overline{D}_2, \overline{T}_1 \cup \overline{T}_2, \overline{A}, \overline{B}}(U_1 \bowtie_{\alpha \wedge \psi} U_2)$$

Fig. 4.   Translation of queries with merge into queries on U-relations.

selection (P2), and then above the join operator (P3). Among the three plans, P1 is clearly the least efficient. However, without statistics about the data, one cannot tell which of P2 and P3 should be preferred. If DATE>2003 is very selective, then merging immediately thereafter as in P2 will lead to the filtering of tuples from $\pi_{\text{CUSTKEY}}(\text{Ord})$ and thus fewer tuples will be processed by the join. Is this not the case, then first merging only increases the number and size of the tuples that have to be processed by the join. Also, in P3 all value attributes except for DATE are projected away after the join as they are not needed for the final result. □

Queries on U-relations. Fig. 4 gives the function $[\![\cdot]\!]$ that translates positive relational algebra queries with *possible* and *merge* operators into relational algebra queries on U-relational databases.

The possible operator applied on a U-relation U closes the possible worlds semantics by computing the set of tuples possible in U. It thus translates to a simple projection on the value attributes of U. The result of a projection is a U-relation whose value attributes are those from the projection list (thus the input ws-descriptors and tuple ids are preserved). Selections apply conditions on the value attributes.

The merge operator that reconstructs a relation from its vertical partitions was already explained. Similarly to the merge, the join uses the ψ-condition to discard tuple combinations with inconsistent ws-descriptors. Fig. 4 gives the translation in case U_1 and U_2 do not contain partitions of the same relation. For the case of self-joins we require aliases for the copies of the relation involved in it such that they do not have common tuple id attributes. Example III.7 will illustrate this.

The union of U_1 and U_2 like the ones from Fig. 4 is sketched next. We assume that $\overline{A}_1 = \overline{A}_2$, $\overline{T}_1 \cap \overline{T}_2 = \emptyset$, and the tuples of different relations have different ids. To bring U_1 and U_2 to the same schema, we first ensure ws-descriptors of the same size by padding the smaller ws-descriptors with already contained variable assignments, and add new (empty) columns $\overline{T}_2$ to U_1 and $\overline{T}_1$ to U_2. We then perform the standard union.

From our translation $[\![\cdot]\!]$ it immediately follows that

Theorem III.5. *Positive relational algebra queries extended with the possible operator can be evaluated on U-relational databases using relational algebra only.*

Example III.6. Recall the U-relational database of Fig. 1 storing information about moving vehicles. Consider a query asking for ids of enemy tanks:

$$S = \pi_{\text{Id}}(\sigma_{\text{Type}='\text{Tank}' \wedge \text{Faction}='\text{Enemy}'}(R))$$

After merging the necessary partitions of relation R and translating it into positive relational algebra, we obtain

$$\pi_{\text{Id}}(\sigma_{\text{Type}='\text{Tank}' \wedge \text{Faction}='\text{Enemy}'}(U_1 \bowtie_{\alpha_1 \wedge \psi_1} U_2 \bowtie_{\alpha_2 \wedge \psi_2} U_3)),$$

where the conditions ψ_1, ψ_2, α_1, and α_2 follow the translation given in Fig. 4. The three vertical partitions are joined on the tuple id attributes (α_1 and α_2) and the combinations with conflicting mappings in the ws-descriptors are discarded (ψ_1 and ψ_2). Before and after translation, the query is subject to optimizations as discussed earlier. (In this case, a good query plan would first apply the selections on the partitions, then project away the irrelevant value attributes Type and Faction, and then merge the partitions).

U_4	D_1	D_2	T_S	Id
	$x \mapsto 1$		c	3
	$x \mapsto 2$		c	2
	$y \mapsto 1$	$z \mapsto 2$	d	4

The above U-relation U_4 encodes the query answer. □

Example III.7. We continue Example III.6 and ask whether it is possible that the enemy has two tanks on the map, and if so, which vehicles are those. For this, we compute the pairs of enemy tanks as a self-join of S: $(S \ s_1) \bowtie_{s_1.\text{Id} \neq s_2.\text{Id}} (S \ s_2)$. This query is in turn equivalent to a self-join of U_4.

U_5	D_1	D_2	D_3	T_{s_1}	T_{s_2}	Id_1	Id_2
	$x \mapsto 1$	$y \mapsto 1$	$z \mapsto 2$	c	d	3	4
	$x \mapsto 2$	$y \mapsto 1$	$z \mapsto 2$	c	d	2	4
	$y \mapsto 1$	$z \mapsto 2$	$x \mapsto 1$	d	c	4	3
	$y \mapsto 1$	$z \mapsto 2$	$x \mapsto 2$	d	c	4	2

The answer is encoded by the above U-relation U_5. Note that the combinations of the first two tuples of U_4 are not in U_5, because they have inconsistent ws-descriptors and are filtered out using the ψ-condition (vehicle c cannot be at the same time at two different positions). To obtain the possible pairs of vehicle ids, we apply the possible operator on U_5. This is expressed as the projection on the value attributes of U_5. □

Our translation yields relational algebra queries, whose evaluation always produces tuple-level U-relations, i.e., U-relations without vertical decompositions, by joining and merging vertical partitions of relations. Following the definition of the merge operator, if the input U-relations are reduced, then the result of merging vertical partitions is also reduced. We thus have that

Proposition III.8. *Given a positive relational algebra query Q and a reduced U-relational database U, $[\![Q]\!](U)$ is a reduced U-relational database.*

Algorithm 1: Normalization of ws-descriptors.

Input: Reduced U-relational database $\mathcal{U} = (U_1, \ldots, U_m, W)$
Output: Normalized reduced U-relational database.
begin

  $R :=$ the relation consisting of all pairs of variables (c_i, c_j) that occur together in some ws-descriptor of $\mathcal{U}$;

  $\mathcal{G} :=$ the graph whose node set is the set of variables and whose edge relation is the refl. and trans. closure of R;

  Compute the connected components of $\mathcal{G}$;

  foreach *U-relation* $U_j(D_1, \ldots, D_n, \overline{T}, \overline{A})$ *of* $\mathcal{U}$ **do**

    $U'_j :=$ empty U-relation over $U'_j(\text{Var}, \text{Rng}, \overline{T}, \overline{A})$;

    foreach $t \in U$ **do**

      $G_i :=$ connected component of $\mathcal{G}$ with id i such that the nodes $t.\text{Var}_1, \ldots, t.\text{Var}_n$ are in G_i;

      $\{c_{i_1}, \ldots, c_{i_k}\} = G_i - \{t.\text{Var}_1, \ldots, t.\text{Var}_n\}$;

      foreach

      $l_{i_1} : (c_{i_1}, l_{i_1}) \in W, \ldots, l_{i_k} : (c_{i_k}, l_{i_k}) \in W$ **do**

        /* Compute a new domain value ($f_{|G_i|}$ is either the identity or better, for atomic l's, an injective function $\text{int}^{|G_i|} \to \text{int}$) */;

        $l := f_{|G_i|}(t.\overline{\text{Rng}}, l_{i_1}, \ldots, l_{i_k})$;

        $U'_j := U'_j \cup \{(G_i, l, t.\overline{T}, t.\overline{A})\}$;

  $W' := \bigcup_i \{(g_i, (l_1, \ldots, l_m)) \mid G_i = \{c_1, \ldots, c_m\}$ and $(c_1, l_1), \ldots, (c_m, l_m) \in W\}$;

  Output $(U'_1, \ldots, U'_m, W')$;

end

U	D_1	D_2	T	A
	$c_1 \mapsto 1$	$c_1 \mapsto 1$	t_1	a_1
	$c_1 \mapsto 1$	$c_2 \mapsto 2$	t_2	a_2
	$c_1 \mapsto 2$	$c_1 \mapsto 2$	t_2	a_3
	$c_3 \mapsto 1$	$c_3 \mapsto 1$	t_3	a_4
	$c_3 \mapsto 2$	$c_3 \mapsto 2$	t_3	a_5

W	Var	Rng
	c_1	1
	c_1	2
	c_2	1
	c_2	2
	c_3	1
	c_3	2

(a) U-relational database

U'	D	T	A
	$c_{12} \mapsto (1,1)$	t_1	a_1
	$c_{12} \mapsto (1,2)$	t_1	a_1
	$c_{12} \mapsto (1,2)$	t_2	a_2
	$c_{12} \mapsto (2,1)$	t_2	a_3
	$c_{12} \mapsto (2,2)$	t_2	a_3
	$c_3 \mapsto 1$	t_3	a_4
	$c_3 \mapsto 2$	t_3	a_5

W'	Var	Rng
	c_{12}	$(1,1)$
	c_{12}	$(1,2)$
	c_{12}	$(2,1)$
	c_{12}	$(2,2)$
	c_3	1
	c_3	2

(b) Database from (a) normalized

Fig. 5. Normalization example.

IV. NORMALIZATION OF U-RELATIONS

U-relations do not forbid large ws-descriptors. The ability to extend the size of ws-descriptors is what yields efficient query evaluation on U-relations. However, large ws-descriptors cause an inherent processing overhead. Also, after query evaluation or dependency chasing on a U-relational database, it may happen that tuple fields, which used to be dependent on each other, become independent. In such a case, it is desirable to optimize the world-set representation [6]. We next discuss one approach to normalize U-relational databases by reducing large ws-descriptors to ws-descriptors of size one. Normalization is an expensive operation per se, but it is not unrealistic to assume that uncertain data is initially in normal form [4], [6] and can subsequently be maintained in this form.

Definition IV.1. A U-relational database is normalized if all ws-descriptors of its U-relations have size one.

Algorithm 1 gives a normalization procedure for U-relations that determines classes of variables that co-occur in some ws-descriptors and replaces each such class by one variable, whose domain becomes the product of the domains of the variables from that class. Fig. 5 shows a U-relational database and its normalization.

Theorem IV.2. *Given a reduced U-relational database, Algorithm 1 computes a normalized reduced U-relational database that represents the same world-set.*

Computing certain answers. Given a set of possible worlds, we call a tuple certain iff it occurs in each of the worlds. It is known that the tuple certainty problem is coNP-hard for a number of representation systems, ranging from attribute-level ones like WSDs to tuple-level ones like ULDBs [6]. In case of tuple-level normalized U-relations, however, we can efficiently compute the certain tuples using relational algebra.

Lemma IV.3. *Tuple $\overline{a}$ is certain in a tuple-level normalized U-relation U iff there exists a variable x such that $(x \mapsto l, \overline{t}, \overline{a}) \in U$ for each domain value l of x and some tuple id $\overline{t}$.*

The condition of the lemma can be encoded as the following domain calculus expression:

$$cert(U) := \{\overline{a} \mid \exists x \forall l \ (x, l) \in W \Rightarrow \exists \overline{t}(x, l, \overline{t}, \overline{a}) \in U\}$$

The equivalent relational algebra query on a tuple-level normalized U-relational database $(U[\text{Var}, \text{Rng}, \overline{T_R}, \overline{A}], W)$ is

$$\pi_{\overline{A}}(\pi_{Var}(W) \times \pi_{\overline{A}}(U) - \pi_{Var, \overline{A}}(W \times \pi_{\overline{A}}(U) - \pi_{Var, Rng, \overline{A}}U)).$$

V. SUCCINCTNESS AND EFFICIENCY

This section compares U-relational databases with WSDs [4], [6] and ULDBs [8] using two yardsticks: succinctness, i.e., how compactly they can represent world-sets, and efficiency of query evaluation. Due to lack of space, we defer a more complete comparison (with proofs and examples) to an extended version of this paper [3].

WSDs vs. U-Relations. WSDs are essentially normalized U-relational databases where each variable c_i of a U-relation corresponds to a WSD *component* relation C_i and each domain value l_i of c_i corresponds to a tuple of C_i. The normalization may lead to an exponential blow-up in the database size and accounts for U-relations with arbitrarily large ws-descriptors being more compact than U-relations with singleton ws-descriptors and thus than WSDs.

Theorem V.1. *U-relational databases are exponentially more succinct than WSDs.*

Positive relational queries have polynomial data complexity for U-relations (Section III) and exponential data complexity for WSDs [6]. This can be explained in close analogy to the difference in succinctness and by the fact that query evaluation creates new dependencies [10]: U-relations can efficiently store the new dependencies by enlarging ws-descriptors, whereas WSDs correspond to U-relations with normalized ws-descriptors, hence the exponential blowup.

ULDBs vs. U-Relations. ULDBs are databases with uncertainty and lineage [8]. Due to lack of space, we only state the salient results concerning our comparison to ULDBs.

Lemma V.2. *ULDBs [8] can be translated linearly into U-relational databases.*

The translation uses a direct encoding of ULDB's lineage into ws-descriptors, where ULDB's tuple and alternative ids become variables and domain values, respectively.

There are U-relations, however, whose ULDB encodings are necessarily exponential in the arity of the logical relation. This is the case of, e.g., or-set relations [13], attribute-level representations that can be linearly encoded as U-relations but exponentially as ULDBs.

Theorem V.3. *U-relational databases are exponentially more succinct than ULDBs.*

Both ULDBs and U-relations have polynomial data complexity for positive relational queries. Differently from ULDBs, evaluating queries on U-relations is possible using relational algebra only. The main difference between their evaluation algorithms concerns dealing with erroneous tuples, i.e., tuples that do not appear in any world. In contrast to U-relations, erroneous tuples may appear in the answers to queries on ULDBs (see [8] for an example). The removal of such tuples is called data minimization, an expensive operation that involves the computation of the transitive closure of lineage [8]. Such tuples occur with ULDBs because the lineage of an alternative in the answer only points to the lineage of alternatives from the input relations, even though these input alternatives may not occur in the same world. This cannot happen with U-relations because each query operation ensures that only valid tuples are in the query answer by (1) using the ψ-condition in the join and merge operations and by (2) carrying all dependencies in the ws-descriptors – and not only to tuples of the input relation.

VI. PROBABILISTIC U-RELATIONS

U-relational databases can be elegantly extended to model probabilistic information by adding a probability column Pr to the world table W. Thus W contains tuples (x, v, p) for all domain values v of a variable x, and p is the probability of $x \mapsto v$. For each variable x defined by W, the sum of the values $\pi_{Pr}(\sigma_{\mathrm{Var}=x})(W)$ must equal one. Fig. 6(a) shows a probabilistic version of the world-table of Fig. 1(b).

We use a function P to define the probability of a valuation

W	Var	Rng	Pr
	x	1	0.1
	x	2	0.9
	y	1	0.3
	y	2	0.7
	z	1	0.6
	z	2	0.4

U_6	Id	conf
	3	$P(\{x \mapsto 1\}) = 0.1$
	2	$P(\{x \mapsto 2\}) = 0.9$
	4	$P(\{y \mapsto 1, z \mapsto 2\}) = 0.12$

(a) Probabilistic world-table.   (b) Computing tuple confidence.

Fig. 6.   Probabilistic U-relations

as the product of probabilities of its variable assignments:

$$P(\{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\}) = \prod_{i=1}^{n} P(\{x_i \mapsto v_i\}) \qquad (*)$$

The probabilistic extension is orthogonal to the techniques for evaluating positive relational algebra queries described in Section III. Since processing relational algebra queries only extends each world with the result of the query in it without changing the world's probabilities, the algorithms carry over with no change to the probabilistic case as well. A different class of queries are those that ask for confidence of tuples in the result of a query. Let U be a U-relation representing the answer to a query q on a U-relational database. Then, the *confidence* of a tuple $\overline{a}$ in the answer to q is the sum of the probabilities of the worlds defined by U that contain $\overline{a}$. Computing the confidence by enumerating all possible worlds, as the above definition suggested, is, however, not feasible. A better approach is to compute the probability of the world-set represented by the union of ws-descriptors associated with $\overline{a}$ in U:

$$P(\{\overline{d} \mid \exists \overline{s}(\overline{d}, \overline{s}, \overline{a}) \in U\})$$

In case only one tuple with ws-descriptor $\overline{d}$ in U matches the given tuple $\overline{a}$, then the confidence of $\overline{a}$ can be trivially computed as $P(\overline{d})$ using formula (*) above. In the general case, however, the computation is #P-complete. This follows from the mutual reducibility of the problem of computing the probability of the union of the (possibly overlapping) world-sets represented by a set of ws-descriptors and of the #P-complete problem of counting the number of satisfying assignments of Boolean formulas in disjunctive normal form. Indeed, we can encode a set of k ws-descriptors $\{x_1^i \mapsto v_1^i, \ldots, x_{m_i}^i \mapsto v_{m_i}^i\}$ $(1 \le i \le k)$ as a formula $\bigvee_{1 \le i \le k} (x_1^i = v_1^i \wedge \ldots \wedge x_{m_i}^i = v_{m_i}^i)$.

Recent work considered efficient solutions for restricted classes of queries and probabilistic databases [10] or by applying approximation techniques [14]. Scalable confidence computation is out of the scope of this paper. Our current approach for exact confidence computation exploits the independence and variable sharing among ws-descriptors and is by far more efficient than approaches based on enumeration of all worlds or on the inclusion-exclusion formula.

Example VI.1. Consider a probabilistic version of the U-relational database of Fig. 1(b) with world-table defined in Fig. 6(a). Consider again relation S from Example III.6 containing the ids of enemy tanks on the map. There are three

Fig. 7. Queries used in the experiments.

different possible enemy tank ids, whose confidence can be computed as $P(\{x \mapsto 1\}), P(\{x \mapsto 2\})$ and $P(\{y \mapsto 1, z \mapsto 2\})$, respectively. The result is given in Fig. 6(b).

The confidence of having at least one enemy tank on the map is computed as $P(\{\{x \mapsto 1\}, \{x \mapsto 2\}, \{y \mapsto 1, z \mapsto 2\}\})$. The three ws-descriptors represent the entire world-set, thus the confidence is 1. □

VII. Experiments

Prototype Implementation. We implemented the query translator of Fig. 4. We also extended the C implementation of the TPC-H population generator version 2.6 build 1 [17] to generate attribute and tuple-level U-relations and ULDBs. The code is available on the MayBMS project page (http://www.cs.cornell.edu/database/maybms).

Setup. The experiments were performed on a 3GHZ/1GB Pentium running Linux 2.6.13 and PostgreSQL 8.2.3.

Generation of uncertain data. The following parameters were used to tune the generation: *scale* (s), *uncertainty ratio* (x), *correlation ratio* (z), and *maximum alternatives per field* (m). The (dbgen standard) parameter s is used to control the size of each world; x controls the percentage of (uncertain) fields with several possible values, and m controls how many possible values can be assigned to a field. The parameter z defines a Zipf distribution for the variables with different dependent field counts (DFC). The DFC of a variable is the number of tuple fields dependent on that variable. We use the parameter z to control the attribute correlations: For n uncertain fields, there are $\lceil C * z^i \rceil$ variables with DFC i, where $C = n(z-1)/(z^{k+1}-1)$, i.e., $n = \sum_{i=0}^{k}(C * z^i)$. Thus greater z values correspond to higher correlations in the data. The number of domain values of a variable with DFC $k > 1$ is chosen using the formula $p^{k-1} * \prod_{i=1}^{k}(m_i)$, where m_i is the number of different values for the i-th field dependent on that variable and p is the probability that a combination of possible values for the k fields is valid. This assumption fits naturally to data cleaning scenarios. Previous work [4] shows that chasing dependencies on WSDs enforces correlations between field values and removes combinations that violate the dependencies. We considered here that after correlating two variables with arbitrary DFCs, only $p * 100$ percent of the combinations satisfy the constraints and are preserved.

The uncertain fields are assigned randomly to variables. This can lead to correlations between fields belonging to different tuples or even to different relations. This fits to scenarios where constraints are enforced across tuples or relations. We do not assume any kind of independence of our initial data as done in several other approaches [10], [8].

For the experiments, we fixed p to 0.25, m to 8, and varied the remaining parameters as follows: s ranges over $(0.01, 0.05, 0.1, 0.5, 1)$, z ranges over $(0.1, 0.25, 0.5)$, and x ranges over $(0.001, 0.01, 0.1)$.

An important property of our generator is that any world in a U-relational database shares the properties of the one-world database generated by the original dbgen: The sizes of relations are the same and the join selectivities are approximately equal. We checked this by randomly choosing one world of the U-relational database and comparing the selectivities of joins on the keys of the TPC-H relations for different scale factors and uncertainty ratios.

Queries. We used the three queries from Fig. 7. Query Q_1 is a join of three relations of large sizes. Query Q_2 is a select-project query on the relation lineitem (the largest in our settings). Query Q_3 is a fairly complex query that involves joins between six relations. All queries use the operator 'possible' to retrieve the set of matches across all worlds. Note that these queries are modified versions of Q_3, Q_6, and Q_7 of TPC-H where all aggregations are dropped (dealing with aggregation is subject to future work).

Fig. 9 shows that our queries are moderately selective and their answer sizes increase with uncertainty x and marginally with correlation z. For scale 1, the answer sizes range from tens of thousands to tens of millions of tuples. There is only one setting ($z = 0.25$ and $x = 0.1$) where one of our queries, Q_3, has an empty answer. Before the execution, the queries were optimized using our U-relation-aware optimizations. Fig. 8 shows Q_1 after optimizations.

Characteristics of U-relations. Following Fig. 8, the U-relational databases are exponentially more succinct than databases representing all worlds individually: while the number of worlds increases exponentially (when varying the uncertainty ratio x), the database size increases only linearly. The case of $x = 0$ corresponds to one world generated using the original dbgen. Interestingly, to represent $10^{8 \cdot 10^6}$ worlds, the U-relational database needs about 6.7 times the size of one world.

An increase of the scaling factor leads to an exponential increase in the number of worlds and only to a linear increase in the size of the U-relational database. Although we only report here on experiments with scale factors up to 1, further experiments confirmed that similar characteristics are obtained for larger scales, too. An increase of the correlation parameter leads to a moderate relative increase in the database size. When compared to one-world databases, the sizes of U-relational databases have increase factors that vary from 6.2 (for $z = 0.1$) to 8.2 (for $z = 0.5$).

| s	z	TPC-H dbsize	#worlds	Rng	dbsize	#worlds	Rng	dbsize	#worlds	Rng	dbsize
0.01	0.1	17	$10^{857.076}$	21	82	$10^{7955.30}$	57	85	$10^{79354.1}$	57	114
0.01	0.5	17	$10^{523.031}$	71	82	$10^{4724.56}$	901	88	$10^{46675.6}$	662	139
0.05	0.1	85	$10^{4287.23}$	22	389	$10^{39913.8}$	33	403	10^{396137}	65	547
0.05	0.5	85	$10^{2549.14}$	178	390	$10^{23515.5}$	449	416	10^{232650}	1155	672
0.10	0.1	170	$10^{8606.77}$	27	773	$10^{79889.9}$	49	802	10^{793611}	53	1090
0.10	0.5	170	$10^{5044.65}$	181	776	$10^{46901.8}$	773	826	10^{466038}	924	1339
0.50	0.1	853	$10^{43368.0}$	49	3843	10^{400185}	71	3987	$10^{3.97e+06}$	85	5427
0.50	0.5	853	$10^{25528.9}$	214	3856	10^{234840}	1832	4012	$10^{2.33e+06}$	2586	6682
1.00	0.1	1706	$10^{87203.0}$	57	7683	10^{800997}	99	7971	$10^{7.94e+06}$	113	11264
1.00	0.5	1706	$10^{51290.9}$	993	7712	10^{470401}	1675	8228	$10^{4.66e+06}$	3392	13312
		x = 0.0	**x = 0.001**			**x = 0.01**			**x = 0.1**		
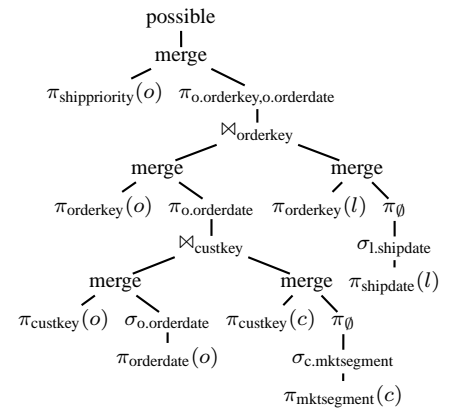
Fig. 8. (left): Total number of worlds, max. number of domain values for a variable (Rng), and size in MB of the U-relational database for each of our settings. (right): Query plan for Q_1 using merge.
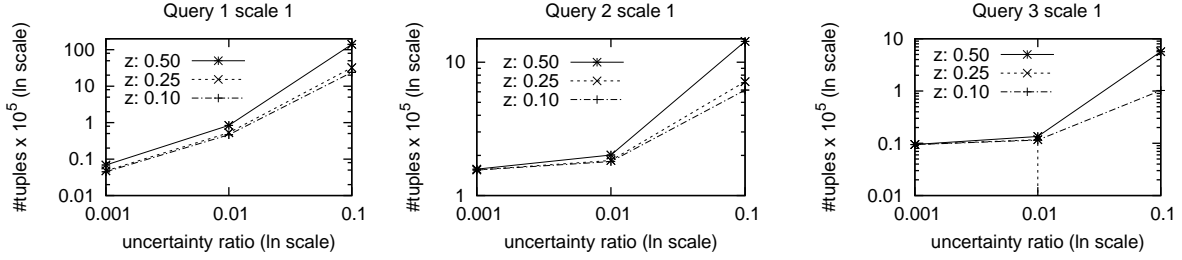

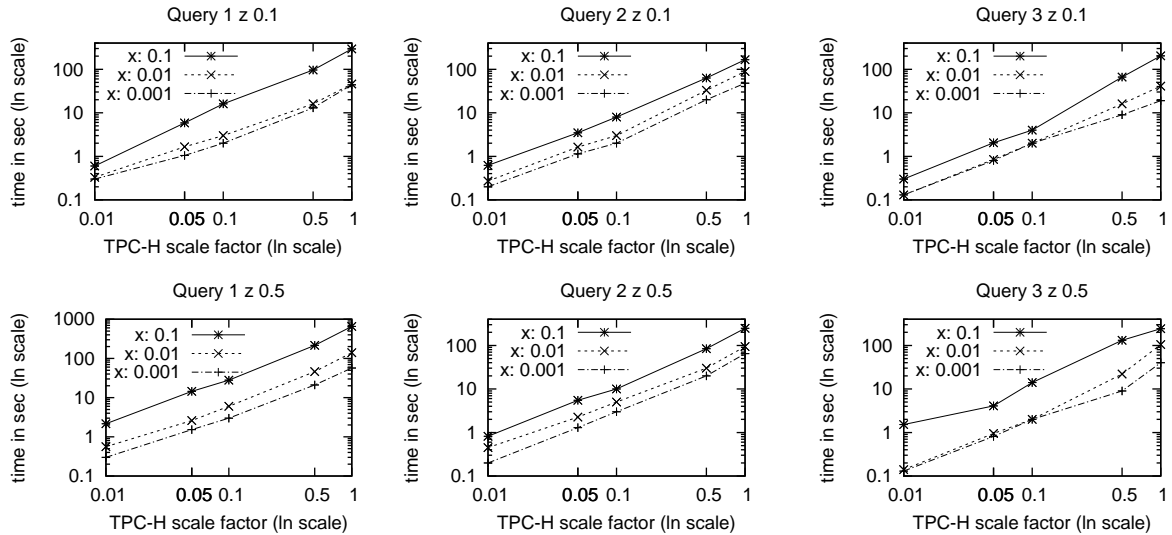Fig. 9. Sizes of query answers for settings with scale 1.

Fig. 10. Performance of query evaluation for various scale, uncertainty, and correlation.

Query Evaluation on U-relations. We run four times our set of three queries on the 45 different datasets reported in Fig. 8. For each query and correlation ratio, Fig. 10 has a log-log scale diagram showing the median evaluation (including storage) time in seconds as a function of the scale and uncertainty parameters ([3] also shows diagrams for $z = 0.25$). The different lines in each of the diagrams correspond to different uncertainty ratios.

Fig. 10 shows that the evaluation of our queries is efficient and scalable. In our largest scenario, where the database has size 13 GB and represents $10^{8 \cdot 10^6}$ worlds with 1.4 GBs each world, query Q_3 involving five joins is evaluated in less than two and a half minutes. One explanation for the good performance is the use of attribute-level representation. This allows to first compute the joins locally using only the join attributes and later merge in the remaining attributes of interest. Another important reason for the efficiency is that due to the simplicity of our rewritings, PostgreSQL optimizes the queries in a fairly good way. ([3] shows an optimized query

plan produced by the PostgreSQL 'explain' statement for the rewriting of Q_2.)

The evaluation time varies linearly with all of our parameters. For Q_1 (Q_2 and Q_3 respectively) we witnessed a factor of up to 6 (4 and 10 respectively) in the evaluation time when varying the uncertainty ratio from 0.001 to 0.1. When the correlation ratio is varied from 0.1 to 0.5, the evaluation time increases by a factor of up to 3; this is also explained by the increase in the input and answer sizes, cf. Fig.s 8 and 9. When the scale parameter is varied from 0.01 to 1, the evaluation time increases by a factor of up to 400; in case of Q_3 and $z = 0.5$, we also noticed some outliers where the increase factor is around 1000.

Effect of attribute-level representation. We also performed query evaluation on tuple-level U-relations, which represent the same world-set as the attribute-level U-relations of Fig. 8, and on Trio's ULDBs [8] obtained by a (rather direct) mapping from the tuple-level U-relations. To date, Trio has no native support for the possible operator or the removal of erroneous tuples in the query answer, though this effect can be obtained as part of the confidence computation[3]. For that reason, we decided to compare the evaluation times of queries without the possible operator and without the (expensive) removal of erroneous tuples or confidence computation (which is an exponential-time problem). Since our data exhibits a high degree of (randomly generated) dependency, its ULDB representation has lineage and thus join queries can introduce erroneous tuples in the answer. The Trio prototype was set to use the (faster) SPI interface of PostgreSQL (and not its default python implementation).
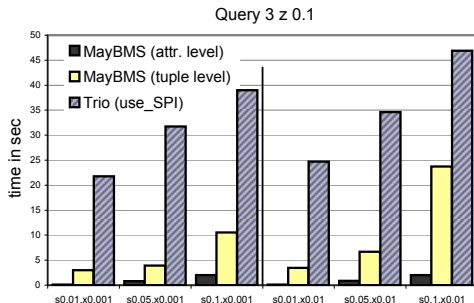

Fig. 11. Querying attribute-level and tuple-level U-relations in MayBMS and ULDBs in Trio.

Fig. 11 compares the evaluation time on attribute- and tuple-level U-relations in MayBMS, and ULDBs for small scenarios of 1% uncertainty, our lowest correlation factor 0.1, and scale up to 0.1. On attribute-level U-relations, the queries perform several times better than on tuple-level U-relations and by an order of magnitude better than ULDBs. This is because attribute-level data allows for late materialization: selections and joins can be performed locally and tuple reconstruction is done only for successful tuples. We witnessed that an increase in any of our parameters would create prohibitively large (exponential in the arity) tuple-level representations. For

[3]Personal communication with the TRIO team as of June 2007.

example, for scale 0.01 and uncertainty 10%, relation lineitem contains more than 15M tuples compared to 80K in each of its vertical partitions.

VIII. CONCLUSION AND FUTURE WORK

This paper introduces U-relational databases, a simple representation system for uncertain data that combines the advantages of existing systems, like ULDBs and WSDs, without sharing their drawbacks. U-relations are exponentially more succinct than both WSDs and ULDBs. Positive relational algebra queries are evaluated purely relationally on U-relations, a property not shared by any other previous succinct representation system. Also, U-relations are a simple formalism which poses a small burden on implementors. Following our recent investigation on uncertainty-aware language constructs beyond relational algebra [5], we identified common physical operators necessary to implement many primitives for the creation and grouping of worlds. It turns out that several other operators described in this work, including choice-of and repair-key, can also be evaluated on U-relational databases using relational algebra only. For others, including confidence computation, it appears that normalizing sets of ws-descriptors in the sense of Section IV plays an important role. We are currently working on secondary-storage algorithms for normalization.

REFERENCES

[1] S. Abiteboul, P. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. *Theor. Comput. Sci.*, **78**(1), 1991.
[2] P. Andritsos, A. Fuxman, and R. J. Miller. Clean Answers over Dirty Databases: A Probabilistic Approach. In *Proc. ICDE*, 2006.
[3] L. Antova, T. Jansen, C. Koch, and D. Olteanu. Fast and Simple Relational Processing of Uncertain Data. Technical Report cs.DB/0707.1644, ACM CORR, 2007.
[4] L. Antova, C. Koch, and D. Olteanu. 10^{10^6} Worlds and Beyond: Efficient Representation and Processing of Incomplete Information. In *Proc. ICDE*, 2007.
[5] L. Antova, C. Koch, and D. Olteanu. From Complete to Incomplete Information and Back. In *Proc. SIGMOD*, 2007.
[6] L. Antova, C. Koch, and D. Olteanu. World-set decompositions: Expressiveness and efficient algorithms. In *Proc. ICDT*, 2007.
[7] D. S. Batory. On Searching Transposed Files. *ACM Trans. Datab. Syst.*, **4**(4):531–544, 1979.
[8] O. Benjelloun, A. D. Sarma, A. Halevy, and J. Widom. ULDBs: Databases with Uncertainty and Lineage. In *Proc. VLDB*, 2006.
[9] R. Cheng, S. Singh, and S. Prabhakar. U-DBMS: a database system for managing constantly-evolving data. In *Proc. VLDB*, 2005.
[10] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *Proc. VLDB*, 2004.
[11] G. Grahne. Dependency Satisfaction in Databases with Incomplete Information. In *Proc. VLDB*, 1984.
[12] T. Imielinski and W. Lipski. Incomplete information in relational databases. *Journal of ACM*, **31**(4), 1984.
[13] T. Imielinski, S. Naqvi, and K. Vadaparty. Incomplete objects — a data model for design and planning applications. In *Proc. SIGMOD*, 1991.
[14] C. Re, N. Dalvi, and D. Suciu. Efficient Top-k Query Evaluation on Probabilistic Data. In *Proc. ICDE*, 2007.
[15] P. Sen and A. Deshpande. Representing and Querying Correlated Tuples in Probabilistic Databases. In *Proc. ICDE*, 2007.
[16] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. J. O'Neil, P. E. O'Neil, A. Rasin, N. Tran, and S. B. Zdonik. C-Store: A Column-oriented DBMS. In *Proc. VLDB*, 2005.
[17] Transaction Processing Performance Council. *TPC Benchmark H (Decision Support)*, revision 2.6.0 edition, 2006. http://www.tpc.org/tpch/spec/tpch2.6.0.pdf.