PXML: A Probabilistic Semistructured Data Model and Algebra

Edward Hung Lise Getoor* V. S. Subrahmanian†
Department of Computer Science
University of Maryland, College Park, MD 20742
{*ehung,getoor,vs*}*@cs.umd.edu*

Abstract

Despite the recent proliferation of work on semistructured data models, there has been little work to date on supporting uncertainty in these models. In this paper, we propose a model for probabilistic semistructured data (PSD). The advantage of our approach is that it supports a flexible representation that allows the specification of a wide class of distributions over semistructured instances. We provide two semantics for the model and show that the semantics are probabilistically coherent. Next, we develop an extension of the relational algebra to handle probabilistic semistructured data and describe efficient algorithms for answering queries that use this algebra. Finally, we present experimental results showing the efficiency of our algorithms.

1 Introduction

In recent years there has been a proliferation of semistructured data models proposed [20, 18, 3, 5], along with associated query languages [1, 10] and algebras [4, 15]. The semistructured data model has the advantage of not placing hard constraints on the *structure* of the data. However, a particular semistructured instance specifies deterministic relationships between objects. It is desirable to have a model that allows us to represent uncertainty over the relationships between objects in the semistructured model.

This uncertainty is necessary when relationships between objects and values for attributes of objects are not known with absolute certainty. A common source for this uncertainty comes when a semistructured representation is constructed from a noisy input source: uncertainty in sensor readings, information extraction using probabilistic parsing of input sources and image processing all may result in a semistructured instance in which there is uncertainty. Another source for this uncertainty comes from the need to represent nondeterministic processes using a semistructured model. In this case, it may be desirable to represent the distribution over possible substructures explicitly, rather than forcing a particular choice. Examples where this may hold include biological domains, manufacturing processes and financial applications.

While there has been a great deal of work on supporting uncertainty in relational models [16, 11, 12, 13], to date, there has been little work on supporting uncertainty in semistructured models. Dekhtyar et al.[9] proposed a model that allows probabilistic information to be stored using semistructured databases. Our proposal does the opposite: we extend the semistructured data model so that paths in such a model can include probabilistic information. More closely related to our work is the work of [19], in which a tree-structured probabilistic database is proposed. As we will see, their model is a special case of our model.

The first contribution of this paper is a flexible probabilistic representation for semistructured data which supports arbitrary distributions over the relationships between an object and its children and arbitrary distributions over the object's value. We do not require the semistructured instance to be tree structured, however we do require that the probabilistic model is acyclic. The second major contribution of this paper is a formal characterization of the probabilistic semantics of the model. This connection is missing in previous approaches to representing probabilistic semistructured data. The third major contribution of this paper is an algebra that supports querying probabilistic semistructured data. In addition, as space permits, we describe efficient algorithms for answering these queries. A companion paper[14] describes an approach which uses interval probabilities and gives a different kind of query language and operational semantics.

The outline of the paper is as follows. We first start with a motivating example in Section 2. In Section 3, we propose the PSD model of probabilistic semistructured databases. We define the semantics for probabilistic semistructured databases in Section 4. Then, in Section 5, we propose an extension of the relational algebra operators to apply to

* University of Maryland Institute for Advanced Computer Studies
† Institute for Advanced Computer Studies and Institute for Systems Research

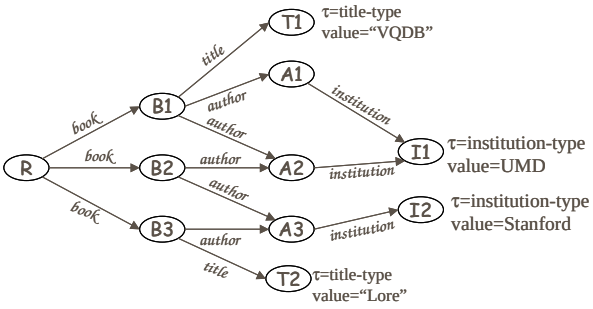


Figure 1: A semistructured instance for a bibliographic domain.

probabilistic semistructured databases. We define the operations of select, project and Cartesian product (for join, renaming, union and intersection, we plan a longer version of the paper). In Section 6, we give algorithms which exploit the local semantics, and result in efficient computation of the results of these algebraic operations. In Section 7, we present experimental results that evaluate the efficiency of the algorithms we have proposed. Due to space reasons, we do not include proofs of all results in this paper.

2 Motivating Example

As our running example, we will use a bibliographic application. This example is rather simple, but we assume it will be accessible to all readers. In this case, we assume that the uncertainty arises from the information extraction techniques used to construct the bibliography. Consider a citation index such as Citeseer [6] or DBLP [7]. In Citeseer, the indexes are created by crawling the web, and operations include parsing postscript and PDF documents. Often, there will be uncertainty over the existence of a reference (have we correctly identified a bibliographic reference?), the type of the reference (is the reference a conference paper, a journal article or a book?), the existence of subfields of the reference such as author, title and year, the identity of the author (does Hung refer to Edward Hung or Sheung-lun Hung?). In such environments, uncertainty abounds.

Semistructured data is a natural way to store such data because for an application of this kind, we have some idea of what the structure of data looks like (e.g. the general hierarchical structure alluded to above). However, semistructured data models do not provide support for uncertainty over the relationships in an instance. In this paper, we will extend this model to naturally store the uncertainty that we have about the structure of the instance as well. Besides, we will see how our algebraic operations and query can handle the following situations:

1. We want to know the authors of all books but we are not interested in the institutions of the authors or the titles of books. However, we want to keep the result in the way that further enquiries (e.g., about probabilities) can be made on it.
2. Now we know that a particular book surely exists. What will the updated probabilistic instance become?
3. We have two probabilistic instances (e.g., the information were collected by two different systems) about books of two different areas and we want to combine them into one.
4. We want to know the probability that a particular author exists.

3 Probabilistic Semistructured Data Model

In this section, we introduce the probabilistic semistructured data (PSD) model. We start by introducing syntactic definitions for this data model, and then we introduce formal semantics. We first review the definition of a semistructured data (SD) model and then we introduce the syntax for the PSD probabilistic semistructured data model.

3.1 Semistructured Data Model

We start by recalling some simple graph concepts.

Definition 3.1 *Let V be a finite set (of vertices), $E \subseteq V \times V$ be a set (of edges) and $\ell : E \to \mathcal{L}$ be a mapping from edges to a set $\mathcal{L}$ of strings called* labels *. The triple $G = (V, E, \ell)$ is an* **edge labeled directed graph**.

Definition 3.2 *Suppose $G = (V, E, \ell)$ is any rooted, edge-labeled directed graph. For $o \in V$:*

- *The* **children** *of o is the set $\mathsf{C}(o) = \{o' \mid (o, o') \in E\}$.*
- *The parents of o, $\mathsf{parents}(o)$, is the set $\{o' \mid (o', o) \in E\}$.*
- *The descendants of o is the set $\mathsf{des}(o) = \{o' \mid$ there is a directed path from o to o' in $G\}$.*
- *The non-descendants of o is the set $\mathsf{non\text{-}des}(o) = \{o' | o' \in V \wedge o' \notin \mathsf{des}(o) \cup \{o\}\}$.*
- *We use $\mathsf{lch}(o, l)$ to denote the set of children of o with label l. More formally,*

$$\mathsf{lch}(o, l) = \{o' \mid (o, o') \in E \wedge \ell(o, o') = l\}.$$

- *A vertex o is called a* **leaf** *iff $\mathsf{C}(o) = \emptyset$.*

It is important to note that our graphs are not restricted to trees— in fact, the above definition allows cycles. However, in our probabilistic semistructured data model, we will restrict attention to directed acyclic graphs.

Definition 3.3 *A* **semistructured instance** *$\mathcal{S}$ over a set of objects $\mathcal{O}$, a set of labels $\mathcal{L}$, and a set of types $\mathcal{T}$ is a 5-tuple $\mathcal{S} = (V, E, \ell, \tau, \mathsf{val})$ where:*

1. $G = (V, E, \ell)$ *is a rooted, directed graph where* $V \subseteq \mathcal{O}$, $E \subseteq V \times V$ *and* $\ell : E \rightarrow \mathcal{L}$;
2. τ *associates a type in* $\mathcal{T}$ *with each leaf object* o *in* G.
3. val *associates a value in the domain* $\mathsf{dom}(\tau(o))$ *with each leaf object* o.

We illustrate the above definition through an example from the bibliographic domain.

Example 3.1 *Figure 1 shows a graph representing a part of the bibliographic domain. The instance is defined over the set of objects* $\mathcal{O} = \{R, B1, B2, B3, T1, T2, A1, A2, A3, I1, I2\}$. *The set of labels is* $\mathcal{L} = \{book, title, author, institution\}$. *There are two types, title-type and instition-type, with domains given by:* $\mathsf{dom}(\textit{title-type}) = \{\textit{VQDB}, \textit{Lore}\}$ *and* $\mathsf{dom}(\textit{institution-type}) = \{\textit{Stanford}, \textit{UMD}\}$. *The graph shows that the relationships between the objects in the domain and the types and values of the leaves.*

3.2 The PSD Probabilistic Data Model

In this section, we develop the basic syntax of the PSD probabilistic data model. However, before defining the important concept of a probabilistic instance, we need some intermediate concepts.

A central notion, that allows us to provide coherent probabilistic semantics, is that of a weak instance. A weak instance describes the objects that can occur in a semistructured instance, the labels that can occur on the edges in an instance and constraints on the number of children an object might have. We will later define a probabilistic instance to be a weak instance with some probabilistic attributes.

Definition 3.4 *A* **weak instance** $\mathcal{W}$ *with respect to* $\mathcal{O}$, $\mathcal{L}$ *and* $\mathcal{T}$ *is a* 5*-tuple* $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ *where:*

1. $V \subseteq \mathcal{O}$.
2. *For each object* $o \in V$ *and each label* $l \in \mathcal{L}$, $\mathsf{lch}(o, l)$ *specifies the set of objects that* **may** *be children of* o *with label* l.
3. τ *associates a type in* $\mathcal{T}$ *with each leaf vertex.*
4. val *associates a value in* $dom(\tau(o))$ *with each leaf object* o.
5. card *is mapping which constrains the number of children with a given label* l. card *associates with each object* $o \in V$ *and each label* $l \in \mathcal{L}$, *an integer-valued interval function,* $\mathsf{card}(o, l) = [min, max]$, *where* $min \geq 0$, *and* $max \geq min$. *We use* $\mathsf{card}(object, l).min$ *and* $\mathsf{card}(object, l).max$ *to refer to the lower and upper bounds respectively.*

A weak instance implicitly defines, for each object and each label, a set of potential sets of children. Consider the following example.

Example 3.2 *Consider a weak instance with* $V = \{R, B1, B2, B3, T1, T2, A1, A2, A3, I1, I2\}$. *We may have* $\mathsf{lch}(R, book) = \{B1, B2, B3\}$ *indicating that B1 and B2 are possible book-children of R. Likewise, we may have* $\mathsf{lch}(B1, author) = \{A1, A2\}$. *If* $\mathsf{card}(B1, author) = [1, 2]$, *then B1 can have between one and two authors. The set of possible* author*-children of B1 is thus* $\{\{A1\}, \{A2\}, \{A1, A2\}\}$. *Likewise, if* $\mathsf{card}(A1, institution) = [1, 1]$ *then A1 must have exactly one (primary) institution.*

We formalize the reasoning in the above example below.

Definition 3.5 *Suppose* $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ *is a weak instance and* $o \in V$ *and* l *is a label. A set* c *of objects in* V *is a* **potential** l*-child set of* o *w.r.t. the above weak instance iff:*

1. *If* $o' \in \mathsf{c}$ *then* $o' \in \mathsf{lch}(o, l)$ *and*
2. *The cardinality of* c *lies in the closed interval* $\mathsf{card}(o, l)$.

We use the notation $\mathsf{PL}(o, l)$ *to denote the set of all potential* l*-child sets of* o.

We are now in a position to define the potential children of an object o.

Definition 3.6 *Suppose* $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ *is a weak instance and* $o \in V$. *A* **potential child set** *of* o *is any set* Q *of subsets of* V *such that* $Q = \bigcup H$ *where* H *is a hitting set*[1] *of* $\{\mathsf{PL}(o, l) \mid (\exists o')o' \in \mathsf{lch}(o, l)\}$. *We use* $\mathsf{PC}(o)$ *to denote the set of all potential child sets of* o *w.r.t. a weak instance.*

Once a weak instance is fixed, $\mathsf{PC}(o)$ is well defined for each o. We will use this to define the *weak instance graph* below. We will need this in our definition of a probabilistic instance.

Definition 3.7 *Given a weak instance* $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$, *the* **weak instance graph**, $\mathcal{G}_{\mathcal{W}} = (V, E)$, *is a graph over the same set of nodes* V, *and for each pair of nodes* o *and* o', *there is an edge from* o *to* o' *iff* $\exists c \in \mathsf{PC}(o)$ *such that* $o' \in c$.

Before we define a probabilistic instance, let us first introduce the notion of a local probability model for a set of children of an object. We adopt the framework of classical probability theory so that the sum of the probabilities of all potential child sets equals 1.

[1] Suppose $\mathbf{S} = \{S_1, \ldots, S_n\}$ where each S_i is a set. A ***hitting set*** for $\mathbf{S}$ is a set H such that (i) for all $1 \leq i \leq n$, $H \cap S_i \neq \emptyset$ and (ii) there is no $H' \subset H$ satisfying condition (i).

Definition 3.8 *Suppose* $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ *is a weak instance. Let* $o \in V$ *be a non-leaf object. An* **object probability function** *(OPF for short) for* o *w.r.t.* $\mathcal{W}$ *is a mapping* $\omega : \mathsf{PC}(o) \rightarrow [0, 1]$ *such that OPF is a legal probability distribution, i.e.,* $\Sigma_{\mathsf{c} \in \mathsf{PC}(o)} \omega(\mathsf{c}) = 1$.

Definition 3.9 *Suppose* $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ *is a weak instance. Let* $o \in V$ *be a leaf object. A* **value probability function** *(VPF for short) for* o *w.r.t.* $\mathcal{W}$ *is a mapping* $\omega : \mathsf{dom}(\tau(o)) \rightarrow [0, 1]$ *such that VPF is a legal probability distribution, i.e.,* $\Sigma_{\mathsf{v} \in \mathsf{dom}(\tau(o))} \omega(\mathsf{v}) = 1$.

An object probability function provides the model theory needed to study a single non-leaf object (and its children) in a probabilistic instance to be defined later. It defines the probability of a set of children of an object existing *given* that the parent object exists. Thus it is the conditional probability of a set of children, given their parent exists in the semistructured instance. Similarly, the value probability function provides the model theory needed to study a leaf object, and defines a distribution over values for the object.

Definition 3.10 *Suppose* $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ *is a weak instance. A* **local interpretation** *is a mapping* $\wp$ *from the set of objects* $o \in V$ *to local probability functions. For non-leaf objects,* $\wp(o)$ *returns an OPF, and for leaf objects,* $\wp(o)$ *returns a VPF.*

Intuitively, a local interpretation specifies, for each object in the weak instance, a local probability function.

Definition 3.11 *A* **probabilistic instance** $\mathcal{I}$ *is a* 6*-tuple* $\mathcal{I} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \wp)$ *where:*

1. $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ *is a weak instance and*
2. $\wp$ *is a local interpretation.*

A probabilistic instance consists of a weak instance, together with probability associated with each potential child of each object in the weak instance.

Example 3.3 *Figure 2 shows a very simple probabilistic instance. The set* $\mathcal{O}$ *of objects is the same as in our earlier* PSD *example. The figure shows the potential* lch *of each object; for example,* $\mathsf{lch}(B1, author) = \{A1, A2\}$. *The cardinality constraints are also shown in the figure; for example, object B1 can have 1 to 2 authors and 0 to 1 titles. The tables on the right of Figure 2 shows the local probability models for each of the objects. The tables show the probability of each potential child of an object. For example, if B2 exists, the probability A1 is one of its authors is* 0.8.

The components $\mathcal{O}, \mathcal{L}, \mathcal{T}$ of a probabilistic instance are identical to those in a semistructured instance. However, in a probabilistic instance, there is uncertainty over:

- The number of sub-objects of an object o;
- The identity of the sub-objects.
- The values of the leaf objects.

This uncertainty is captured through the function $\wp(o)$. We may define $\wp(o)$ more compactly, in the case where there are some symmetries or independence constraints that can be exploited in the representation. For example, if the occurrence of each category of labeled objects is independent, then we can simply specify a probability for each subset of objects with the same label and compute the joint probability as the product of the individual probabilities. For example, if the existence of author and title objects is independent, then we only need to specify a distribution over authors and a distribution over titles. Furthermore, in some domains it may be the case that some objects are indistiguishable. For example in an object recognition system, we may not be able to distinguish between vehicles. Then if we have two vehicles, vehicle1 and vehicle2, and a bridge bridge1 in a scene S1, we may not be able to distinguish between a scene that has a bridge1 and vehicle1 in it from a scene that has bridge1 and vehicle2 in it. In this case, $\wp(S1)(\{bridge1, vehicle1\}) = \wp(S1)(\{bridge1, vehicle2\})$. The semantics of the model we have proposed is fully general, in that we can have arbitrary distributions over the sets of children of an object. However in the case where there is additional structure that can be exploited, we plan to allow compact representations of the distributions and make use of the additional structure effectively when answering queries.

4 Semantics

In this section, we develop the semantics of probabilistic semistructured databases. A PSD model defines a distribution over possible semistructured instances; we can represent our uncertainty about the world as a distribution over possible semistructured instances. A probabilistic instance *implicitly* is shorthand for a set of (possible) semistructured instances—these are the only instances that are *compatible* with the information we do have about the actual world state which is defined by our weak instance. We begin by defining the notion of the set of semistructured instances that are compatible with a weak instance.

Definition 4.1 *Let* $\mathcal{S} = (V_{\mathcal{S}}, E, \ell, \tau_{\mathcal{S}}, \mathsf{val}_{\mathcal{S}})$ *be a semistructured instance over a set of objects* $\mathcal{O}$*, a set of labels* $\mathcal{L}$ *and a set of types* $\mathcal{T}$ *and let* $\mathcal{W} = (V_{\mathcal{W}}, \mathsf{lch}_{\mathcal{W}}, \tau_{\mathcal{W}}, \mathsf{val}_{\mathcal{W}}, \mathsf{card})$ *be a weak instance.* $\mathcal{S}$ *is compatible with* $\mathcal{W}$ *if for each* o *in* $V_{\mathcal{S}}$*:*

- *The root of* $\mathcal{W}$ *is in* $\mathcal{S}$.
- o *is also in* $V_{\mathcal{W}}$.
- *If* o *is a leaf in* $\mathcal{S}$*, then* o *is also a leaf in* $\mathcal{W}$*,* $\tau_{\mathcal{S}}(o) = \tau_{\mathcal{W}}(o)$ *and and* $\mathsf{val}_{\mathcal{S}}(o) \in \tau_{\mathcal{S}}(o)$.

o	l	$\mathsf{lch}(o,l)$ [a]
R	book	{B1, B2, B3}
B1	title	{T1}
B1	author	{A1, A2}
B2	author	{ A1, A2, A3}
B3	title	{T2}
B3	author	{A3}
A1	institution	{I1}
A2	institution	{I1, I2}
A3	institution	{I2}

o	l	$\mathsf{card}(o,l)$
R	book	[2,3]
B1	author	[1,2]
B1	title	[0,1]
B2	author	[2,2]
B3	author	[1,1]
B3	title	[1,1]
A1	institution	[0,1]
A2	institution	[1,1]
A3	institution	[1,1]

[a] We only show non-empty lch

$c \in \mathsf{PC}(R)$	$\wp(R)(c)$
{B1, B2}	0.2
{B1, B3}	0.2
{B2, B3}	0.2
{B1, B2, B3}	0.4

$c \in \mathsf{PC}(B1)$	$\wp(B1)(c)$
{A1}	0.3
{A1, T1}	0.35
{A2}	0.1
{A2, T1}	0.15
{A1, A2}	0.05
{A1, A2, T1}	0.05

$c \in \mathsf{PC}(B2)$	$\wp(B2)(c)$
{A1, A2}	0.4
{A1, A3}	0.4
{A2, A3}	0.2

$c \in \mathsf{PC}(B3)$	$\wp(B3)(c)$
{A3, T2}	1.0

$c \in \mathsf{PC}(A1)$	$\wp(A1)(c)$
{}	0.2
{I1}	0.8

$c \in \mathsf{PC}(A2)$	$\wp(A2)(c)$
{I1}	0.5
{I2}	0.5

$c \in \mathsf{PC}(A3)$	$\wp(A3)(c)$
{I2}	1.0

Figure 2: A probabilistic instance for the bibliographic domain.

- *If o is not a leaf in $\mathcal{S}$ then*
  - *For each edge (o, o') with label l in $\mathcal{S}$, $o' \in \mathsf{lch}_{\mathcal{W}}(o, l)$,*
  - *For each label $l \in \mathcal{L}$, let $k = |\{o' | (o, o') \in E \wedge \ell(E) = l\}|$, then $card(o, l).min \leq k \leq card(o, l).max$.*

We use $Domain(\mathcal{W})$ to denote the set of all semistructured instances that are compatible with a weak instance $\mathcal{W}$. Similarly, for a probabilistic instance $\mathcal{I} = (V, \mathsf{lch}_{\mathcal{I}}, \tau_{\mathcal{I}}, \mathsf{val}_{\mathcal{I}}, \mathsf{card}, \wp)$, we use $Domain(\mathcal{I})$ to denote the set of all semistructured instances that are compatible with $\mathcal{I}$'s associated weak instance $\mathcal{W} = (V, \mathsf{lch}_{\mathcal{I}}, \tau_{\mathcal{I}}, \mathsf{val}_{\mathcal{I}}, \mathsf{card})$.

Next we can define a *global interpretation* based on the set of a compatible instances of a weak instance.

Definition 4.2 *Suppose we have a weak instance $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$. A* **global interpretation** *$\mathcal{P}$ is a mapping from $Domain(\mathcal{W})$ to $[0, 1]$ such that $\Sigma_{S \in Domain(\mathcal{W})} \mathcal{P}(S) = 1$.*

Intuitively, a global interpretation is a distribution over the set semistructured instances compatible with a weak instance.

Recall that a local interpretation defines the local semantics for an object. In addition, it enables us to define the global semantics for our model. First we must put an acyclicity requirement on the weack instance graph. This is required to ensure that our probabilistic model is coherent.

Definition 4.3 *Let $\mathcal{W} = (V_{\mathcal{W}}, \mathsf{lch}_{\mathcal{W}}, \tau_{\mathcal{W}}, \mathsf{val}_{\mathcal{W}}, \mathsf{card})$ be a weak instance. $\mathcal{W}$ is* **acyclic** *if its associated weak instance graph $\mathcal{G}_{\mathcal{W}}$ is acyclic.*

Given a probabilistic instance $\mathcal{I}$ over an acyclic weak instance $\mathcal{W}$, the probability of any particular instance can be computed from the OPF and VPF entries corresponding to each object in the instance and its children. We are now going to define the relationship between the local interpretation and the global interpretation.

Definition 4.4 *Let $\wp$ be local interpretation for a weak instance $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$. Then $\mathcal{P}_{\wp}$ returns a function defined as follows: for any instance $S \in Domain(\mathcal{W})$, $\mathcal{P}_{\wp}(S) = \prod_{o \in S} \wp(o)(\mathsf{c}_S(o))$, where if o is not a leaf in $\mathcal{W}$, then $\mathsf{c}_{\mathcal{S}}(o) = \{o' | (o, o') \in E\}$, i.e., the set of children of o in instance $\mathcal{S}$; otherwise, $\mathsf{c}_{\mathcal{S}}(o) = \mathsf{val}_{\mathcal{S}}(o)$, i.e., the value of o in instance $\mathcal{S}$.*

In order to use this definition for the semantics of our model, we must first show that the above function is in fact a legal global interpretation.

Theorem 1 *Suppose $\wp$ is a local interpretation for a weak instance $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$. Then $\mathcal{P}_{\wp}$ is a global interpretation for $\mathcal{W}$.*

Example 4.1 *Consider S_1 in Figure 3 and the probabilistic semistructured instance from Figure 2.*

$$\begin{aligned} P(S_1) &= P(B1, B2 \mid R)\, P(A1, T1 \mid B1) \\ &\quad P(A1, A2 \mid B2)\, P(I1 \mid A1)\, P(I1 \mid A2) \\ &= 0.2 \cdot 0.35 \cdot 0.4 \cdot 0.8 \cdot 0.5 \\ &= 0.00448 \end{aligned}$$

An important question is whether we can go the other way: from a global interpretation, can we find a local interpretation for a weak instance $\mathcal{W}(V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$? It

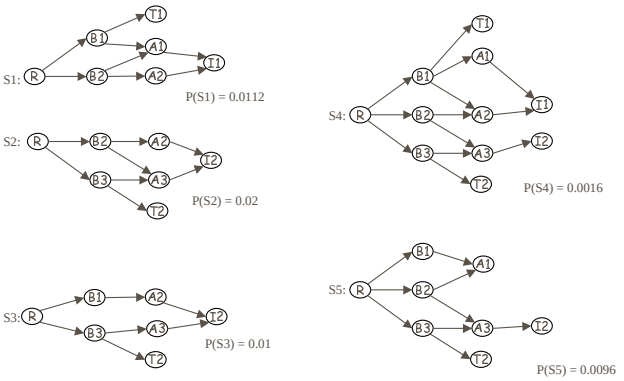


Figure 3: Some of semistructured instances compatible with the probabilistic instance in Figure 2.

turns out that we can **if** the global interpretation can be factored in a manner consistent with the structure constraints imposed by $\mathcal{W}(V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$.

One way to ensure this is to impose a set of independence constraints on the distribution $\mathcal{P}$.

Definition 4.5 *Suppose $\mathcal{P}$ is a global interpretation and $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ is a weak instance. $\mathcal{P}$* **satisfies** *$\mathcal{W}$ iff for every non-leaf object $o \in V$ and each $c \in \mathsf{PC}(o)$ (and for every leaf object $o \in V$ and each $c \in \mathsf{dom}(\tau(o))$), it is the case that $\mathcal{P}(c|\mathsf{non\text{-}des}_{\mathcal{W}}(o)) = \mathcal{P}(c)$ where $\mathsf{non\text{-}des}_{\mathcal{W}}(o)$ are the nondescendants of o in the weak instance graph $\mathcal{G}_{\mathcal{W}}$.*

In other words, given that o occurs in the instance, the probability of any potential children c of o is independent of the nondescendants of o in the instance.

Theorem 2 *Suppose $\mathcal{P}$ is a global interpretation for a weak instance $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$. If $\mathcal{P}$ satisfies $\mathcal{W}$, then there exists a local interpretation $\wp$ such that $\mathcal{P}_\wp = \mathcal{P}$.*

5 Probabilistic Semistructured Algebra

This section describes several algebraic operations on probabilistic instances. Due to space limitations, we cannot present the full set or all details of operations included in this paper. For convenience, we use the term *an instance* to refer to *a probabilistic instance* when there is no ambiguity.

Relational algebra is based on relation names and attribute names while our algebra is based on probabilistic instance names and *path expressions*. The definition of path expressions is a variation of the standard definition[2].

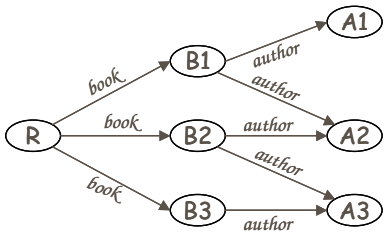


Figure 4: The result of the ancestor projection on the semistructured instance in Figure 1 with the path expressions R.*book.author*.

Definition 5.1 *An* **edge sequence** *is a sequence $l_1.....l_n$, where l_i are labels of edges. A* **path expression** *$p = r.l_1.....l_n$ is an object (oid) r, followed by a (possibly empty) edge sequence $l_1.....l_n$; p denotes the set of objects that can be reached via the sequence of edges with labels $l_1.....l_n$.*

A path expression is used to locate objects in an instance. $o \in p$ iff there is a path p to reach o. For example, in the example instance in Figure 1, $A2 \in R.book.author$ because there is a path from R to reach $A2$ through a path that is labeled $book.author$.

In this section we will define the following operators: projection, selection, and cross product (join can be defined in terms of these operations in the standard way). For each operator, we will first describe how it works on an ordinary semistructured instance, then on a probabilistic instance.

5.1 Projection

We propose several projection operators including ancestor projection, descendant projection and single projection. Here we give details only on the first. The *ancestor projection* operation extracts subgraphs composed of objects located by a path expression and those objects' ancestors up to the root. Note that only those ancestors and edges on the paths to those objects are extracted.

Example 5.1 *Consider the semistructured instance shown earlier in Figure 1. Suppose we have a path expression $R.book.author$, then the ancestor projection will first locate the set V' of objects that satisfy the path expression. Here, $V' = \{A1, A2, A3\}$. Then it will add into the set V' those objects that are on the path from the root to the objects in V' (here, they are B1, B2 and B3), and the root of the instance (R). Now, V' is the set of the objects in our new instance. For the edges in the new instance, we will connect*

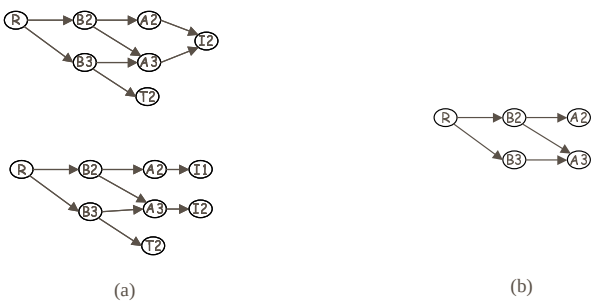


Figure 5: Given the two instances S1 and S2 on the left, the ancestor projection of R.*book.author* gives the same resulting semistructured instance shown as S3 on the right. Because these are the only two compatible instances that produce this result, the probability of the result is simply the sum of the two probabilities, $P(S3) = P(S1) + P(S2)$.

those objects which are also connected in the original instance. Finally, the labels of the edges in the new instance will be identical to those in the old instance. The resulting instance is shown in Figure 4.

Definition 5.2 *[ancestor projection (Λ)] Suppose $G = (V, E)$ is an instance, r is the root of G and p is a path expression. The* ancestor-projection *of G on p, denoted $\Lambda_p(G) = (V', E')$, is defined as follows:*

- $V' = \{o \mid o \in V \wedge (o \in p \vee \exists o' \in V,$ *edge sequences* s, s' $(p = r.s.s' \wedge o \in r.s \wedge o' \in o.s'))\} \cup \{r\}$
- $E' = \{(o, o') \mid (o, o') \in E \wedge o, o' \in V' \wedge$ *$\exists$ edge sequences s, s' a label l and an object o''* $\in V'$ $(p = r.s.l.s' \wedge o \in r.s \wedge o' \in o.l \wedge o'' \in o'.s')\}$
- $\forall (a, b) \in E'$ $(\ell'(a, b) \leftarrow \ell(a, b))$

We have seen how ancestor projection works on a semistructured instance. Now, we are going to see what it means for a probabilistic instance. For example, recall the first situation we want to handle in Section 2. We can use an ancestor projection with a path expression $R.book.author$ on the probabilistic instance. The result keeps the authors and their ancestors, which can be used to deduce the global probabilities of compatible instances or the probability of a particular author in the future. Recall that from a probabilistic instance, we can obtain a set of compatible instances and a distribution over the probability of each of the compatible instances. We can perform the ancestor projection on each of the compatible instances to obtain a resulting set of semistructured instances. We then combine the probabilities of identical instances by summing up them.

For example, after the ancestor projection with a path expression $R.book.author$ on the set of instances S1 and S2 in Figure 5(a), we will have $S3$ as the result, shown in Figure 5(b). We can combine the probabilities of $S1, S2$, i.e., $P(S1) + P(S2)$, which is the probability of the resulting instance.

Definition 5.3 *Suppose $\mathcal{I} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \wp)$ is a probabilistic instance, $\mathcal{P} = \tilde{W}(\wp)$ and p is a path expression. Then, the probabilities of the result of the ancestor projection with path expression p on $\mathcal{I}$ are defined as follows: for every $\mathcal{S} \in Domain(\Lambda_p(\mathcal{I}))$, the probability of $\mathcal{S}$ is $\sum_{\mathcal{S}'' \in Domain(\mathcal{I})\ s.t.\ \Lambda_p(\mathcal{S}'')=\mathcal{S}} \mathcal{P}(\mathcal{S}'')$.*

In reality, there are much more efficient ways of computing the projection. We will describe them in Section 6.

5.2 Selection

In this section we will describe the selection operation. We define two types of selection conditions, an *object selection condition* and a *value selection condition*.

Definition 5.4 (object selection condition) *An* object selection condition *is of the form of $p = o$ where p is a path expression starting from the root and o is an object id.*

Definition 5.5 (value selection condition) *A* value selection condition *is of the form of $\mathsf{val}(p) = v$ where p is a path expression starting from the root to some leaf and v is a possible value for $o \in p$, i.e., $v \in \mathsf{dom}(\tau(o))$.*

Besides conditions based on object and value, other kinds of selection conditions with comparisons based on, for example, cardinality, OPF or VPF , work in a similar way and have similar semantics.

With a given selection condition sc and a probabilistic instance $\mathcal{I}$, the global approach will give a set of semistructured instances (with normalized probabilities) compatible with the selection operation. The global approach works as follows: among the set of instances compatible with the probabilistic instance $\mathcal{I}$, only those instances satisfying the selection condition sc will be selected; then their resulting probabilities will be obtained by normalizing their original probabilities using the formula in the following definition.

Definition 5.6 (selection (σ)) *Suppose $\mathcal{I} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \wp)$ is a probabilistic instance, sc is a selection condition. Let $Domain(\sigma_{sc}(\mathcal{I})) = \{\mathcal{S} \in Domain(\mathcal{I}) \mid \mathcal{S}$ satisfies $sc\}$ be the set of compatible instances satisfying the selection condition. Then, $\forall \mathcal{S} \in Domain(\sigma_{sc}(\mathcal{I}))$, $\mathcal{P}'(\mathcal{S}) = \frac{\mathcal{P}(\mathcal{S})}{\sum_{\mathcal{S}' \in Domain(\sigma_{sc}(\mathcal{I}))} \mathcal{P}(\mathcal{S}')}$*

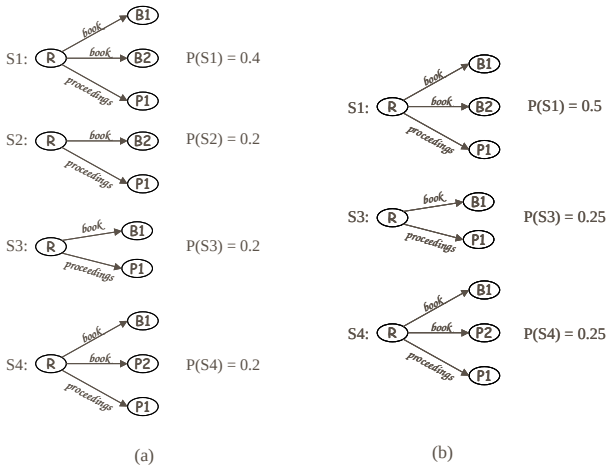


Figure 6: (a) The set of compatible instances of a probabilistic instance along with their probabilities. (b) The result of the selection R.*book* = *B1*.

Example 5.2 *Suppose we have a (simplified) probabilistic instance with the $\mathcal{P}$ shown in Figure 6(a). Recall the second situation in Section 2 and suppose now we know the book $B1$ surely exists. So, how will the probabilities be affected? The result of the selection $R.book = B1$ is shown in Figure 6(b). The set of compatible instances are shown, along with their updated probabilitites. Among the four compatible instances shown in Figure 6(a), only $S1, S3, S4$ satisfy the selection condition. Then we normalize the probabilities of the selected instances as follows. For example, consider $S1$:* $\mathcal{P}'(S1) = \frac{\mathcal{P}(S1)}{\mathcal{P}(S1)+\mathcal{P}(S3)+\mathcal{P}(S4)} = \frac{0.4}{0.4+0.2+0.2} = 0.4$

5.3 Cartesian Product

In Section 2, we mentioned a situation where we want to combine two probabilistic instances into one. In this section we will briefly describe how to do that by the operation of Cartesian product. On two semistructured instances, the Cartesian product operation simply merges the two roots of the two input instances. Similar to the Cartesian product of relational algebra, objects with identical object ids in the two instances need to be renamed. For simplicity, we assume that the object ids are unique (after renaming, if necessary). Readers may wonder why we do not simply add a new root as the parent of the two original roots. The reason is to ensure that the same path expressions can be applicable on the two input semistructured instances before and after the Cartesian product. The Cartesian product of two probabilistic instances is defined similarly. However, we need to specify how to combine the probabilities. Here we make an independence assumption and assume that the probability of the product is the product of the component probabilities. In a manner similar to what is done for projection, the probabilities of identical resulting structures are combined.

Definition 5.7 (Cartesian product ($\times$)) *Suppose $\mathcal{I} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \wp), \mathcal{I}' = (V', \mathsf{lch}', \tau', \mathsf{val}', \mathsf{card}', \wp')$ are two probabilistic instances, r, r' are the roots of $\mathcal{I}, \mathcal{I}'$. The* Cartesian product *of $\mathcal{I}, \mathcal{I}'$, denoted $\mathcal{I} \times \mathcal{I}' = \mathcal{I}'' = (V'', \mathsf{lch}'', \tau'', \mathsf{val}'', \mathsf{card}'', \wp'')$, rooted at r'', is defined as follows:*

- $V'' = (V \cup V' \cup \{r''\}) - \{r\} - \{r'\}$, $\tau'' = \tau \cup \tau'$, $\mathsf{val}'' = \mathsf{val} \cup \mathsf{val}'$.
- $\mathsf{lch}'' = \mathsf{lch} \cup \mathsf{lch}'$ *and* $\mathsf{card}'' = \mathsf{card} \cup \mathsf{card}'$ *with the modification such that the two old roots are merged into r'' with all children of r, r' become the children of r''.*
- $\forall b \in \mathsf{lch}''(a, l)$ *(for every label l),*
  - *if $a = r''$, then: if $b \in V$, then $(\ell''(a, b) \leftarrow \ell(r, b))$; otherwise, $(\ell''(a, b) \leftarrow \ell'(r', b))$;*
  - *else: if $b \in V$, then $(\ell''(a, b) \leftarrow \ell(a, b))$; otherwise, $(\ell''(a, b) \leftarrow \ell'(a, b))$.*
- *$\wp''$ is defined such that $\forall o \in V$, $\wp''(o) = \wp(o)$; $\forall o \in V'$, $\wp''(o) = \wp'(o)$. The root r'' requires a special treatment: $\forall \mathsf{c}'' \in \mathsf{PC}(r'')$ such that $c'' = c \cup c'$ where $\mathsf{c} \in \mathsf{PC}(r), \mathsf{c}' \in \mathsf{PC}(r')$, $\wp''(r'')(\mathsf{c}'') = \wp(r)(\mathsf{c}) \times \wp'(r')(\mathsf{c}')$.*

6 Probabilistic XML (PXML)

This section describes PXML, an efficient implementation of the algebraic operations defined in the previous section. We begin by describing algorithms for operations such as ancestor projection which return an updated probabilistic instance. Next we describe queries that simply return the probability that a selection condition is satisfied in the probabilistic instance.

In general, as we saw in Theorem 1, there is a mapping between a probabilistic instance and a Bayesian network. For any query, there is a mapping to an equivalent query in the Bayesian network. Inference in Bayesian networks has been studied extensively [21, 17, 8], and many off the shelf implementations are available. In general, if the network is tree structured, the inference will be linear in the number of nodes in the network. If the network is not a tree, the complexity depends on the connectivity of the graph and the induced tree width of the graph. In practice, if the graph is not highly connected, as in our example, the inference is quite efficient. And, regardless of the structure, the inference algorithms are significantly more efficient than naively

computing the probability by marginalizing over all of the compatible instances.

Rather than presenting the generic inference algorithm here, we show how to compute a variety of simple queries. Here we give an efficient algorithm with the assumption that all compatible instances are tree-structured. Note that this is not a requirement, but it does simplify the algorithms.

6.1 Ancestor Projection

As we saw in the algebra section, ancestor projection on a probabilistic instance results in a new probabilistic instance, with the probability of an instance S' in the projection computed as the sum of the probabilities of the instances that map to S' in the original probabilistic instance.

We can treat the probabilistic instance as an ordinary semistructured instance and perform ancestor projection on it and update the card and $\wp$. The update of the $\wp$ and card are done starting from the immediate parents of leaves. The update is bottom up; it will be performed on an object only if the updates have been done on all of its children.

Here, we use o_i to denote the non-leaf object whose $\wp(o_i)$ and card are to be updated. We denote the original set of children before projection as $\mathsf{C}(o_i)$, the new set of children after projection as $\mathsf{C}'(o_i)$, and $C_d = \mathsf{C}(o_i) - \mathsf{C}'(o_i)$. Similary, we use $\wp'$ and card' to denote the new local interpretation and cardinality.

- **Marginalization.** First, consider the immediate parent of a leaf. Intuitively, for each $c' \subseteq \mathsf{C}'(o_i)$, we project all the children in the original, $c \subseteq \mathsf{C}(o_i)$, where c' is the result of projection of c (after removing the deleted children), to c':

$$\wp'(o_i)(c') = \sum_{d \subseteq C_d \text{ s.t. } c=(c' \cup d) \in \mathsf{PC}(o_i)} \wp(o_i)(c' \cup d).$$

  Normalization.

  A non-leaf object (except the root) in the result of ancestor projection should not exist in a compatible instance if none of its children exists in the compatible instance (by the definition of ancestor projection). We will compute ϵ_{o_i}, the probability that o_i has some child still existing in the result of the ancestor projection:

$$\epsilon_{o_i} = \sum_{c' \in \mathsf{PC}'(o_i) \wedge c' \neq \emptyset} \wp'(o_i)(c').$$

  Then, we renormalize the probabilities so that $\wp'(o_i)(c)$ will represent the conditional probability of o_i having children c given the condition that some of the children exist. We set $\wp'(o_i)(\{\}) = 0$ and do the normalization as follows: $\forall c \subseteq \mathsf{C}'(o_i)$,

$$\wp'(o_i)(c) = \frac{\wp'(o_i)(c)}{\epsilon_{o_i}}$$

- For other non-leaf object (except the root), for each $c' \in \mathsf{C}'(o_i)$, we project all the children in the original $c \in \mathsf{C}(o_i)$, where c' is a subset of c, to c', and multiply by the probability that each exists:

$$\wp'(o_i)(c') = \sum_{c \in \mathsf{PC}(o_i) \wedge c' \subseteq c} \wp(o_i)(c) \prod_{o_j \in c'} \epsilon_{o_j} \prod_{o_j \in (c - c') \wedge o_j \in \mathsf{PC}'(o_i)} (1 - \epsilon_{o_j}).$$

  As, above, we will record the probability ϵ_{o_i}, set $\wp'(o_i)(\{\}) = 0$ and renormalize the probabilities by dividing by ϵ_{o_i}.
- For the root r, we marginalize as above. However, we do not need to set $\wp'(r)(\{\})$ to 0 and do normalization. In essence, $\wp'(r)(\{\})$ is the probability that a compatible instance in the original has no object satisfying the path expression of the ancestor projection and, as a result, only the root object is returned.

The process of update of card is the same for all non-leaf objects: for an object o_0 and an edge label l_j,

$$\mathsf{card}'(o_0, l_j).lb = \min_{C \subseteq \mathsf{C}'(o_0) \text{ s.t. } \wp'(o_0)(C) > 0} (\text{number of objects in } C \text{ that have edge label } l_j)$$

$$\mathsf{card}'(o_0, l_j).ub = \max_{C \subseteq \mathsf{C}'(o_0) \text{ s.t. } \wp'(o_0)(C) > 0} (\text{number of objects in } C \text{ that have edge label } l_j)$$

6.2 Probabilistic Point Queries

In this section, we will describe an algorithm to compute the probability that an object exists satisfying some constraints on the path to the object.

To begin, we consider computing the probability of a simple object chain. To compute the probability of a simple object chain $c = r.o_1.o_2.\ldots.o_i$, we consider all possible ways that the chain can be achieved:

$$P(c) = \sum_{c_1 \in \mathsf{PC}(r) \wedge o_1 \in c_1} \wp(r)(c_1) \times \sum_{c_2 \in \mathsf{PC}(c_1) \wedge o_2 \in c_2} \wp(o_1)(c_2) \times \cdots \times \sum_{c_i \in \mathsf{PC}(c_{i-1}) \wedge o_i \in c_i} \wp(o_{i-1})(c_i)$$

Next we consider *probabilistic point queries*, which allow us to compute the probability that an object satisfies a path expression. This kind of query can be used to answer the last situation in Section 2: we want to know the probability that a particular author exists.

Definition 6.1 *Given a path expression p and an object o in a probabilistic instance, a probabilistic point query returns the probability that $o \in p$ in a compatible instance.*

Here we assume that $o \in p$ in the probabilistic instance, otherwise it is obvious that the probability must be zero. First, let us define the *path ancestors* of o as all o's ancestors such that for every such ancestor o_a, there exist a path identical to the path expression p from the root to o_a then to o. We notice that if we extract only the object o and its path ancestors from the probabilistic instance, and use the same method described in the previous section to calculate ϵ_r, then ϵ_r will be the answer to this problem. The reason is that the root of the result of the ancestor projection on a compatible instance will have a child if and only if there is some object in that compatible instance satisfying the path expression. Here, because we only keep o and its path ancestors, the root of the result of the ancestor projection on a compatible instance will have a child if and only if o in that compatible instance satisfies the path expression p. Recall the meaning of ϵ_r is that, given that r exists in a compatible instance (which is true always), r still has a child that should exist after the ancestor projection on that compatible instance, so ϵ_r also gives the probability that o satisfies p.

An extension to this problem is to find the probability that there exists some object satisfying a given path expression. We can solve it by keeping all objects satisfying the path expression in the probabilistic instance and their path ancestors and calculate ϵ_r as the answer.

7 PXML Experiments

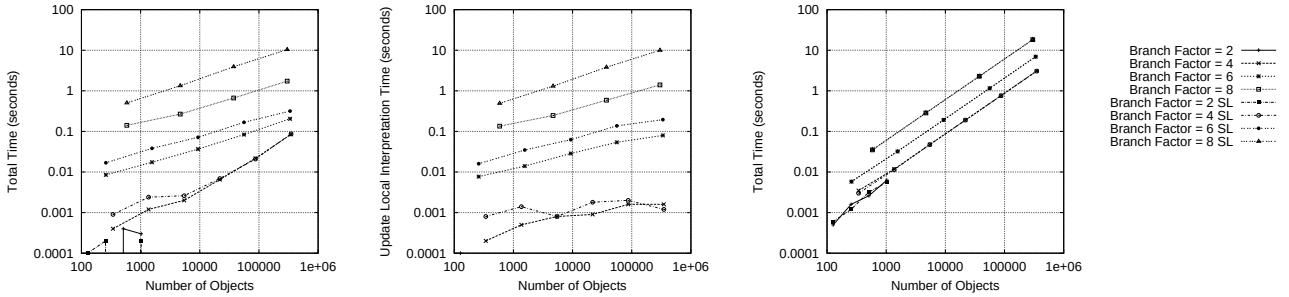
7.1 Experimental Design

We have implemented a prototype system in C on a Dell PowerEdge with 1.13 Ghz PIII processors, 4GB RAM running Linux. We generated probabilistic instances as balanced trees with every non-leaf nodes having the same number of children. Probabilistic instances were generated with the depth (of the tree) ranging from 3 to 9 and the branching factor (the number of children of each non-leaf node) ranging from 2 to 8. We assume that there is no cardinality constraint, so the total number of entries in a local interpretation for each non-leaf object is 2^b where b is the branching factor. There are two kinds of random labeling of the edges. In the *same label* (or SL) labeling, all children of the same parent have the same labels (shown as SL in the figures). The *fully random* (or FR) labeling assigns random labels to all children of the same parent. We evaluated the performance of ancestor projection and selection. We did not evaluate the Cartesian product because it only involves the update of the roots, whose running time is very short and independent of the the size of the instances.

In this paper we include graphs of total query time and the time required to update the local interpretation ($\wp$). The total query time is the sum of the time to make a copy of the input instance, the time to locate objects satisfying a path expression (and the object id of the object to be selected in the case of selection operation), the time to update the structure of the instance (for ancestor projection only), the time to update the local interpretation, and the time to write the resulting instance onto a disk. For each depth, each branching factor and each operation, we generated 10 instances. For each instance, we kept track of labels used by edges of objects in each depth and generated 10 random queries that returned results not only consisting of a root. For example, in an instance of depth 2 where the edges connecting objects of depth 1 to their parents have labels from the set $\{a, b\}$ and the edges connecting objects of depth 2 to their parents have labels from the set $\{c, d\}$, the path expression of an ancestor projection query generated has the form $r.x_1.x_2$ where r is the root id, $x_1 \in \{a, b\}$ and $x_2 \in \{c, d\}$. We accepted this query in the performance measurement in our experiment only if there were objects satisfying the path expression of this query. For each selection query, we generated a path expression p (in the same way) and similarly found a set $SelObj$ of objects satisfying the path expression. The selection queries used have the form $p = o$ where o is an object id selected randomly from $SelObj$. In our experiments, we only consider single path expressions as defined in Definition 5.1. Besides, we set the length of the query (the length of the path expression) equal to the depth of the instance because, according to the definition of ancestor projection and selection, the objects whose depth exceeds the length of the query will not be considered and will not affect the query results and the local interpretation of such objects does not need updating. For each combination of depth and branching factor, we took the average of 100 such queries.

7.2 Performance results

Figure 7 (a) and (b) show the total query processing time and the time of updating $\wp$ in an ancestor projection. By comparing the two graphs, we can see that the time of updating $\wp$ dominates the total query processing time. Figure 7 (b) shows that the time to update is linear to the number of objects, which can be explained by the fact that $\wp(o)$ is updated only once for each object o. Recall that the time to propagate probabilities from children of an object o to o is quadratic in the size of $\wp(o)$. When we fix the number of objects, we can see from Figure 7 (b) that the time increases by a multiple less than 16 when the branching factor increases by 2, i.e. the number of entries in $\wp(o)$ is multiplied by 4. Besides, under the setting of having the same labels for all children of the same parent, the time is longer than the other setting. One possible reason is that in the former setting, there is a higher chance that more objects are located by the path expression, and so there are more objects to be kept whose local interpretations are to be updated. The final note is that the updating time for 299593



(a) (b) (c)

Figure 7: (a) Total query time of ancestor projection, (b) update local interpretation time of ancestor projection and (c) total query time of selection for instances of sizes ranging from 100 to 100000, branching factors ranging from 2 to 8 and two different labeling schemes (SL = same labels for children of the same parent; otherwise, all random labels).

objects and branch factor 8 SL (the top rightmost point) is 10.4s, which seems to be long. However, it is reasonable after considering the fact that about 700 - 5000 objects are kept and about 28000 - 200000 $\wp(o)$ entries are processed.

Figure 7 (c) shows the total query processing time for selection. The result of selection is different from that of ancestor projection as the time to write the result onto the disk dominates the total query processing time (the time of updating $\wp$ only involves less than 0.001 second). The reason is that the structure of the resulting instance does not change after selection. Hence, the amount of data to be written is much larger than the number of objects whose $\wp(o)$ needs to be updated (the number is the same as the depth). The total time is linear in the number of objects and linear in the number of entries in $\wp(o)$ of each object o. The quantity of data to be written is independent of whether SL or FR labelings are used.

8 Related Work

ProTDB proposed by Nierman and Jagadish[19] is similar in spirit to our model, however there are a few important differences. In ProTDB, independent probabilities are assigned to each individual child of an object; PXML supports arbitrary distributions over sets of children. Furthermore, dependencies are required to be tree-structured in ProTDB, whereas PXML allows arbitrary acyclic dependency models. Thus PXML data model subsumes ProTDB data model. In addition, here we prove that the semantics of PXML are probabilistically coherent. Another important difference is the queries supported. There is no direct mapping among our algebra and their query language. For example, in their conjunctive query, given a query pattern tree, they return a set of subtrees (with some modified node probabilities) from the given instance, each with a global probability. There is no direct mapping between their conjunctive query and our ancestor projection because they find subtrees matching the pattern tree, while we use a path expression. Each of their subtrees is restricted to match the query pattern tree and has a fixed structure while our output is a probabilistic instance which implicitly includes many possible structures.

The work of Dekhtyar et al.[9] was the first to deal with probabilities and semistructured data. It appears to be similar to PXML but in fact it is quite different. They introduce a semistructured model to allow us to use an object (semistructured probabilistic object or SPO) to represent the probability table of one or more random variables, the extended context and the extended conditionals. An SPO itself can be represented in a semistructured way, but its main body is just a flat table. It cannot show the semistructured relationship among variables. In contrast, our model is based on the widely used model OEM[20], which allows data to be represented in a truly semistructured manner. We modify the syntax and semantics of the model by introducing cardinality and object probability functions to demonstrate the uncertainty of the number and the identity of objects existing in possible worlds. Every possible world is a semistructured instance compatible with the probabilistic instance. The representation of a possible world (semistructured instance) is the same as the one widely accepted nowadays. However, the model of Dekhtyar et al. cannot do this. Their model also requires random variables to have distinct variable names (or edge labels) (in our model, they are the children connected to their parents with the same edge label). Consequently, their model cannot allow two or more variables with the same variable names (no matter their values are the same or different) in a single possible world. Their model also cannot capture the uncertainty of cardinality. On the other hand, our model can represent their table. For each random variable, define a set of children (with the possible variable values) connected to their parent with the same edge label (set as the variable name). The cardinality associates with the parent object with each label is set

to $[1, 1]$ so that each random variable can have exactly one value in each possible world.

XPath[22] and XQuery[23] are languages that use path expressions (defined in XPath) to extract objects. SAL [4] and TAX [15] are two algebras for semistructured data. The reason that we cannot use XPath, Xquery and SAL directly is that the original parent-children relationships and probabilities associated with them cannot be kept directly in the results since individual objects are selected during the process. However, our algebra uses the well-defined path expressions as a tool to locate the objects we are interested and manipulates the graph structure of semistructured data directly. On the other hand, TAX uses a pattern tree to extract subsets of nodes (called witness trees), one for each embedding of the pattern tree in an input tree (instance). The reason that we cannot use theirs directly is the fixed structure of the result, e.g., fixed number of children, which restricts the representation of the uncertainty in cardinality.

9 Conclusions

We have presented a new probabilistic semistructured data model, given semantics for the model and proven that the semantics are probabilistically coherent. We have presented an algebra for the data model. The algebra has some interesting differences from existing XML algebras. We have shown how queries can be answered efficiently in our system PXML.

The marriage of semistructured model with probabilistic models is a natural pairing. It supports uncertainty not only over the schema but also over the instance. This is particularly useful in the processing of complex, noisy data that abounds in real world domains. Our future work includes extending our model to allow cycles.

Acknowledgement

Edward Hung is supported by the Croucher Foundation Scholarships. Work is funded in part by the Army Research Lab under contract number DAAL0197K0135, the Army Research Office under grant number DAAD190010484, by DARPA/RL contract number F306029910552, the ARL CTA on Advanced Decision Architectures, and NSF grant 0205489.

References

[1] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel query language for semistructured data. *Journal of Digital Libraries*, 1(1):68–88, November 1996.

[2] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J.L. Wiener. The lorel query language for semistructured data. *Journal of Digital Libraries*, 1(1):68–88, November 1996.

[3] D. Beech, S. Lawrence, M. Maloney, N. Mendelsohn, and H. Thompson. XML schema part 1: Structures. W3C Working Draft, May 1999. Available at `http://www.w3.org/TR/1999/xmlschema-1/`.

[4] C. Beeri and Y. Tzaban. SAL: An algebra for semistructured data and XML. In *Informal Proceedings of the ACM International Workshop on the Web and Databases (WebDB'99)*, Philadelphia, Pennsylvania, USA, June 1999.

[5] P. Biron and A. Malhotra. XML schema part 2: Datatypes. W3C Working Draft, May 1999. Available at `http://www.w3.org/TR/1999/xmlschema-2/`.

[6] Citeseer (NEC researchindex). Available at `http://citeseer.nj.nec.com/cs/`.

[7] DBLP (computer science bibliography). Available at `http://www.informatik.uni-trier.de/ ley/db/`.

[8] R. Dechter. Bucket elimination: A unifying framework for probabilistic inference. In *Proceedings of the 12th Conference on Uncertainty in Artifi cial Intelligence*, pages 211–219, 1996.

[9] A. Dekhtyar, J. Goldsmith, and S.R. Hawkes. Semistructured models for interval probabilities. *Submitted paper*.

[10] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL: A query language for XML. Available at `http://www.w3.org/xml/`, August 1998.

[11] D. Dey and S. Sarkar. A probabilistic relational model and algebra. *ACM Transactions on Database Systems*, 21(3), 1996.

[12] T. Eiter, T. Lukasiewicz, and M. Walter. Extension of the relational algebra to probabilistic complex values. In *Proc. of Int. Symposium on Foundations of Information and Knowledge Systems*, pages 94–115, 2000.

[13] N. Fuhr and T. Rolleke. A probabilistic relational algebra for the intergration of information retrieval and database systems. *ACM Transactions on Information Systems*, 15(1), 1997.

[14] E. Hung, L. Getoor, and V.S. Subrahmanian. Probabilistic Interval XML. In *Proc. of International Conference on Database Theory (ICDT)*, Siena, Italy, January 2003.

[15] H.V. Jagadish, V.S. Lakshmanan, and Divesh Srivastava. TAX: A tree algebra for XML. In *Proc. of Int. Workshop on Database Programming Languages (DBPL'01)*, Roma, Italy, September 2001.

[16] V.S. Lakshmanan, N. Leone, R. Ross, and V.S. Subrahmanian. ProbView: A flexible probabilistic database system. *ACM Transactions on Database Systems*, 22(3):419–469, 1997.

[17] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society*, B 50(2), 1988.

[18] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3):54–66, September 1997.

[19] A. Nierman and H.V. Jagadish. ProTDB: Probabilistic data in XML. In *Proc. of the 28th VLDB Conference*, Hong Kong, China, August 2002.

[20] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Proc. of the Eleventh International Conference on Data Engineering*, pages 251–260, Taipei, Taiwan, March 1995.

[21] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, San Francisco, 1988.

[22] XML Path Language (XPath) 2.0, W3C working draft. Available at `http://www.w3.org/TR/xpath20/`.

[23] XQuery 1.0: An XML Query Language, W3C working draft. Available at `http://www.w3.org/TR/xquery/`.