Representing Probabilistic Information in XML

Wenzhong Zhao, Alex Dekhtyar and Judy Goldsmith

Department of Computer Science

University of Kentucky

Lexington, KY40506, USA

$\{wzhao0, dekhtyar, goldsmit\}@cs.uky.edu$

Abstract. Traditional databases have been successful in managing large amounts of deterministic information. However, the real world is filled with uncertainty. Up until now, there has been limited success in managing probabilistic information. In 2001, Dekhtyar, Goldsmith and Hawkes [1] proposed a formal Semistructured Probabilistic Object (SPO) data model to represent probabilistic information and Semistructured Probabilistic Algebra (SP-Algebra) to query it. In this paper, we discuss the XML-based implementation of this model. We show how SPOs are represented in XML. We discuss our storage strategies for efficiently mapping XML representations of SPOs into relational tables. We also describe a query translation mechanism which automatically generates a set of SQL queries for evaluating SP-Algebra queries. Based on this framework, a prototype semistructured probabilistic DBMS (SPDBMS) has been implemented on top of a relational DBMS. We also report on the results of the experiments testing the performance of the SPDBMS.

1 Introduction

Probabilistic information occurs in applications such as medical diagnosis, image processing, fault analysis, stock market prediction, and applications of Bayesian Nets. Interest in storing and managing probabilistic information has been growing in the last two decades. There have been relational and object data models (see [1] for a survey of prior research) proposed to support storage and querying of probabilistic information. Unfortunately, these approaches are not sufficiently flexible to handle different contexts in which probabilities must be managed to analyze and model stochastic systems. Consider, for instance, academic advising, where the expectation of a student's success in a course may be represented in a variety of forms: a simple probability distribution for one course or a joint probability distribution for several courses, or a simple or joint conditional probability distribution (success in later courses may depend on earlier grades).

In [1], we have proposed a flexible database model for probabilistic information: Semistructured Probabilistic Objects (SPOs). The SPO model stores information about a probability distribution of one or more discrete random variables with finite domains together with additional identifying and conditioning information in a single object. We have also proposed Semistructured Probabilistic Algebra (SP-Algebra, SPA) — the query algebra for the SPO model. SP-Algebra included operations of selection, projection, Cartesian product, join and conditionalization.

In this paper, we describe our progress in building the Semistructured Probabilistic DBMS (SPDBMS) implementing the SPO data model. First, we propose a markup meta-language SPO-ML to represent SPOs (Section 2). The SPO-ML representation of SPOs will have specific well-defined structure, which we take advantage of in designing the relational schema to store SPO-ML documents (Section 3). We then describe in Section 4 how SP-Algebra queries are executed in SPDBMS by transforming them into sequences of SQL queries to the underlying RDBMS and performing postprocessing of results of those queries. Finally, we describe the experiments that tested the performance of our implementation in Section 5.

2 Data Model

2.1 SPO Data Model

In this section we briefly summarize the SPO model as described in [1]. Consider a universe $\mathcal{V}$ of *discrete random variables* $\{v_1, \ldots, v_m\}$. With each random variable $v \in \mathcal{V}$ we associate a finite set $dom(v)$ of its *possible values*. Given a set $V = \{v_1, \dots v_q\} \subseteq \mathcal{V}$, $dom(V)$ denotes $dom(v_1) \times \dots dom(v_q)$.

Let $\mathcal{R} = (A_1, \dots, A_n)$ be a collection of relational attributes. We call elements of $\mathcal{R}$ *context attributes*. For $A \in \mathcal{R}$, let $dom(A)$ denote the domain of A. We define a *context schema* $R^\star$ over $\mathcal{R}$ as a **multiset** of attributes from $\mathcal{R}$. For example, if $\mathcal{R} = \{\mathsf{year}, \mathsf{major}, \mathsf{college}\}$, the following are valid context schemas over $\mathcal{R}$: $R_1^\star = \{\mathsf{year}, \mathsf{college}\}$; $R_2^\star = \{\mathsf{year}, \mathsf{year}, \mathsf{major}, \mathsf{college}\}$; $R_3^\star = \{\mathsf{major}, \mathsf{major}, \mathsf{major}\}$. We define the key notion of our framework: *Semistructured Probabilistic Objects (SPOs)* as follows.

Definition 1. *Given a universe $\mathcal{V}$ of random variables and a set $\mathcal{R}$ of context attributes,* ***Semistructured Probabilistic Object (SPO)*** *S over $\langle \mathcal{V}, \mathcal{R} \rangle$ is a quadruple $S = \langle T, V, P, C \rangle$, where*

- *T, the* **context** *of S, is a relational tuple over a* context schema *$R^\star$ over $\mathcal{R}$.*
- *$V = \{v_1, \dots, v_q\} \subseteq \mathcal{V}$ is a set of* **random variables** *that participate in S. We require that $V \neq \emptyset$.*
- *$P : dom(V) \longrightarrow [0, 1]$ is the* **probability table** *of S. We require $\sum_{\bar{x} \in dom(V)} P(\bar{x}) \leq 1$.*
- *$C = \{(u_1, X_1), \dots (u_s, X_s)\}$, where $\{u_1, \dots, u_s\} = U \subseteq \mathcal{V}$ and $X_i \subseteq dom(u_i)$, $1 \leq i \leq s$, such that $V \cap U = \emptyset$. C is called the* **conditional** *of S.*

A semistructured probabilistic relation *(*SP-relation*) is a collection of SPOs, and a* semistructured probabilistic database *(*SP-database*) is a collection of SP-relations.*

We explain the definition of the SPO with the following example.

Example 1. Consider the *joint probability distribution of student grades in* Databases (DB) *and* Operating Systems (OS) *for College of Engineering majors who have a grade of* A *or* B *in* Data Structures (DS), as defined in the left part of Figure 1. We can break this information into the four categories as follows:

college : Engineering

DB	OS	P
A	A	0.09
A	B	0.12
A	C	0.03
A	F	0.005
B	A	0.12
B	B	0.16
B	C	0.13
B	F	0.01
C	A	0.03
C	B	0.08
C	C	0.11
C	F	0.045
F	A	0.0
F	B	0.01
F	C	0.02
F	F	0.04

$\mathsf{DS} \in \{ \mathsf{A}, \mathsf{B} \}$

```
<?xml version="1.0"?>
<spo>
<context>
  <college> Engineering </college>
</context>
<table>
  <row> <DB>A</DB> <OS>A</OS> <P>0.09 </P> </row>
  <row> <DB>A</DB> <OS>B</OS> <P>0.12 </P> </row>
  <row> <DB>A</DB> <OS>C</OS> <P>0.03 </P> </row>
  <row> <DB>A</DB> <OS>F</OS> <P>0.005</P> </row>
   ...                ...                  ...
  <row> <DB>F</DB> <OS>A</OS> <P>0.0  </P> </row>
  <row> <DB>F</DB> <OS>B</OS> <P>0.01 </P> </row>
  <row> <DB>F</DB> <OS>C</OS> <P>0.02 </P> </row>
  <row> <DB>F</DB> <OS>F</OS> <P>0.04 </P> </row>
</table>
<conditional>
  <DS> {A B} </DS>
</conditional>
</spo>
```

Fig. 1. A typical SPO object for Student Performance in Databases and Operating Systems Courses, and its XML representation.

```
<!DOCTYPE spo [
<!ELEMENT spo (context?, table, conditional?)>
<!ELEMENT context ((A1|A2|A3| ... |Ak)*)>
<!ELEMENT table (row+)>
<!ELEMENT conditional (v1?, v2?, v3?, ... ,vn?)>
<!ELEMENT row (v1?, v2?, v3?, ... ,vn?, P)>
<!ELEMENT Ai (#PCDATA)>
<!ELEMENT vj (#PCDATA)>
<!ELEMENT P  (#PCDATA)>
]>
```

Fig. 2. XML DTD template for SPOs in the specified application domain.

context: information about student's college is **not** represented by a random variable in our universe rather — by an attribute college. Then college: Engineering is the **context**.

participating random variables: $V = \{\mathsf{DB}, \mathsf{OS}\}$.

probability table: stores the joint probability distribution of DB and OS.

conditional: there is a single conditional $\mathsf{DS} \in \{\mathsf{A}, \mathsf{B}\}$ representing the information that the probability distribution is defined for students who received A or B in Data Structures associated with this distribution. It is stored in an SPO as $C = \{(\mathsf{DS}, \{\mathsf{A}, \mathsf{B}\})\}$.

2.2 Representing SPOs in XML

We represent SPOs in XML using a meta-language we call SPO-ML for markup. A number of XML representations are possible: we choose to utilize the names of random variables and context attributes as element names in the markup language. Thus, actual DTD/XML schema of the markup language depends on the application domain, namely the pair $\langle \mathcal{V}, \mathcal{R} \rangle$. SPO-ML represents the general markup rules for any domain. Given the universe $\mathcal{V}$ of random variables {v1, v2, ..., vn} and a collection of context attributes $\mathcal{R} = (A1, \ldots, Ak)$ we construct the appropriate markup language as shown on the template DTD in Figure 2 [1].

Figure 1 (right) shows the SPO-ML encoding of the SPO shown on the left side. The top layer of the XML document for the SPO consists of three elements `<context>`, `<table>` and `<conditional>`. The contents of context and conditional parts are fairly straightforward. The probability table is modeled as a collection of rows each of which consists of a sequence of random variables with values and the corresponding probability.

Semistructured Probabilistic Objects are complex structures and not all their properties can be captured by XML validity checks. An SPO representation in SPO-ML should satisfy the following extra validity constraints. First, all `<row>` elements inside the `<table>` elements have to have exactly the same sequence of participating random variables. Second, the set of random variable elements inside `<table>` and the set of random variable elements inside `<conditional>` must be disjoint. Finally, the content of <P> elements inside `<row>` elements is expected to be real numbers between 0 and 1, and their sum must be less than or equal to 1. These additional constraints mean that validation on an XML representation of an SPO is a two-step process: first the XML is validated against the appropriate DTD/XML schema, and then the additional constraints are verified.

[1] DTD representation of SPO-ML is chosen here for its simplicity and succinctness. We also maintain and use the corresponding XML schemas.

3 Mapping SPOs to relational tables

A relational database system has been used as a back end to store SPO-ML encoded data by mapping the XML schema onto a set of relational tables. All SPO objects are stored as records in a set of relational tables. The structure of each SPO object is preserved by utilizing a special database design. The relational schemas for an SP-Database are defined as follows:

RELATION (rid integer, name varchar, schema varchar) contains SP-relation level information. It connects all other tables by using the table naming convention that every table uses the unique identifier of the corresponding SP-relation *rid* as a prefix for its name. The attributes *name* and *schema* represent the SP relation name and the corresponding schema URL, respectively.

rid_SPO (id integer, head varchar, numvar integer) contains SPO level information. The association between this table and other tables is established by the unique identifier of an SPO *id*. The attribute *head* stores the prolog of an XML document and *numvar* stores the number of participating random variables in an SPO.

rid_SPO_CONS (id integer, type char, elemname varchar, elemvalue varchar, idref varchar) contains all the information about SPO context and conditional. The attribute *id* is a foreign key, and *type* tells whether it's a context or conditional. The attributes *elemname* and *elemvalue* give the pair of element name and element value.

rid_SPO_VAR (id integer, position integer, varname varchar) contains all the information of SPO participating variables. The attributes *position* and *varname* represent a pair of position and variable name.

rid_SPO_num [2] **(id integer, var_1 char, ... , var_num char, p decimal)** contains all the information of the probability tables for SPOs which have *num* participating random variables. The attribute *p* stores the probability value.

[2] *num* is a variable, which equals the number of participating variables in a particular SPO. Thus, *num* may vary from 1 to $|\mathcal{V}|$.

In order to improve data integrity and query performance, we created primary keys and foreign keys, such as primary key *rid* for relation RELATION, primary key *id* for relation *rid*_SPO and foreign keys *id* for all other relations. We also created indexes for the last three type of relations, for instance, multicolumn index on (*id*, *elemname*) for relation *rid*_SPO_CONS, multicolumn index on (*id*, *varname*) for relation *rid*_SPO_VAR and multicolumn index on (*id*, *var*_1,..., *var*_num) for relation *rid*_SPO_num.

The database system stores SPOs from each SP-relation in a separate set of relational tables. The CREATE algorithm starts by storing the SP-relation name and the schema url, generates a unique identifier *rid* for the SP-relation. It also creates all the empty tables with the schema defined above and associates them with the SP-relation. In order to store SPOs in an SP-relation, the SPOs must be parsed to a DOM tree and decomposed into four components, Head, Context, Table and Conditional, based on a predefined schema template. The INSERT algorithm gets a unique SPO object identifier *id*, then stores the XML prolog information and the number of participating random variables in the SPO in the rid_SPO table. It stores SPO context and conditional into the table rid_SPO_CONS, the probability table into the table rid_SPO_num and participating random variables into the table rid_SPO_VAR, respectively. Figure 3 shows the resulting tables after storing the SPO defined in Figure 1 into an SP-relation.

4 Querying the SPDBMS

The SP-Algebra proposed in [1] has been implemented to query the SP-Database. Here we briefly summarize the SP-Algebra and refer readers to [1] for the formal definitions of the SP-Algebra operations.

SP-Algebra extends the definitions of standard relational operations and includes *selection* (σ), *projection* (π), *conditionalization*(μ), *Cartesian product* ($\times$) and *join* ($\bowtie$) operations. Given an SPO $S = \langle T, V, P, C \rangle$, selection operation may query information stored in each of its components: context, participating vari-

1_SPO_2			
id	var_1	var_2	P
2	A	A	0.09
2	A	B	0.12
2	A	C	0.03
2	A	F	0.005
2	B	A	0.12
2	B	B	0.16
2	B	C	0.13
2	B	F	0.01
2	C	A	0.03
2	C	B	0.08
2	C	C	0.11
2	C	F	0.045
2	F	A	0.0
2	F	B	0.01
2	F	C	0.02
2	F	F	0.04

RELATION		
rid	name	schema
1	First	schema_url

1_SPO		
id	head	numvar
2	null	2

1_SPO_VAR		
id	position	varname
2	1	DB
2	2	OS

1_SPO_CONS				
id	type	elemname	elemvalue	idref
2	cont	college	Engineering	null
2	cond	DS	{A,B}	null

Fig. 3. internal representation for the SPO defined in Figure 1.

ables, conditional information and the probability table. Each type of selection operation requires its own language of selection conditions. Given a selection condition on context, participating variables or conditionals, an SPO will either match the condition and be returned in the result of selection operation, or will not match and be omitted from the result. Selection conditions that operate on the contents of the probability table (these are either conditions of the rows or on the probability values) are processed in a slightly different manner. Such selection operations return only the rows of the probability table that match the conditions, while keeping context and conditional parts intact.

Projection operation may apply to context, participating variables or conditionals. Projection on context or conditionals is similar to classical relational projection: it removes the unwanted context or conditional entries from the SPO. Given a list $\mathcal{F}$ of random variables, $\pi_{\mathcal{F}}(S)$, the projection on participating variables computes the marginal probability distribution of random variables $\mathcal{F} \cap V$ from the joint probability distribution for V stored in P (the context and conditional parts are preserved).

Given a constraint $v = a$ where v is a random variable and $a \in dom(v)$, conditionalization operation $\mu_{v=a}(S)$ computes from P the conditional probability distribution of random variables $V - \{v\}$ given $v = a$. The resulting SPO, will have the same context, and $(v, \{a\})$ will be added to its conditional part.

Cartesian product and join compute joint probability distributions from the probability tables of the participating SPOs. Cartesian product is applicable only to a pair of SPOs which do not share common variables, while join applies only to a pair of SPOs that *do* share common variables.

The query language allows us to navigate through the entire database with structured queries, including any combination of SELECTION, PROJECTION, CONDITIONALIZATION, CARTESIAN PRODUCT and JOIN. In the current implementation the query processing proceeds as follows. Structured queries are first parsed and then transformed into a query tree [3]. Each internal node in the resulting parse tree is an SP-Algebra operator and each of the leaves is an SP-relation. Each operator is translated in a straightforward manner into a set of corresponding SQL statements that can be executed by the underlying RDBMS.

This, however, is not enough for some SP-Algebra queries. Conditionalization, projection, Cartesian product and join change the probability tables in the results according to the semantics of each operation. These computations are performed at the SPDBMS server during the special *postprocessing* stages of the query processing. Postprocessing also includes the assembly of the resulting XML document.

Space limitations prevent us from describing query translations for all SP-Algebra queries. Here we give two examples of probabilistic queries, illustrating how to map from an SP-Algebra query to a set of SQL statements; other translations can be found in the extended version of this paper [4]. First, consider **selection on probabilities**. Given a selection condition P op x and an SP-relation $\mathcal{S} = \{S_1, \ldots, S_n\}$, this operation returns SPOs that have at least one

[3] Work on query optimization in SPDBMS is underway, but the current version of the SPDMBS server does not have this feature.

row with probability that satisfies the selection condition. Selection preserves the context, participating random variables and conditionals in the original SPOs, but returns only those rows of the probability table satisfying the selection condition. Consider a sample query $\sigma_{p>0.1}(\mathcal{S})$. Figure 4 shows the sequence of SQL statements needed in order to evaluate this query.

Our second example is the operation of **conditionalization**. This operation computes conditional probability distributions and thus, is specific to *probabilistic* algebras. Given the constraint $v = a$ and an SP-relation $\mathcal{S} = \{S_1, \ldots, S_n\}$ conditionalization $\mu_{v=a}$ first selects SPOs in which v is a participating random variable. For each selected SPO, conditionalization preserves the context, conditions the joint probability distribution, replaces the probability table with a new conditional probability distribution over the remaining random variables, and adds the condition $(v, \{a\})$ to the conditional part of each of the resulting SPOs. Consider a sample query $\mu_{DB='A'}(\mathcal{S})$. We need the following sequence of SQL statements to perform the query, as shown in Figure 5.

```
step 1. Get the SPO ID list based
        on given probability value:
SELECT DISTINCT id
FROM rid_SPO_1
WHERE p > 0.1
...
UNION ALL
SELECT DISTINCT id
FROM rid_SPO_i
WHERE p > 0.1
...
UNION ALL
SELECT DISTINCT id
FROM rid_SPO_max
WHERE p > 0.1

step 2. Get the variable list for
        each SPO in the ID list:
SELECT id, varname, position
FROM rid_SPO_VAR
WHERE id IN {ID list}

step 3. Retrieve context/conditional:
SELECT id, elemname, elemvalue, idref
FROM rid_SPO_CONS
WHERE id IN {ID list}

step 4. Retrieve probability table
SELECT *
FROM rid_SPO_1
WHERE id IN {ID list}
      AND p > 0.1
...
UNION ALL
SELECT *
FROM rid_SPO_i
WHERE id IN {ID list}
      AND p > 0.1
...
UNION ALL
SELECT *
FROM rid_SPO_max
WHERE id IN {ID list}
      AND p > 0.1
```

Fig. 4. Steps to evaluate the selection query

```
step 1. Get the SPO ID list based
        on the variable name(s):
SELECT DISTINCT id
FROM rid_SPO_VAR
WHERE varname = 'DB'

step 2. Get the variable list for
        each SPO in the ID list:
SELECT id, varname, position
FROM rid_SPO_VAR
WHERE id IN {ID list}

step 3. Retrieve context/conditional:
SELECT id, elemname, elemvalue, idref
FROM rid_SPO_CONS
WHERE id IN {ID list}

step 4. Retrieve probability table
SELECT id, {var_i list}, P
FROM rid_SPO_1
WHERE id IN {ID list}
      AND var_position = 'A'
...
UNION ALL
SELECT id, {var_j list}, P
FROM rid_SPO_i
WHERE id IN {ID list}
      AND var_position = 'A'
...
UNION ALL
SELECT id, {var_k list}, P
FROM rid_SPO_max
WHERE id IN {ID list}
      AND var_position = 'A'
```

Fig. 5. Steps to evaluate the conditionalization query

5 Experimental results

We have implemented a prototype semistructured probabilistic database system on top of a RDBMS in java, JDK1.3. Figure 6 depicts the overall architecture of our system. The core of the system is the SPDBMS application server which processes query requests from a variety of client applications. The application server provides a JDBC-like API, through which client applications can send standard database management instructions, such as CREATE DATABASE, DROP DATABASE, CREATE SP-RELATION, DROP SP-RELATION, INSERT INTO SP-RELATION, DELETE FROM SP-RELATION, as well as SP-Algebra queries to the server.

In this section we present the results of tests conducted with the prototype system. The algorithms for storing and querying SPO objects in a relational database are completely independent of the underlying RDBMS. All features specific to a RDBMS are implemented in the special database adapter layer, so the system can port to any relational database with little modification. Current system uses Oracle8i as the RDBMS backend. To avoid network delays during tests, both the application server and Oracle DB server were running on the

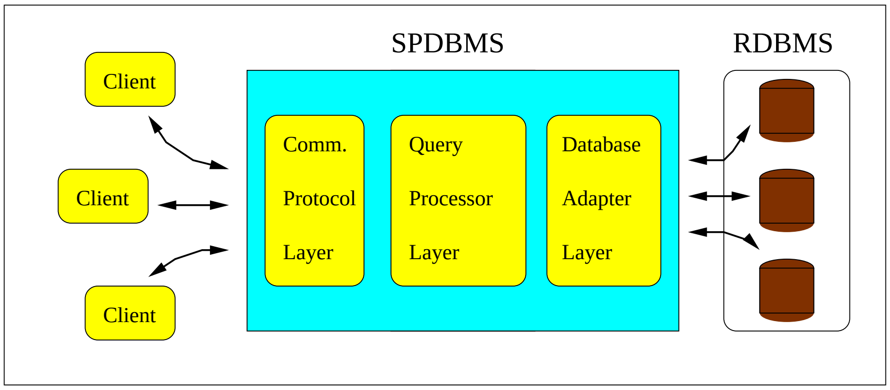

Fig. 6. The overall architecture of SPDBMS.

same machine, a 440 MHz Sun Ultra 10 running Solaris OS with 1GB of main memory, and the timing was done on the server side.

In order to ensure consistency, each experiment consists of 20 runs, and each point on a graph represents the average running time for the 20 runs. Most test data sets used in the experiments are generated randomly by a custom data generator[4]. However, for Cartesian product and join, we generated specific data sets in order to control the selectivity for each query. Each data set was generated based on the following three parameters: number of SPOs per SP-relation, number of participating random variables in an SPO, and size of the domain of participating random variables. The first parameter affects the number of objects to be stored in the database while the other two affect the size of individual objects. Table 1 shows some typical data sets with corresponding file size and number of tuples in the underlying Oracle database.

We examined the running time for each type of atomic SP-Algebra query [1]. Most queries are generated randomly at runtime by a custom query generator[4] in the client application running on another machine. We have collected both the total running time and the time consumed by the Oracle DB server for executing

[4] Both the SPO data generator and the SP-Algebra query generator utilize a linear congruential pseudo-random number generator, which comes with Sun JDK1.3 package.

Table 1. File size and number of tuples in Oracle database for typical data sets.

number of SPOs	1,000	1,000	1,000	10,000	10,000	10,000
number of variables	2	4	2	2	4	2
size of domain	2	2	4	2	2	4
size of original XML file (MB)	0.38	1.64	1.10	3.81	16.4	11.0
number of tuples in Oracle DB	10,000	24,000	22,000	100,000	240,000	220,000

SQL statements. The Oracle server consumes 75 - 95% of the total execution time for most queries, and the percentage increases with the size of the XML files.

To study the effects of the number of SPOs in an SP-relation on the query running time, different experiments were conducted with the number of SPOs varying from 100 to 10,000. The results are plotted in Figure 7. It can be observed that all types of unary SPA queries scale well as the size of SP-relations increases: the running time increases sublinearly with the number of SPOs for large SP-relations, but at a much slower rate for SP-relations of small size. In Figure 8, the effects of domain size for participating variables are shown. Notice that the number of tuples considered grows exponentially with the size of the domain, leading to a near-exponential growth in the size of the XML files. The running time increases with the size of domain, but not as quickly as does the size of the XML files. Figure 9 shows the effects of the number of variables in SPOs on the time for the conditionalization operation. Finally, Figure 10 shows the dependence of running time on the size of the SP-relation and query selectivity for the join operation. The graph shows that the running time for join increases with the number of SPOs at a nearly quadratic rate, and also increases with the selectivity, but at a much slower rate. One reason is that the number of SPOs output increases quadratically with the number of SPOs in the initial SP-relations.

6 Conclusions, Future and Related Work

In this paper, we described a prototype semistructured probabilistic DBMS based on the SPO data model for storing and managing semistructured proba-

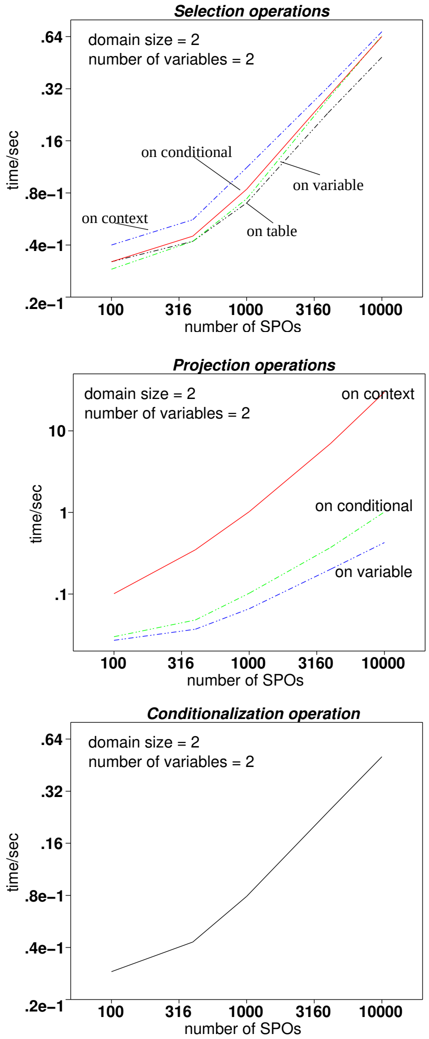

Fig. 7. Effect of number of SPOs in an SP-relation.

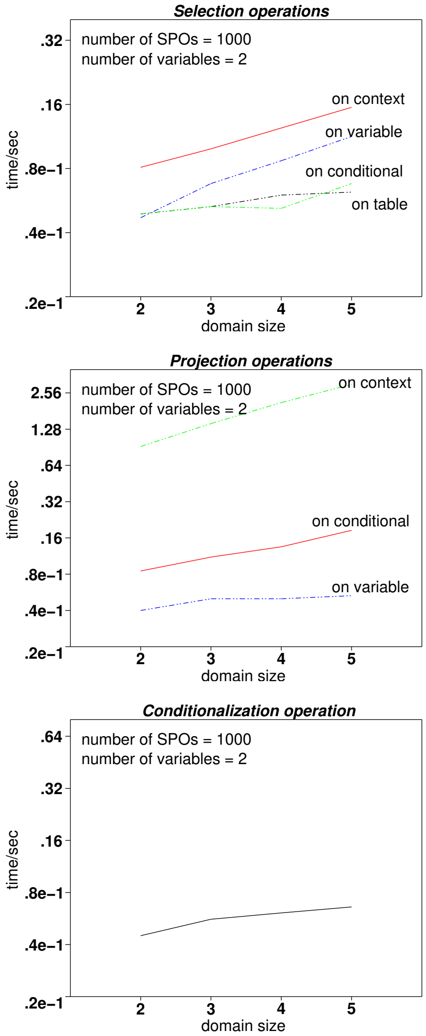

Fig. 8. Effect of domain size of participating random variables.

bilistic objects. We also presented a performance evaluation of the SPDBMS for different types of SP-Algebra queries.

There are three major foci of our ongoing work: (i) implementation of a query optimizer for the current semistructured probabilistic DBMS; (ii) extension of the data model and the algebra to handle interval probabilities, and (iii) study of data fusion and conflict resolution problems that arise in this framework.

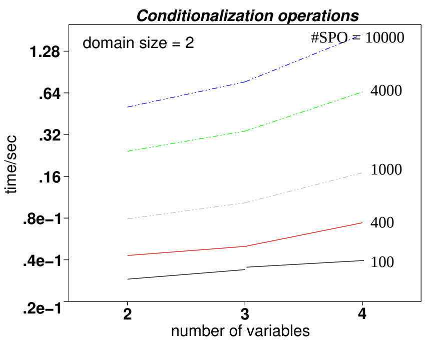

Fig. 9. Effect of number of variables on conditionalization operation.

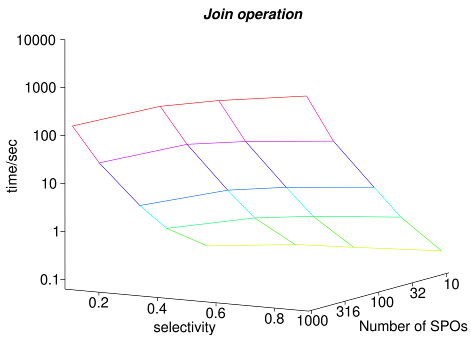

Fig. 10. Effect of number of SPOs and selectivity on join operation.

We refer the reader to [1, 2] for surveys of work on relational and object models for probabilistic databases, and to two more studies: Probabilistic Interval XML by Hung, Getoor and Subrahmanian [2] and ProTDB by Nierman and Jagadish [3] investigate the use of XML to represent uncertainty. While both those frameworks deal with uncertainty in the *structure* of XML documents, the work described here concentrates on representing uncertainty about real world events, encoded as probability distributions of random variables in XML form.

References

1. Alex Dekhtyar, Judy Goldsmith, and Sean Hawkes. Semistructured probabilistic databases. In *Proc. Statistical and Scientific Database Management Systems*, 2001.
2. Edward Hung, Lise Getoor, and V.S. Subrahmanian. Probabilistic interval XML. In *Proc. International Conference on Database Theory*, 2003.
3. Andrew Nierman and H. V. Jagadish. ProTDB: Probabilistic data in XML. In *Proceedings of the 28th VLDB Conference*, 2002.
4. Wenzhong Zhao, Alex Dekhtyar, and Judy Goldsmith. Representing probabilistic information in xml. Technical Report 770-03, Department of Computer Science, University of Kentucky, 2003.