On the Complexity of Managing Probabilistic XML Data

Pierre Senellart  Serge Abiteboul

Principles Of Database Systems, 13th June 2007
Outline

1 Introduction
   - Motivation
   - Probabilistic Data Management
   - Complexity Issues

2 Prob-Trees

3 Equivalence of Prob-Trees

4 Prob-Trees with Additional Constraints

5 Conclusion
Imprecise Data and Imprecise Tasks

Observations

- Many tasks generate **imprecise** data, with some **confidence** value.
- Need for a way to manage this imprecision, to work with it **throughout an entire complex process**.
Imprecise Data and Imprecise Tasks

Observations

- Many tasks generate **imprecise** data, with some **confidence** value.
- Need for a way to manage this imprecision, to work with it **throughout an entire complex process**.
A Probabilistic XML Warehouse

Module 1

Update transaction + confidence

Update interface

Query interface

Probabilistic XML Warehouse

Query

Results + confidence

Module 2

Module 3
A Probabilistic XML Warehouse (Hidden Web)

- Topic crawler
- Form analyzer
- Inf. Extractor

Update interface
- Update transaction
- + confidence

Query interface
- Query
- Results
- + confidence

Probabilistic XML Warehouse
A Probabilistic XML Warehouse (Hidden Web)

Module 1
Module 2
Module 3

Update interface
Query interface

Topic crawler
Form analyzer
Inf. Extractor

Crawled URLs + confidence
Query
Results + confidence

Probabilistic XML Warehouse

Pierre Senellart, Serge Abiteboul
On the Complexity of Probabilistic XML
A Probabilistic XML Warehouse (Hidden Web)

- Topic crawler
- Form analyzer
- Inf. Extractor

Update transaction + confidence

Form URL?

Results + confidence

Update interface

Query interface

Probabilistic XML Warehouse
A Probabilistic XML Warehouse (Hidden Web)

Topic crawler

Form analyzer

Inf. Extractor

Update transaction + confidence

Query

URLs + confidence

Update interface

Query interface

Probabilistic XML Warehouse

Pierre Senellart, Serge Abiteboul

On the Complexity of Probabilistic XML
A Probabilistic XML Warehouse (Hidden Web)

Topic crawler

Form analyzer

Inf. Extractor

Update interface

Query interface

Probabilistic XML Warehouse

Analyzed form + confidence

Query + confidence

Results + confidence

Pierre Senellart, Serge Abiteboul

On the Complexity of Probabilistic XML
A Probabilistic XML Warehouse (Hidden Web)

- Topic crawler
- Form analyzer
- Inf. Extractor

Update interface
- Update transaction
  + confidence

Query interface
- Form?
  + confidence
- Results
  + confidence

Probabilistic XML Warehouse
A Probabilistic XML Warehouse (Hidden Web)

Module 1
- Topic crawler
- Form analyzer
- Inf. Extractor

Update interface
- Update transaction
  + confidence

Query interface
- Query
  + confidence

Person → ·
+ confidence

Probabilistic XML Warehouse
A Probabilistic XML Warehouse (Hidden Web)

**Module 1**
- Topic crawler
- Update interface

**Module 2**
- Form analyzer

**Module 3**
- Inf. Extractor
- Query interface

**Update**
- transaction
- + confidence

**Query**
- Results
- + confidence

**Probabilistic XML Warehouse**
- Person → ISBN
- + confidence
Probabilistic Trees

Framework

- Unordered data trees
- Details: no attributes, no mixed content...

Sample space: Set of all such data trees.

Probabilistic tree (prob-tree): Representation of a discrete probability distribution over this sample space.
Probabilistic Trees

Framework

- **Unordered** data trees
- Details: no attributes, no mixed content...

Sample space: Set of all such data trees.

Probabilistic tree (prob-tree): Representation of a discrete probability distribution over this sample space.

A

B

C

D

A

B

B

C

D

(multiset semantics)
Probabilistic Trees

Framework

- Unordered data trees
- Details: no attributes, no mixed content...

Sample space: Set of all such data trees.

Probabilistic tree (prob-tree): Representation of a discrete probability distribution over this sample space.
Probabilistic Trees

Framework
- **Unordered** data trees
- Details: no attributes, no mixed content...

![Diagram]

Sample space: Set of all such data trees.

**Probabilistic tree (prob-tree):** Representation of a discrete probability distribution over this sample space.
Probabilistic Trees

Framework
- **Unordered** data trees
- Details: no attributes, no mixed content...

Sample space: Set of all such data trees.

Proabilistic tree (prob-tree): Representation of a discrete probability distribution over this sample space.
Complexity Issues

Prob-tree model defined in [Abiteboul & Senellart 2006]. Here, we tackle *complexity questions* about it:

- What is the complexity of *queries* and *updates*?
- Is this complexity *inherent* to the problem of managing tree-like probabilistic information?
- How can we check if two prob-trees are *equivalent*?
- Can we compute efficiently *restrictions* of prob-trees (e.g., by a DTD)?
Complexity Issues

Prob-tree model defined in [Abiteboul & Senellart 2006]. Here, we tackle complexity questions about it:

- What is the complexity of queries and updates?
- Is this complexity inherent to the problem of managing tree-like probabilistic information?
- How can we check if two prob-trees are equivalent?
- Can we compute efficiently restrictions of prob-trees (e.g., by a DTD)?
Prob-tree model defined in [Abiteboul & Senellart 2006]. Here, we tackle complexity questions about it:

- What is the complexity of queries and updates?
- Is this complexity inherent to the problem of managing tree-like probabilistic information?
- How can we check if two prob-trees are equivalent?
- Can we compute efficiently restrictions of prob-trees (e.g., by a DTD)?
Prob-tree model defined in [Abiteboul & Senellart 2006]. Here, we tackle complexity questions about it:

- What is the complexity of queries and updates?
- Is this complexity inherent to the problem of managing tree-like probabilistic information?
- How can we check if two prob-trees are equivalent?
- Can we compute efficiently restrictions of prob-trees (e.g., by a DTD)?
Prob-tree model defined in [Abiteboul & Senellart 2006]. Here, we tackle complexity questions about it:

- What is the complexity of queries and updates?
- Is this complexity inherent to the problem of managing tree-like probabilistic information?
- How can we check if two prob-trees are equivalent?
- Can we compute efficiently restrictions of prob-trees (e.g., by a DTD)?
Outline

1. Introduction

2. Prob-Trees
   - The Prob-Tree Model
   - Queries and Updates

3. Equivalence of Prob-Trees

4. Prob-Trees with Additional Constraints

5. Conclusion
The Prob-Tree Model

- Data tree with event conditions (conjunction of probabilistic events or negations of probabilistic events) assigned to each node.
- Probabilistic events are boolean random variables, assumed to be independent, with their own probability distribution.
- Representation à la [Imieliński & Lipksi 1984].

\[
\begin{array}{c|c|c}
\text{Event} & \text{Prob.} \\
\hline
w_1 & 0.8 \\

w_2 & 0.7 \\
\end{array}
\]
Semantics of Prob-Trees

Semantics of a Prob-Tree $T$: Set of Possible Worlds $[T]$ (probability distribution over the set of data trees).

<table>
<thead>
<tr>
<th>Event</th>
<th>Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1$</td>
<td>0.8</td>
</tr>
<tr>
<td>$w_2$</td>
<td>0.7</td>
</tr>
</tbody>
</table>

$p_1 = 0.06$  \hspace{1cm} p_2 = 0.70 \hspace{1cm} p_3 = 0.24$

Actually, fully expressive.
Semantics of a Prob-Tree $T$: Set of Possible Worlds $[T]$ (probability distribution over the set of data trees).

<table>
<thead>
<tr>
<th>Event</th>
<th>Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1$</td>
<td>0.8</td>
</tr>
<tr>
<td>$w_2$</td>
<td>0.7</td>
</tr>
</tbody>
</table>

$p_1 = 0.06$  $p_2 = 0.70$  $p_3 = 0.24$

Actually, fully expressive.
Semantics of Prob-Trees

Semantics of a Prob-Tree $T$: Set of Possible Worlds $\{T\}$
(probability distribution over the set of data trees).

<table>
<thead>
<tr>
<th>Event</th>
<th>Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1$</td>
<td>0.8</td>
</tr>
<tr>
<td>$w_2$</td>
<td>0.7</td>
</tr>
</tbody>
</table>

$p_1 = 0.06$  $p_2 = 0.70$  $p_3 = 0.24$

Actually, fully expressive.
Semantics of Prob-Trees

Semantics of a Prob-Tree $T$: Set of Possible Worlds $[T]$ (probability distribution over the set of data trees).

<table>
<thead>
<tr>
<th>Event</th>
<th>Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1$</td>
<td>0.8</td>
</tr>
<tr>
<td>$w_2$</td>
<td>0.7</td>
</tr>
</tbody>
</table>

$p_1 = 0.06 \quad p_2 = 0.70 \quad p_3 = 0.24$

Actually, fully expressive.
Locally Monotone Queries

**Query**: function that maps a data tree $t$ to a set of subtrees of $t$ containing its root.

**Definition**

A query $Q$ is **locally monotone** if, for any data trees $u$, $t'$ and $t$ such that $u \leq t' \leq t$, $u \in Q(t) \iff u \in Q(t')$.

**Examples**

- Tree-pattern queries with joins are locally monotone.
- “Return the root node if it has no A child, nothing otherwise.” is not locally monotone.
Locally Monotone Queries

Query: function that maps a data tree \( t \) to a set of subtrees of \( t \) containing its root.

Definition

A query \( Q \) is locally monotone if, for any data trees \( u, t', t \) such that \( u \leq t' \leq t \), \( u \in Q(t) \iff u \in Q(t') \).

Examples

- Tree-pattern queries with joins are locally monotone.
- “Return the root node if it has no A child, nothing otherwise.” is not locally monotone.

Pierre Senellart, Serge Abiteboul

On the Complexity of Probabilistic XML
Locally Monotone Queries

**Query**: function that maps a data tree $t$ to a set of subtrees of $t$ containing its root.

**Definition**

A query $Q$ is **locally monotone** if, for any data trees $u$, $t'$ and $t$ such that $u \leq t' \leq t$, $u \in Q(t) \iff u \in Q(t')$.

**Examples**

- Tree-pattern queries with joins are locally monotone.
- “Return the root node if it has no A child, nothing otherwise.” is **not** locally monotone.
Locally Monotone Queries

**Query**: function that maps a data tree $t$ to a *set of subtrees* of $t$ containing its root.

**Definition**

A query $Q$ is *locally monotone* if, for any data trees $u$, $t'$ and $t$ such that $u \leq t' \leq t$, $u \in Q(t) \iff u \in Q(t')$.

**Examples**

- Tree-pattern queries with joins are locally monotone.
- “Return the root node if it has no $A$ child, nothing otherwise.” is *not* locally monotone.
Queries on Prob-Trees

Illustration of how to query prob-trees on an example.

Query: //C
Queries on Prob-Trees

Illustration of how to query prob-trees on an example.

Query: //C
Queries on Prob-Trees

Illustration of how to **query prob-trees** on an example.

Query: `//C`

Underlying data tree.
Illustration of how to query prob-trees on an example.

Query: //C
Queries on Prob-Trees

Illustration of how to query prob-trees on an example.

Query: //C
Queries on Prob-Trees

Illustration of how to query prob-trees on an example.

Query: //C
Queries on Prob-Trees

Illustration of how to query prob-trees on an example.

Query: //C
Consistence of Queries on Prob-Trees

Theorem

Prob-Tree \rightarrow \text{query} \rightarrow \text{Prob-Tree}

Possible Worlds \rightarrow \text{query} \rightarrow \text{Possible Worlds}
What about updates?

- We consider sets of elementary **insertions** and **deletions**.
- Defined with respect to a query (mapping between nodes of the query and nodes to insert/delete).
- More **involved** definitions...
- ... but a similar result:

**Theorem**

\[
\text{Prob-Tree} \xrightarrow{\text{update}} \text{Prob-Tree} \\
\text{Possible Worlds} \xrightarrow{\text{update}} \text{Possible Worlds}
\]
What about updates?

- We consider sets of elementary **insertions** and **deletions**.

- Defined **with respect to a query** (mapping between nodes of the query and nodes to insert/delete).

- More **involved definitions**...

- ... but a similar result:

**Theorem**

\[
\text{Prob-Tree} \xrightarrow{\text{update}} \text{Prob-Tree} \\
\text{Possible Worlds} \xrightarrow{\text{update}} \text{Possible Worlds}
\]
What about updates?

- We consider sets of elementary **insertions** and **deletions**.
- Defined **with respect to a query** (mapping between nodes of the query and nodes to insert/delete).
- More **involved** definitions...
- ... but a similar result:

**Theorem**

\[
\begin{align*}
\text{Prob-Tree} & \xrightarrow{\text{update}} \text{Prob-Tree} \\
\text{Possible Worlds} & \xrightarrow{\text{update}} \text{Possible Worlds}
\end{align*}
\]
What about updates?

- We consider sets of elementary **insertions** and **deletions**.
- Defined **with respect to a query** (mapping between nodes of the query and nodes to insert/delete).
- More **involved** definitions...
- ... but a similar result:

**Theorem**

\[
\text{Prob-Tree} \xrightarrow{\text{update}} \text{Prob-Tree} \\
\downarrow \text{semantics} \quad \quad \quad \downarrow \text{semantics} \\
\text{Possible Worlds} \xrightarrow{\text{update}} \text{Possible Worlds}
\]
**Complexity Results**

\( T \): prob-tree with underlying data tree \( t \).

\( \text{time}(Q(t)) \): complexity of the query \( Q \) over the data tree \( t \).

Upper bounds for operations on \( T \):

<table>
<thead>
<tr>
<th>Operation</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query</td>
<td>( \text{time}(Q(t)) + \text{polynomial in the size of } T, Q(t) )</td>
</tr>
<tr>
<td>Insertion</td>
<td>( \text{time}(Q(t)) + \text{polynomial in the size of } T, Q(t) )</td>
</tr>
<tr>
<td>Deletion</td>
<td>( \text{time}(Q(t)) + \text{exponential in the size of } T, Q(t) )</td>
</tr>
</tbody>
</table>

**Proposition**

*If the query language is not trivial, the result of a deletion may necessarily be exponential.*
**Complexity Results**

\[ T \]: prob-tree with underlying data tree \( t \).

\[ \text{time}(Q(t)) \]: complexity of the query \( Q \) over the data tree \( t \).

Upper bounds for operations on \( T \):

<table>
<thead>
<tr>
<th>Operation</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query</td>
<td>( \text{time}(Q(t)) + \text{polynomial in the size of } T, Q(t) )</td>
</tr>
<tr>
<td>Insertion</td>
<td>( \text{time}(Q(t)) + \text{polynomial in the size of } T, Q(t) )</td>
</tr>
<tr>
<td>Deletion</td>
<td>( \text{time}(Q(t)) + \text{exponential in the size of } T, Q(t) )</td>
</tr>
</tbody>
</table>

**Proposition**

*If the query language is not trivial, the result of a deletion may necessarily be exponential.*
**Complexity Results**

*T*: prob-tree with underlying data tree *t*.

\textit{time}(Q(t)): complexity of the query *Q* over the data tree *t*.

Upper bounds for operations on *T*:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query</td>
<td>\textit{time}(Q(t)) + \text{polynomial} in the size of <em>T</em>, <em>Q(t)</em></td>
</tr>
<tr>
<td>Insertion</td>
<td>\textit{time}(Q(t)) + \text{polynomial} in the size of <em>T</em>, <em>Q(t)</em></td>
</tr>
<tr>
<td>Deletion</td>
<td>\textit{time}(Q(t)) + \text{exponential} in the size of <em>T</em>, <em>Q(t)</em></td>
</tr>
</tbody>
</table>

**Proposition**

If the query language is not trivial, the result of a deletion may necessarily be exponential.
### Complexity Results

Let $T$: prob-tree with underlying data tree $t$.

$\text{time}(Q(t))$: complexity of the query $Q$ over the data tree $t$.

Upper bounds for operations on $T$:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query</td>
<td>$\text{time}(Q(t)) + \text{polynomial}$ in the size of $T$, $Q(t)$</td>
</tr>
<tr>
<td>Insertion</td>
<td>$\text{time}(Q(t)) + \text{polynomial}$ in the size of $T$, $Q(t)$</td>
</tr>
<tr>
<td>Deletion</td>
<td>$\text{time}(Q(t)) + \text{exponential}$ in the size of $T$, $Q(t)$</td>
</tr>
</tbody>
</table>

**Proposition**

*If the query language is not trivial, the result of a deletion may necessarily be exponential.*
**Complexity Results**

\( T \): prob-tree with underlying data tree \( t \).

\( \text{time}(Q(t)) \): complexity of the query \( Q \) over the data tree \( t \).

Upper bounds for operations on \( T \):

<table>
<thead>
<tr>
<th>Operation</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query</td>
<td>( \text{time}(Q(t)) + \text{polynomial} ) in the size of ( T, Q(t) )</td>
</tr>
<tr>
<td>Insertion</td>
<td>( \text{time}(Q(t)) + \text{polynomial} ) in the size of ( T, Q(t) )</td>
</tr>
<tr>
<td>Deletion</td>
<td>( \text{time}(Q(t)) + \text{exponential} ) in the size of ( T, Q(t) )</td>
</tr>
</tbody>
</table>

**Proposition**

*If the query language is not trivial, the result of a deletion may necessarily be exponential.*
Outline

1 Introduction

2 Prob-Trees

3 Equivalence of Prob-Trees
   - Two Notions of Equivalence
   - Structural Equivalence
   - Semantic Equivalence

4 Prob-Trees with Additional Constraints

5 Conclusion
Two Notions of Equivalence

What does it mean for two prob-trees to represent the same information?

Two different notions:

Structural Equivalence: we keep the same event variables.

Semantic Equivalence: we only consider the possible worlds semantics.

Complexity results? Relation between these two notions?
Two Notions of Equivalence

What does it mean for two prob-trees to represent the same information?

Two different notions:

Structural Equivalence: we keep the same event variables.

Semantic Equivalence: we only consider the possible worlds semantics.

Complexity results? Relation between these two notions?
Two Notions of Equivalence

What does it mean for two prob-trees to represent the same information?

Two different notions:

- **Structural Equivalence**: we keep the same event variables.
- **Semantic Equivalence**: we only consider the possible worlds semantics.

Complexity results? Relation between these two notions?
Two Notions of Equivalence

What does it mean for two prob-trees to represent the same information?

Two different notions:

Structural Equivalence: we keep the same event variables.

Semantic Equivalence: we only consider the possible worlds semantics.

Complexity results? Relation between these two notions?
Structural Equivalence

**Definition**

Two prob-trees $T$ and $T'$ are *structurally equivalent* ($T \equiv_{\text{struct}} T'$) if they have the same event variables, the same probability distribution, and if they define the same possible world for every valuation of the event variables.
Definition

Two prob-trees \( T \) and \( T' \) are **structurally equivalent** \((T \equiv_{\text{struct}} T')\) if they have the same event variables, the same probability distribution, and if they define the same possible world for every valuation of the event variables.
Complexity of Structural Equivalence

Theorem

*Structural Equivalence is a *coRP* problem: there exists a randomized polynomial-time algorithm that returns true if two prob-trees are equivalent, and false with probability \( \geq \frac{1}{2} \) otherwise.*

Based on the notion of *count-equivalence:*

Definition

Two propositional formulas \( \psi, \psi' \) in DNF are *count-equivalent* \( (\psi \equiv \psi') \) if, for every valuation of the variables of \( \psi \) and \( \psi' \), the same number of disjuncts of \( \psi \) and \( \psi' \) are satisfied.

\[
A \equiv A \lor (A \land B) \quad \text{but} \quad A \ncong A \lor (A \land B)
\]
Complexity of Structural Equivalence

**Theorem**

Structural Equivalence is a coRP problem: there exists a randomized polynomial-time algorithm that returns true if two prob-trees are equivalent, and false with probability $\geq 1/2$ otherwise.

Based on the notion of count-equivalence:

**Definition**

Two propositional formulas $\psi$, $\psi'$ in DNF are count-equivalent ($\psi \equiv \psi'$) if, for every valuation of the variables of $\psi$ and $\psi'$, the same number of disjuncts of $\psi$ and $\psi'$ are satisfied.

$$A \equiv A \lor (A \land B) \quad \text{but} \quad A \not\equiv A \lor (A \land B)$$
Complexity of Structural Equivalence

Theorem

Structural Equivalence is a coRP problem: there exists a randomized polynomial-time algorithm that returns true if two prob-trees are equivalent, and false with probability \( \geq 1/2 \) otherwise.

Based on the notion of count-equivalence:

Definition

Two propositional formulas \( \psi, \psi' \) in DNF are count-equivalent (\( \psi \equiv^\pm \psi' \)) if, for every valuation of the variables of \( \psi \) and \( \psi' \), the same number of disjuncts of \( \psi \) and \( \psi' \) are satisfied.

\[
A \equiv A \lor (A \land B) \quad \text{but} \quad A \not\equiv^\pm A \lor (A \land B)
\]
Idea behind the Probabilistic Algorithm

In a very simple case:

\[
\begin{align*}
\iff & \quad w_1 \lor (w_2 \land \neg w_3) \\ & \iff X_1 + X_2(1 - X_3)
\end{align*}
\]

\[
\begin{align*}
\iff & \quad (w_1 \land w_2 \land \neg w_3) \lor (w_1 \land w_2) \lor \\
& \quad (w_1 \land \neg w_2) \lor (\neg w_1 \land w_2 \land \neg w_3) \\
& \quad X_1X_2(1 - X_3) + X_1X_2 + \\
& \quad X_1(1 - X_2) + (1 - X_1)X_2(1 - X_3)
\end{align*}
\]

(see [Green, Karvounarakis & Tannen 2007]).

Polynomial-time **randomized algorithm** for determining if a multivariate polynomial is zero [Schwartz 1980].
Idea behind the Probabilistic Algorithm

In a very simple case:

\[ \equiv_{\text{struct}} \quad w_1, w_2, \neg w_3 \quad \equiv \quad (w_1 \land w_2 \land \neg w_3) \lor (w_1 \land w_2) \lor (w_1 \land \neg w_2) \lor (\neg w_1 \land w_2 \land \neg w_3) \]

\[ \iff \quad w_1 \lor (w_2 \land \neg w_3) \quad \iff \quad X_1 + X_2(1 - X_3) \quad = \quad X_1 X_2 (1 - X_3) + X_1 X_2 + X_1 (1 - X_2) + (1 - X_1) X_2 (1 - X_3) \]

(see [Green, Karvounarakis & Tannen 2007]).

Polynomial-time randomized algorithm for determining if a multivariate polynomial is zero [Schwartz 1980].
Idea behind the Probabilistic Algorithm

In a very simple case:

\[
\begin{align*}
A & \quad \equiv \text{struct} \\
B & \quad \equiv (w_1 \lor (w_2 \land \neg w_3)) \\
B & \quad \equiv X_1 + X_2(1 - X_3)
\end{align*}
\]

(see [Green, Karvounarakis & Tannen 2007]).

Polynomial-time randomized algorithm for determining if a multivariate polynomial is zero [Schwartz 1980].
Idea behind the Probabilistic Algorithm

In a very simple case:

\[ A \quad \equiv \quad \text{struct} \quad w_1, w_2, w_3 \]

\[ B \quad \Leftrightarrow \quad w_1 \lor (w_2 \land \lnot w_3) \]

\[ B \quad \Leftrightarrow \quad X_1 + X_2 (1 - X_3) \]

(see [Green, Karvounarakis & Tannen 2007]).

Polynomial-time randomized algorithm for determining if a multivariate polynomial is zero [Schwartz 1980].
Idea behind the Probabilistic Algorithm

In a very simple case:

\[
\begin{align*}
\leftrightarrow & \quad w_1 \lor (w_2 \land \neg w_3) \\
\iff & \quad X_1 + X_2(1 - X_3)
\end{align*}
\]

(see [Green, Karvounarakis & Tannen 2007]).

Polynomial-time \textit{randomized algorithm} for determining if a multivariate polynomial is zero [Schwartz 1980].
Semantic Equivalence

Definition

Two prob-trees $T$ and $T'$ are **semantically equivalent** ($T \equiv_{sem} T'$) if $[T] = [T']$.

<table>
<thead>
<tr>
<th>Event</th>
<th>Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1$</td>
<td>0.5</td>
</tr>
<tr>
<td>$w_2$</td>
<td>0.8</td>
</tr>
<tr>
<td>$w_3$</td>
<td>0.4</td>
</tr>
</tbody>
</table>

Pierre Senellart, Serge Abiteboul

On the Complexity of Probabilistic XML
Semantic Equivalence

Definition

Two prob-trees $T$ and $T'$ are semantically equivalent ($T \equiv_{sem} T'$) if $[T] = [T']$.

<table>
<thead>
<tr>
<th>Event</th>
<th>Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1$</td>
<td>0.5</td>
</tr>
<tr>
<td>$w_2$</td>
<td>0.8</td>
</tr>
<tr>
<td>$w_3$</td>
<td>0.4</td>
</tr>
</tbody>
</table>

$A \quad A$

$\equiv_{sem}$

$B \quad B$

$w_1, w_2$

$w_3$
Semantic Equivalence

**Definition**

Two prob-trees $T$ and $T'$ are **semantically equivalent** ($T \equiv_{sem} T'$) if $[T] = [T']$.

<table>
<thead>
<tr>
<th>Event</th>
<th>Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1$</td>
<td>0.5</td>
</tr>
<tr>
<td>$w_2$</td>
<td>0.8</td>
</tr>
<tr>
<td>$w_3$</td>
<td>0.4</td>
</tr>
</tbody>
</table>

![Diagram](image.png)
**Facts**

1. If $T \equiv_{\text{struct}} T'$, then $T \equiv_{\text{sem}} T'$

2. If $T \equiv_{\text{sem}} T'$ for every possible probability distribution, then $T \equiv_{\text{struct}} T'$.

Complexity of semantic equivalence: open issue. Easy EXPTIME upper bound.
Semantic and Structural Equivalence

Facts

1. If $T \equiv_{\text{struct}} T'$, then $T \equiv_{\text{sem}} T'$

2. If $T \equiv_{\text{sem}} T'$ for every possible probability distribution, then $T \equiv_{\text{struct}} T'$.

Complexity of semantic equivalence: open issue. Easy EXPTIME upper bound.
Fact 1: If $T \equiv_{\text{struct}} T'$, then $T \equiv_{\text{sem}} T'$.

Fact 2: If $T \equiv_{\text{sem}} T'$ for every possible probability distribution, then $T \equiv_{\text{struct}} T'$.

Complexity of semantic equivalence: open issue. Easy EXPTIME upper bound.
Outline

1. Introduction
2. Prob-Trees
3. Equivalence of Prob-Trees
4. Prob-Trees with Additional Constraints
   - Restriction to a Probability Threshold
   - DTD Validation
5. Conclusion
Restriction to a Probability Threshold

Is it possible to remove from a prob-tree least probable worlds?

\[ [T]_{\geq p} \] : set of possible worlds in \([T]\) whose probabilities are greater than \(p\).

**Proposition**

The prob-tree representation of \([T]_{\geq p}\) is sometimes necessarily exponential.
Restriction to a Probability Threshold

- Is it possible to remove from a prob-tree least probable worlds?

- \([T]_{\geq p}\) : set of possible worlds in \([T]\) whose probabilities are greater than \(p\).

**Proposition**

The prob-tree representation of \([T]_{\geq p}\) is sometimes necessarily exponential.
Restriction to a Probability Threshold

Is it possible to remove from a prob-tree least probable worlds?

\([T]_{\geq p}\) : set of possible worlds in \([T]\) whose probabilities are greater than \(p\).

Proposition

The prob-tree representation of \([T']_{\geq p}\) is sometimes necessarily exponential.
DTD Validation

- Is it possible to compute the restriction of a prob-tree to worlds valid against a given DTD?
- DTD definition adapted to the case of unordered trees, and without disjunction.

Proposition

- Deciding if, given a prob-tree, there exists a possible world valid against a DTD is \( NP \)-complete.
- Deciding if, given a prob-tree, all possible worlds are valid against a DTD is \( coNP \)-complete.
- In some cases, the prob-tree representation of the restriction of a prob-tree to a given DTD is of exponential size.
DTD Validation

- Is it possible to compute the restriction of a prob-tree to worlds valid against a given DTD?
- DTD definition adapted to the case of unordered trees, and without disjunction.

**Proposition**

- Deciding if, given a prob-tree, there exists a possible world valid against a DTD is NP-complete.
- Deciding if, given a prob-tree, all possible worlds are valid against a DTD is coNP-complete.
- In some cases, the prob-tree representation of the restriction of a prob-tree to a given DTD is of exponential size.
Is it possible to compute the restriction of a prob-tree to worlds valid against a given DTD?

DTD definition adapted to the case of unordered trees, and without disjunction.

Proposition

Deciding if, given a prob-tree, there exists a possible world valid against a DTD is \( \text{NP} \)-complete.

Deciding if, given a prob-tree, all possible worlds are valid against a DTD is \( \text{coNP} \)-complete.

In some cases, the prob-tree representation of the restriction of a prob-tree to a given DTD is of exponential size.
DTD Validation

- Is it possible to compute the restriction of a prob-tree to worlds valid against a given DTD?
- DTD definition adapted to the case of unordered trees, and without disjunction.

Proposition

- Deciding if, given a prob-tree, there exists a possible world valid against a DTD is $\text{NP}$-complete.
- Deciding if, given a prob-tree, all possible worlds are valid against a DTD is $\text{coNP}$-complete.
- In some cases, the prob-tree representation of the restriction of a prob-tree to a given DTD is of exponential size.
DTD Validation

- Is it possible to compute the restriction of a prob-tree to worlds valid against a given DTD?
- DTD definition adapted to the case of unordered trees, and without disjunction.

Proposition

- **Deciding if, given a prob-tree, there exists a possible world valid against a DTD is NP-complete.**
- **Deciding if, given a prob-tree, all possible worlds are valid against a DTD is coNP-complete.**
- **In some cases, the prob-tree representation of the restriction of a prob-tree to a given DTD is of exponential size.**
Is it possible to compute the restriction of a prob-tree to worlds **valid** against a given DTD?

DTD definition adapted to the case of **unordered trees**, and without disjunction.

**Proposition**

Deciding if, given a prob-tree, there exists a possible world **valid** against a DTD is **NP-complete**.

Deciding if, given a prob-tree, all possible worlds **are valid** against a DTD is **coNP-complete**.

In some cases, the prob-tree representation of the restriction of a prob-tree to a given DTD **is of exponential size**.
Outline

1. Introduction
2. Prob-Trees
3. Equivalence of Prob-Trees
4. Prob-Trees with Additional Constraints

5. Conclusion
   - Summary
   - Perspectives
Summary

- A model for representing probabilistic information in semi-structured databases.

- Polynomial complexity for queries and insertions.

- Unavoidable exponential complexity for deletions.

- Characterization of the complexity of key problems.

- Structural equivalence: randomized polynomial algorithm.
Summary

- A model for representing **probabilistic information** in semi-structured databases.

- **Polynomial** complexity for queries and insertions.

- Unavoidable **exponential** complexity for deletions.

- Characterization of the complexity of **key** problems.

- Structural equivalence: **randomized polynomial** algorithm.
Summary

- A model for representing **probabilistic information** in semi-structured databases.

- **Polynomial** complexity for queries and insertions.

- **Unavoidable exponential** complexity for deletions.

- Characterization of the complexity of **key** problems.

- Structural equivalence: **randomized polynomial** algorithm.
Summary

- A model for representing probabilistic information in semi-structured databases.
- Polynomial complexity for queries and insertions.
- Unavoidable exponential complexity for deletions.
- Characterization of the complexity of key problems.
- Structural equivalence: randomized polynomial algorithm.
Summary

- A model for representing probabilistic information in semi-structured databases.
- Polynomial complexity for queries and insertions.
- Unavoidable exponential complexity for deletions.
- Characterization of the complexity of key problems.
- Structural equivalence: randomized polynomial algorithm.
Perspectives

- Complexity of **semantic equivalence**.
- Prob-tree **simplification**.
- Top-$k$ possible worlds from a prob-tree.
- Aggregate functions in queries.
Perspectives

- **Complexity of semantic equivalence.**
- **Prob-tree simplification.**
- **Top-k** possible worlds from a prob-tree.
- **Aggregate** functions in queries.
Perspectives

- Complexity of semantic equivalence.
- Prob-tree simplification.
- Top-\(k\) possible worlds from a prob-tree.
- Aggregate functions in queries.
Perspectives

- Complexity of **semantic equivalence**.
- Prob-tree **simplification**.
- **Top-k** possible worlds from a prob-tree.
- **Aggregate** functions in queries.
Merci.
Proof of the Exponential Complexity of Deletion

Proof.

**Deletion $d$:** “If the root has a $C$-child, then delete all $B$-children of the root.”

\[ T = \]

Then, it can be shown that if $T' \equiv_{\text{struct}} d(T')$, at least $2^n$ literals appear in $T'$.
Tomasz Imieliński and Witold Lipski. Incomplete information in relational databases. 


Serge Abiteboul and Pierre Senellart. Querying and updating probabilistic information in XML. In Extending Database Technology, Munich, Germany, March 2006.