Validity-Sensitive Querying of XML Databases

Slawomir Staworko  Jan Chomicki

Department of Computer Science and Engineering
University at Buffalo, SUNY
Motivation

Querying Invalid XML

- Integration of XML documents
- Slight differences between schemas (e.g. different cardinality constraints)
- Legacy XML databases
- Database updates
Querying Invalid XML

- Integration of XML documents
- Slight differences between schemas (e.g. different cardinality constraints)
- Legacy XML databases
- Database updates

Document Type Definition ($D_0$)

\[
\begin{align*}
\text{proj} & \rightarrow (\text{name}, \text{emp}, \text{proj}^*, \text{emp}^*) \\
\text{emp} & \rightarrow (\text{name}, \text{salary}) \\
\text{name} & \rightarrow \#PCDATA \\
\text{salary} & \rightarrow \#PCDATA
\end{align*}
\]

Query: get salaries of all employees that are not managers

```xml
//proj/name/emp/following-sibling::emp/salary
```
Motivation

Querying Invalid XML

- Integration of XML documents
- Slight differences between schemas (e.g. different cardinality constraints)
- Legacy XML databases
- Database updates

Document Type Definition ($D_0$)

```
proj → (name, emp, proj*, emp*)
emp → (name, salary)
name → #PCDATA
salary → #PCDATA
```

Query: get salaries of all employees that are not managers

```
//proj/name/emp/following-sibling::emp/salary
```
**Motivation**

- Integration of XML documents
- Slight differences between schemas (e.g. different cardinality constraints)
- Legacy XML databases
- Database updates

**Querying Invalid XML**

**Document Type Definition ($D_0$)**

- proj → (name, emp, proj*, emp*)
- emp → (name, salary)
- name → #PCDATA
- salary → #PCDATA

**Query**: get salaries of all employees that are not managers

```xml
//proj/name/emp/following-sibling::emp/salary
```
Motivation

Querying Invalid XML

- Integration of XML documents
- Slight differences between schemas (e.g. different cardinality constraints)
- Legacy XML databases
- Database updates

Document Type Definition (D₀)

proj → (name, emp, proj*, emp*)
emp → (name, salary)
name → #PCDATA
salary → #PCDATA

Query: get salaries of all employees that are not managers

//proj/name/emp/following-sibling::emp/salary
Positive Core XPath

Positive Core XPath Queries

- text values but without attributes
- all standard axes
- tests with subexpressions and equality:
  
  \( //*[A/B], //*[B//text()=\text{\textquoteleft}abcd\text{\textquoteright}] \)
- but no negation in tests:
  
  \( //*[\text{not} A/B], //*[B/text() \neq \text{\textquoteleft}abcd\text{\textquoteright}] \)
- and no functions

Tree Reachability Fact: \((x, Q, y)\)

Basic facts use only */ and following_sibling:

\[(N_0,/*,N_1), (N_1,/*,N_2)\]

Other facts are inferred with (Horn) rules
\[(X, Q/P, Y) \Leftarrow (X, Q, Z) \land (Z, P, Y).\]
\[(N_0,/*/*,N_2).\]

Query Answers

- given query \(Q\) and document \(T\) with the root node \(r\)
- find all tree facts that hold in \(T\)
- \(x\) in an answer to \(Q\) in \(T\) iff the tree fact \((r, Q, x)\) holds in \(T\)
Query Evaluation

**Bottom-up approach**
- computing tree facts for query $Q$
- tree facts for $T_1, \ldots, T_m$ computed before
- including inferred facts (involving subqueries of $Q$)

**Algorithm**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>start with $\emptyset$</td>
<td></td>
</tr>
<tr>
<td>II</td>
<td>for subtree $T_i$ ($i = 1, \ldots, m$)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 add all facts of the subtree (obtained by recursion)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 add $(N_0,/*,N_i)$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 if $i &gt; 1$ add $(N_{i-1},following_sibling::*,N_i)$</td>
<td></td>
</tr>
</tbody>
</table>

**Tree**

![Diagram of a tree structure](image)
Editing operations

A

B
C
D

Cost: 3

Delete:

B
C
D
H

Cost: 2

Modify:

Cost: 1
Editing operations

- Inserting a subtree
  
  Cost: 3

- Deleting a subtree
  
  Cost: 2

- Modify
  
  Cost: 1
Editing operations

- Inserting a subtree
- Deleting a subtree

Cost: 3

Cost: 2
Editing operations

- Inserting a subtree
- Deleting a subtree
- Modifying node’s label
Edit Distance and Repairs

**Distance between documents**

\[ \text{dist}(T, S) \] is the minimal cost of transforming \( T \) into \( S \)

**Distance to a DTD**

\[ \text{dist}(T, D) \] is the minimal cost of repairing \( T \) w.r.t \( D \) i.e.,

\[ \min \{ \text{dist}(T, S) | S \text{ valid w.r.t } D \} \]

**DTD**

\[ \text{C} \rightarrow (\text{A,B})^* \]
\[ \text{A} \rightarrow \text{EMPTY} \]
\[ \text{B} \rightarrow \text{EMPTY} \]

**Repair**

\( T' \) is a repair of \( T \) w.r.t \( D \) iff

\[ \text{dist}(T', T) = \text{dist}(T, D) \]

There can be an exponential number of repairs
Valid Query Answers

\[ \text{x is a valid answer to query } Q \text{ in } T \text{ w.r.t. } D \text{ iff } \]
\[ \text{x is an answer to } Q \text{ in every repair of } T \text{ w.r.t. } D. \]

Document Type Definition (\(D_0\))

\[
\begin{align*}
\text{proj} & \rightarrow (\text{name, emp, proj*, emp*}) \\
\text{emp} & \rightarrow (\text{name, salary}) \\
\text{name} & \rightarrow \#PCDATA \\
\text{salary} & \rightarrow \#PCDATA
\end{align*}
\]

Queries and Valid Answers

\[
\begin{align*}
//\text{proj[emp[1]/salary='}$90K']$/name/text() & \rightarrow \{\text{Pierogies}\} \\
//\text{proj[name='Pierogies'/emp[1]/salary/text() & } \rightarrow \{\text{$90K}\} \\
//\text{proj[name='Pierogies']/emp[1]/name/text()} & \rightarrow \emptyset \\
//\text{proj[name='Pierogies']/emp[1]/salary} & \rightarrow \emptyset
\end{align*}
\]
Trace graph

**DTD**

\[
\begin{align*}
C & \rightarrow (A,B)^* \\
A & \rightarrow \text{EMPTY} \\
B & \rightarrow \text{EMPTY}
\end{align*}
\]
Trace graph

DTD
- C → (A,B)*
- A → EMPTY
- B → EMPTY

A → EMPTY

B → EMPTY
Repairing Paths:

- (Read, Read, Del)
- (Read, Read, Ins A, Read)
- (Read, Del, Read)

Compact representation of all repairs:

A → (A, B)*
A → EMPTY
B → EMPTY

Trace graph
Trace graph

**DTD**

\[
\begin{align*}
C & \rightarrow (A,B)^* \\
A & \rightarrow \text{EMPTY} \\
B & \rightarrow \text{EMPTY}
\end{align*}
\]

**Trace graph**
Trace graph

- **DTD**
  - $C \rightarrow (A,B)^*$
  - $A \rightarrow \text{EMPTY}$
  - $B \rightarrow \text{EMPTY}$

- **Set Difference Tree**
  - Initial state: $Q_0$
  - Transitions:
    - $Q_0 \xrightarrow{\text{Del}} Q_0$
    - $Q_0 \xrightarrow{\text{Read A}} Q_0$
    - $Q_0 \xrightarrow{\text{Ins A}} Q_1$
    - $Q_1 \xrightarrow{\text{Del}} Q_1$
    - $Q_1 \xrightarrow{\text{Read B}} Q_1$
    - $Q_1 \xrightarrow{\text{Ins B}} Q_1$

- **Action Symbols**
  - Read
  - Del
  - Ins A
  - Ins B

- **Graph Structure**
  - Nodes: $Q_0$, $Q_1$
  - Edges:
    - $Q_0 \xrightarrow{\text{Del}} Q_0$
    - $Q_0 \xrightarrow{\text{Read A}} Q_0$
    - $Q_0 \xrightarrow{\text{Ins A}} Q_1$
    - $Q_1 \xrightarrow{\text{Del}} Q_1$
    - $Q_1 \xrightarrow{\text{Read B}} Q_1$
    - $Q_1 \xrightarrow{\text{Ins B}} Q_1$
Trace graph

DTD

C → (A,B)*
A → EMPTY
B → EMPTY

TRACE

Q₀,₀ → Q₀,₁ → Q₀,₀ → Q₀,₁ → Q₀,₁

Q₁,₁ → Q₁,₀ → Q₁,₁ → Q₁,₁ → Q₁,₂
_trace graph_

_DTD_

C \rightarrow (A,B)^* 
A \rightarrow \text{EMPTY} 
B \rightarrow \text{EMPTY} 

- \text{Compact representation of all repairs} 

_DTD_

C \rightarrow (A,B)^* 
A \rightarrow \text{EMPTY} 
B \rightarrow \text{EMPTY} 

- \text{Compact representation of all repairs} 

\text{Trace graph}
Compact representation of all repairs

Repairing Paths:
- (Read, Read, Del)
- (Read, Read, Ins A, Read)
- (Read, Del, Read)
Computing Valid Query Answers

Certain Tree Facts
Tree facts present in every repair of a given tree

Bottom-up approach
Precomputed values:
- certain facts for all children
- certain facts common for every minimal tree satisfying DTD

Obtain certain facts by
Intersection of the sets of all repairing paths

For every repairing path, construct:
set of facts collected “so far”:
- start with $∅$
- Read adds certain facts of the corresponding child
- Del adds nothing
- Ins A adds certain facts common for minimal trees labeled with A
Eager intersection

Problem
Possibly an exponential number of paths

Solution: Eager Intersection
For queries without tests of form (joins)
\[ Q/text() = P/text() \].
Intersect all sets of certain facts for paths sharing the same last adding operation (Read/Ins).

Data complexity of VQA
Computation of valid answers to positive core XPath queries without joins can be performed in polynomial time in the size of the document.

Data complexity of VQA with joins
There exists a query with joins for which computing VQA is co-NP-complete.

Combined complexity of VQA
Combined complexity of computing valid query answers is co-NP-complete.
Experimental Results: Edit Distance Computation

Compared algorithms

<table>
<thead>
<tr>
<th>Parse</th>
<th>base line</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validate</td>
<td>regular automata</td>
</tr>
<tr>
<td>Dist</td>
<td>distance computation</td>
</tr>
</tbody>
</table>

Data generation

1. random valid document
2. removing and adding random nodes
3. invalidity ratio
   \[ \text{dist}(T, D)/|T| \approx 0.1\% \]
4. small height (8-10)
Experimental Results: Edit Distance Computation

Compared algorithms

- Parse: base line
- Validate: regular automata
- Dist: distance computation

Data generation

1. random valid document
2. removing and adding random nodes
3. invalidity ratio
   \[ \text{dist}(T, D) / |T| \approx 0.1\% \]
4. small height (8-10)
Experimental Results: Edit Distance Computation

Compared algorithms
- **PARSE** base line
- **VALIDATE** regular automata
- **DIST** distance computation

Data generation
1. random valid document
2. removing and adding random nodes
3. invalidity ratio
   \[ \text{dist}(T, D)/|T| \approx 0.1\% \]
4. small height (8-10)
Experimental Results: Valid Query Answer Computation

**Compared algorithms**

- QA base line
- VQA eager intersection

**Data generation**

- invalidity ratio

\[ \text{dist}(T, D) / |T| \approx 0.1\% \]

- small height (8-10)

**Implemented Queries**

- //, /, following_sibling::
- name()=A, text()='str'
- QA works in linear time
Experimental Results: Valid Query Answer Computation

**Compared algorithms**
- QA base line
- VQA eager intersection

**Data generation**
- invalidity ratio
  \[ \frac{dist(T, D)}{|T|} \approx 0.1\% \]
- small height (8-10)

**Implemented Queries**
- `//, /, following_sibling::`
- `name()=A,text()='str'`
- QA works in linear time

---

**Graph**
- Green line: QA
- Brown line: VQA

- Y-axis: Time (sec.)
- X-axis: Document size (MB)
Experimental Results: Valid Query Answer Computation

**Compared algorithms**
- QA base line
- VQA eager intersection

**Data generation**
- invalidity ratio
  \[ \text{dist}(T, D)/|T| \approx 0.1\% \]
- small height (8-10)

**Implemented Queries**
- `//, /, following_sibling::`
- `name()=A,text()='str'`
- QA works in linear time

![Graph showing time vs. DTD size for VQA]

- VQA

![Graph showing time vs. DTD size for VQA]

- DTD size $|D|$
- Time (sec.)

Data generation
- invalidity ratio
- $\text{dist}(T, D)/|T| \approx 0.1\%$
- small height (8-10)
Conclusions and Future Work

Conclusions

- Framework for querying of documents with validity violations of local nature (missing or superfluous nodes)
- Efficient algorithm for computing valid answers to a class of XPath queries

Future Work

- Valid answers by query rewriting
- Valid answers to queries with negation
- Other tree operations (swap, shift, contraction, . . .)
- Semantic inconsistencies (keys, functional dependencies, . . .)