Structured Materialized Views for XML Queries

Ioana Manolescu\textsuperscript{1}
Joint work with: Andrei Arion\textsuperscript{1,2} Véronique Benzaken\textsuperscript{2}
Yannis Papakonstantinou\textsuperscript{3} Ravi Vijay \textsuperscript{4}

\textsuperscript{1}GEMO group, INRIA Saclay – Île-de-France
\textsuperscript{2}LRI, Université Paris XI
\textsuperscript{3}University of California, San Diego
\textsuperscript{4}Indian Institute of Technology, Bombay

January 25, 2008
Outline

1. Overview
2. XML Access Modules
3. Answering queries using XAMs
4. Related works
5. Conclusion
Motivation: physical data independence in XML databases

- Fundamental concept in relational database management systems (RDBMSs): *physical data independence*
- Main goal of this work: physical data independence in XML databases
General RDBMS architecture

Query (SQL) -> Query analyzer

Data -> Algebraic optimizer

Loader -> Execution engine

Storage manager

data store  indices  materialized views

Table expressions
Logical model
Relational algebra
Physical model
Physical (rel) op.
(hash join, nlj,...)

Tables on disk
B+ trees
Hash Indexes
Physical data independence in RDBMs

- Query and applications refer to the data at the **logical level**
- **Transparent access path selection algorithm**
  - choose between equivalent data access methods

**Advantages:**
- applications ignore the details on how the data is stored and accessed
- **flexibility** when confronted to storage changes
Physical data independence in RDBMs

- Query and applications refer to the data at the **logical level**
- **Transparent access path selection algorithm**
  - choose between equivalent data access methods
- **Advantages:**
  - applications ignore the details on how the data is stored and accessed
  - **flexibility** when confronted to storage changes
Most research prototypes and commercial systems:
- fixed storage strategies, some indices
- optimizer and execution engine are tuned for this setting

We need:
- a high-level language for describing disk-resident storage structures
- algorithms for query answering
Physical data independence for XML

Most research prototypes and commercial systems:

- fixed storage strategies, some indices
- optimizer and execution engine are tuned for this setting

We need:

- a high-level language for describing disk-resident storage structures
- algorithms for query answering
Proposed architecture

New layer describing the information contained in persistent storage structures

- Approach
  - describe storage module (storage, indices, views) using XAMs
  - optimize queries over the XAMs
  - update the storage
  - update the XAM set.

- Query answering problem
  - query rewriting using views
Proposed architecture

New layer describing the information contained in persistent storage structures

**Approach**
- describe storage module (storage, indices, views) using XAMs
- optimize queries over the XAMs
- update the storage → update the XAM set.

**Query answering problem** → query rewriting using views
Proposed architecture

New layer describing the information contained in persistent storage structures

- **Approach**
  - describe storage module (storage, indices, views) using XAMs
  - optimize queries over the XAMs
  - update the storage
  - update the XAM set.

- **Query answering problem**
  - query rewriting using views
Contributions

1. XML Access Modules: expressive language for storage structures [XIMEP2005]

2. Methodology for answering XQuery queries based on XAMs
   - algorithm for extracting XAMs from conjunctive XQuery [FQAS2006]
   - XAM containment and rewriting under summary constraints [VLDB 2007]
   - implemented in the ULoad tool [VLDB 2005]
Contributions

1. XML Access Modules: expressive language for storage structures [XIMEP2005]
2. Methodology for answering XQuery queries based on XAMs
   - algorithm for extracting XAMs from conjunctive XQuery [FQAS2006]
   - XAM containment and rewriting under summary constraints [VLDB 2007]
   - implemented in the ULoad tool [VLDB 2005]
Contributions

1. XML Access Modules: expressive language for storage structures [XIMEP2005]
2. Methodology for answering XQuery queries based on XAMs
   - algorithm for extracting XAMs from conjunctive XQuery [FQAS2006]
   - XAM containment and rewriting under summary constraints [VLDB 2007]
   - implemented in the ULoad tool [VLDB 2005]
Outline

1. Overview
2. XML Access Modules
3. Answering queries using XAMs
4. Related works
5. Conclusion
Describing XML storage structures, indices and views

- **Genericity**: many fragmentation strategies
- **Document order** may be preserved
- **Document structure** may be preserved
- **Node identifiers**
- **Various degrees of fragmentation**
- **Model**: trees with nested tuples semantics

Different data sets and workloads may required any combination
Describing XML storage structures, indices and views

- **Genericity**: many fragmentation strategies
- **Document order** may be preserved
- **Document structure** may be preserved
- **Node identifiers**
- **Various degrees of fragmentation**
- **Model**: trees with nested tuples semantics

Different data sets and workloads may required any combination
An XML document
Another XML document
Modeling XML views: XQueries

for $x in //item return
  <res>
    {$x/name/text()}
    {$x//@keyword}
  </res>

- Use an XQuery view that materialize the whole query.
- Problems:
  - Complex queries return new elements (not part of the document)
  - Difficult to combine multiple XQuery views
Modeling XML views: XQueries

```xml
for $x in //item return
  <res>
    {$x/name/text()}
    {$x//keyword}
  </res>
```

- Use an XQuery view that materialize the whole query.
- Problems:
  - Complex queries return new elements (not part of the document)
  - Difficult to *combine* multiple XQuery views
Modeling XML views: XPath

for $x$ in //item return

<res>
  {$x/name/text()}
  {$x//keyword}
</res>

- XPath: $V = //item$
  - $Q = E_1(V)$
  - Problems: $V$ is big, $E_1$ is complex

- XPath: $V_1 = //item/name$, $V_2 = //item//keyword$
  - $Q = E_2(V_1, V_2)$
  - XQuery semantics: output an empty res element even if no name, keyword!
  - Problem: How to combine $V_1$ and $V_2$? Add some IDs!
Modeling XML views: XPath

for $x$ in //item return

$$<\text{res}>
\{x\text{/name/text()}
\{x\text{/keyword}
</\text{res}>$$

- XPath: $V=\text{//item}$
  - $Q=E_1(V)$
  - Problems: $V$ is big, $E_1$ is complex
- XPath: $V_1=\text{//item/name}$, $V_2=\text{//item//keyword}$
  - $Q = E_2(V_1, V_2)$
  - XQuery semantics: output an empty res element even if no $name$, $keyword$!
  - Problem: How to combine $V_1$ and $V_2$? Add some IDs!
Modeling XML views: XPath

for $x$ in //item return
  <res>
    {$x/name/text()}
    {$x//keyword}
  </res>

- XPath: $V$=//item
  - $Q=E_1(V)$
  - Problems: $V$ is big, $E_1$ is complex

- XPath: $V_1$=//item/name, $V_2$=//item//keyword
  - $Q = E_2(V_1, V_2)$
  - XQuery semantics: output an empty res element even if no name, keyword!
  - Problem: How to combine $V_1$ and $V_2$? Add some IDs!
Modeling XML views: XPath

for $x$ in //item return
  <res>
    {$x/name/text()}
    {$x//keyword}
  </res>

- XPath: $V=$//item
  - $Q=E_1(V)$
  - Problems: $V$ is big, $E_1$ is complex

- XPath: $V_1=$//item/name, $V_2=$//item//keyword
  - $Q = E_2(V_1, V_2)$
  - XQuery semantics: output an empty res element even if no name, keyword!
  - Problem: How to combine $V_1$ and $V_2$? Add some IDs!
Modeling XML views: enhanced tree patterns

for $x$ in //item return
    <res>
        {$x/name/text()} 
        {$x//keyword} 
    </res>

- Store IDs in the view: $V_1$=//item ID, $V_2$=//name ID, $V_3$=//keyword ID
  - $Q = V_1 \bowtie ID V_2 \bowtie ID V_3$
  - Problem: still store too much!

- ID, Val, Cont in the view definition: $V_1$=//item ID, $V_2$=//name ID Val, $V_3$=//keyword ID Cont
  - $Q = V_1 \bowtie ID V_2 \bowtie ID V_3$
  - Can I eliminate the joins? Express outer joins in view!
Modeling XML views: enhanced tree patterns

for $x$ in //item return
  <res>
    {$x//keyword}
    {$x//name/text()}
  </res>

- Store IDs in the view $V_1=//item$ ID, $V_2=//name$ ID, $V_3=//keyword$ ID
  - $Q = V_1 \bowtie_{ID} V_2 \bowtie_{ID} V_3$
  - Problem: still store too much!

- ID, Val, Cont in the view definition $V_1=//item$ ID, $V_2=//name$ ID Val, $V_3=//keyword$ ID Cont
  - $Q = V_1 \bowtie_{ID} V_2 \bowtie_{ID} V_3$
  - Can I eliminate the joins? Express outer joins in view!
Modeling XML views: enhanced tree patterns

for $x$ in //item return
  <res>
    {$x/name/text()}
    {$x//keyword}
  </res>

- Store IDs in the view $V_1=$//item ID, $V_2=$//name ID, $V_3=$//keyword ID
  - $Q = V_1 \bowtie_{ID} V_2 \bowtie_{ID} V_3$
  - Problem: still store too much!
- ID, Val, Cont in the view definition $V_1=$//item ID, $V_2=$//name ID Val, $V_3=$//keyword ID Cont
  - $Q = V_1 \bowtie_{ID} V_2 \bowtie_{ID} V_3$
  - Can I eliminate the joins? Express outer joins in view!
Modeling XML views: enhanced tree patterns

```
for $x in //item return
  <res>
    {$x/name/text()}
    {$x//keyword}
  </res>
```

- Store ID, Cont, Val + optional edges

<table>
<thead>
<tr>
<th>V</th>
<th>item ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>name</td>
</tr>
<tr>
<td></td>
<td>keyword</td>
</tr>
</tbody>
</table>

- $Q = \text{GroupBy}_{ID}(V)$
- Can I eliminate the need for group by? Add nesting!
Modeling XML views: enhanced tree patterns

for $x$ in //item return
<res>
  {$x/name/text()} 
  {$x//keyword} 
</res>

- Store ID, Cont, Val + optional edges

$Q = \text{GroupBy}_{ID}(V)$

Can I eliminate the need for group by? Add nesting!
Modeling XML views: enhanced tree patterns

for $x$ in //item return
   <res>
      {$x/name/text()}
      {$x//keyword}
   </res>

- Store ID, Cont, Val + optional edges

- $Q = \text{GroupBy}_{ID}(V)$
- Can I eliminate the need for group by? Add nesting!
XML Access Modules = tree patterns with IDs, Val, Cont + optional and nested edges + value predicates

\[ Q = V \] (modulo element construction)
XAMs by example

\[
\begin{array}{ccccccc}
\text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{a} \\
\text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} \\
\text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{a} \\
\text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} \\
\text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{a} \\
\text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} \\
\text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{a} \\
\text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} \\
\end{array}
\]
XAMs by example

Can you...
XAMs by example

![XML Access Modules Diagram]

<table>
<thead>
<tr>
<th>ID</th>
<th>A</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>n4</td>
<td>&lt;listitem&gt;&lt;keyword&gt;Columbus&lt;/keyword&gt;&lt;text&gt;Italic fountain pen&lt;/text&gt;&lt;/listitem&gt;</td>
<td>gold plated</td>
</tr>
<tr>
<td></td>
<td>&lt;listitem&gt;&lt;text&gt;Stainless steel, &lt;bold&gt;gold plated&lt;/text&gt;&lt;/listitem&gt;</td>
<td></td>
</tr>
<tr>
<td>n21</td>
<td>&lt;listitem&gt;&lt;text&gt;Monteverdi Invincia pen&lt;/text&gt;&lt;/listitem&gt;</td>
<td></td>
</tr>
</tbody>
</table>
XAMs by example

```
<table>
<thead>
<tr>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
</tr>
</thead>
</table>
| *   |  b  |  ID |  b  |  IDs|  b  |  V  |  b  | Cont|  b  |  V="7"
```
XAMs by example

```
<table>
<thead>
<tr>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
</tbody>
</table>

```

```
<table>
<thead>
<tr>
<th>a</th>
<th>a</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>b</td>
<td>ID</td>
</tr>
</tbody>
</table>

```

```
<table>
<thead>
<tr>
<th>a</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>IDs</td>
</tr>
</tbody>
</table>

```

```
<table>
<thead>
<tr>
<th>a</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>V</td>
</tr>
</tbody>
</table>

```

```
<table>
<thead>
<tr>
<th>a</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>Cont</td>
</tr>
</tbody>
</table>

```

```
<table>
<thead>
<tr>
<th>a</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>V=&quot;7&quot;</td>
</tr>
</tbody>
</table>

```
Overview

XML Access Modules

Answering queries using XAMs

Related works

Conclusion
Answering XQuery queries using XAMs
Query answering approach

- XQuery query
- Pattern extraction
- XMLize
- Query XAMs
- Storage XAM definitions
- Answering queries using XAMs
- Logical operators
- Plan Finalizer
- XMLize
- Logical operators
- XAMScan
- XAMScan
- XAMScan
- XAMScan
Query rewriting: problem statement (1)

**Input:** Path summary $S$, XQuery $Q$, set of XAMs $X_1, X_2, \ldots, X_n$

**Output:** all minimal algebraic expressions $e(X_1, X_2, \ldots, X_n)$ (up to algebraic equivalence) s.t. $\forall d$

$$Q(d) = e(X_1, X_2, \ldots, X_n)(d)$$

**Algebra:** $\sigma$, $\Pi$, $\bowtie_{ID}$, $\bowtie_{ID}$, $\bowtie_{\prec}$, $\bowtie_{\preceq}$ (variants: ancestor-descendant, nested joins), $\text{Nest}$, $\text{Unnest}$, $\text{Nav}$ and $\bigcup$
Input: Path summary $S$, XQuery $Q$, set of XAMs $X_1, X_2, \ldots, X_n$

Output: all minimal algebraic expressions $e(X_1, X_2, \ldots, X_n)$ (up to algebraic equivalence) s.t. $\forall d$

$$Q(d) = e(X_1, X_2, \ldots, X_n)(d)$$

Algebra: $\sigma$, $\Pi$, $\bowtie_{ID}$, $\bowtie_{ID}$, $\bowtie_{\prec}$, $\bowtie_{\prec}$ (variants: ancestor-descendant, nested joins), $\text{Nest}$, $\text{Unnest}$, $\text{Nav}$ and $\cup$
Query rewriting: problem statement (1)

Input: Path summary $S$, XQuery $Q$, set of XAMs $X_1, X_2, \ldots, X_n$

Output: all minimal algebraic expressions $e(X_1, X_2, \ldots, X_n)$ (up to algebraic equivalence) s.t. $\forall d$

$$Q(d) = e(X_1, X_2, \ldots, X_n)(d)$$

Algebra: $\sigma$, $\Pi$, $\Join_{ID}$, $\Join_{ID}$, $\Join_\prec$, $\Join_\preceq$ (variants: ancestor-descendant, nested joins), $Nest$, $Unnest$, $Nav$ and $\cup$
Query rewriting: problem statement (2)

Input: Path summary $S$, XQuery $Q$, set of XAMs $X_1, X_2, \ldots, X_n$

Output: all minimal algebraic expressions $e(X_1, X_2, \ldots, X_n)$ (up to algebraic equivalence) s.t. $\forall d$ conforming to path summary (or dataguide) $S$

$$Q(d) = e(X_1, X_2, \ldots, X_n)(d)$$

Algebra: $\sigma$, $\Pi$, $\bowtie_{ID}$, $\bowtie_{ID}$, $\bowtie_{\prec}$, $\bowtie_{\prec}$ (variants: ancestor-descendant, nested joins), $\text{Nest}$, $\text{Unnest}$, $\text{Nav}$ and $\cup$
**Path summary**

**Simple summary** of a document $d$:
- root, label and parent-preserving mapping $\phi : d \rightarrow S$
- the children of a summary node have distinct labels

A document $d'$ conforms to a path summary $S$ ($S \models d'$) iff
$\exists \phi' : d' \rightarrow S$
Simple summary of a document $d$:  
- root, label and parent-preserving mapping $\phi : d \rightarrow S$  
- the children of a summary node have distinct labels  

A document $d'$ conforms to a path summary $S$ ($S \models d'$) iff  
$\exists \phi' : d' \rightarrow S$
Path summary

*Simple summary* of a document $d$:

- root, label and parent-preserving mapping $\phi : d \rightarrow S$
- the children of a summary node have distinct labels

A document $d'$ conforms to a path summary $S$ ($S \models d'$) iff

$$\exists \phi' : d' \rightarrow S$$
Path summaries and XAMs in query rewriting
Path summaries and XAMs in query rewriting
Path summaries and XAMs in query rewriting

\[ Q \]
\[ a \]
\[ b \]
\[ e \]

\[ S \]
\[ a \]
\[ b \]
\[ c \]
\[ d \]
\[ b \]
\[ e \]

\[ V1 \]
\[ a \]
\[ b \]
\[ c \]

\[ V2 \]
\[ c \]
\[ b \]
\[ d \]
\[ b \]
\[ e \]
Pattern extraction

Answering queries using XAMs

XMLize

Storage XAM definitions

XQuery

Equations

Logical operators

Plan Finalizer

Related works

Conclusion

Overview

XML Access Modules
Pattern extraction overview

- Algebraic translation to capture XQuery semantics
- Exploits nesting and optional edges to find maximal XAMs that:
  - cut across nested FLWOR expressions
  - are larger than the views identified by previous approaches
  - allow to use larger views in query rewriting ⇒ better performance
Pattern extraction example

for $x$ in doc("XMark.xml")//namerica//item
return ⟨res1⟩ {for $y$ in $x$//parlist return
    ⟨res2⟩ {for $z$ in $y$//text return ⟨res3⟩ {$z$} ⟨/res3⟩ } ⟨/res2⟩ }
⟨/res1⟩
XAM rewriting using XAM views

Diagram:
- **Pattern extraction**: XQuery query
- **XMLize**: XQuery, XQ_1, XQ_2, ..., XQ_n
- **Storage XAM definitions**: EQ^r
- **Answering queries using XAMs**: EQ^l
- **XMLize**
- **Logical operators**
- **XAMScan**
Query rewriting algorithm

\[ \text{XQuery } Q \rightarrow E \]

\[ \text{XQ}_1 \quad \text{XQ}_2 \quad \ldots \quad \text{XQ}_m \]
Query rewriting algorithm

\[ \text{XQuery } Q \rightarrow XQ_1 \rightarrow XQ_2 \rightarrow \ldots \rightarrow XQ_m \rightarrow E \]

\[ X_1, X_2, \ldots, X_n \]
Query rewriting algorithm

XQuery Q → \[E\]

\[XQ_1, XQ_2, \ldots, XQ_m\]

\[S\] → \[e_{1,1}\]
Query rewriting algorithm
Query rewriting algorithm

\[
\text{XQuery } Q \rightarrow XQ_1 \rightarrow XQ_2 \rightarrow \ldots \rightarrow XQ_m
\]

\[
S = \{X_1, X_2, \ldots, X_n\}
\]

\[
E = \{e_{1,1}, e_{1,2}, \ldots, e_{1,k_1}\}
\]
Query rewriting algorithm

\[ \text{XQuery } Q \rightarrow \text{XQ}_1 \rightarrow \ldots \rightarrow \text{XQ}_n \]

\[ S \rightarrow e_{1,1} \rightarrow e_{1,2} \rightarrow \ldots \rightarrow e_{1,k1} \]

\[ \text{XQ}_1 \rightarrow e_{2,1} \rightarrow e_{2,2} \rightarrow \ldots \rightarrow e_{2,k2} \]

\[ \ldots \rightarrow E \]
Query rewriting algorithm
Query rewriting algorithm
Query rewriting algorithm
Query rewriting algorithm

XQuery Q → E

X1, X2, ..., Xn → e1,1, e1,2, ..., e1,k1, e2,1, e2,2, ..., e2,k2, ..., em,1, em,2, ..., em,km
Query rewriting algorithm
Query rewriting algorithm

XQuery Q

XQ1, XQ2, ..., XQm

S

e1,1, e1,2, ..., e1,k1

e2,1, e2,2, ..., e2,k2

... e2,1, e2,2, ..., e2,k2

... e1,1, e1,2, ..., e1,k1

E

e1,1, e2,2, ..., em,km

E

e1,1, e2,2, ..., em,km

Zoom on XAM rewriting

Rewriting one query XAM

- In the presence of a summary bucket algorithms would be incomplete
- Inflationary algorithm combining \((plan, equivalent\text{XAM})\) pairs:
  - combine (plan, pattern) pairs via (structural) joins
  - a plan is a full rewriting when its equivalent XAM is \(\equiv_S\) to the query XAM.
  - XAM containment algorithm under \(S\) constraints
- When to stop: no new plans or plans outgrow a certain size
- Aggressive summary-based pruning of plans to keep search space manageable
Plan combination: zipping algorithm

\[ p_1 \bowtie p_2 \]

\[ b.ID = b.ID \]

\[ p_1 \]

\[ p_2 \]

\[ \text{a, b, ID} \]

\[ \text{x, ID} \]
Plan combination: zipping algorithm

\[ p_1 \quad p_2 \quad p_1 \Join p_2 \quad b.ID = b.ID \]

ID \ a
ID \ b
ID \ y
ID \ x
ID \ x \ y
ID \ x
ID \ b
ID \ y
ID \ a
Plan combination: zipping algorithm

\[
p_1 \quad p_2
\]

\[
p_3 \quad p_4\]

\[
p_1 \bowtie p_2
\]

\[
b, ID = b, ID
\]

\[
a\]

\[
b, ID \quad y, ID
\]

\[
x, ID
\]

\[
f
\]

\[
d
\]

\[
b, ID \quad b, ID
\]

\[
x, ID
\]

\[
x, ID
\]
Plan combination: zipping algorithm

\[ p_1 \quad p_2 \quad p_3 \quad p_4 \]

\[ p_1 \bowtie p_2 \]
\[ b.ID=b.ID \]

\[ p_3 \bowtie p_4 \]
\[ b.ID=b.ID \]
Plan combination: zipping algorithm
Plan combination: zipping algorithm

Full algorithm handles branches below and above the join point and structural joins.
Rewriting XQuery using XAMs

for $x$ in //item[//mail] return
<res> {$x/name/text(),
  for $y$ in $x$//listitem return
  <key> {$y//keyword} </key>} </res>

Can you...
Rewriting XQuery using XAMs

If document structure is known

for $x$ in //item[//mail] return
<res> {$x/name/text(),
  for $y$ in $x$//listitem return
  <key> {$y//keyword} </key>} </res>
Full results

- **XAM language**
  - Value and structure predicates
  - Nesting
  - Optional edges
  - Structural and non-structural IDs

- Enhanced summaries: integrity constraints (required children)
Summary based containment

Definition

\( p \subseteq_S p' \) iff for any \( t \) such that \( S \models t \), \( p(t) \subseteq p'(t) \).

Algorithm:
- consider trees from a finite set canonical model of \( S \)
- if no canonical tree is a counter-example then \( p \subseteq_S p' \)
Summary based containment

**Definition**

\[ p \subseteq_S p' \text{ iff for any } t \text{ such that } S \models t, \ p(t) \subseteq p'(t). \]

**Algorithm:**

- consider trees from a finite set canonical model of \( S \)
- if no canonical tree is a counter-example then \( p \subseteq_S p' \)
Summary based containment

**Definition**

$p$ is $S$-contained in $p'$ iff for any $t$ such that $S \models t$, $p(t) \subseteq p'(t)$.

**Proposition:** $p \subseteq_S p'$ iff

$\forall \ t_p \in \text{mod}_S(p)$ the return tuple of $p(t_p) \in p'(t_p)$. 
Experimental setting

**ULoad prototype**, Java 1.5, 1GHz CPU, 1GB RAM

- **Documents**
  - DBLP 2005 (280MB)
  - XMark (233MB)

- **Queries**
  - XMark query patterns
  - Randomly generated patterns based on DBLP and XMark path summaries

- **Views**
  - 1 view per each XMark tag (IDs V)
  - randomly generated patterns based on XMark and DBLP
Pattern containment results

Containment of randomly generated patterns, XMark summary (548 nodes)
Pattern containment results

Containment of randomly generated patterns

DBLP '05 summary

- contained, conjunctive patterns
- not contained, conjunctive patterns
- contained, optional
- not contained, optional edges
Query rewriting

- 183 views: 100 generated + 83 tag partition

Rewriting XMark queries

<table>
<thead>
<tr>
<th>Total time (ms)</th>
<th>Setup and pruning time (ms)</th>
<th>Time to 1st solution (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>Q2</td>
<td>Q3</td>
</tr>
<tr>
<td>Q4</td>
<td>Q5</td>
<td>Q6</td>
</tr>
<tr>
<td>Q7</td>
<td>Q8.1</td>
<td>Q8.2</td>
</tr>
<tr>
<td>Q9.1</td>
<td>Q9.2</td>
<td>Q9.3</td>
</tr>
<tr>
<td>Q10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The chart shows the comparison of total time, setup and pruning time, and time to the first solution for different queries (Q1 to Q10).
Outline

1. Overview
2. XML Access Modules
3. Answering queries using XAMs
4. Related works
5. Conclusion
Related works

XPath containment and equivalence
- No constraints:
  - S. Amer-Yahia 2001, Deutsch and Tannen, 2001; Miklau and Suciu, 2002
- DTD constraints:
  - Wood 2003, Neven and Schwentick 2003

XPath rewriting
- Balmin et al, 2004, weak path usage
- Wanhong Xu et al 2005
- Lakshmanan et al, 2006, MCR under path summary constraints

XQuery containment and rewriting
- Halevy et al. 2004
- Onose et al. 2006, equivalent rewriting

We address XAM containment and rewriting under path summaries constraints.
Related works

XPath containment and equivalence
- No constraints:
  - S. Amer-Yahia 2001, Deutsch and Tannen, 2001; Miklau and Suciu, 2002
- DTD constraints:
  - Wood 2003, Neven and Schwentick 2003

XPath rewriting
- Balmin et al, 2004, weak path usage
- Wanhong Xu et al 2005
- Lakshmanan et al, 2006, MCR under path summary constraints

XQuery containment and rewriting
- Halevy et al. 2004
- Onose et al. 2006, equivalent rewriting

We address XAM containment and rewriting under path summaries constraints.
Summary vs. DTD constraints

- **Constraint-free containment**

- **DTD-based containment**
  - type alternatives
    - A=B|C
  - sibling co-occurrence
    - A=BC

- **Summary based containment**
  - closer to the instance
  - bounds for recursion
  - unfold the // edges
**Summary vs. DTD constraints**

Example: Is //a a rewriting for //b/a + //c/a?
Outline

1. Overview
2. XML Access Modules
3. Answering queries using XAMs
4. Related works
5. Conclusion
Conclusions

Contributions

- XAMs: a language for describing storages/indices and materialized views
- Practical algorithms for tree pattern containment and rewriting
  - XAM pattern extraction from XQuery
  - XML query pattern containment and equivalent rewriting based on summary constraints
  - exploiting detailed information about view contents and IDs expressed by XAMs

Future works

- Constraint-free rewriting algorithm
- Cost-based materialized view selection
XAM publications


<table>
<thead>
<tr>
<th>Overview</th>
<th>XML Access Modules</th>
<th>Answering queries using XAMs</th>
<th>Related works</th>
<th>Conclusion</th>
</tr>
</thead>
</table>

Thank you