Structured Materialized Views for XML Queries

Andrei Arion$^{1,2}$ Véronique Benzaken$^2$
Ioana Manolescu$^1$ Yannis Papakonstantinou$^3$

$^1$GEMO Project, INRIA Futurs
firstname.lastname@inria.fr

$^2$Laboratoire de Recherche en Informatique, University Paris XI
firstname.lastname@lri.fr

$^3$University of California San Diego, USA
firstname@cs.ucsd.edu

VLDB 2007, September 25th, 2007
Outline

1. Context: rewriting XQuery using nested views
2. Pattern containment under summary constraints
   - Path summary
   - Summary based containment
3. Summary based query rewriting
4. Experimental results
5. Conclusions
Outline

1. Context: rewriting XQuery using nested views

2. Pattern containment under summary constraints
   - Path summary
   - Summary based containment

3. Summary based query rewriting

4. Experimental results

5. Conclusions
Context: rewriting XQuery using nested views
Context: rewriting XQuery using nested views
Structured Materialized Views for XML Queries

Context: rewriting XQuery using nested views

Diagram showing the process of rewriting XQueries using nested views.
Context: rewriting XQuery using nested views

The diagram illustrates a system for processing XQueries using nested views. The process begins with an XML document, which is loaded into the system. The query analyzer takes the query as input and produces an optimized plan. The execution engine then executes this plan to generate the results. The storage manager manages the storage of index and materialized views, as well as other data access structures, to efficiently retrieve the required data for processing the queries.
Structured Materialized Views for XML Queries

Queries

XML doc

Loader

Query analyzer

Algebraic optimizer

Execution engine

Storage manager

Index

A_1

materialized views

other data access structures

A_1

Answers

Q_1

Q_2
Context: rewriting XQuery using nested views
Context: rewriting XQuery using nested views

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.
Materialized views for XML: Query

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

- Use an XQuery view that materialize the whole query.

Problems:
- The complexity of the XQuery language
- Difficult to understand and combine multiple XQuery views
Materialized views for XML: Query

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

- Use an XQuery view that materialize the whole query.
- Problems:
  - The complexity of the XQuery language
  - Difficult to understand and combine multiple XQuery views
for $x$ in //item return
  <res>
    {$x/name/text()}
    {$x//keyword}
  </res>

- Use an XQuery view that materialize the whole query.
- Problems:
  - The *complexity* of the XQuery language
  - Difficult to *understand and combine* multiple XQuery views
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!
Materialized views for XML: 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!
Materialized views for XML: 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!
Materialized views for XML: 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!
Materialized views for XML: 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!
Materialized views for XML: 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!
Materialized views for XML: 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!
Materialized views for XML: 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!
Materialized views for XML: 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!
Materialized views for XML: 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!
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!
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!
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!
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!
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
  
  $Q = V_1 \bowtie\bowtie ID V_2 \bowtie\bowtie ID V_3$

  **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
  
  $Q = V_1 \bowtie\bowtie ID V_2 \bowtie\bowtie ID V_3$

  **Can I eliminate the joins?** Express outer joins in view!
for $x$ in //item return
  <res>
    {$x/name/text()}
    {$x//keyword}
  </res>

- Store ID, Cont, Val + optional edges

\[ V \quad \text{item ID} \]
\[ V \quad \text{name} \quad \text{keyword} \quad \text{Cont} \]

- \( Q = \text{GroupBy}_{ID}(V) \)
- Can I eliminate the need for group by? Add nesting?
for $x$ in //item return
  <res>
    {$x/name/text()}
    {$x//keyword}
  </res>

- Store ID, Cont, Val + optional edges

\[
V \quad \text{item ID}
\]
\[
\text{V name keyword Cont}
\]

- \( Q = \text{GroupBy}_{\text{ID}}(V) \)
- Can I eliminate the need for group by? Add nesting!
for $x$ in //item return
  <res>
    {$x/name/text()}
    {$x//keyword}
  </res>

- Store ID, Cont, Val + optional edges

```
V
  ^   item ID
    \
    V name  keyword Cont
```

- $Q = \text{GroupBy}_{ID}(V)$
- Can I eliminate the need for group by?

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

- Store ID, Cont, Val + optional edges

\[
V \quad \text{item ID}
\]

\[
V \quad \text{name} \quad \text{keyword} \quad \text{Cont}
\]

- \( Q = \text{GroupBy}_{ID}(V) \)
- Can I eliminate the need for group by? **Add nesting!**
for $x$ in //item return
  <res>
    {$x/name/text()}  
    {$x//keyword}
  </res>

**XML Access Modules** = tree patterns with IDs, Val, Cont +
optional and nested edges + value predicates

\[
V \quad \text{item ID}
\]
\[
\text{n} \quad \text{n}
\]
\[
V \quad \text{name} \quad \text{keyword} \quad \text{Cont}
\]

\[
Q = V
\]
XAMs by examples

\[
\begin{align*}
&\text{a \ a \ a \ a \ a} \\
&\text{b \ b \ b \ b \ b} \\
&\text{a \ a \ a \ a \ a} \\
&\text{b \ b \ b \ b \ b} \\
&\text{a \ a \ a \ a} \\
&\text{b \ ID \ b \ IDs \ b \ V \ b \ Cont \ b \ V=\text{7}}
\end{align*}
\]
XAMs by examples

```
V_1

<table>
<thead>
<tr>
<th>ID</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>n_5</td>
<td>Columbus pen</td>
</tr>
<tr>
<td>n_22</td>
<td>Monteverdi pen</td>
</tr>
</tbody>
</table>
```

Site: n_1
Regions: n_2
Asia: n_3

Item: n_4

Mailbox: n_15
Name: n_22
Description: n_23
Mail: n_16
Monteverdi pen
Parlist: n_24
Mail: n_28
Listitem: n_25
From: n_29
to: n_30
date: n_31
text: n_32

Can you...
XAMs by examples

Context: rewriting XQuery using nested views

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

If document structure is known

V1

<table>
<thead>
<tr>
<th>ID</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>n5</td>
<td>Columbus pen</td>
</tr>
<tr>
<td>n22</td>
<td>Monteverdi pen</td>
</tr>
</tbody>
</table>

V2

<table>
<thead>
<tr>
<th>ID</th>
<th>A</th>
<th>C</th>
<th>V</th>
</tr>
</thead>
</table>

item ANCESTOR−OF name

site $n_1$

regions $n_2$

Asia $n_3$

item $n_4$

name $n_5$
description $n_6$

parlist $n_7$

Columbus pen

Italic fountain pen

Stainless steel, gold plated

listitem $n_8$
text $n_9$

listitem $n_10$
text $n_11$

listitem $n_12$
text $n_13$

listitem $n_14$
text $n_15$

from $n_16$

bill@aol.com

to $n_17$

jane@u2.com

date $n_18$

4/6/2006

text $n_19$

Hello,...

listitem $n_20$
text $n_21$

Monteverdi Invincia pen

item $n_22$
description $n_23$

parlist $n_24$

Columbus pen

Monteverdi pen

listitem $n_25$
text $n_26$

text $n_27$

text $n_31$
to $n_30$

jim@gmail.com

bob@u2.com

date $n_32$


Can you...
XAMs by example

\[
\begin{align*}
&\text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{n} & \text{a} & \text{a} & \text{a} & \text{a} & \text{n} \\
&\text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} \\
&\text{a} & \text{a} & \text{a} & \text{a} & \text{n} & \text{n} & \text{n} & \text{n} & \text{n} & \text{n} & \text{n} & \text{n}
\end{align*}
\]

\[
\begin{align*}
&\text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{a} \\
&\text{*} & \text{b} & \text{ID} & \text{b} & \text{IDs} & \text{b} & \text{V} & \text{b} & \text{Cont} & \text{b} & \text{V}="7"
\end{align*}
\]
XAMs by example

Structured Materialized Views for XML Queries
Outline

1. Context: rewriting XQuery using nested views

2. Pattern containment under summary constraints
   - Path summary
   - Summary based containment

3. Summary based query rewriting

4. Experimental results

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

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

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

Q

a

b
e

V1

a

b
c

V2

c

b
d

b
e
Path summaries and XAMs in query rewriting

\[ Q \]

\[ V1 \]

\[ S \]

\[ V2 \]
Path summaries and XAMs in query rewriting
**Summary based containment**

**Definition**

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

**Problem:** infinitely many trees \( t \)

**Solution:**
- consider trees from a finite set canonical model of \( S \)
- if no canonical tree is a counter-example then \( p \subseteq_S p' \)
**Definition**

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

**Problem:** infinitely many trees \( t \)

**Solution:**
- consider trees from a finite set canonical model of \( S \)
- if no canonical tree is a counter-example then \( p \subseteq_S p' \)
**Definition**

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

**Problem:** infinitely many trees \( t \)

**Solution:**
- consider trees from a finite set *canonical model of* \( S \)
- if no canonical tree is a counter-example then \( p \subseteq_S p' \)
Pattern containment under summary constraints

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

\[ \forall t_p \in mod_S(p) \text{ the return tuple of } p(t_p) \subseteq p'(t_p). \]
Outline

1. Context: rewriting XQuery using nested views

2. Pattern containment under summary constraints
   - Path summary
   - Summary based containment

3. Summary based query rewriting

4. Experimental results

5. Conclusions
Summary based query rewriting

Query rewriting: problem statement

**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 $S$

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

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

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 $S$

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

Algebra: $\sigma$, $\Pi$, $\bowtie_{ID}$, $\bowtie_{ID}$, $\bowtie_{\prec}$, $\bowtie_{\prec}$ (variants: ancestor-descendant, nested joins), $Nest$, $Unnest$, $Nav$ and $\cup$
Summary based query rewriting

Query rewriting: problem statement

**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 $S$

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

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

XQuery Q
Query rewriting algorithm

XQuery $Q \rightarrow XQ_1 \rightarrow \ldots \rightarrow XQ_m$
Query rewriting algorithm

\[ X_{Q_1}, X_{Q_2}, ..., X_{Q_m} \]

\[ X_1, X_2, ..., X_n \]

Summary based query rewriting

Structured Materialized Views for XML Queries
Summary based query rewriting

Query rewriting algorithm

\[ X_{\text{Query}} Q \rightarrow \begin{array}{c}
XQ_1 \\
XQ_2 \\
\vdots \\
XQ_m
\end{array} \]

\[ X_1, X_2, \ldots, X_n \rightarrow e_{1,1} \]
Summary based query rewriting

Query rewriting algorithm

XQuery Q → E

XQ_1, XQ_2, ..., XQ_m

X_1, X_2, ..., X_n → e_{1,1}, e_{1,2}
Query rewriting algorithm
Summary based query rewriting

Query rewriting algorithm

XQuery Q

\( X_1, X_2, \ldots, X_n \)

\( S \)

\( E \)

\( XQ_1 \)

\( XQ_2 \)

\( XQ_m \)

\( e_{1,1} \)

\( e_{1,2} \)

\( e_{1,k1} \)

\( e_{2,1} \)

\( e_{2,2} \)

\( e_{2,k2} \)
Summary based query rewriting

Query rewriting algorithm

\[
\begin{align*}
XQuery \; Q & \rightarrow \cdots \\
XQ_1 & \rightarrow c_{1,1} & \rightarrow e_{1,1} & \rightarrow XQ_2 & \rightarrow c_{2,1} & \rightarrow e_{2,1} & \rightarrow \cdots \\
& \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
S & \rightarrow e_{1,k1} & \rightarrow \cdots & \rightarrow e_{2,k2} & \rightarrow \cdots & \rightarrow e_{m,km}
\end{align*}
\]

Arion, Benzaken, Manolescu, Papakonstantinou  Structured Materialized Views for XML Queries
Summary based query rewriting

Query rewriting algorithm

Structured Materialized Views for XML Queries
Summary based query rewriting

Query rewriting algorithm

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

\[ X_1, X_2, \ldots, X_n \]

\[ S \]

\[ e_{1,1}, e_{1,2}, \ldots, e_{1,k_1} \]

\[ e_{2,1}, e_{2,2}, \ldots, e_{2,k_2} \]

\[ e_{m,1}, e_{m,2}, \ldots, e_{m,k_m} \]

E
Query rewriting algorithm

XQuery Q → XQ₁ → e₁,1 → e₁,2 → e₁,k₁
                      ↓             ↓             ↓
                       XQ₂ → e₂,1 → e₂,2 → e₂,k₂
                              ↓             ↓             ↓
                                XQₘ → eₘ,1 → eₘ,2 → eₘ,kₘ
                                      ↓             ↓             ↓
                                            E
Summary based query rewriting

Query rewriting algorithm

XQuery Q → XQ1 → XQ2 → ... → XQm → E
S → e1,1 → e1,2 → e1,k1 → e2,1 → e2,2 → e2,k2 → em,1 → em,2 → em,km → E

Arion, Benzaken, Manolescu, Papakonstantinou
Structured Materialized Views for XML Queries
Summary based query rewriting

Query rewriting algorithm

XQuery Q → E

X_Q_1 → E

X_Q_2 → E

X_Q_m → E

S

X_1, X_2, ..., X_n

e_1,1 → e_2,1 → e_m,1

e_1,2 → e_2,2 → e_m,2
...

e_1,k_1 → e_2,k_2 → e_m,k_m
Rewriting one query XAM

- In the presence of a summary bucket algorithms would be incomplete
- Inflationary algorithm:
  - Builds (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
Full results

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

- Enhanced summaries: integrity constraints (required children)
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 recursivity
- unfold the // edges
Summary based query rewriting

Summary vs. DTD constraints

Rewriting without constraints

Rewriting with DTDs
- type alternatives
  - A=B|C
- sibling co-occurrence
  - A=BC

Rewriting with summaries
- closer to the instance
- bounds for recursivity
- unfold the // edges

Example: Is there a rewriting //a//=b/a + //c/a ?
Outline

1. Context: rewriting XQuery using nested views
2. Pattern containment under summary constraints
   - Path summary
   - Summary based containment
3. Summary based query rewriting
4. Experimental results
5. Conclusions
Experimental results

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

Time (ms)

Pattern size

- contained, conjunctive patterns
- not contained, conjunctive patterns
- contained, optional
- not contained, optional edges
183 views: 100 generated + 83 tag partition
Outline

1. Context: rewriting XQuery using nested views
2. Pattern containment under summary constraints
   - Path summary
   - Summary based containment
3. Summary based query rewriting
4. Experimental results
5. Conclusions
Conclusions

Contributions

- XML query pattern containment and equivalent rewriting based on summary constraints
- exploiting detailed informations about view contents and IDs expressed by XAMs
- query rewriting using XAMs is feasible

Future work

- Cost-based materialized view selection
- View maintenance in the presence of updates
More information

- ULoad, XAM home: http://gemo.futurs.inria.fr/projects/XAM/