Answering XPath Queries over Networks by Sending Minimal Views

Keishi Tajima   Yoshiki Fukui
Japan Advanced Institute of Science and Technology (JAIST)

31 Aug. 2004

Target

XML Information Services on networks

- **On-line XML databases**

a query

client $\xrightarrow{}$ $\xleftarrow{}$ server

an answer set

- **Subscription Systems**

register a set of queries

client $\xrightarrow{}$ $\xleftarrow{}$ server

answer sets (on Week 1)

answer sets (on Week 2)

$\vdots$

Problem

Answers to XML Queries Can be Redundant

When issuing a set of queries and getting answer sets ...

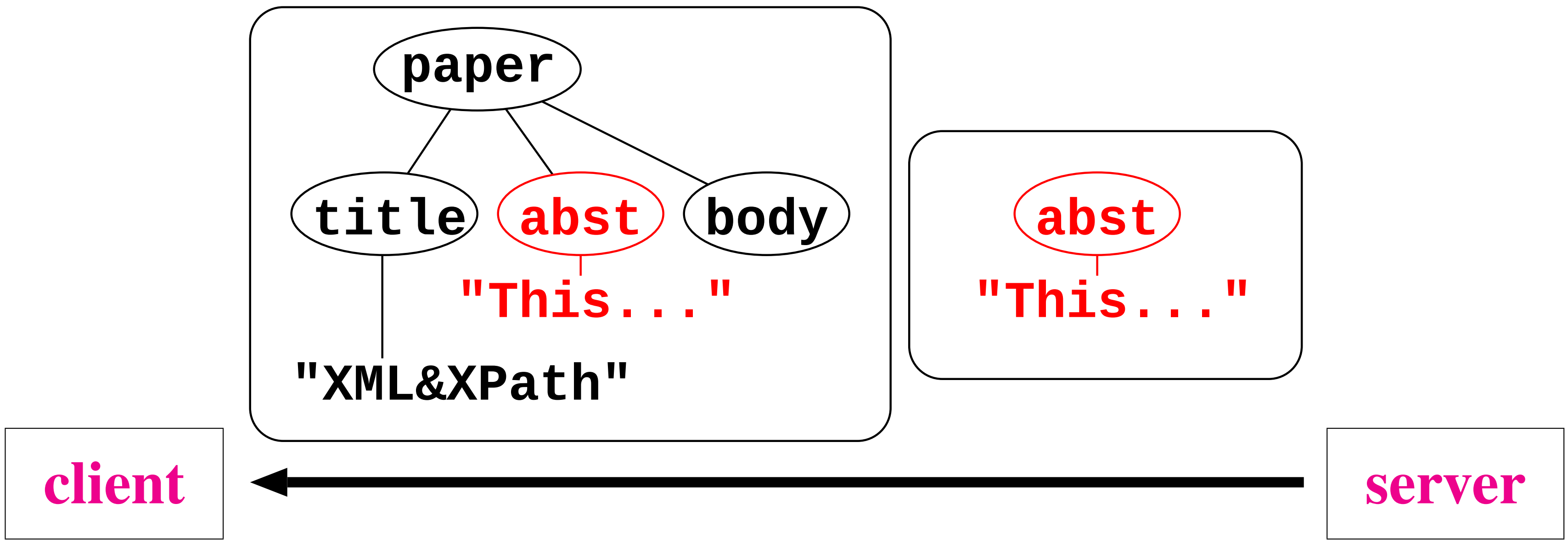
- **an element may appear in more than one answer set**
- **an element in one answer set may be a subelement of an answer in another answer set**
- **an element in one answer set may be a subelement of another answer in the same answer set**

Problem

Example 1

Q_1: **abstracts** of papers including **XML** in their titles

Q_2: **entire papers** including **XML and XPath** in their titles

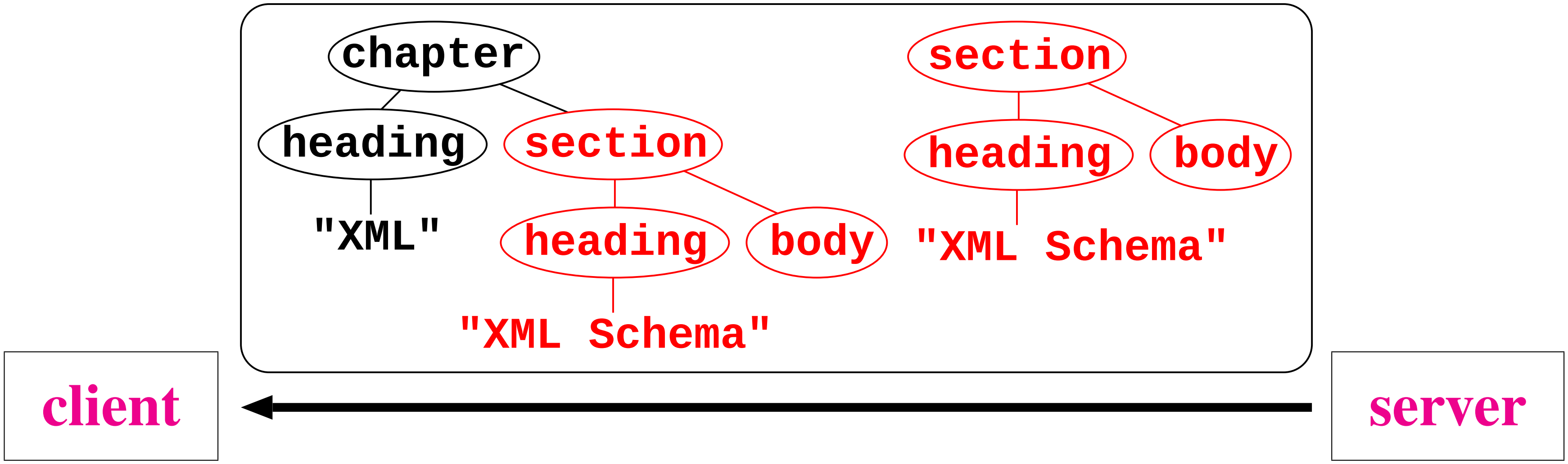


Communication cost is not optimal

Problem

Example 2

Q: **chapters, sections, . . . etc including XML in the headings**



Communication cost is not optimal even with a single query

Problem

Assumption

Here, we assume:

- **Databases are services provided by someone else.**
  - **All we can do is to submit queries and get answers.**
  - **No special encodings or protocols can be used.**
- **Servers provide XPath interface.**
  - **Full-fledged QLs are too expensive for large-scale services on the Internet.**
  - **Subtree extraction only. Queries cannot delete redundant parts or embed some markers in the answers.**

$\boxed{\text{Our Solution}}$

Example 1

Q_1: **abstracts of papers with XML in their titles**
Q_2: **entire papers with XML and XPath in their titles**

$$\Downarrow$$

V_1: **abstracts of papers with XML but not XPath in the titles**
V_2: **entire papers with XML and XPath in their titles**

$$\Downarrow$$

The answer to Q_1 is the union of:

- **the answer to V_1, and**
- **the abstracts extracted from the answer to V_2**

Our Solution

Example 2

Q: chapters, sections, ...etc with XML in their headings

$$\Downarrow$$

V: chapters, sections, ...etc with XML in their headings, but with no such ancestor

$$\Downarrow$$

- The answer to V includes all the top-most answers to Q.
- All the other nesting answers can be extracted from them.

Our Solution

1. Given $Q_1, \ldots, Q_n$, the client submits $V_1, \ldots, V_m$ s.t.
   - the answers to $Q_1, \ldots, Q_n$ can be extracted from the answers to $V_1, \ldots, V_m$, and
   - the total size of the answers to $V_1, \ldots, V_m$ is minimal.

   $V_1, \ldots, V_m$ is a minimal-size view set that can answer all the original queries.

2. The server sends the answers.
3. The client extracts the final answers from those answers.

The Goal of This Research

We develop an algorithm for computing a minimal-size view set that can answer all the given queries.

Here,

- **we consider (a fragment of) XPath,**
- **we do not consider minimization of the number of queries.**

Organization of the Rest of the Presentation

1. XPath fragment we use
2. more examples and intuitions behind the algorithm
3. the algorithm
4. related work
5. conclusion

Preliminary

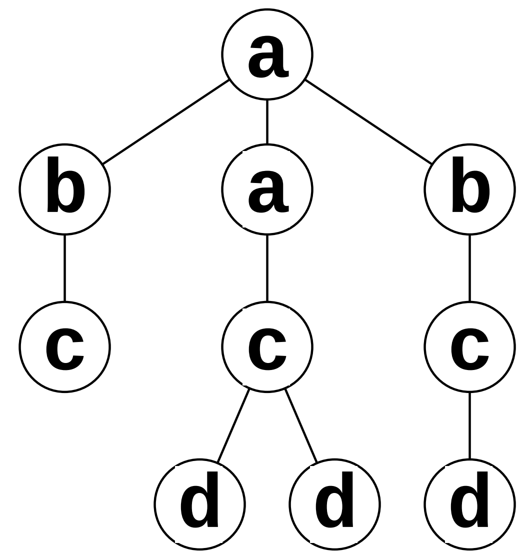
A Fragment of XPath

$$q ::= /p \mid //p \mid q \cup q \mid q - q$$

$$p ::= a \mid \overline{\{a_1, \ldots, a_n\}} \mid * \mid p/p \mid p//p \mid p[p] \mid p[\overline{p}]$$

e.g.

- **/a/b[c]**
- **//a/$\overline{\{\mathbf{b}\}}$**



Preliminary

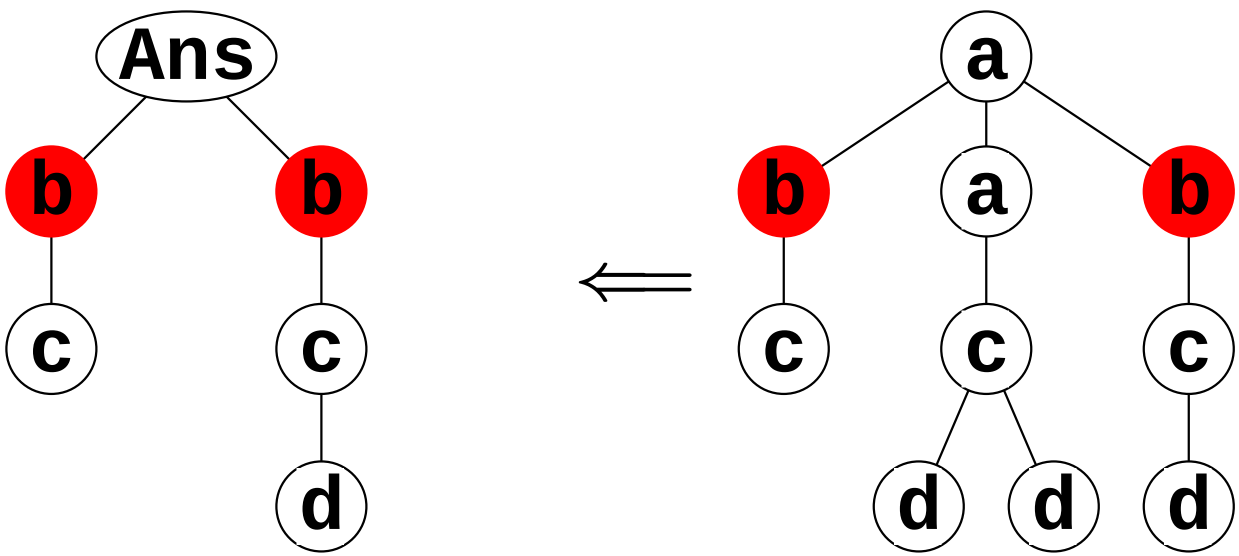
A Fragment of XPath

$$q ::= /p \mid //p \mid q \cup q \mid q - q$$

$$p ::= a \mid \overline{\{a_1, \ldots, a_n\}} \mid * \mid p/p \mid p//p \mid p[p] \mid p\overline{[p]}$$

e.g.

- /a/b[c]
- //a/$\overline{\{b\}}$



Preliminary

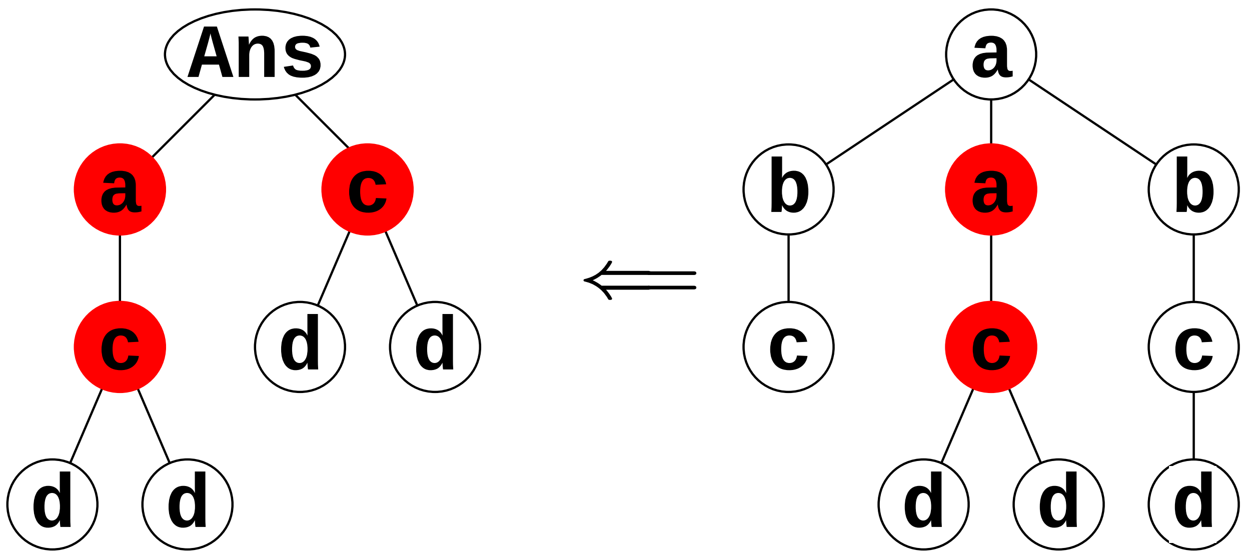
A Fragment of XPath

$$q \ ::= \ /p \mid //p \mid q \cup q \mid q - q$$

$$p \ ::= \ a \mid \overline{\{a_1, \ldots, a_n\}} \mid \ * \ \mid p/p \mid p//p \mid p[p] \mid p[\overline{p}]$$

e.g.

- **/a/b[c]**
- **//a/$\overline{\{\text{b}\}}$**



Example 1: Non-Recursive Queries of the Same Length

Given:

$$\begin{cases} Q_1 : \textsf{/a/}\overline{\{\textsf{b}\}}\textsf{/d} \\ Q_2 : \textsf{/a/}\{\textsf{c}\}\textsf{/d} \end{cases}$$

then, we submit:

$$\begin{cases} V_{1-2} : \textsf{/a/c/d} \\ V_{1\cap 2} : \textsf{/a/}\overline{\{\textsf{b,c}\}}\textsf{/d} \\ V_{2-1} : \textsf{/a/b/d} \end{cases}$$

and produce the final answers:

$$\begin{aligned} Q_1 &\leftarrow (V_{1-2},\ \textsf{/Ans/}*) \\ Q_1 &\leftarrow (V_{1\cap 2},\ \textsf{/Ans/}*) \\ Q_2 &\leftarrow (V_{1\cap 2},\ \textsf{/Ans/}*) \\ Q_2 &\leftarrow (V_{2-1},\ \textsf{/Ans/}*) \end{aligned}$$

Example 1: Non-Recursive Queries of the Same Length

Given:

$$\begin{cases} Q_1 : \mathbf{/a/}\overline{\{\mathbf{b}\}}\mathbf{/d} \\ Q_2 : \mathbf{/a/}\overline{\{\mathbf{c}\}}\mathbf{/d} \end{cases}$$

then, we submit:

$$\begin{cases} V_{1-2} : \mathbf{/a/c/d} & \leftarrow Q_1 - Q_2 \\ V_{1\cap 2} : \mathbf{/a/}\overline{\{\mathbf{b,c}\}}\mathbf{/d} & \leftarrow Q_1 \cap Q_2 \\ V_{2-1} : \mathbf{/a/b/d} & \leftarrow Q_2 - Q_1 \end{cases}$$

and produce the final answers:

$$Q_1 \leftarrow (V_{1-2},\ \mathbf{/Ans/}*)$$
$$Q_1 \leftarrow (V_{1\cap 2},\ \mathbf{/Ans/}*)$$
$$Q_2 \leftarrow (V_{1\cap 2},\ \mathbf{/Ans/}*)$$
$$Q_2 \leftarrow (V_{2-1},\ \mathbf{/Ans/}*)$$

Example 1: Non-Recursive Queries of the Same Length

Given $Q_1, \ldots, Q_n$ of the same length,

$$\{V(S) \mid S \neq \emptyset,\ S \subseteq \{1, \ldots, n\}\}$$

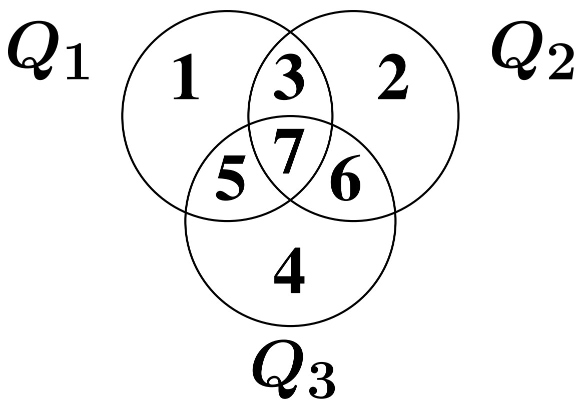
is a minimal view set, where

$$V(S) = \bigcap_{i \in S} Q_i - \bigcup_{i \in \{1,\ldots,n\} - S} Q_i$$

and we need

$$Q_i \leftarrow (V(S),\ \textsf{/Ans/}*) \quad \text{for } i \in S$$

$V(S)$ correspond to the regions in the Venn diagram.



Example 2: Non-Recursive Queries of Different Length

Given:

$$\begin{cases} Q_1 : \text{/a/}\overline{\{\text{b}\}}\text{/d} \\ Q_2 : \text{/a/}\overline{\{\text{c}\}}\text{/d/e} \end{cases}$$

then, we submit:

$$\begin{cases} V_{1-2} : \text{/a/c/d} \\ V_{1\cap 2} : \text{/a/}\overline{\{\text{b,c}\}}\text{/d} \\ V_{2-1} : \text{/a/b/d/e} \end{cases}$$

and produce the final answers:

$$\begin{aligned} Q_1 &\leftarrow (V_{1-2},\ \text{/Ans/}*) \\ Q_1 &\leftarrow (V_{1\cap 2},\ \text{/Ans/}*) \\ Q_2 &\leftarrow (V_{1\cap 2},\ \text{/Ans/}*\text{/e}) \\ Q_2 &\leftarrow (V_{2-1},\ \text{/Ans/}*) \end{aligned}$$

Example 2: Non-Recursive Queries of Different Length

Given:

$$\begin{cases} Q_1 : \text{/a/}\overline{\{\text{b}\}}\text{/d} \\ Q_2 : \text{/a/}\{\text{c}\}\text{/d/e} \end{cases} \qquad \begin{pmatrix} \text{pre}(Q_2) = \text{/a/}\overline{\{\text{c}\}}\text{/d} \\ \text{suf}(Q_2) = \text{/e} \end{pmatrix}$$

then, we submit:

$$\begin{cases} V_{1-2} : \text{/a/c/d} & \leftarrow Q_1 - \text{pre}(Q_2) \\ V_{1\cap 2} : \text{/a/}\overline{\{\text{b,c}\}}\text{/d} & \leftarrow Q_1 \cap \text{pre}(Q_2) \\ V_{2-1} : \text{/a/b/d/e} & \leftarrow (\text{pre}(Q_2) - Q_1).\text{suf}(Q_2) \end{cases}$$

and produce the final answers:

$$\begin{aligned} Q_1 &\leftarrow (V_{1-2}, \text{/Ans/}*) \\ Q_1 &\leftarrow (V_{1\cap 2}, \text{/Ans/}*) \\ Q_2 &\leftarrow (V_{1\cap 2}, \text{/Ans/}*\text{/e}) \quad \leftarrow \text{suffix}(Q_2) \\ Q_2 &\leftarrow (V_{2-1}, \text{/Ans/}*) \end{aligned}$$

Example 3: A Single Recursive Query

Given:

$$\{\ Q : //\mathsf{a}/\mathsf{b}/*/\mathsf{b}$$

then, we submit:

$$\begin{cases} V_1 : //\mathsf{a}/\mathsf{b}/\mathsf{a}/\mathsf{b} & -//\mathsf{a}/\mathsf{b}/*/\mathsf{b}//* \\ V_2 : //\mathsf{a}/\mathsf{b}/\overline{\{\mathsf{a}\}}/\mathsf{b} & -//\mathsf{a}/\mathsf{b}/*/\mathsf{b}//* \end{cases}$$

and produce the final answers:

$$Q \leftarrow (V_1, /\mathsf{Ans}/*)$$
$$Q \leftarrow (V_2, /\mathsf{Ans}/*)$$

$$Q \leftarrow (V_1, /\mathsf{Ans}//\mathsf{a}/\mathsf{b}/*/\mathsf{b})$$
$$Q \leftarrow (V_2, /\mathsf{Ans}//\mathsf{a}/\mathsf{b}/*/\mathsf{b})$$

$$Q \leftarrow (V_1, /\mathsf{Ans}/*/*/\mathsf{b})$$

Example 3: A Single Recursive Query

Given:

$\{ \; Q \; : \; //\mathsf{a}/\mathsf{b}/*/\mathsf{b}$

then, we submit:

$$\begin{cases} V_1 \; : \; //\mathsf{a}/\mathsf{b}/\mathsf{a}/\mathsf{b} & -//\mathsf{a}/\mathsf{b}/*/\mathsf{b}//* \\ V_2 \; : \; //\mathsf{a}/\mathsf{b}/\overline{\{\mathsf{a}\}}/\mathsf{b} & -//\mathsf{a}/\mathsf{b}/*/\mathsf{b}//* \end{cases} \longleftarrow$$

to retrieve only top-most answers

and produce the final answers:

$$Q \leftarrow (V_1, /\mathsf{Ans}/*)$$
$$Q \leftarrow (V_2, /\mathsf{Ans}/*)$$

$$Q \leftarrow (V_1, /\mathsf{Ans}//\mathsf{a}/\mathsf{b}/*/\mathsf{b})$$
$$Q \leftarrow (V_2, /\mathsf{Ans}//\mathsf{a}/\mathsf{b}/*/\mathsf{b})$$

$$Q \leftarrow (V_1, /\mathsf{Ans}/*/*/\mathsf{b})$$

Example 3: A Single Recursive Query

Given:

$$\{\ Q : //a/b/*/b \qquad (\mathrm{pre}(Q) = //a/b,\ \ \mathrm{suf}(Q) = /*/b)$$

then, we submit:

$$\begin{cases} V_1 : //a/b/a/b & -//a/b/*/b//* \qquad \leftarrow Q \cap \mathrm{pre}(Q) \\ V_2 : //a/b/\overline{\{a\}}/b & -//a/b/*/b//* \qquad \leftarrow Q - \mathrm{pre}(Q) \end{cases}$$

and produce the final answers:

$$Q \leftarrow (V_1, /Ans/*)$$
$$Q \leftarrow (V_2, /Ans/*)$$

$$Q \leftarrow (V_1, /Ans//a/b/*/b)$$
$$Q \leftarrow (V_2, /Ans//a/b/*/b)$$

$$Q \leftarrow (V_1, /Ans/*/*/b) \qquad \leftarrow \mathrm{suf}(Q)$$

Algorithm

Input:

$$\begin{cases} Q_1 : \ /_1^1 \ p_1^1 \ /_1^2 \ p_1^2 \ \dots \ /_1^{l_1} \ p_1^{l_1} \\ \vdots \\ Q_n : \ /_n^1 \ p_n^1 \ /_n^2 \ p_n^2 \ \dots \ /_n^{l_n} \ p_n^{l_n} \end{cases}$$

(each $/_i^j$ is either / or //)

Output:

- a set of view queries $\{V_1, \dots, V_m\}$, and
- a set of triplets of the form $Q_i \leftarrow (V_j, q_i^j)$

$\boxed{\text{Algorithm}}$

Auxiliary Function pp_i^j

$$pp_i^j \equiv \begin{cases} \emptyset \ \text{(empty pattern)} & \text{if } j = 0,\ /_i^1 = / \\ //* & \text{if } j = 0,\ /_i^1 = // \\ /_i^1 p_i^1 \ldots /_i^j p_i^j & \text{if } j > 0,\ /_i^{j+1} = / \\ /_i^1 p_i^1 \ldots /_i^j p_i^j \cup /_i^1 p_i^1 \ldots /_i^j p_i^j //* & \text{if } j > 0,\ /_i^{j+1} = // \end{cases}$$

E.g.:

$$\begin{cases} Q_1 : //\mathbf{a} \\ Q_2 : /\mathbf{b}//\overline{\{\mathbf{c}\}} \end{cases} \longrightarrow \begin{pmatrix} pp_1^0 = //* \\ pp_2^0 = \emptyset \\ pp_2^1 = /\mathbf{b} \cup /\mathbf{b}//* \end{pmatrix}$$

$\boxed{\text{Algorithm}}$

Main Routine

1. For every S, T, s.t.:

- $S \subseteq \{1, \ldots, n\}, \;\; S \neq \emptyset$
- $T \subseteq \{(i, j) \mid 1 \leq i \leq n, \;\; 0 \leq j \leq l_i - 1\}$

create a view query $V(S, T)$:

$$\Big(\bigcap_{i \in S} Q_i - \bigcup_{i \notin S} Q_i\Big) \cap \Big(\bigcap_{(i,j) \in T} pp_i^j - \bigcup_{(i,j) \notin T} pp_i^j\Big) - \bigcup_{1 \leq i \leq n} Q_i//*$$

2. For each $V(S, T)$, create triplets:

$Q_i \leftarrow (V(S, T), /\textbf{Ans}/*)$ **for** $i \in S$

$Q_i \leftarrow (V(S, T), /\textbf{Ans}/*/_i^{j+1} p_i^{j+1} \ldots /_i^{l_i} p_i^{l_i})$ **for** $(i, j) \in T$

$\boxed{\text{Algorithm}}$

Main Routine

1. For every S, T, s.t.:

- $S \subseteq \{1, \ldots, n\}, \;\; S \neq \emptyset$
- $T \subseteq \{(i, j) \mid 1 \leq i \leq n, \;\; 0 \leq j \leq l_i - 1\}$

create a view query $V(S, T)$:

$$\left(\bigcap_{i \in S} Q_i - \bigcup_{i \notin S} Q_i \right) \cap \left(\bigcap_{(i,j) \in T} pp_i^j - \bigcup_{(i,j) \notin T} pp_i^j \right) - \bigcup_{1 \leq i \leq n} Q_i//*$$

$$\Downarrow$$

classifying elements based on which Q_i it matches

$\boxed{\text{Algorithm}}$

Main Routine

1. For every S, T, s.t.:

- $S \subseteq \{1, \ldots, n\}, \;\; S \neq \emptyset$
- $T \subseteq \{(i, j) \mid 1 \leq i \leq n, \;\; 0 \leq j \leq l_i - 1\}$

create a view query $V(S, T)$:

$$\Big(\bigcap_{i \in S} Q_i - \bigcup_{i \notin S} Q_i\Big) \cap \Big(\bigcap_{(i,j) \in T} pp_i^j - \bigcup_{(i,j) \notin T} pp_i^j\Big) - \bigcup_{1 \leq i \leq n} Q_i//*$$

$$\Downarrow$$

classifying elements based on which prefixes it matches

$\boxed{\textbf{Algorithm}}$

Main Routine

1. For every S, T, s.t.:

- $S \subseteq \{1, \ldots, n\}, \;\; S \neq \emptyset$
- $T \subseteq \{(i, j) \mid 1 \leq i \leq n, \;\; 0 \leq j \leq l_i - 1\}$

create a view query $V(S, T)$:

$$\left(\bigcap_{i \in S} Q_i - \bigcup_{i \notin S} Q_i\right) \cap \left(\bigcap_{(i,j) \in T} pp_i^j - \bigcup_{(i,j) \notin T} pp_i^j\right) - \bigcup_{1 \leq i \leq n} Q_i//*$$

$$\Downarrow$$

to only retrieve the top-most answers

Algorithm

Main Routine

1. For every S, T, s.t.:

- $S \subseteq \{1, \ldots, n\}, \;\; S \neq \emptyset$
- $T \subseteq \{(i, j) \mid 1 \leq i \leq n, \;\; 0 \leq j \leq l_i - 1\}$

create a view query $V(S, T)$:

$$\Big(\bigcap_{i \,\in\, S} Q_i - \bigcup_{i \,\notin\, S} Q_i\Big) \cap \Big(\bigcap_{(i,j) \,\in\, T} pp_i^j - \bigcup_{(i,j) \,\notin\, T} pp_i^j\Big) - \bigcup_{1 \leq i \leq n} Q_i//*$$

2. For each $V(S, T)$, create triplets:

$Q_i \leftarrow (V(S, T), /\mathbf{Ans}/*)$ **for** $i \in S$

$Q_i \leftarrow (V(S, T), /\mathbf{Ans}/* /_i^{j+1} p_i^{j+1} \ldots /_i^{l_i} p_i^{l_i})$ **for** $(i, j) \in T$

Algorithm

An Example

For example, given:

$$\begin{cases} Q_1 : //\mathbf{a} \\ Q_2 : /\mathbf{b}//\overline{\{\mathbf{c}\}} \end{cases}$$

Then,

$$\begin{cases} pp_1^0 = //* \\ pp_2^0 = \emptyset \\ pp_2^1 = /\mathbf{b} \cup /\mathbf{b}//* \end{cases}$$

- **Views for T including pp_2^0 or not including pp_1^0 are empty.**
- **$\cap pp_1^0$ and $-pp_2^0$ can be omitted.**

Algorithm

An Example

Views:

$$V_1 : (Q_1 \cap Q_2) \cap pp^1_2 - (Q_1//* \cup Q_2//*)$$
$$V_2 : (Q_1 \cap Q_2) - pp^1_2 - (Q_1//* \cup Q_2//*)$$
$$V_3 : (Q_1 - Q_2) \cap pp^1_2 - (Q_1//* \cup Q_2//*)$$
$$V_4 : (Q_1 - Q_2) - pp^1_2 - (Q_1//* \cup Q_2//*)$$
$$V_5 : (Q_2 - Q_1) \cap pp^1_2 - (Q_1//* \cup Q_2//*)$$
$$V_6 : (Q_2 - Q_1) - pp^1_2 - (Q_1//* \cup Q_2//*)$$

Triplets:

$$Q_1 \leftarrow (V_i, //\mathsf{Ans}/*) \quad \text{where } i \in \{1,2,3,4\}$$
$$Q_2 \leftarrow (V_i, //\mathsf{Ans}/*) \quad \text{where } i \in \{1,2,5,6\}$$
$$Q_1 \leftarrow (V_i, //\mathsf{Ans}/*//\mathsf{a}) \quad \text{where } i \in \{1,2,3,4,5,6\}$$
$$Q_2 \leftarrow (V_i, //\mathsf{Ans}/*//\overline{\{\mathsf{c}\}}) \quad \text{where } i \in \{1,3,5\}$$

Algorithm

The view set computed by our algorithm is minimal

because what it does is:

to retrieve only top-most answers and classify them

and therefore,

- **it retrieves only necessary elements, and**
- **no element appears more than once.**

Discussion

Efficiency of the Algorithm

1. **The number of view queries:**

   In our algorithm, it grows exponential with:

   - **the number of given queries (even for non-recursive queries), and**
   - **the total length of given queries (for recursive queries)**

   but it is inevitable to minimize the view size.

2. **Evaluation cost on the server:**

   In our experiments,

   - **we could even reduce the server cost in many cases.**

   It is because the view queries are more complicated, but have smaller answers than the original queries.

Related Work

- Minimal views for relational queries [Chirkova, Li 03]

redundancy caused by join operations

$\Updownarrow$

our work: redundancy caused by nested structure of XML

- Reminder query [Dar et al 96]
  1. submit Q_1, and cache the result
  2. given Q_2, retrieve only $Q_2 - Q_1$

Not always possible to extract $Q_1 \cap Q_2$ from the cached Q_1

$\Updownarrow$

our work: given Q_1 and Q_2,
retrieve $Q_1 - Q_2$, $Q_1 \cap Q_2$, $Q_2 - Q_1$

Conclusion

We showed an algorithm to compute a minimal view for a given set of XPath queries.

Our algorithm works as long as:

1. **the fragment is closed under $\cup$ and $-$,**
   - **we use those operations to compute view queries**
2. **it only supports child and descendant axis.**
   - **some axes (e.g. following) make it harder to extract nesting answers from top-most answers.**

Future Work

- **Efficient evaluation of view queries on the server.**
  Queries produced by our algorithm have some pattern.
  - $Q_1 - Q_2, Q_1 \cap Q_2, Q_2 - Q_1$
  - $-Q_i//*$
- **Interaction with the compression approach.**
  1. **compress answers on the server before sending**
  2. **send the compressed answers**
  3. **decompress on the client**

  Compression removes redundancy and may offset the benefit of our approach.