FIX: Feature-based Indexing Technique for XML Documents

Ning Zhang

University of Waterloo
http://www.cs.uwaterloo.ca/~nzhang

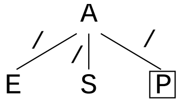
Joint work with M. Tamer Özsu, Ihab F. Ilyas, and Ashraf Aboulnaga

Motivating Example

- Twig Query (root axis could be //, others are /):
  Q_1: Find phone numbers (P) of all authors (A) who also have email (E) and school (S).

  `//A[./E][./S]/P`

- Find all subtrees satisfying a pattern tree:

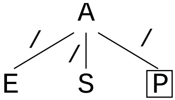


Motivating Example

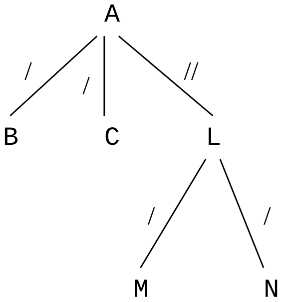
- Twig Query (root axis could be //, others are /):
  Q_1: Find phone numbers (P) of all authors (A) who also have email (E) and school (S).

  `//A[./E][./S]/P`

- Find all subtrees satisfying a pattern tree:



- A general path containing // in the middle can be decomposed into interconnected twig queries.

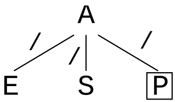


Motivating Example

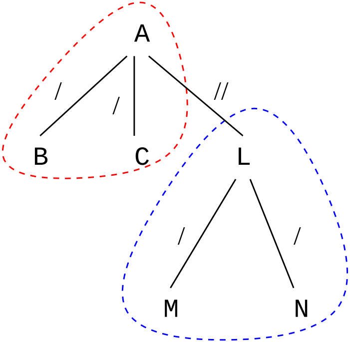
- Twig Query (root axis could be //, others are /):
  Q_1: Find phone numbers (P) of all authors (A) who also have email (E) and school (S).

  `//A[./E][./S]/P`

- Find all subtrees satisfying a pattern tree:



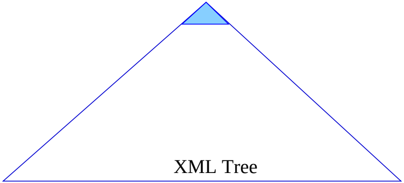
- A general path containing // in the middle can be decomposed into interconnected twig queries.



Approaches to Evaluating Twig Queries

Navigational Approach

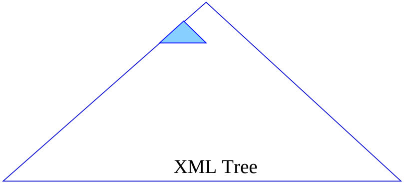
Traverse the XML tree and perform Tree Pattern Matching (TPM) operation on every tree node



Approaches to Evaluating Twig Queries

Navigational Approach

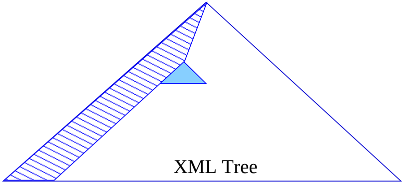
Traverse the XML tree and perform Tree Pattern Matching (TPM) operation on every tree node



Approaches to Evaluating Twig Queries

Navigational Approach

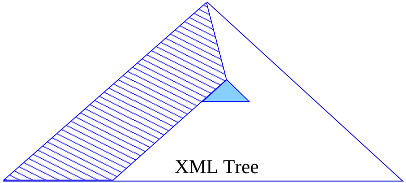
Traverse the XML tree and perform Tree Pattern Matching (TPM) operation on every tree node



Approaches to Evaluating Twig Queries

Navigational Approach

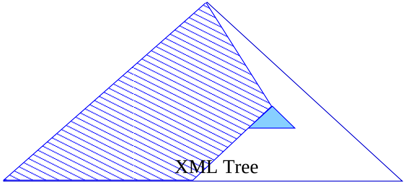
Traverse the XML tree and perform Tree Pattern Matching (TPM) operation on every tree node



Approaches to Evaluating Twig Queries

Navigational Approach

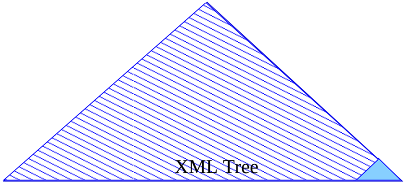
Traverse the XML tree and perform Tree Pattern Matching (TPM) operation on every tree node



Approaches to Evaluating Twig Queries

Navigational Approach

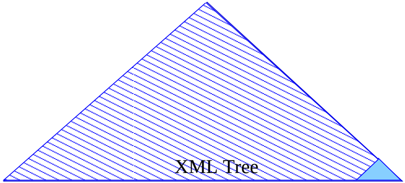
Traverse the XML tree and perform Tree Pattern Matching (TPM) operation on every tree node



Approaches to Evaluating Twig Queries

Navigational Approach

Traverse the XML tree and perform Tree Pattern Matching (TPM) operation on every tree node

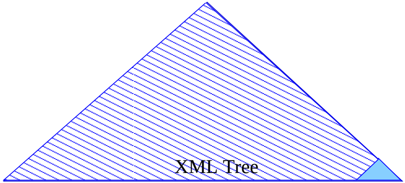


- Analogous to sequential scan, very expensive:
    - 4,000,000+ TPM operations on DBLP.

Approaches to Evaluating Twig Queries

Navigational Approach

Traverse the XML tree and perform Tree Pattern Matching (TPM) operation on every tree node

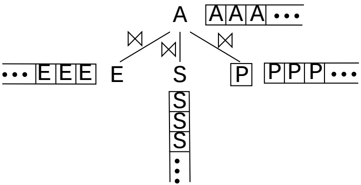


- Analogous to sequential scan, very expensive:
  - 4,000,000+ TPM operations on DBLP.
- Many unnecessary operations for highly selective queries.

Approaches to Evaluating Twig Queries

Join-based Approach

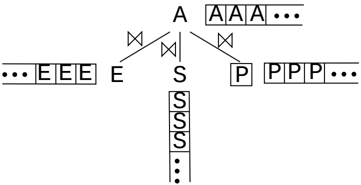
XML elements are clustered by tag-names; elements matched with the tag-names are structurally joined



Approaches to Evaluating Twig Queries

Join-based Approach

XML elements are clustered by tag-names; elements matched with the tag-names are structurally joined

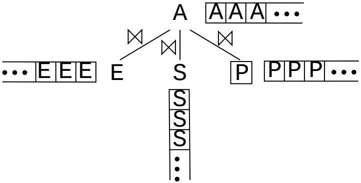


- Analogous to index-based join.

Approaches to Evaluating Twig Queries

Join-based Approach

XML elements are clustered by tag-names; elements matched with the tag-names are structurally joined



- Analogous to index-based join.
- Tag-name indexes are not discriminative enough:
  - Elements are selected to join solely based on their tag names, without considering their descendants.

Objectives

- Build an index that does not only consider root tags, but also the whole subtree.

Objectives

- Build an index that does not only consider root tags, but also the whole subtree.

```
abz)e)cf)g))cf)g))cf)g))c)c))cz)e)df)g))i)c)))
```

TPM Starting Points using sequential scan

Objectives

- Build an index that does not only consider root tags, but also the whole subtree.

```
abz)e)cf)g))cf)g))cf)g))c)c))cz)e)df)g))i)c)))
```

Starting Points Using Tag−name Index

Objectives

- Build an index that does not only consider root tags, but also the whole subtree.

```
abz)e)cf)g))cf)g))cf)g))c)c))cz)e)df)g))i)c)))
```

Starting points using index considering the whole subtree

Objectives

- Build an index that does not only consider root tags, but also the whole subtree.

```
abz)e)cf)g))cf)g))cf)g))c)c))cz)e)df)g))i)c)))
```

Starting points using index considering the whole subtree

- Exploiting existing indexes (e.g., B^+ tree) to build the new index.

Objectives

- Build an index that does not only consider root tags, but also the whole subtree.

```
abz)e)cf)g))cf)g))cf)g))c)c))cz)e)df)g))i)c)))
```

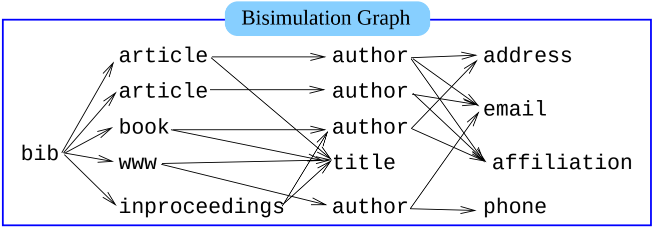
Starting points using index considering the whole subtree

- Exploiting existing indexes (e.g., B^+ tree) to build the new index.
- Incorporating both structures and values in the index.

Related Work — Structural Indexes

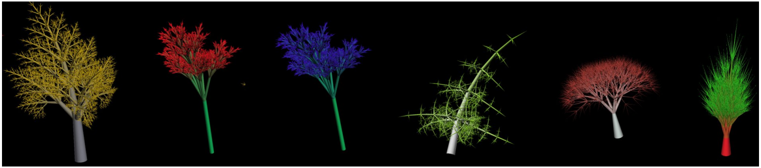
Cluster XML tree nodes having similar structures in terms of:

- Tag-name: tag-name indexes: tag-names as keys for B^+ tree (the pruning power comes from the root of query tree).
- Rooted path: DataGuide, 1-index, A(k)-index (simple path expressions only)
- Subtree: bisimulation graph
- Rooted path & subtree: F&B bisimulation graph



The bisimulation and F&B bisimulation graphs could be very large (3×10^5 vertices and 2×10^6 edges for Treebank).

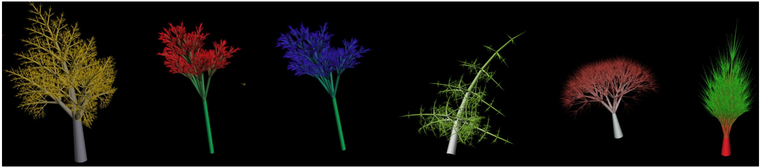
Our Approach: Feature-based Index (FIX)



Key Idea:

1. Data and query trees are all converted to bisimulation graphs.
   1.1 Bisimulation graph is much smaller than the XML tree
   1.2 Bisimulation graph preserves all structural information
2. Enumerate all subgraphs of depth k (indexable units) in the data bisimulation graph.
3. Insert indexable units based on their distinctive features.
4. Calculate the features of query bisimulation graph and use them to filter out indexed units by comparing their features.

Our Approach: Feature-based Index (FIX)



Key Idea:

1. Data and query trees are all converted to bisimulation graphs.
   1.1 Bisimulation graph is much smaller than the XML tree
   1.2 Bisimulation graph preserves all structural information
2. Enumerate all subgraphs of depth k (indexable units) in the data bisimulation graph.
3. Insert indexable units based on their distinctive features.
4. Calculate the features of query bisimulation graph and use them to filter out indexed units by comparing their features.

What are the *features* for labeled trees?

Features of XML Tree

Three features of indexable units (after converted to a special matrix):

1. minimum eigenvalue λ_{min}
2. maximum eigenvalue λ_{max}
3. root label r

Features of XML Tree

Three features of indexable units (after converted to a special matrix):

1. minimum eigenvalue $\lambda_{\min}$
2. maximum eigenvalue $\lambda_{\max}$
3. root label r

Theorem

Given two graphs G and H, if H is an induced subgraph of G, then $\lambda_{\min}(G) \leq \lambda_{\min}(H) \leq \lambda_{\max}(H) \leq \lambda_{\max}(G)$.

Features of XML Tree

Three features of indexable units (after converted to a special matrix):

1. minimum eigenvalue λ_{min}
2. maximum eigenvalue λ_{max}
3. root label r

Theorem

Given two graphs G and H, if H is an induced subgraph of G, then $\lambda_{\text{min}}(G) \leq \lambda_{\text{min}}(H) \leq \lambda_{\text{max}}(H) \leq \lambda_{\text{max}}(G)$.

Necessary conditions for a query Q having positive answers in data tree D:

$$\begin{aligned} & \lambda_{\text{min}}(D) \leq \lambda_{\text{min}}(Q) \leq \lambda_{\text{max}}(Q) \leq \lambda_{\text{max}}(D) \\ \wedge \quad & r(Q) = r(D) \end{aligned}$$

Features of XML Tree

Three features of indexable units (after converted to a special matrix):

1. minimum eigenvalue λ_{min}
2. maximum eigenvalue λ_{max}
3. root label r

Theorem

Given two graphs G and H, if H is an induced subgraph of G, then $\lambda_{\mathrm{min}}(G) \leq \lambda_{\mathrm{min}}(H) \leq \lambda_{\mathrm{max}}(H) \leq \lambda_{\mathrm{max}}(G)$.

Necessary conditions for a query Q having positive answers in data tree D:

$$\begin{array}{rl} & \lambda_{\mathrm{min}}(D) \leq \lambda_{\mathrm{min}}(Q) \leq \lambda_{\mathrm{max}}(Q) \leq \lambda_{\mathrm{max}}(D) \\ \wedge & r(Q) = r(D) \end{array}$$

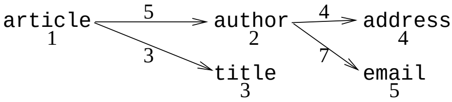
Returned results may have false-positives: need refinement.

Calculating Features

1. Compute bisimulation graph for an XML tree
2. Convert labeled directed graph into weighted directed graph (encode labeled edge into edge weight). e.g.,

$$(article, title) \rightarrow 3 \qquad (article, author) \rightarrow 5$$
$$(author, address) \rightarrow 4 \qquad (author, email) \rightarrow 7$$

3. Convert weighted directed graph into anti-symmetric matrix

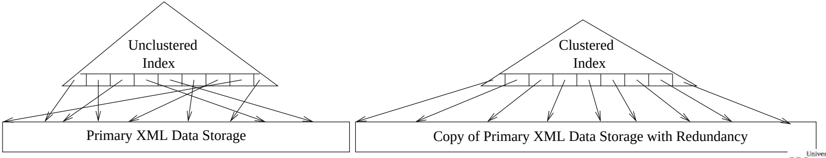


$$\mathbf{M} = \begin{bmatrix} 0 & 5 & 3 & 0 & 0 \\ -5 & 0 & 0 & 4 & 7 \\ -3 & 0 & 0 & 0 & 0 \\ 0 & -4 & 0 & 0 & 0 \\ 0 & -7 & 0 & 0 & 0 \end{bmatrix}$$

4. Calculate eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ of the matrix. The λ_{min}, λ_{max} and the root note label r are three features.

Building Index

- Bisimulation graph could be too large — restrict the depth of XML trees to a small k.
- For each document in a collection:
  - if the document's maximum depth is less than k, convert the whole tree into bisimulation graph.
  - otherwise, enumerate all subtrees having depth under the limit k, convert them into bisimulation graph.
- Both clustered and unclustered indexes can be built for a collection



Query Processing

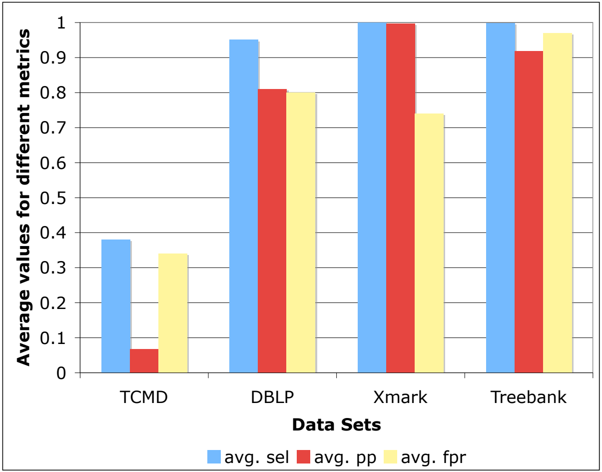
- Convert the query tree into bisimulation graph
- Convert bisimulation graph into anti-symmetric matrix
- Calculate $\lambda_{\text{min}}(Q)$ and $\lambda_{\text{max}}(Q)$
- Reduced to range query:
  - find all $[\lambda_{\text{min}}, \lambda_{\text{max}}]$ in the index that contain $[\lambda_{\text{min}}(Q), \lambda_{\text{max}}(Q)]$ and $r = Q.root$.
- Refinement: evaluating Tree Pattern Matching on returned candidate results.

Incorporating Values

- Treat values as special tag names:
  - Values are hashed into a small domain D_v outside of tag name encodings D_t, i.e., $D_v \cap D_t = \emptyset$.
  - (tag, value) edges are also mapped to a distinct integer.
- Can answer equality constrained queried, e.g.,

`//book[title="TCP/IP Illustrated"]/price`

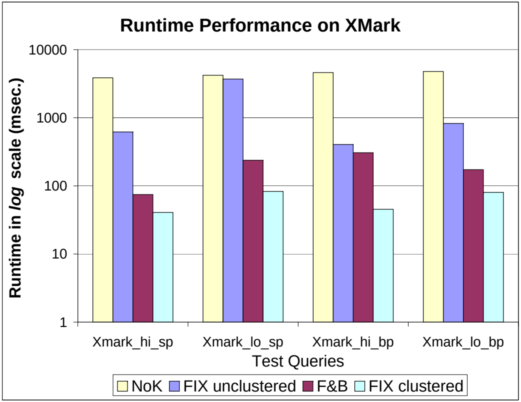
Performance Evaluation: Implementation-independent Metrics



Implementation-independent metrics:

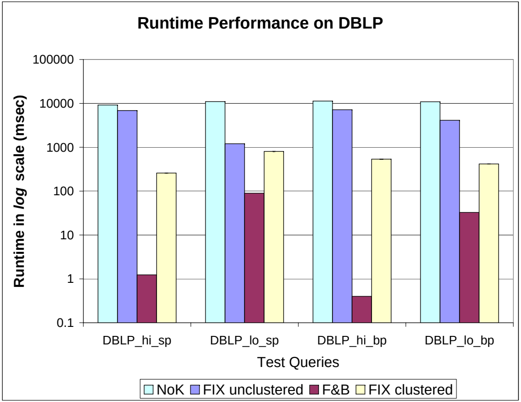
$$\begin{aligned} sel &= 1 - rst/ent \\ pp &= 1 - cdt/ent \\ fpr &= 1 - rst/cdt \end{aligned}$$

Performance Evaluation: Runtime



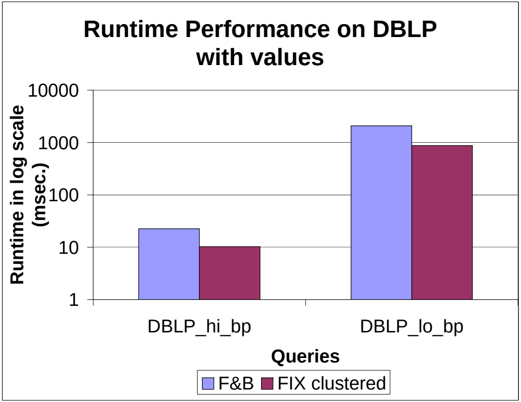
FIX improves performance significantly on structure-rich data sets.

Performance Evaluation: Runtime (cont.)



FIX does not perform as well on simple structured data sets.

Performance Evaluation: Runtime (cont.)



But when considering values, FIX performs better.

Conclusion and Future Work

Summary:

- We identify three features for pruning subtrees during query processing.
  - Easy to calculate.
  - Pruning uses simple numeric comparisons.
- A unified structure and value index (FIX) can be built based on these features to improve query performance significantly.
- Query evaluating based on FIX is simple and based on well-studied techniques.

Future Work:

- Try R-tree or other high-dimensional index instead of B^+ tree.
- Support wider range of queries.
- Find more features!

Thank you!