Workload-aware Trie Indices for XML

Yuqing Wu Sofía Brenes Hyungdae Yi

Indiana University, Bloomington, USA
{yuqwu, sbrenesb, yih}@cs.indiana.edu

ABSTRACT

Well-designed indices can dramatically improve query performance. Including query workload information can produce indices that yield better overall throughput while balancing the space and performance trade-off at the core of index design. In the context of XML, structural indices have proven to be particularly effective in supporting XPath queries by capturing the structural correlation between data components in an XML document. In this paper, we propose a family of novel workload-aware indices by taking advantage of the disk-based $\mathcal{P}[k]$-Trie index framework, which indexes node pairs of an XML document to facilitate index-only evaluation plans. Our indices are designed to be optimal for answering frequent path queries in one index lookup and efficient for answering non-frequent path queries using an index-only plan. Experimental results prove that our indices outperform the APEX index in overall throughput and excel in answering non-frequent queries, queries with predicates, and queries that yield empty results.

Categories and Subject Descriptors

H.2.2 [**Information Systems**]: Database Management—*Physical Design (Access Method)*

General Terms

Index

Keywords

XML, structural index, workload, query processing

1. INTRODUCTION

The explosive growth of data and search on the Internet and the growing demands for managing increasingly large amounts of data have helped XML emerge as the data format for representing, storing, and querying semi-structured data. As this trend is likely to continue, efficient query evaluation techniques for XML must be developed. The structural correlation between XML data components, as seen in Figure 1, has required the development of structural indices for XML. These structural indices have proved to be of significant importance in improving the performance of XPath queries, which are at the core of all XML queries.

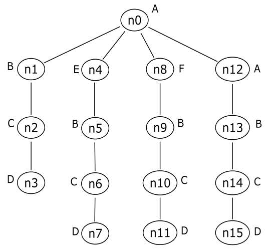


Figure 1: A Sample XML Document

Good index design requires a careful balance between index size and the precision it may provide in query evaluation. Workload information can be leveraged to mitigate the trade-off between index size and precision, producing an index that yields better overall throughput, while maintaining its footprint under control. Numerous structural indices (including workload-aware approaches) have been proposed [5], however, there is room for improvement in the areas of space and query efficiency.

Our research goal is to design workload-aware indices for XML that are (1) optimal for answering frequent path queries and efficient for answering non-frequent path queries; (2) efficient in identifying queries that yield empty results; (3) efficient in updates induced by the changes in either the workload or the data itself; (4) efficient in size and easily adjustable to space allowance requirements.

2. PRELIMINARIES

We define an XML document as $\mathcal{X} = (V, Ed, r, \lambda)$, with V the finite set of nodes, $Ed \subseteq V \times V$ the set of parent-child edges, $r \in V$ the root, and $\lambda\colon V \to \mathcal{L}$ a node-labeling function into the set of labels $\mathcal{L}$.

Given an XML document $\mathcal{X}$ and a number $k \in \mathbb{N}$, we define $DownPairs(\mathcal{X}, k)$ as the set of node pairs (m, n) such that (1) $length(m, n) \leq k$, and (2) m is an ancestor of n. A *label-path* $LP(m, n)$ is defined as the unique sequence of labels between two nodes m and n in V, with the *k-label-path* $LP(n, k)$ of a node $n \in V$ and $k \in \mathbb{N}$ defined as the label-path of the unique downward path of length l into n where $l = min(height(n), k)$. We define $DownPaths(\mathcal{X}, k)$ as the set of *k-label-paths* of the node pairs in $DownPairs(\mathcal{X}, k)$. Two node pairs (m_1, n_1) and (m_2, n_2) are $\mathcal{P}[k]$-equivalent if (1) (m_1, n_1) and $(m_2, n_2) \in DownPairs(\mathcal{X}, k)$ and (2) $LP(m_1, n_1) = LP(m_2, n_2)$.

The $\mathcal{P}[k]$*-partition* of an XML document $\mathcal{X}$ is the partition of the node pairs in $\mathcal{X}$ induced by the $\mathcal{P}[k]$-equivalence relation. Each partition class C in the $\mathcal{P}[k]$*-partition* can be associated with the unique k-label-path lp of the node

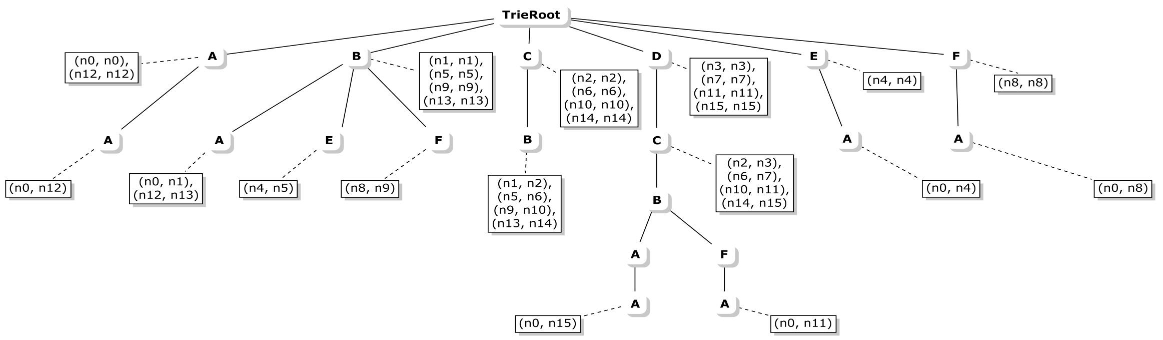

TrieRoot
A
B
C
D
E
F
(n0, n0), (n12, n12)
(n1, n1), (n5, n5), (n9, n9), (n13, n13)
(n2, n2), (n6, n6), (n10, n10), (n14, n14)
(n3, n3), (n7, n7), (n11, n11), (n15, n15)
(n4, n4)
(n8, n8)
A
A
E
F
B
C
A
A
(n0, n12)
(n0, n1), (n12, n13)
(n4, n5)
(n8, n9)
(n1, n2), (n5, n6), (n9, n10), (n13, n14)
(n2, n3), (n6, n7), (n10, n11), (n14, n15)
(n0, n4)
(n0, n8)
B
A
F
A
A
(n0, n15)
(n0, n11)

Figure 2: $\mathcal{WP}[1]$-Trie of $\mathcal{X}$ with $F = \{(A, A, B, C, D), (A, F, B, C, D)\}$.

pairs in C. Additionally, a label-path lp uniquely identifies a $\mathcal{P}[k]$-partition class C, denoted as $\mathcal{P}[k][lp]$.

XPath queries are the core of almost all XML query languages. The path semantics of the core XPath expressions that are frequently studied [1] are defined as:

$$E := \epsilon \mid \phi \mid \widehat{l} \mid \uparrow \mid \downarrow \mid E_1 \circ E_2 \mid E_1[E_2] \mid E_1 * E_2$$ [1]

The node semantics of an XPath algebra expression E on $\mathcal{X}$, denoted $E^{nodes}(\mathcal{X})$, is the set $\{n \mid \exists m : (m, n) \in E(\mathcal{X})\}$.

The downwards algebra $\mathcal{D}$, as studied in [2, 4], contains only label matching and downward navigation and is the simplest form of a path query. We further define the $\mathcal{D}[k]$ expressions to be the $\mathcal{D}$ expressions with no more than k $\downarrow$'s. $\mathcal{D}[k]$ queries can be answered easily under path semantics by applying the union operation to the $\mathcal{P}[k]$-partition classes.

Theorem 2.1. *Let $\mathcal{X}$ be an XML document and E an expression in $\mathcal{D}[k]$, Let $LPS(E, \mathcal{X})$ be the set of label-paths in $\mathcal{X}$ that satisfy the node-labels and structural containment relationships specified by E. Then,*

$$E(\mathcal{X}) \quad = \bigcup_{lp \in LPS(E,\mathcal{X})} \mathcal{P}[k][lp]$$

3. WORKLOAD-AWARE TRIE INDICES

Given a set of XML queries Q, we define the workload to be a set of $\mathcal{D}$ queries whose appearance is above a certain threshold. Since $\mathcal{D}$ queries can be represented by *label-paths*, we call it *frequent label-path set*, abbreviated as F. Given a document $\mathcal{X}$ and a workload F, we are interested in designing workload-aware indices for $\mathcal{X}$ that support efficient evaluation of core XPath queries with a very small space and maintenance overhead.

To achieve this, we take advantage of the $\mathcal{P}[k]$-Trie index framework [2], in which all $DownPairs(\mathcal{X}, k)$ of an XML document $\mathcal{X}$ are indexed. The $\mathcal{P}[k]$-partition classes of $\mathcal{X}$ are organized in a trie structure with the inverted label-path (lp^{-1}) as an index key. Because the $\mathcal{P}[k]$-Trie index indexes node pairs, any core XPath query can be decomposed into $\mathcal{D}[k]$ sub-queries, which can be answered by an index scan over the $\mathcal{P}[k]$-Trie index, and subsequent natural joins that compute the result of the query. Additionally, we take advantage of two important properties of the $\mathcal{P}[k]$-Trie index: (1) the independence of the trie branches in terms of the degree of local bi-similarity; (2) the suitability of the $\mathcal{P}[k]$-Trie index for answering all core XPath queries and its good performance with a modest $k \geq 1$ value. Thus, we propose a family of workload-aware Trie indices that index a frequent label-path set and a selected set of *complementary* label-paths that include all paths of length $\leq k$ (with a rather small k value).

[1]Where $*$ is $\cap$, $\cup$ or $-$.

3.1 Extending the $\mathcal{P}[k]$-Trie with F

Definition 3.1. *Given an XML document $\mathcal{X}$ and a workload F, the* k-extension *of F in $\mathcal{X}$, denoted $F^{\mathcal{X},k+}$ (or simply F^+), is defined as $F^{\mathcal{X},k+} = F \cup DownPaths(\mathcal{X}, k)$.*

The $\mathcal{WP}[k]$-Trie index is the simplest form of a $\mathcal{P}[k]$-Trie based, workload-aware index. It indexes the label-paths in the *k-extension* of a given workload. Figure 2 shows a $\mathcal{WP}[1]$-Trie index for the XML document in Figure 1.

The *lookup* function of a $\mathcal{WP}[k]$-Trie index takes a label-path query lp as input and retrieves the extent of the index entry with key lp^{-1}, or $\emptyset$ if $lp \notin DownPaths(\mathcal{X}, k)$. To assist in effective query optimization, the $\mathcal{WP}[k]$-Trie index also provides a *probe* function, which takes a label-path query lp as input and returns the label-path that is the *best* (longest) match to lp, or $\emptyset$ when there is enough information in the index to determine that $lp(\mathcal{X}) = \emptyset$.

The $\mathcal{P}[k]$-Trie index is a special case of the $\mathcal{WP}[k]$-Trie index where $F \subseteq DownPaths(\mathcal{X}, k)$. Thus, the $\mathcal{WP}[k]$-Trie index is also capable of answering all core XPath queries over $\mathcal{X}$ using index-only plans. Furthermore, it can answer path queries in F^+ with one index lookup. However, there is still room for improvement in terms of index size and its ability in efficiently identifying queries that yield empty results and answering non-frequent path queries longer than k.

3.2 Annotated Workload-aware Trie

By indexing only a subset of the downward label-paths beyond length k, the $\mathcal{WP}[k]$-Trie index no longer "fully represents" the structural distribution of the XML document, resulting in over-shredding of queries and unnecessary index lookups. To address this, we define the notion of label-path representativeness at a structural and instance level.

Definition 3.2. *Given a label-path $lp \in DownPaths(\mathcal{X}, k)$, $|lp| = k$, and a label-path set S, we say that lp is structurally represented by S with respect to $\mathcal{X}$, denoted $lp \prec^s S$, if $\forall$ $lp' \in DownPaths(\mathcal{X}, k+1)$ such that lp is a proper suffix of lp', $\exists$ $lp'' \in S$ such that lp' is a suffix of lp''.*

Given an XML document $\mathcal{X}$ and a workload F, for every lp that is either in F^+ or is the suffix of a label-path in F^+, we can easily compute the boolean flag $struct(lp) = lp \prec^s F^+$. This flag will allow us to identify more queries that yield empty results than the $\mathcal{WP}[k]$-Trie index by applying the following:

Lemma 3.1. *Given an XML document $\mathcal{X}$, a $\mathcal{WP}[k]$-Trie index T on $\mathcal{X}$ that is sensitive to a workload F, and a label-path query lp such that $lp \notin F^+$, let lp_s be the longest proper suffix of lp in F^+. Then, we can conclude that $lp(\mathcal{X}) = \emptyset$ if $lp_s \prec^s F^+$.*

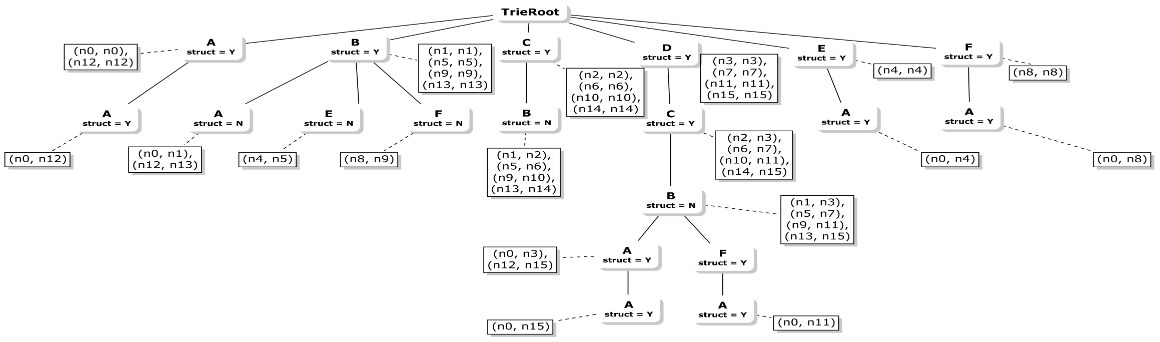


Figure 3: $\mathcal{AW}[1]$-Trie of $\mathcal{X}$ with $F = \{(A, A, B, C, D), (A, F, B, C, D)\}$.

Let us recall Theorem 2.1 stated in Section 2. We can simplify the evaluation of query E if the label-paths in $LPS(E, \mathcal{X})$ are all indexed.

DEFINITION 3.3. *Given an XML document $\mathcal{X}$, a set of label-paths S and a label-path $lp \in DownPaths(\mathcal{X})$, we say that lp is represented at the instance level by S with respect to $\mathcal{X}$, denoted $lp \prec^i S$, if there exists a label-path set $S' \subset \{lp' \mid lp' \in S \wedge lp$ is a suffix of $lp'\}$, such that $lp(\mathcal{X}) = \bigcup_{lp' \in S'} (lp'(\mathcal{X}))$.*

Note that S' does not necessarily include all label-paths that have lp as their suffix, but rather, there exists a smallest S' which includes only the "closest" label-paths that have lp as their suffix. To summarize our study of label-path representativeness, we define the notion of *full representation*:

DEFINITION 3.4. *Given an XML document $\mathcal{X}$, a set of label-paths S and a label-path $lp \in DownPaths(\mathcal{X})$, we say that lp is* fully represented *by S with respect to $\mathcal{X}$, denoted $lp \prec S$, if $lp \prec^s S \wedge lp \prec^i S$.*

EXAMPLE 3.1. *Consider the 1-extension $F^{\mathcal{X},1+}$ of $F = \{(A, A, B, C, D), (A, F, B, C, D)\}$ on the XML document shown in Figure 1. Label-path $(B, C, D) \not\prec^s F^{\mathcal{X},1+}$ since $(E, B, C, D) \in DownPaths(\mathcal{X}, 3)$, of which lp is a proper suffix, $\notin F^{\mathcal{X},1+}$. On the other hand, label-path $(A, B, C, D) \not\prec^i F^{\mathcal{X},1+}$ since we cannot compute the answer to query $A/B/C/D$ through the union of other label-paths in $F^{\mathcal{X},1+}$.*

We are now ready to define the notion of a *self-sustaining* label-path set and the *self-sustaining closure* of a label-path set.

DEFINITION 3.5. *Given an XML document $\mathcal{X}$ and a label-path set S, S is* self-sustaining *with respect to $\mathcal{X}$ if for each label-path lp that is a suffix of a path in S, it is the case that (1) $lp \in S$ or (2) $lp \prec S$.*

DEFINITION 3.6. *Given an XML document $\mathcal{X}$ and a label-path set S, the* self-sustaining closure *of S, denoted $S(\mathcal{X})^*$ (or simply S^*), is the minimal set among all sets that has S as a subset and is* self-sustaining *with respect to $\mathcal{X}$.*

The $\mathcal{AW}[k]$-Trie index of an XML document $\mathcal{X}$, sensitive to workload F, indexes the label-paths in $(F^+)^*$, and includes annotations that reflect the structural representativeness property for the suffix paths of every label-path in $(F^+)^*$.

EXAMPLE 3.2. *Figure 3 shows an example $\mathcal{AW}[1]$-Trie index of the XML document $\mathcal{X}$ shown in Figure 1. In this index structure, every node has an associated* struct *flag. For example, $struct((A, A)) = TRUE$, since there is no label-path in $DownPaths(\mathcal{X})$ that has (A, A) as a proper suffix; $struct((B, C, D)) = FALSE$, since the label-path (E, B, C, D) of $DownPaths(\mathcal{X})$ is not in $(F^{\mathcal{X},1+})^*$. Label-paths $lp_1 = (B, C, D)$ and $lp_2 = (A, B, C, D)$, which were not indexed in the $\mathcal{WP}[1]$-Trie index are indexed here since $lp_1 \not\prec^s F^{\mathcal{X},1+}$, and $lp_2 \not\prec^i F^{\mathcal{X},1+}$.*

We define two *lookup* methods for the $\mathcal{AW}[k]$-Trie index: (1) the direct lookup function is the same as the lookup in the $\mathcal{WP}[k]$-Trie index; (2) the sub-tree lookup uses Theorem 2.1 to find all the "closest" label-paths lp_i in the sub-tree rooted at the index entry associated with lp and computes $lp(\mathcal{X})$ under node semantics.

The *probe* function of the $\mathcal{AW}[k]$-Trie index takes a label-path as input and returns the *best* (longest) label-path for direct lookup, and the *best* label-paths for sub-tree lookup.

3.3 Sparse $\mathcal{AW}[k]$-Trie

The $\mathcal{AW}[k]$-Trie index is better equipped than the $\mathcal{WP}[k]$-Trie index in its ability to identify queries that yield empty results and answer non-frequent path queries longer than k, due to the introduction of the *struct* flag and the sub-tree lookup. However, there is still room for improvement in terms of index size.

We propose a variant of the $\mathcal{AW}[k]$-Trie index to further improve space efficiency with a minimum impact to query performance.

Given an XML document $\mathcal{X}$ and a workload F, we define the *restricted k-extension* of F, denoted $F^{\mathcal{X},(1,k)+}$, as the union of F and all downward label-paths in $\mathcal{X}$ that are of length 1 and k. The $\mathcal{W}[k]$-Trie index indexes all label-paths in the *self-sustaining closure* of $F^{\mathcal{X},(1,k)+}$, with annotations that indicate the *structural representativeness* of the label-paths in $(F^{\mathcal{X},(1,k)+})^*$ and their suffixes. The structures of the $\mathcal{AW}[k]$-Trie index and the $\mathcal{W}[k]$-Trie index are exactly the same for label-paths with length $\geq k$. A label-path lp with length $< k$ will only be indexed in the $\mathcal{W}[k]$-Trie index if (1) $lp \in F$, or (2) its length is 1, or (3) lp does not pass the instance-level representativeness test in $(F^{\mathcal{X},(1,k)+})^*$. Therefore, the $\mathcal{W}[k]$-Trie index is much more "sparse" on the top k levels and much smaller in size when compared to its $\mathcal{AW}[k]$-Trie counterpart, but it bears the same query evaluation power in answering frequent path queries and identifying queries that yield empty results.

Figure 4 shows an example $\mathcal{W}[1]$-Trie index for the document $\mathcal{X}$ shown in Figure 1. Note that there is no index entry associated with any label-paths of length 0 (except for (A) which fails the instance-level representativeness test) since none of them are part of $(F^{\mathcal{X},(1,1)+})^*$.

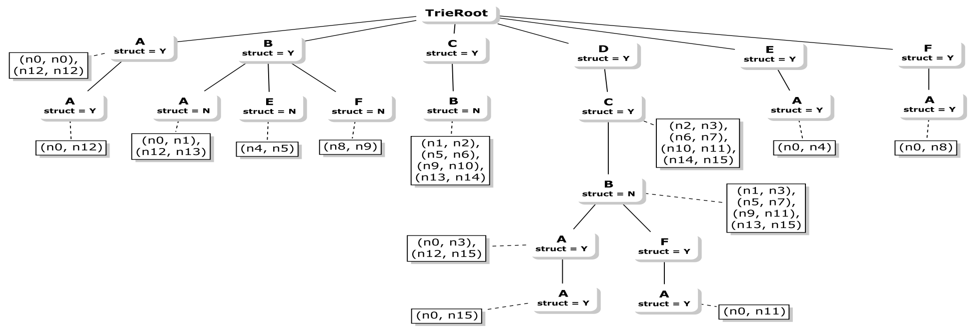
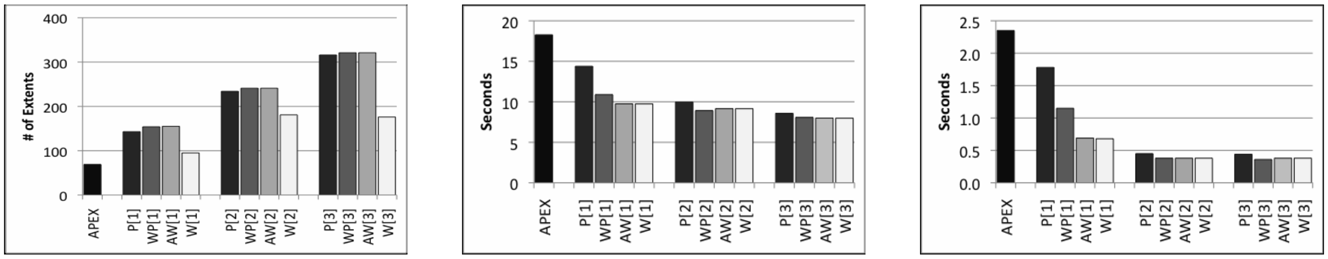


Figure 4: $\mathcal{W}[1]$-Trie of $\mathcal{X}$ with $F = \{(A, A, B, C, D), (A, F, B, C, D)\}$.



(a) Index Size (b) Query Set Evaluation (c) Q09 Evaluation

Figure 5: Experimental Results

4. EXPERIMENTAL EVALUATION

We compared the workload-aware Trie indices against each other and against two existing indices: (1) the $\mathcal{P}[k]$-Trie index [2], a non workload-aware, structural index, and (2) APEX [3], a workload-aware structural index that claims to provide index-only query evaluation plans for any label-path query. We implemented all indices in the Timber native XML database system [6]. We present the results of the experiments conducted on a NASA XML document that contains 1.4M nodes, a workload that combines longer and shorter label-paths (w.r.t. k), and a query set that contains 10 automatically generated queries that honor the workload.

Among the Trie index family, the index size (Figure 5(a)) is directly correlated to the k parameter. The sizes of the $\mathcal{WP}[k]$-Trie and the $\mathcal{AW}[k]$-Trie indices are comparable to the size of the $\mathcal{P}[k]$-Trie index with the same k value. Most importantly, the $\mathcal{W}[k]$-Trie index is *significantly* smaller than other indices in the Trie family with the same k value, with an average 35% decrease over the $\mathcal{P}[k]$-Trie index, and only a 25% increase over the APEX index with the same workload.

The ultimate goal of our workload-aware Trie index is to improve the overall query throughput when workload information is available. Figure 5(b) shows how the workload-aware Trie indices clearly outperform the $\mathcal{P}[k]$-Trie index and APEX in evaluating our query set. More importantly, our workload-aware indices perform very well with a small k parameter, making them a perfect solution for improving query performance when the available space is limited.

We attribute the performance gain to the fact that our workload-aware indices can facilitate efficient query processing for different types of queries. The workload-aware Trie indices and APEX perform equally well on frequent queries, outperforming the $\mathcal{P}[k]$-Trie index for queries longer than k. The Trie indices all outperform APEX on non-frequent path queries and queries with predicates (such as Q09 in Figure 5(c)), as longer sub-queries can be evaluated, while APEX can only evaluate sub-queries of length 0, resulting in multiple joins.

```
Q09   //dataset[textFile/description/para/footnote]
              /fitsFile/description/para/footnote
```

5. CONCLUSIONS

In this paper, we: (1) take advantage of the disk-based $\mathcal{P}[k]$-Trie index [2] and propose a family of workload-aware Trie indices for indexing frequent label-paths while maintaining efficient support for non-frequent queries; (2) introduce the concept of structural and data instance representativeness of a set of paths with respect to an XML document to guide the selection of the complementary label-path set and the annotation of the index structure; (3) propose a powerful probe function for the $\mathcal{AW}[k]$-Trie index, which opens the door for more sophisticated query optimization strategies; and (4) perform extensive experiments to compare the proposed indexing and query evaluation techniques with existing techniques in terms of the space footprint and query performance.

6. REFERENCES

[1] M. Benedikt, *et al.* Structural properties of XPath fragments. *Theor. Comput. Sci.*, 336(1), 2005.

[2] S. Brenes, *et al.* Trie Indexes for Efficient XML Query Evaluation. In *WebDB*, 2008.

[3] C.-W. Chung, *et al.* APEX: An Adaptive Path Index for XML Data. In *SIGMOD*, 2002.

[4] G. H. L. Fletcher, *et al.* A Methodology for Coupling Fragments of XPath with Structural Indexes for XML Documents. In *DBPL*, 2007.

[5] G. Gou, *et al.* Efficiently Querying Large XML Data Repositories: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(10), 2007.

[6] H. Jagadish, *et al.* TIMBER: A Native XML Database. *The International Journal on Very Large Data Bases*, 11, 2004.