Structural Twig Hash Indexing Scheme for XML Twig Query

Yi Mon Thet, ThiThiSoeNyunt, Yuzana
yimonthet.ucsy@gmail.com

Abstract

Since XML (eXtensible Markup) is the popular language for the data over the Internet, querying XML data is interested topic in research area. In order to efficiently process the XML query, indexing schemes are vital role in XML query processing systems. Structural path summary indexing scheme is efficiently support for path queries and achieve precise answer. It is also capable for solving the twig queries but additional join processing steps are needed to achieve precise answer. In this paper, the extendible hash table is proposed as twig indexing table. It is combined with existing 1-index (backward bisimilarity) structural path indexing scheme for processing twig queries that can be achieved precise answer. As a result, precise answers for twig query can be achieved without additional join processing steps. We have conducted on a series of experiments on the DBLP XML datasets to evaluate the performance of the proposed system.

***Keywords**: structural path indexing,twig indexing, twig query*

1. Introduction

As the XML has become a popularity for data representation and information exchange on the web, XML data management and query processing have attracted a lot of interesting in database community. Elements, attributes and value nodes are the basic structure of XML data and element nodes are represented in nested hierarchy. According to the tree structure of XML data, queries are specified as the path expressions to retrieve data from the XML tree. Many XML query languages [2, 3, 4, 9, 11] are proposed in literature. Among them XPath [3] and XQuery [2] are recommended by W3C. Because of the variety of structural relationships between various elements in XML data, structural index that reflects all of the structural relationships plays an important role in XML query processing. Node indexing schemes[6, 7, 8, 16] and path indexing (structural path summary) schemes [5, 10, 13, 15, 18] are proposed. Node indexing schemesdepend on many labeling approaches [12, 14, 21]and hold values that reflect the nodes' positions within the structure of an XML tree [17]. Pathindexing schemes can be used in both tree-shaped XML data and graph structured XML data.1-indexing scheme (backward bisimilarity) [18] is one of the structural path summary indexing schemes that summarizes the structural paths in XML data.It can process the simple path queriesand can achieve precise answers. However, additional join processing operations are needed for twig queries that can be achieved precise answer.

In this paper,extendible hash table is proposed as twig indexing table for XML twig pattern query. XML twig query is performed by combining structural path indexing (1-index) and extendible hash indexing schemes. Twig query is processed on both structural summary path index tree and extendible hash table.Twig query consists of two parts: filtering part(s) and result part. The filtering part(s) is processed on structural path summary tree and the result part is extracted from twig hash table**.** In structural path summary index tree, structural relationships and content searching (keys for hash table) are processed. In this process, partial results (key nodes) for hash table are achieved.After achieving the partial results from path summary index tree, keys are hashed to generate pseudokey. It is used to find values that are associated of keys in hash table. When the value of key is extracted from hash table, common (root) node and result node of the twig query is needed to be checked with the extracted value. Since structural related nodes are stored as values in hash table, structural relationships of result part are also satisfied and achieve precise answer for twig query without join processing steps.

The remaining of this paper is organized as follows. Related works are discussed in section 2. In section 3, background theory is presented. Data flow diagram, algorithms and overview of theproposed system architectureare introduced in section 4. The performance evaluation isdescribedin section 5. Finally, conclusion and future work are presented in section 6.

2. Related Works

In this section,some research results which are related to this paper are discussed.

Milo and Suciu [18] proposed 1-indexing scheme when the source data is tree shaped data. The

1-index partitions the data nodes of a document into equivalence classes based on their backward bisimilarity from the root node to the indexed node. For tree shaped data, 1-indexing scheme reduces the size of the structural summary to less than that of a Strong Data Guide However, It is complete and precise for evaluating path queries but not précis for evaluating twig queries. To reduce the size of a 1-index, Kaushik et al [15] proposed A(k) index. It also partitions the data nodes into equivalence classes based on backward bisimilarity. Like 1-index, A(k) index cannot achieve précised answers for twig queries. Chen et al. [10] proposed D(k) index, which assigns different k values to different index nodes based on a specific query workloads. Therefore, D(k)-index is more efficient than A(k)-index with regard to processing time and storage space. However, D(k) index cannot support for twig query and post processing steps are needed to achieve precise answers. Abiteboul, Bunemon, et al [5] introduced F&B index for twig queries that can be achieved precise answers. It is based on both incoming and outgoing (forward and backward) paths. Therefore, it can achieve precise answers in the initial steps and improve efficiency. However, there are insufficient memory problems for very large size indexes.

3. Background Theory

In this section, XML data model and twig pattern query are described.

3.1. XML Data Model

As the nature of the XML document is hierarchical and nested structure, it is modeled as node labeled tree T = (R,V,E), where R is the root node which is the parent of the all other nodes and V is the set of nodes (element nodes, attribute nodes and text nodes.) Among them, element and attribute nodes are the internal nodes of tree and leaf nodes represent the data values which are either a text in an element or an attribute value. E is the set of edges which connect element, sub-element, element-attribute, element-value, and attribute-value pairs. Two nodes connected by a tree edge are in parent-child (PC) relationship, and the two nodes on the same path are in ancestor-descendant (AD) relationship. Figure 1(a) shows a fragment of DBLP XML document and fig 1(b) shows data tree model of the XML document in fig 1(a).

```
<Bib>
<book>
<author>M. Tamer </author>
</book>
<paper></paper>
<paper>
<author>Frank Manola</author>
</paper>
<paper @reviewer="Jim Gray">
<author>AmerDiwan</author>
</paper>
</Bib>
```

Figure 1(a). Fragment of DBLP XML document

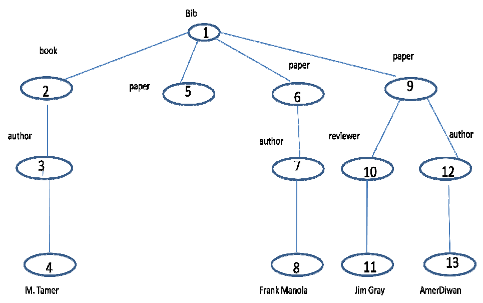


Figure 1(b). XML data tree model of figure 1(a).

3.2. Twig Pattern Query

The core query pattern in most standard XML query languages (e.g., XPathand XQuery) is also in a tree-like structure, which is often referred as a twig pattern. In particular, an XPath query is normally modeled as a twig patternquery. In a twig pattern query, an edge can be either single-lined or double-lined,which constraints the two matched nodes in either a PC relationship or an ADrelationship. Since a twig pattern normally models an XPath expression, leaf nodes of a twig pattern query are set to be a value based predicate condition. The process to find all the occurrences of a twig pattern in an XML document is called *twig pattern matching*. A *match* of a twig pattern Q in a document tree T is identified by a mapping from the query nodes in Q to the document nodes in T, such that: (i) each query node either has the same string name as or is evaluated true based on the corresponding document node, depending onwhether the query node is an element/attribute node. (ii) the relationship between the query nodes at the ends of each "/" or "//" edge in Q is satisfied by the relationship between the corresponding document nodes.For exampleXPathQuery Q1:*Bib/paper[/author="AmerDiawn"]/reviewer*is represented as twig pattern query in figure 2.

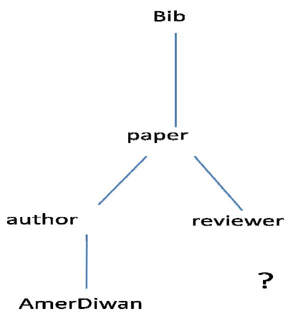


Figure 2.Example of Twig Pattern Query Expression.

3.3.Bisimilarity Algorithm

Bisimilarity algorithm [20] is applied in both graph structure and tree structure of XML data. There are two types of bisimilarity, namely forward and backward bisimilarity. In this section, backward bisimilarityconcept is presented because it is important in this paper. Two nodes u and v in tree T are said to be backward bisimilar if any two nodes u and v with , we have that (a) u and v have the same label, and (b) if u’ is a parent of u, then there is a parent v’ of v such that and vice versa. Figure 3 shows the structural path summary index tree of the DBLP XML document in figure 1(a).

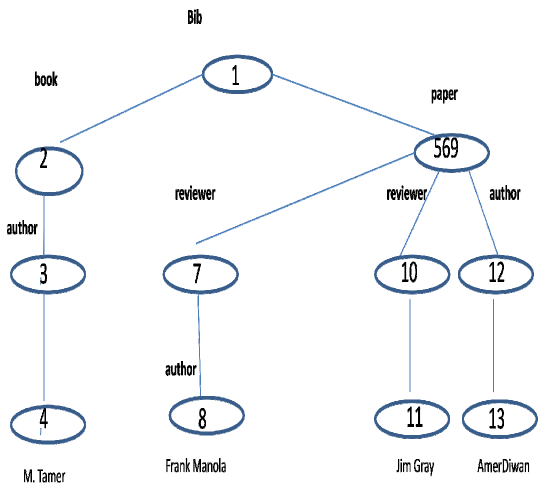


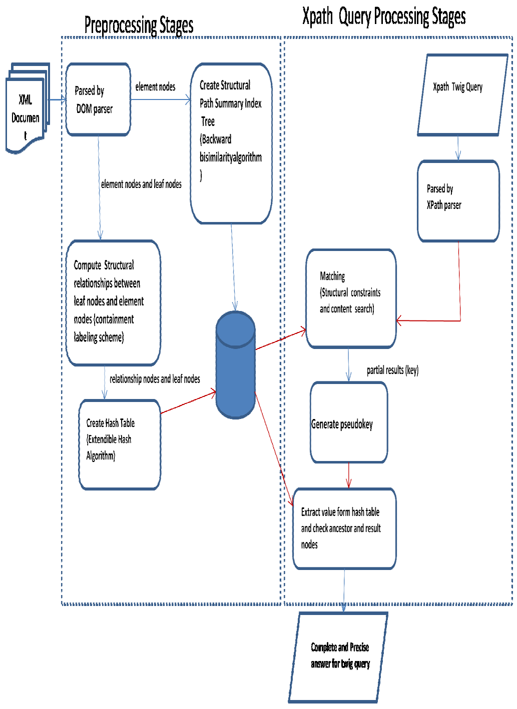
Figure 3. Structural Path Summary Index Tree

4. The Proposed System Architecture

In this section, the data flow diagram of the proposed system architecture and algorithmsof hash twig indexing scheme are described and then the query processing on Twig Hash Indexing Scheme is explained.

4.1. Data Flow Diagram of the Proposed System Architecture

The proposed system consists of two steps: preprocessing steps and XML query processing step. In preprocessing steps, backward bisimilarity method is used to build the structural path summary index tree and extendible hashing scheme is used for creating twig index. In addition, containment labeling scheme is used to compute the structural relationships of nodes that are stored as values in twig hash table.



Preprocessing Stages
Xpath Query Processing Stages
XML Document
Parsed by DOM parser
element nodes
Create Structural Path Summary Index Tree (Backward bisimilarityalgorithm)
element nodes and leaf nodes
Compute Structural relationships between leaf nodes and element nodes (containment labeling scheme)
relationship nodes and leaf nodes
Create Hash Table (Extendible Hash Algorithm)
Xpath Twig Query
Parsed by XPath parser
Matching (Structural constraints and content search)
partial results (key)
Generate pseudokey
Extract value form hash table and check ancestor and result nodes
Complete and Precise answer for twig query

Figure 4.Data Flow Diagram of the Proposed System Architecture.

4.2. Overview of Twig Hash Indexing Scheme

For XML twig pattern queries, common (root) node of the branch paths is the ancestor node that count form the leaf nodes in XML document tree is observed. Based on this observation, ancestor nodes, parent nodes and child (leaf) nodes are stored together as values in extendible hash table for processing the twig query. Twig queries in our system is processed on both structural path summary index tree (1-index) and extendible hash twig index. Twig queries consists of two parts: filtering part(s) and result part. The filtering part(s) is processed onstructural path summary tree (1-index) and the result part is extracted from the extendible twig hash table. In structural path summary index tree, structural relationships of filtering part(s) and keys for extendible twig hash table are processed. In this process, partial results (key nodes) for extendible twig hash table are achieved. After achieving the partial results, keys are hashed to generate the pseudo key for extracting the precise answer from extendible twig hash table. When the associated value of key is extracted from extendible twig hash table, common (root) node and result node of the twig query is needed to be checked with the associated relationship nodes of the extracted value. Since structural related nodes are stored as value in extendible twig hash table, structural relationships of result part are not needed to be computed again and achieve precise answer for twig query without join processing steps.

4.3. Creating extendible twig hash table

Leaf (value) nodes are used as keys and structural relationship nodes are set as associated values of keys that are stored in extendible twig hash table after the DBLP XML document is parsed by DOM parser.Containment labeling scheme [21] is used to compute the relationship nodes of XML data tree. Ancestor nodes, parent nodes and child nodes are stored as value in twig hash table. Figure 5(a) shows the pseudo keys for leaf nodes and figure 5(b) shows the extendible twig hash table.

H(M.Tamer)= 00000000000000001111111100000000

H(Frank Manola) = 01010100000000111111111111110000

H(Jim Gray)   = 10001000000000011100000000000111

H(AmerDiwan)   = 11111000000000001111110000000101

Figure 5(a). Pseudo keys for value(leaf) nodes.

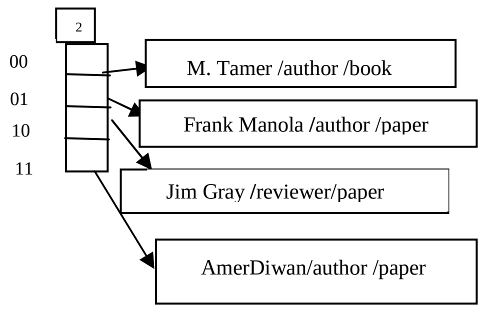


Figure 5(b). Extendible Twig Hash Table

4.4. Query Processing in Twig Hash Table

In our system, twig query is processed on both existing structural path summary index tree (1-index) and extendible twig hash table. The query processing algorithm is presented in algorithm 1.

Algorithm 1:TwigQMatch (Structural path summary Tree, Twig Hash table, twig query)

```
Input :  a twig query Q with n nodes {$q_1$, $q_2$, …, $q_n$} with P-C and A-D relationships, X be Structural path summary tree, H be Twig hash table.
Output : precise answer for twig query
1.     for each $X_i$ in X
2.     begin
3.     while $q_i$ {$q_1$, $q_2$, …, $q_n$} in Q
4.     begin
5.     if $q_i$ matches with $X_i$ then
6.     check $q_i$ contains in parent-child relationship of $X_i$ in X
7.      if qi contains in ancestor-descendent relationship of $X_i$ in X then
8.     expand  ancestor-descendent to parent-child relationship
9.     if $q_i$  is leaf nodes then
10.    generate $q_i$ as pseudo keys and retrieve answer from twig hash table
11.    $R_t \leftarrow$ TQMatch ($X_i$,$q_i$)
12.    end
13.       end
14.    end
```

The filtering part(s) of twig query are first processed. Line 5 checks whether the query node q_i and XML document element node X_i are matched or not. If match, it will continue to match the node in parent-child relationship. Line7 checks if q_i contains in ancestor-descendant relationship of X_i, then we expand ancestor-descendent to parent-child relationship. Line 10 -12 generate the pseudo key for leaf nodes and retrieved precise answer from extendible twig hash table.

Twig pattern query in figure 2 is used to illustrate how Extendible Twig hash Table. In above this twig pattern query, two parts are divided. Bib/paper/[author=AmerDiwan] is the filtering part and Bib/ paper/ reviewer is the result part. Filtering part is processed on structural path summary index tree (1-index). When the leaf node (AmerDiwan) is achieved, pseudo key is generated for this node and search the associated value of this pseudo key in extendible twig hash table. Finally, the common node (paper) and result (review) node of twig query and their related nodes of extracted value are needed to be checked. Since the structural related nodes are stored as values in extendible twig hash table, the precise answer is achieved for twig query without join processing steps.

5. Performance Evaluation

In this section, performance over twig queries is evaluated in our Twig hash indexing scheme. In our approach, we combine the hash table with existing structural indexing scheme. Hash-table is used as Twig index for processing the twig queries to avoid the joining process. The relationship XML data nodes are stored as value in twig hash table. Twig query is processed on both structural path summary tree and extendible twig hash table. By using hash table, precise answers are achieved for twig query without join processing steps. The comparison execution time of our approach and backward bisimilarity indexing scheme for twig queries is illustrated in figure6. All our experiments are tested over the 127 MB of DBLP data set [1].DBLP dataset is an XMLdocument, including information about papers, thesis, books and authors. In our system, we selected the general three tested twig queries for dblp data set. Non specific predicate query, predicate of equality comparison and multiple predicate of different comparisons under one object. The queries are shown in Table 1.

Table 1. The experimental queries

Data Set	Query	Twig Queries
	DQ1	dblp/phdthesis/[publisher]/author
DBLP	DQ2	dblp/article/[volume=TR-0244-12-93-165]/author
	DQ3	dblp/article/[author=Sai Choi Kwan]/[author=H.Raymond Strong]/[month=January]/title

Table 2. Extracted Results of Backward Bisimilarity and Twig Hash Table

Twig Queries	Backward Bisimilarity Indexing		Twig Hash Indexing	
	Extracted Results	Execution Time (second)	Extracted Results	Execution Time (second)
Q1	2	1.2	2	0.55
Q2	1	1.8	1	1
Q3	1	2.5	1	2.1

In backward bisimilarity indexing scheme, the returned answers for three tested twig queries are not precise and many extra documents are included and lead to the lower accuracy. Additional joining processes (post processing steps) are needed to achieve precise answers. In our twig hash indexing scheme, returned answer for twig queries are precise without joining processing steps. In both system, Completeness (recall) is achieved 100% because all relevant documents are retrieved by the query.The following figure shows returned answers of three tested queries and the execution time of 1-indexing scheme and twig hash indexing scheme. Since our approach doesn't need to join for twig queries, the execution time of our approach is faster than the 1-indexing scheme.

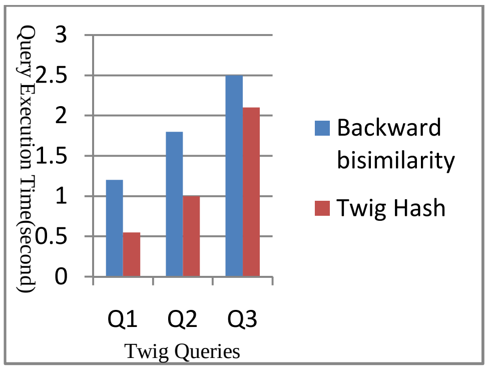


Figure 6. Execution Time of Backward bisimilarity Indexing scheme and Twig hash Indexing scheme.

6.Conclusion and Future Work

In this paper, extendible twig hash indexing table is combined with backward bisimilarity indexing scheme for processing the twig query that can be achieved precise answers without join processing steps. However, the proposed indexing scheme can only evaluate the equality operators of twig queries because extendible hash table is used as Twig indexing scheme. We will continue how to process inequality predicate operators in extendible twig hash table as a part of future work.

References

[1] http://www.cs.washington.edu/research/xmldatasets/data/dblp/dblp.xml

[2] W3C XML Query Specification, Latest.http://www.w3.org/TR/xquery

[3] W3C XML Path Language Specification, Latest.http://www.w3.org/TR/xpath

[4] S. Abiteboul*et al*. The Lorel Query Language for Semistructured Data. In International Journal on Digital Libraries, 1(1):68-88.1997.

[5] S. Abiteboul, P. Buneman, and D. Suciu, Data on the Web: From Relations to Semistructured Data and XML. Morgan Kaufmann, 1999.

[6] S.Al-Khalifa, H.V.Jagadish, J.M.Patel, Y.Wu, N.Koudas and D.Srivastava. Structural Joins: A primitive for efficient XML query pattern matching. In Proc.of ICDE, 2002, pp.141-154.

[7] N.Bruno, N.Koudas and D.Srivastava. Holistic twig joins: Optimal XML pattern matching. In Proc.of SIGMOD, 2002, pp.310-321.

[8] S.Chein,Z.Vagena, D.Zhang, V.Tsotras, C.Zaniolo. Efficient structural joins on indexed XML documents. In Proc. of 28th International Conference on Very Large Data Bases, 2002, pp. 263-274.

[9] T. Chinenyanga and N. Kushmerick. An Expressive and Efficient Language for XML Information Retrieval. In Journal of the American Society for Inf.Sci. and Tech., 53 (6):438-453, 2002.

[10] Q. Chen, A. Lim, and K.W. Ong, "D(K)-Index: An Adaptive Structural Summary for Graph-Structured Data," Proc. of 22nd ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '03), 2003.

[11] A. Deutsch, M. Fernandez, D. Florescu, A.Levy and D.Suciu. XML-QL:A query language for XML.InProc.of 8thInterntational World Wide Web Conference,1999.

[12] G.Gou, R.Chirkova. Efficiently Querying Large XML Data Repositories: A Survey. Transactions on Knowledge and Data Engineering, *2007,* 19(10), 1381-1403.

[13] R. Goldman and J. Widom, "DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases,"Proc. 23rd Int'l Conf. Very Large Data Bases (VLDB '97), 1997.

[14] D.K.Fisher, F.Lam, W.M.Shui and R.K.Wong.Dynamic Labeling Schemes for Ordered XML Based on Type Information. In Proc.of the 17th Australasian Database Conference, 2006, pp. 59-68.

[15] R. Kaushik, P. Shenoy, P. Bohannon, and E. Gudes, "Exploiting Local Similarity for Indexing Paths in Graph-Structured Data," Proc. 18th IEEE Int'l Conf. Data Eng. (ICDE '02), 2002.

[16] Q.Li, B.Moon. Indexing and querying XML data for regular path expressions. In Proc. of 27th International Conference on Very Large Data Bases, 2001, pp. 361-370.

[17] S.Mohammad and P.Martin. XML Structural Indexes. Technical Report 2009-560, School of Computing, Queen's University, June 2009.

[18] T. Milo and D. Suciu, "Index Structures for Path Expressions," Proc. of 7th Int'l Conf. Database Theory (ICDT '99), 1999.

[19] P.O'Neil, E.O'Neil, S.Pal, I.Cseri, G.Schaller and N.Westbury. ORDPATHs: Insert-Friendly XML Node Labels. In Proc.of SIGMOD, 2004, pp. 903-908.

[20] D. Park. Concurrency and automata on infinite sequences. In Theoretical Computer Science, 5th GI-Conf., LNCS 104, pages 167–183. Springer-Verlag, Karlsruhe, Mar. 1981.

[21] C.Zhang, J.Naughton, D.DeWitt, Q.Luo and G.Lohman. On Supporting containment Queries in Relational Database Management Systems. In Proc. of SIGMOD, 2001, pp.425