An efficient Path Query Processing support for Parent-Child Relationship in Native XML Databases

Journal of Digital
Information Management

Su-Cheng Haw, G.S.V. Radha Krishna Rao
Faculty of Information Technology
Multimedia University
63100 Cyberjaya, Malaysia
{schaw, gsvradha}@mmu.edu.my

ABSTRACT: *Due to its flexibility and efficiency in transmission of data, XML has become the emerging standard for data transfer and exchange across the Internet. In native XML database, XML documents are usually modeled as trees, and XML queries are typically specified in path expression. The primitive structural relationships are parent-child and ancestor-descendant in the path expression. Thus, finding all occurrences of these relationships is crucial. We adopt the decomposition-matching-merging approach and propose INLAB, a novel hybrid query processing merging both indexing and labeling technologies. Experimental results show that INLAB can process XML path queries by up to an order of magnitude faster than conventional top-down approach.*

Categories and Subject Descriptors
H.2 [Database Management]; H 2.3 [Languages]; Query Languages:

General Terms
XML Database, XML Query

Keywords: XML, query processing, query pattern matching, query evaluation, path query, path expression

Received 12 August 2006; Revised and accepted 22 Jan. 2007

1. Introduction

With the ever-increasing popularity of XML as data representation and exchange across the Internet, querying XML data has become an important issue to be addressed. Being a semi-structured data, there are two main approaches in XML query processing for native XML databases (NXD). The first approach is to traverse the XML tree sequentially to find the matching pattern. This approach certainly poses a new challenge, because it may not meet the processing requirements under heavy access requests [1]. Besides, it requires a large search space since there is no schema fixed in advance. As a result, some researchers had utilized index-based approach in order to reduce the portion to be scanned during query evaluation [2-6].

The second approach is executed through a series of processes involving decomposition, matching and merging [7-9, 18]. Firstly, a complex query pattern can be decomposed into a set of basic binary structural relationship between pairs of nodes. These relationships could be either of ancestor-descendant or parent-child relationship. The query pattern can then be matched by matching each of the binary structural relationships against the XML tree. Next, these matches are merged together to form the path solution. The disadvantage of this approach is that it may produce large size of intermediate results, which may not contribute to the final results at the end of query evaluation. This certainly imposes serious scalability and efficiency issues.

In this paper, we adopt the decomposition-matching-merging approach and propose a novel hybrid query processing technique, INLAB comprising both indexing and labeling which support parent-child relationship efficiently. The index structures of INLAB allow us to efficiently find all elements that belongs to the same parent or ancestor, which is one of the most common operations to evaluate path queries. The proposed labeling scheme quickly determines the parent-child relationship between elements in the XML tree.

Our contribution can be summarized as follows: -

- We propose a novel indexing and labeling scheme, INLAB to enhance the query processing performance.
- The proposed INLAB labeling scheme can be used for determining (i) ancestor-descendant and (ii) parent-child relationships efficiently.
- The proposed INLAB query processing can process path expression queries without traversing the whole XML tree. We present and show the substantial performance benefit of our approach on a range of real and synthetic data.

The rest of the paper is organized as follows. Section 2 presents the background and related work. Section 3 gives an overview of INLAB indexing, labeling and processing technique. Section 4 presents the experimental setup, findings and preliminary results. Lastly, section 5 concludes the paper and suggests future work.

2. Background and Related Work

2.1 XML Document And Query Model

In this paper, Object Exchange Model (OEM) is used to represent data in NXD [10]. It can be viewed as a rooted, ordered, node-labeled tree where each node represents an element or a value. Element-subelement or element-value is represented by the labeled edge. For the sample XML document of Figure 1(a), its OEM representation is shown in Figure 1(b).

```
<publications>
   <book ISBN = “0-7895-6514-5”>
        <title category = “Multimedia”> Discovering Computer
2005 </title>
        <author>Gary B. Shelly </author>
        <author>Thomas J. Cashman </author>
        <author>Misty E. Vermaat </author>
        <publicationInfo>
                <place>Boston</place>
                <publisher>Thomson Learning</publisher>
                <dateIssued>2002</dateIssued>
                <dateRevised>2003</dateRevised>
        </publicationInfo>
          <price>60.99</price>
   </book>
   <journal number = “J1” year=”2003">
        <title category = “XML”> XML Query …</title>
        <author>S.C. Haw </author>
        <price>20.00</price>
   </journal>
</publications>
```

Figure 1. (a) A sample XML document

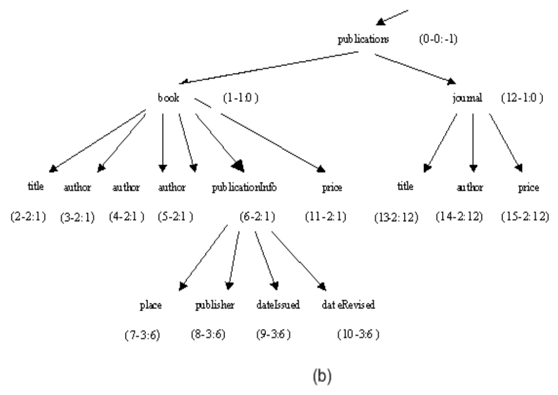


Figure 1. (b) OEM representation [11].

In fact, XML query languages were a hot research topic. In this context, several XML query languages have been proposed such as Lorel [12], XPath [13], XQuery [14] and YATL [15]. Although the query languages differ in detailed grammars and representation, they share a common feature, that is, queries usually make use of path expression as the matching criteria to create a join between each binary structural relationship [16]. Parent-child relationship is denoted by "/" while ancestor-descendant relationship is denoted by "//". Table 1 summarized some representational notations for path expressions.

There are two types of queries namely full-text queries and structural queries. Nevertheless, this paper is mainly concerned with structural queries. Structural relationship can be categorized into two main classes, namely path query and twig pattern query as depicted in Figures 2(a) and 2(b) respectively [17-19]. Both path query and twig pattern query can be decomposed into a set of basic binary structural relationship between pairs of nodes. However, the focus of this paper is only on path query. Hence, Figure 3 illustrates the decomposition process of path query.

Symbol	Description
//	Separator indicating ancestor-descendant relationship
/	Separator indicating parent-child relationship
?	Zero or one occurrence of a node
+	One or more occurrences of a node
*	Zero or more occurrences of a node
[]	Encloses a predicate expression
@	Representing attributes
()	Indicates precedence

Table 1. Notations for path expressions

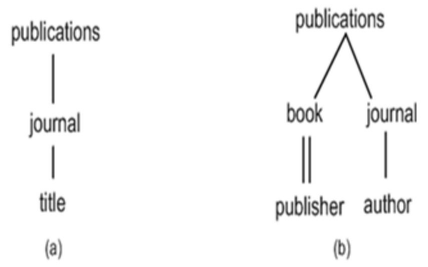


Figure 2. (a) Path query (b) twig pattern query

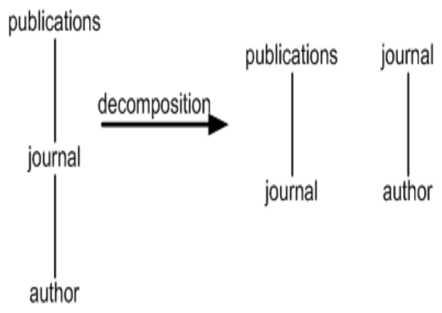


Figure 3. Decomposition process on path query

2.2 XML Query Processing

Path traversal plays an important role in query processing. McHugh and Widom propose three approaches that process the path traversal: top-down, bottom-up and hybrid approach [2]. Top-down approach starts the traversal from root of the tree traversing down level by level to find the matching nodes while bottom-up approach starts the traversal from the bottom of the tree traversing up level by level to find the matching nodes. The hybrid approach, however, combines both the top-down and bottom-up approaches, and stops when a convergence is found. In the worst cases, all these approaches need to search the whole tree, hence are inefficient.

Query pattern matching are typically decomposition-matching-merging processes involving (1) decomposition of query pattern into binary relationships between pairs of nodes, (2) matching each binary component of the query pattern against the XML database and (3) merge-join them to obtain the final results.

MPMGJN [8], PathStack [9] and Stack-Tree [20] algorithms focus on the second sub process: matching the binary structural relationships. The difference between MPMGJN and both PathStack and Stack-Tree is that MPMGJN requires multiple scans on input lists for the matching process. The PathStack and Stack-Tree algorithms are more efficient as they use stack to maintain the ancestor or parent nodes and therefore require only one time scan per input list. These approaches use the labeling of *(docno, begin: end, level)* for an element and *(docno, wordno, level)* for a text word as the positional representation of XML elements and texts. However, we use <*self–level: parent*> as the positional representation instead. Details on this will be explained in section 3. Some other XML query processing performed in a streaming fashion include XMLTK [21], XSQ [22], EXPedite [23], XSM [24] and YFilter [25].

3. Overview of INLAB

3.1. INLAB labeling scheme

In INLAB labeling scheme, given an XML tree, any label consists of <*self-level:parent*> representation, where (i) *self* is obtained by doing a pre-order traversal of the tree nodes (ii) *level* of a node is its distance from the root and (iii) *parent* is the direct node which relates to the *self* node.

Now, we introduce how to assign the label using INLAB labeling scheme. Basically, assignment on *self*, *level* and *parent* attributes can be obtained easily as follows. The *self* attribute is computed based on pre-order traversal the beginning of the XML tree. For instance, in Figure 1(a), *publications* is the root of the tree and therefore, it has 0 as the *self* attribute. The first child of root, *book* will be assigned with the *self* attribute as 1. Next, the first child of *book* (the first child of root) will be assigned with *self* attribute as 2. So, the leaf of the last child (node *price*) will be assigned with highest *self* attribute (in this case, 15).

The *level* attribute of a node is the nesting depth of the node form the root. Because the root has a zero distance from itself, the root is at level 0. The children of the root are at *level* 1, their children are *level* 2 and so forth.

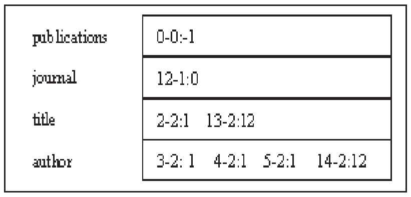
The *parent* attribute is obtained by tracing to which node the outdegree edge connects to. For example, node *publicationInfo* has an outdegree edge which links to node *book*. Thus, the *parent* attribute for *publicationInfo* is 1.

Structural relationship between nodes can be efficiently determined from the label <*self-level:parent*>:-

1. parent-child relationship
   $node_1$ is the parent of $node_2$ if and only if $node_1$.self = $node_2$.parent. For example, *book* (1-1:0) is the parent of *title* (2-2:1) element because *book* has *self* attribute 1 that is equal to *title parent's* attribute 1.
2. ancestor-descendant relationship
   $node_1$ is an ancestor of $node_2$ if and only if leveldiff = $node_2$.level - $node_1$.level > 1. Further explanation is shown in section 3.2.

A set of encoded XML streams is generated to store each node label, groups by their node name. Figure 4 shows the fragment streams generated based on the sample XML data in Figure 1(a). Consequently, instead of parsing regular XML data, we parse the set of INLAB encoded XML to be evaluated. There are several advantages to parsing INLAB encoded XML data over parsing regular XML data:-

1. Integer processing is very efficient
2. The size of the label is 12 bytes, which is much shorter than the previous labeling schemes.



publications	0-0:-1
journal	12-1:0
title	2-2:1 13-2:12
author	3-2: 1 4-2:1 5-2:1 14-2:12

Figure 4. Fragment of INLAB encoded XML

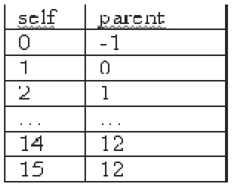
3.2.CheckAncestor Function

Function *CheckAncestor()* (Function 1) take two nodes (*q* and *n*) for comparison and returns whether the two nodes is of ancestor-descendant relationship. This function loops as long as the level different (*leveldiff*) between the two nodes is greater than zero (line 7-8). If it is not, it returns in line 16 and 18. In line 9, function *hashPCTable()* is being invoked. During this process, the index table, PCTable (storing parent-child relationship) is being hashed to retrieve each node's parent for comparison. Fragment of PCTable is depicted in Figure 5. Function *getSelf()* and *getLevel()* returns the *self* and *level* attributes of the node in the stream that are being processed.

Function 1

```
function checkAncestor(q, n) {
1. input : two nodes
2. output : boolean true or false
3. int leveldiff=0, current = 0, cursorUp = 0
4. leveldiff = getLevel(n) – getLevel(q)
5. current = getSelf(n)
6. if (getSelf(n) != eof) {
7.    if (leveldiff> 0) {
8.       while (leveldiff > 0) {
9.          cursorUp = hashPCTable(current)
10.           current = cursorUp
11.         leveldiff—
12.      }
13.      if (current = getSelf(q)) return true
14.      else return false
15.   }
16. return false
17. }
18. return false
19. } //end function

function hashPCTable (q) {
1. input : self label of current node
2. output : parent node of the current node
```



self	parent
0	-1
1	0
2	1
...	...
14	12
15	12

Figure 5. Fragment of PCTable index table

3.3 INLAB Processing

Each node in the path query is associated with a stream. Each stream contains the positional representations of the node appearance in the XML tree (as shown in Figure 4). The nodes in the stream are sorted by their *self* attribute, and thus, this will determine the order of the node to be process. Associated with each stream is a stack. Stack is used to store the possible intermediate results.

Using the path query defined in Figure 2(a) as an example, there exist two structural binary relationships after the decomposition process, namely, *publications-journal* and *journal-author.* We first try to find the matches for the *publications-journal* relationship against the XML tree. The *publications* stream is being accessed to retrieve the first node to be processed. There is only one occurrence, that is node <0-0:-1>. Since *publications* is the root of the path query, node <0-0:-1> is push directly into the *publications* stack. Next, *journal* stream is being accessed. There is only one occurrence, that is node <12-1:0>. The *parent* attribute of this node is equal to the *self* attribute of *publications* node. Thus, node <12-1:0> is pushed into *journal* stack.

Next, we need to find matches for the other binary relationship, *journal-author*. The *author* stream will be accessed first

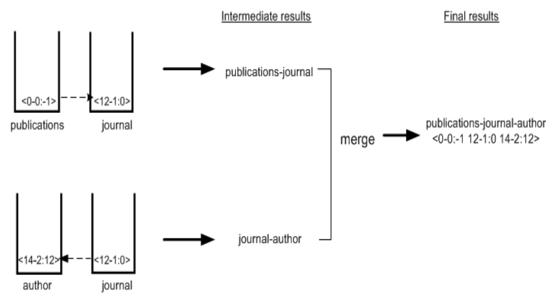


Figure 6. Stacks operation for INLAB processing

because the *self* attribute of the first occurrence is less than the *self* attribute of the first occurrence in *journal* stream. However, node <3-2:1> *parent* attribute does not occurs as any *self* attribute in *journal* stream. Therefore, this node is ignored. The next node will be <4-2:1>, <5-2:1> and <14-2:12> respectively. Only node <14-2:12> *parent* attribute occurs as *self* attribute in the *journal* stream. Thus, node <14-2:12> is pushed into *author* stack. After finding the matches for each structural binary relationship, the intermediate results are being merged to produce the final solution. Figure 6 illustrates the stack operation on matching-merging processes.

4. Experimental Evaluation

We have implemented INLAB using Java API for XML Processing (JAXP). In this section, we describe experimental setup and present the preliminary results.

4.1. Experimental Setup

We run experiments on four datasets, two synthetic and two real datasets. The real datasets are obtained from the International Protein Sequence Database [26] and Sigmod database [27] where else the synthetic datasets, DBLP Computer Science Bibliography and TreeBank are obtained from the University of Washington XML repository [28]. All our experiments were performed on 1.7GHz Pentium IV processor with 1.024 GB SDRAM running on windows 2000 systems. We benchmark our results with the conventional top-down traversal approach [2] by using the set of query listed in Table 2 and Table 3 over the modified TreeBank dataset. TreeBank dataset has been modified into smaller scale (approximately 3MB file size) to cater for conventional top-down approach. These queries are composed in such a way that all edges in Q1 to Q4 are parent-child relationship while in Q5 is ancestor-descendant relationship.

All numbers presented here are produced by running the experiments multiple times and averaging the execution times of several consecutive runs.

Query	Path expression
Q1	S/NP
Q2	NP/PP
Q3	S/NP/NN
Q4	S/VP/NP/NN
Q5	S//NP

Table 2. Queries over Treebank dataset

Query	Path expression	Path length
Q1	FILE/EMPTY	2
Q2	FILE/EMPTY/S	3
Q3	FILE/EMPTY/S/VP	4
Q4	FILE/EMPTY/S/VP/NP	5
Q5	FILE/EMPTY/S/VP/NP/NN	6

Table 3. Queries with different path length

4.2. Performance Results

Figure 7(a) shows that INLAB encoding outperforms in terms of reducing the XML file size. XML data is usually much smaller, about 15%-60% than the original XML file. For a larger XML file size (as shown by Protein dataset) by using INLAB encoding, there is a major reduction in file size, about 63%. Thus, it is very suitable especially in reducing the size for a large-scale dataset.

Figure 7(b) shows the performance of a conventional SAX parser and the XML encoded parser, INLAB. From the result, we see that the parsing time using INLAB is slightly higher than SAX parser. This is because the parsing time using INLAB comprises both parsing and encoding time. Although it is slightly higher (around 1%-2%), the different is not significant and thus, could be ignored.

Figure 8(a-b) show the execution time of queries defined in Table 2 and Table 3 respectively for INLAB and conventional top-down approach. As can be observed, conventional top-down approach is much slower compared to INLAB (generally over an order of magnitude). This is because conventional top-down approach is too conservative when backtracking and reads several times unnecessary nodes in the XML document when comparing for matches. In Figure 8(b), we see that the performance of conventional top-down approach degrades drastically with the increasing size of the path length. On the other hand, by using INLAB technique, the execution time increase slightly with the increment of path length. Thus, INLAB is much more efficient and scalable as compared to the conventional top-down approach.

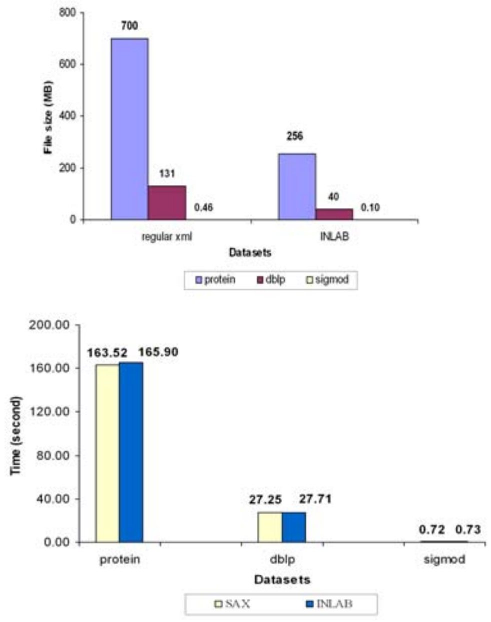
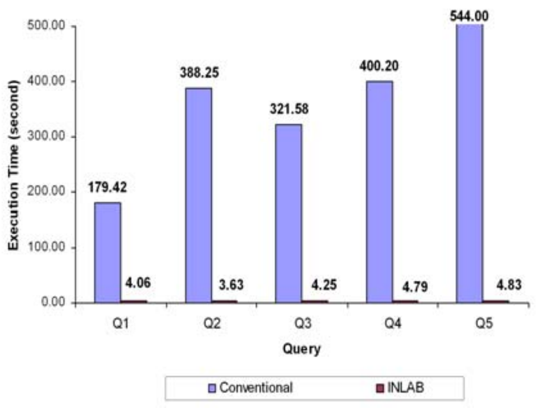


Figure 7 (a) File size on regular XML and INLAB encoded (b) Parsing time for parsers on different datasets



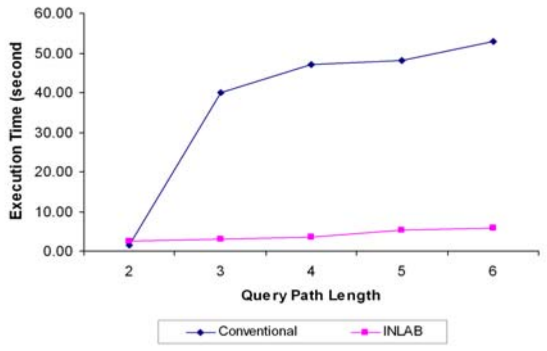


Figure 8 (a) Execution time over TreeBank query set (b) Execution time by varying path query length

5. Conclusions

In this paper, we have proposed a hybrid query processing technique, INLAB comprising both indexing and labeling. Using the INLAB labeling scheme, structural relationships can be determined easily. The extensive experimental results showed that INLAB labeling scheme is efficient and yet simple. We complement INLAB with indexing technologies to speed up the matching and merging processes among each binary structural relationship. Performance results show that our technique outperforms the conventional top-down approach in terms of reducing the XML file sizes, faster parsing and execution time and scalability in most cases.

We are currently exploring a number of optimization issues in INLAB processing. The study can be further extended in future. Some of the future approaches could includes 1): utilizing the sibling and ordered query relationship to optimize the decomposition-matching-merging processes, 2): supports branching path query (twig pattern query), 3): performance tuning on the matching sub-process especially to determine the ancestor-descendant relationship.

References

[1] Li, Q., Moon, B. (2001). Indexing and Querying XML Data for Regular Path Expressions, *In*: Proceedings of 27th VLDB Conference. 361-370.

[2] McHugh, J., Widom, L (1999). Query Optimization for XML. Proceeding 25th International Conference on Very Large Databases. 315-326.

[3] Kaushik, R., Shenoy, D., Bohannon, P., Gudes, E. (2002). Exploiting Local Similarity to Efficiently Index Paths in Graph-Structured Data, *In*: Proceedings of International Conference on Data Engineering. 129-140.

[4] Kim, J.,Kim, H-J. (2003). Efficient processing of regular path joins using PID. Information and Software Technology 45. 241-251.

[5] Milo, T., Suciu, D. (1999). Index structures for path expression, *In*: Proceedings of 7th International Conference on Database Theory. 277-295.

[6] Chung, C.W., Min, J. K., Shim, K. (2002). APEX : An Adaptive Path Index for XML data, *In*: Proceedings of ACM SIGMOD. 121-132.

[7] Yao, J.T., Zhang, M. (2004). A Fast Tree Pattern Matching Algorithm for XML Query, *In*: Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence. 235-241.

[8] Zhang, C., Naughton, J., DeWitty, D., Luo, Q., Lohman, G. (2001). On Supporting Containment Queries in Relational Database Management Systems, *In:* Proceedings of ACM SIGMOD. 425-436.

[9] Bruno, N., Srivastava, D., Koudas, D. (2002). Holistic twig joins: optimal XML pattern matching, *In*: Proceedings of ACM SIGMOD. 310-321.

[10] Papakonstantinou, Y., Abiteboul, S., Garcia-Molina, H. (1996). Object Fusion in Mediator Systems, *In*: Proceedings of the 22nd International Conference on Very Large Data Bases. 413-424.

[11] Haw, S.C and Rao, G.S.V.R.K. (2006). INLAB : A Hybrid XML Query Optimization Technique, *In*: Proceedings of International Conference on Computer and Communication Engineering,ICCCE'06, Vol 1. 219-224.

[12] Abiteboul, S. et al. (1997). The Lorel Query Language for Semistructed Data, *Journal of Digital Libraries*. 1(1). 68-88.

[13] W3C, XML Path Language (XPath). Available http://www.w3.org/TR/xpath-datamodel/

[14] W3C, XML Query (XQuery). Available http://www.w3.org/XML/XQuery

[15] Christophides, V., Cluet, S., Simeon, J. (2000). On Wrapping Query Languages and Efficient XML Integration, ACM SIGMOD International Conference on Management of Data, ACM Press. 141-152.

[16] Haw, S.C and Rao, G.S.V.R.K. (2005). Query Optimization Techniques for XML Databases. International Journal of Information Technology, 2(1). 97-104.

[17] Aghili, S.A., Li, H-G., Agrawal, D., Abbai, A. E. TWIX: Approximate and Exact Twig Structure and Content Matching over XML Document Collections using Binary Labeling. University of California, Santa Barbara.

[18] Kim, J., Lee, S.H., Kim, H-J. (2004). Efficient structural joins with clustered extents. Information Processing Letters 91. Elsevier. 69-75

[19] Rao, P., Moon, B. (2004). PRIX: Indexing And Querying XML Using Prüfer Sequences. Proceedings of International Conference of Data Engineering. 288-300.

[20] Al-Khalifa, S., Jagadish, H.V., Koudas, N., Patel, J.M., Srivastava, D., Wu Y. (2002). Structural Joins: A Primitive for Efficient XML Query Pattern Matching. Proceedings of International Conference of Data Engineering. 141-152

[21] Green, T.J., Miklau, G., Onizuka, M., Suciu, D. (2003). Processing XML Streams with Deterministic Automata. Proceedings of International Conference of Database Theory. 173-189.

[22] Peng, F. and Chawathe, S.S. (2003) XPath queries on streaming data. Proceedings of ACM SIGMOD. 431-442.

[23] Chen, Y., Padmanabhan, S., Mihaila, G.A., Davidson, S.B. (2004). Efficient Path Query Processing on Encoded XML. Proceedings of High Performance XML Processing.

[24] Ludascher, B., Mukhopadhayn, P., Papakonstantinou, Y. (2002). A Transducer-Based XML Query Processor. Proceedings of VLDB. 227-238.

[25] Diao, Y., Altinel, M., Franklin, M.J., Zhang, H., Fischer, P. (2003). Path Sharing and Predicate Evaluation for High-Performance XML Filtering. ACM Transactions on Database Systems, 28 (4). 467-516.

[26] Georgetown Protein Information Resource, Protein Sequence Database. (2001).
Available at http://pir.georgetown.edu/

[27] Sigmod Database, ACM. (2005) Available at http://www.sigmod.org/record/xml/

[28] Treebank and DBLP dataset, University of Washington XML Repository. (2002)
Available at http://www.cs.washington.edu/research/xmldatasets/

Su-Cheng Haw is currently pursuing her Ph.D (Information Technology) in Multimedia University, Malaysia. Her research interests include XML database, query optimization, database tuning, data warehousing, Entity-Relationship Approach and web programming.

Radha Krishna Rao is currently associated with the Faculty of Information Technology at Multimedia University, Malaysia. His research interests include Software Test Automation, Web Services Security, Network Processors, Hyperthreading Technology, Databases, Operating Systems.