XSeq: An Indexing Infrastructure for Tree Pattern Queries

Xiaofeng Meng Yu Jiang Yan Chen
Renmin University of China
Beijing 100872, China
{jiangyu,chenyan,xfmeng}@ruc.edu.cn

Haixun Wang
IBM T. J. Watson Research Center
Hawthorne, NY 10532
haixun@us.ibm.com

1. INTRODUCTION

Given a tree-pattern query, most XML indexing approaches decompose it into multiple sub-queries, and then join their results to provide the answer to the original query. Join operations have been identified as the most time-consuming component in XML query processing.

XSeq is a powerful XML indexing infrastructure which makes tree patterns a first class citizen in XML query processing. Unlike most indexing methods that directly manipulate tree structures, XSeq builds its indexing infrastructure on a much simpler data model: sequences. That is, we represent both XML data and XML queries by structure-encoded sequences. We have shown that this new data representation preserves query equivalence [9], and more importantly, through subsequence matching, structured queries can be answered directly without resorting to expensive join operations. Moreover, the XSeq infrastructure unifies indices on both the content and the structure of XML documents, hence it achieves an additional performance advantage over methods indexing either just content or structure, or indexing them separately.

XSeq originated from our early work ViST [10], which introduced sequence-based XML indexing. We refer the readers to [9] for the latest theoretical and technical details of XSeq.

2. THE MOTIVATION OF XSEQ

The semi-structured nature of XML data and the demand on query flexibility pose unique challenges to database indexing methods.

Querying XML data is close to finding sub structures of the data graph that match the query structure. The query structure may contain values, wild-cards ('*' and '//'), tree patterns, etc. The purpose of XML indexing is to provide efficient support for structured queries. However, in state-of-the-art indexing solutions [1, 7, 6, 4, 3, 5, 2], tree pattern is not a first class citizen, or the most basic query unit. Instead, the most commonly supported query interface is the following:

$$\text{Simple Paths} \Rightarrow \mathcal{P}(\text{Node Ids})$$

That is, given a path, the index structure returns a set of nodes that represent such a path. Some index methods extend the above interface to support paths that *start with* a '*' or '//' with a much larger index [3].

Still, for tree-pattern queries or queries with '*' or '//' inside, we have to decompose them into a set of simple path queries. Then, we use join operations to merge their results to answer the original query. To avoid expensive join operations, some index methods create special index entries for a limited set of frequently queried patterns [3, 5].

We have proposed sequence-based XML indexing [10], which represents a major departure from previous XML indexing approaches. The new method supports a more general query interface:

$$\text{Tree Pattern} \Rightarrow \mathcal{P}(\text{Doc Ids})$$

That is, given a tree pattern, the index returns a set of XML documents that contain such a pattern. Instead of disassembling a structured query into multiple sub queries, the tree structure itself is used as the basic query unit.

3. TREE STRUCTURE SEQUENCING

Instead of directly manipulating tree structures, XSeq adopts a much simpler data model: sequences. There are many ways to transform a tree structure to a sequence. For instance, we can encode each node by the path leading from the root to the node, and represent an XML tree by its preorder sequence. In the same spirit, we convert XML queries to preorder sequences.

Our purpose is to perform XML queries by subsequence matching so that structured queries can be processed as a whole instead of being broken down into smaller query units, because combining the results of these sub queries through join operations is expensive. In other words, we want to use tree patterns as the basic unit of query.

Thus, the most important step is to establish query equivalence between a structure match and a subsequence match. For certain query structures, such equivalence seems obvious. However, a subsequence match does not always correspond to a structure match. The sequence-based approach opens up many new research issues. XSeq addresses the following challenges: i) Query equivalence, or, does a subsequence match always correspond to a structure match? ii) What are the sequencing methods that preserve query equivalence? iii) How to perform subsequence match under query equivalence? iv) For datasets based on a given

SIGMOD 2004 June 13-18, 2004, Paris, France.

schema, which sequencing method shall we use in order to maximize index and query performance? v) How to support dynamic index updates using only standard data structures such as the B$^+$Tree and the hash table? We refer the readers to [9] for a detailed discussion of these challenges and their solutions.

Query Equivalence. Naïve subsequence matching cannot preserve query equivalence. The two tree structures in Figure 1 are apparently different, however, there is a non-contiguous subsequence match between their sequence representations, that is, $\mathcal{Q} \subset \mathcal{D}$. XSeq adopts an indexing algorithm that preserves query equivalence.

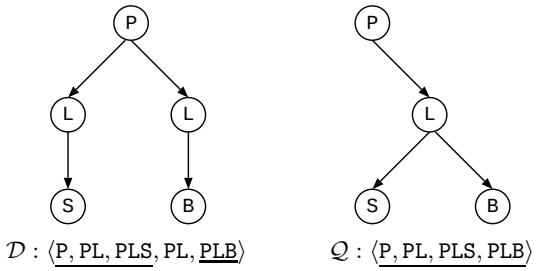


Figure 1: False Alarm

Performance-oriented Sequencing. XSeq identifies a set of sequential representations for a tree structure, each of which preserves query equivalence. The question is, which sequence should be used? XSeq makes the decision based on the DTD schema or the distribution of the dataset, which leads us to the 'best' sequencing method in terms of index and query performance.

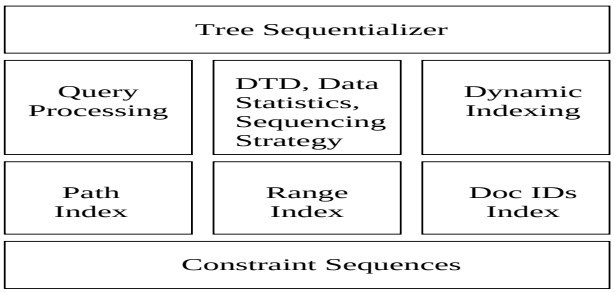
The XSeq Storage Infrastructure. XSeq's underlying data model is sequence, and its storage model for the sequence is based on the virtual suffix tree [10]. An important feature of the virtual suffix tree is that it can be realized by standard data structures, such as the B$^+$Tree and the hash table, which are well supported in commercial DBMSs. Thus, XSeq offers an easily deployable solution to XML data management.

4. ABOUT THE DEMO

Our demonstration consists of the following parts.

A Web-based Query Interface. The web interface allows users to submit structured queries in the form of a graph. Users construct queries from graph components such as elements, attributes, values, and wildcards '*', '//'. Users can inspect the internal representation of the query structure, which is a constraint sequence. After processing the query, the web interface returns a list of XML documents, and users can verify that each of them contains the query structure as an embedded graph.

A Web-based XSeq Tuner. The XSeq tuner enables users to adjust the node occurrence probabilities XSeq derives from the data. XSeq's tree sequencing strategy can be changed through the adjustment of such probabilities. The demo will show that different sequencing strategies result in indices of different sizes, which create a direct impact on query performance. The demo will also compare XSeq's dynamic sequencing strategies with predefined sequencing strategies, including depth-first sequences, breadth-first sequences, etc.



The XSeq Infrastructure

The demo will reveal XSeq's internal architecture. The data model of XSeq is the constraint sequence, which preserves query equivalence between a structure match and a subsequence match. The constraint sequences are managed by three indices, namely the path index, the range index, and the document id index. The indices are accessed by the query processing module as well as the dynamic indexing module. For a given dataset, XSeq's sequencing strategy is derived from its schema and its data distribution.

5. REFERENCES

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the web: from relations to semistructured data and XML.* Morgan Kaufmann Publishers, Los Altos, CA 94022, USA, 1999.

[2] C. Chung, J. Min, and K. Shim. APEX: An adaptive path index for XML data. In *ACM SIGMOD*, June 2002.

[3] B. F. Cooper, N. Sample, M. Franklin, G. Hjaltason, and M. Shadmon. A fast index for semistructured data. In *VLDB*, pages 341–350, September 2001.

[4] R. Goldman and J. Widom. DataGuides: Enable query formulation and optimization in semistructured databases. In *VLDB*, pages 436–445, August 1997.

[5] R. Kaushik, P. Bohannon, J. Naughton, and H. Korth. Covering indexes for branching path queries. In *ACM SIGMOD*, June 2002.

[6] Q. Li and B. Moon. Indexing and querying XML data for regular path expressions. In *VLDB*, pages 361–370, September 2001.

[7] T. Milo and D. Suciu. Index structures for path expression. In *Proceedings of 7th International Conference on Database Theory (ICDT)*, pages 277–295, January 1999.

[8] Praveen R. Raw and Bongki Moon. PRIX: Indexing and querying XML using prüfer sequences. In *ICDE*, 2004.

[9] Haixun Wang, Xiaofeng Meng, Wei Fan, and Philip S. Yu. Sequential and structural query equivalence in XML query processing. Technical report, http://wis.cs.ucla.edu/~ hxwang/xseq.pdf, 2003.

[10] Haixun Wang, Sanghyun Park, Wei Fan, and Philip S. Yu. ViST: A dynamic index method for querying XML data by tree structures. In *SIGMOD*, 2003.