

Indexing for XML Siblings

SungRan Cho
L3S, University of Hannover
scho@l3s.de

ABSTRACT

Efficient querying XML documents is an increasingly important issue considering the fact that XML becomes the de facto standard for data representation and exchange over the Web, and XML data in diverse data sources and applications is growing rapidly in size. Given the importance of XPath based query access, Grust proposed R-tree index, we refer to as *whole-tree indexes* (WI). Such index, however, has a very high cost for the following-sibling and preceding-sibling axes. In this paper we develop a family of index structures, which we refer to as *split-tree indexes* (SI), to address this problem, in which (i) XML data is horizontally split by a simple, yet efficient criteria, and (ii) the split value is associated with tree labeling. While the SI is straightforward to construct, it incurs the overlap problem between bounding boxes. We resolve this problem by designing the *transformed split-tree indexes* (TSI). We also study the most promising existing method of constructing R-tree, the Hilbert tree, so that we take advantage of its benefit for XML siblings. Lastly, we experimentally demonstrate the benefits of the TSI for siblings over the WI using benchmark data sets.

1. INTRODUCTION

With the advent of XML as the de facto standard for data representation and exchange over the Web, querying XML documents has become more important. In this context, XML query evaluation engines need to be able to efficiently identify the elements along each location step in the XPath query. Several index structures for XML documents have been proposed [4, 5, 9, 10, 13, 15], in a way to efficiently querying XML documents.

As XML documents are modeled by a tree structure, a numbering scheme, labeling tree elements, allows for managing the hierarchy of XML data. For example, each element has the position, a pair of its beginning and end locations in a depth

first search. In general, the numbering approach has the benefit of easily determining the ancestor-descendant relationship in a tree. In this respect, R-tree index using node's preorder and postorder, we refer to as *whole-tree indexes* (WI), has been proposed in [5]. Such index, however, does not consider issues related to the costs of the preceding-sibling and following-sibling axes.

In this paper, we discuss index techniques to reduce the cost of performing XML siblings. This also addresses an issue of what efficient packing for XML tree data is. An efficient packing method for a tree is not only to group together data elements which are close in a tree, but also to reduce dead space resulting in false positives (no data in indexed space). The packing method of the WI, taking a whole tree, may cover considerable dead space, which influences querying XML siblings. We design the *split-tree index* (SI) to address the problem, in which (i) an XML tree is horizontally split by a simple, but efficient criteria, and (ii) the split value is associated with tree labeling. The SI uses standard R-tree index lookup algorithms to match elements along XPath location steps.

Since data trees are indexed separately in the SI, bounding boxes representing data elements may overlap, which impacts the performance. We resolve this problem by developing the *transformed split-tree index* (TSI), in which all elements are transformed into new dimensions. To take advantage of the semantics of the index structure, we develop index lookup for XPath axes in the TSI.

We next consider the most promising existing method of constructing R-tree for XML siblings, the Hilbert tree, which preserves data locality well in dimensions. While the WI, SI, and TSI use the preorder to cluster node elements, the Hilbert R-tree uses the Hilbert ordering generated from node's positional numbers in a tree. It is shown in our experiment that the Hilbert R-tree has an even page access across all XPath axes.

Finally, we perform an experimental comparison between the WI, SI, TSI, and Hilbert R-tree. Using the XMark benchmark data set, we demonstrate that the index lookup costs in the TSI is superior to the WI for siblings and comparable to other axes in the WI.

This paper is organized as follows. Section 2 presents some background material along with the structure of our indexes (WI, SI, and TSI) and motivates our indexes (SI and TSI). In Section 3, we present the split criteria of a tree as the base of building the SI. In Section 4, we design a new index TSI by enhancing SI and presents the index lookup for the TSI. Section 5

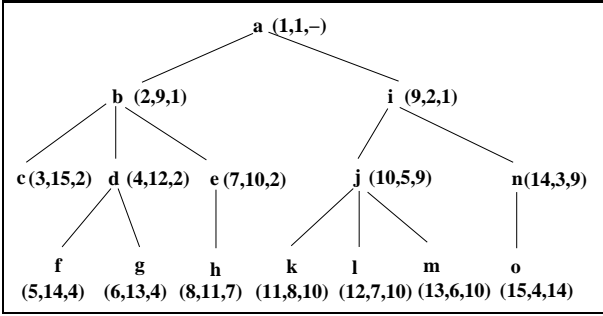


Figure 1: XML data tree encoded with L_n , R_n , PL_n

describes the Hilbert R-tree for XML tree. The benefits of SI, TSI, and Hilbert indexes over WI are evaluated experimentally in Section 6. We present related work in Section 7 and conclude in Section 8.

2. BACKGROUND

We assume familiarity with XML and XPath expressions. Here, we review the encodings used on XML nodes to facilitate matching of XPath axes.

2.1 Whole-Tree Indexes: WI

A preorder and a postorder numbering of nodes (see, e.g., [9, 5]) in an (ordered, tree-structured) XML document suffice to reconstruct the XML document unambiguously.

In this paper, we use an encoding scheme, L_n and R_n , for nodes in XML documents that has the same effect as preorder and postorder. L_n is the rank at which the node is encountered in a *left to right* depth first search (DFS) of the XML data tree, and R_n is the rank at which the node is encountered in a *right to left* DFS. In order to handle level sensitive matching, such as child and parent axes (matching nodes one level apart), and following-sibling and preceding-sibling axes (matching nodes with the same parent), the parent node’s L_n , written as PL_n is associated with each node. Thus each XML element node is labeled with three numbers: L_n , R_n , and PL_n . These numbers become coordinates in multi-dimensions. Figure 1 is an example of XML data tree where each node is encoded with L_n , R_n , and PL_n .

The *whole-tree index* (WI) is obtained by the R-tree index on the (L_n, R_n, PL_n) dimensions, in which XML data is loaded using L_n ordering.

2.2 Matching XPath Location Steps on WI

XML nodes are packed in a *bounding box* which denotes the set of leaf pages rooted at a non-leaf index page entry. Non-leaf pages in the index structure maintain the low and high ranges, represented in $(B_{L_n_{low}}, B_{R_n_{low}}, B_{PL_n_{low}}, B_{L_n_{high}}, B_{R_n_{high}}, B_{PL_n_{high}})$. Given a query node Q_{L_n, R_n, PL_n} and a non-leaf index page, let us review the conditions, for matching XPath location steps along eight XPath axes (i.e., descendant, child, ancestor, parent, preceding, following, preceding-sibling, and following-sibling).

- **descendant:** $(B_{L_n_{high}} > Q_{L_n}) \ \& \ (B_{R_n_{high}} > Q_{R_n})$.

Axes	Leaf	Axes	Leaf
ancestor	3.15	descendant	1.16
parent	1	child	1.11
preceding	846	preceding-sibling	7.56
following	847	following-sibling	166.9

Table 1: Leaf page accesses

- **child:** $(B_{L_n_{high}} > Q_{L_n}) \ \& \ (B_{R_n_{high}} > Q_{R_n}) \ \& \ (B_{PL_n_{low}} \leq Q_{L_n} \leq B_{PL_n_{high}})$.
- **ancestor:** $(B_{L_n_{low}} < Q_{L_n}) \ \& \ (R_{n_{low}} < Q_{R_n})$.
- **parent:** $(B_{L_n_{low}} < Q_{L_n}) \ \& \ (R_{n_{low}} < Q_{R_n}) \ \& \ (B_{L_n_{low}} \leq Q_{PL_n} \leq B_{L_n_{high}})$.
- **preceding:** $(B_{L_n_{low}} < Q_{L_n}) \ \& \ (B_{R_n_{high}} > Q_{R_n})$.
- **following:** $(B_{L_n_{high}} > Q_{L_n}) \ \& \ (B_{R_n_{low}} < Q_{R_n})$.
- **preceding-sibling:** $(B_{L_n_{low}} < Q_{L_n}) \ \& \ (B_{R_n_{high}} > Q_{R_n}) \ \& \ (B_{PL_n_{low}} \leq Q_{PL_n} \leq B_{PL_n_{high}})$.
- **following-sibling:** $(B_{L_n_{high}} > Q_{L_n}) \ \& \ (B_{R_n_{low}} < Q_{R_n}) \ \& \ (B_{PL_n_{low}} \leq Q_{PL_n} \leq B_{PL_n_{high}})$.

2.3 Motivation

We first ran the XMark benchmark dataset (see <http://monetdb.cwi.nl/xml/>) with 10 MBytes dataset and a page capacity of 100 nodes to observe the number of page accesses for sibling axes in the WI. We counted the number of leaf page accesses of the index tree needed to find the results while exploring XPath axes. The leaf page access results for XPath location steps are given in Table 1.

As observed in Table 1, the WI has a high cost of the following-sibling and preceding-sibling, on the average 166 page accesses for the following-sibling and 7 page accesses for the preceding-sibling. However the ancestor, parent, child, and descendant axes rather do well (on the average between 1 and 3 page accesses). For preceding and following axes, a node has (on the average) half of the document preceding and following it, respectively. The experiment results motivate us to propose a new indexing technique that can reduce the cost of XML siblings. Next we would like to discuss a question of why the costs of the preceding-sibling and following-sibling axes are not symmetric. One reason for this is that false positives (due to R-tree bounding boxes) of following-sibling axis, is much higher than those in other axes.

3. SPLIT-TREE INDEXES: SI

In this section, we discuss a split strategy for XML data tree as the base of obtaining the *split-tree indexes* (SI).

3.1 An Horizontal Split Strategy

An XML data tree is split using node’s L_n and R_n . A key of the split strategy is α where α can be the total number of nodes in a tree, the maximum level of a tree, or etc. Since the value of α is still an open issue, we don’t specify the value of α in this paper. Instead we will study the costs of XPath

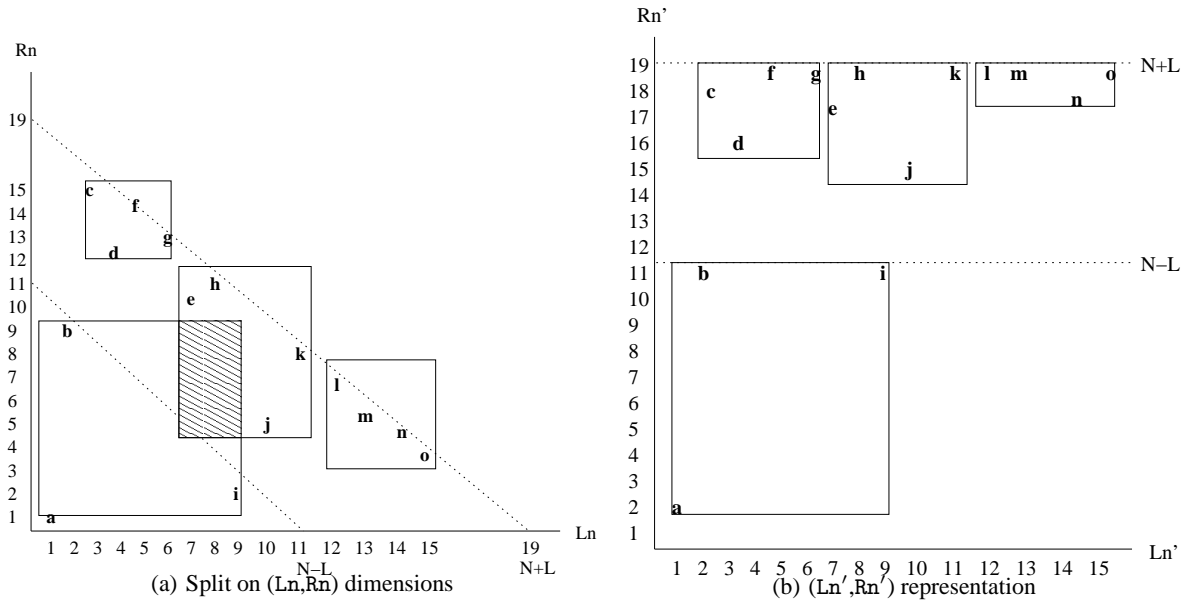


Figure 2: (L_n, R_n) and (L_n', R_n') dimensions

axes with varying α (experiment results provide more details in Section 6).

XML data nodes on the (L_n, R_n, PL_n) dimensions are divided on the center of the line $R_n = -L_n + \alpha$, and in effect the data tree is split horizontally. For example, Figure 2 (a) is a 2-dimensional representation of the data tree of Figure 1, where x and y axes represent L_n and R_n , respectively. It shows that the dimensions are divided by the line $R_n = -L_n + (N - L)$, where N is the total number of nodes and L is the maximum level in the tree. The property of the split regions in this example is that all leaf nodes and some intermediate nodes in the data tree lie above the line $R_n = -L_n + (N - L)$, which might be dense (we refer to as an *upper region*), and some intermediate nodes lie under the line, which might be sparse (we refer to as a *lower region*).

3.2 Designing SI

The *split-tree index* (SI) is constructed by indexing XML data nodes in the upper and lower regions separately. The separate packing reduces long thin boundary boxes, produced by WI, that may contain dead space (space which is indexed but does not have data). As for index lookup, the SI uses the same WI lookup algorithms (see Section 2.2) to identify the desired elements along an XPath axis from a specified element.

Since we pack each region separately, we obtain the overlap between bounding boxes at each region of the tree. For example, Figure 2 (a) where the pack capacity is four nodes, shows an overlap area highlighted in a shaded rectangle. Due to overlap, multiple paths from the root downwards on the SI may need to be traversed, which results in increasing page accesses. We address this problem by designing the TSI next.

4. TRANSFORMED SI: TSI

In this section we design a family of transformed split-tree

indexes (TSI) to avoid possible overlap in the SI.

4.1 Coordinate Transformation

New coordinates of nodes are determined by their origins. A node element n with coordinates (L_n, R_n, PL_n) is transformed into $n'=(L_n', R_n', PL_n')$, such that

$$\begin{aligned} L_n' &= L_n \\ R_n' &= L_n + R_n \\ PL_n' &= PL_n \end{aligned}$$

The original coordinates are extended with in the R_n direction with respect to L_n . Thus the new dimensions, (L_n', R_n', PL_n') , are at most $N \times L \times N$ larger than the original dimensions, where N is the total number of nodes and L is the maximum level in the tree. More importantly this result does not break the hierarchy of a tree represented in multi-dimensions. Figure 2 (b) shows the tree on (L_n', R_n') dimensions transformed from the data tree of Figure 2 (a). In the transformed dimensions, without allowing overlap, TSI is constructed in manner of building SI whose packing is based on L_n ordering.

4.2 Matching XPath Location Steps on TSI

We discuss the conditions for matching XPath location steps along XPath axes on (L_n', R_n', PL_n') dimensions. First, we present a comparison of XPath axes crossed on between (L_n, R_n) and (L_n', R_n') dimensions, which is shown in Figure 3. This suggests the new conditions for XPath axes in the TSI. Given a query node $Q_{L_n', R_n', PL_n'}$ and a non-leaf index page represented in $(B_{L_n'_{low}}, B_{R_n'_{low}}, B_{PL_n'_{low}}, B_{L_n'_{high}}, B_{R_n'_{high}}, B_{PL_n'_{high}})$, we develop the conditions for XPath location steps along eight XPath axes (more details are provided in Appendix).

- **descendant:** $(B_{R_n'_{high}} - B_{L_n'_{low}} > Q_{R_n'} - Q_{L_n'}) \ \& \ (B_{L_n'_{high}} > Q_{L_n'}) \ \& \ (B_{R_n'_{high}} > Q_{R_n'})$.

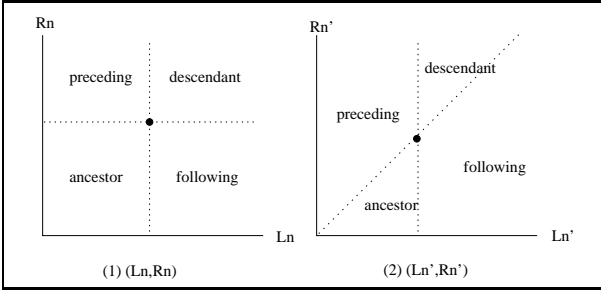


Figure 3: XPath conditions on (L_n, R_n) and (L_n', R_n')

- **child:** $(B_{R_n'} - B_{L_n'} > Q_{R_n'} - Q_{L_n'}) \&$
 $(B_{L_n'} > Q_{L_n'}) \& (B_{R_n'} > Q_{R_n'}) \&$
 $(B_{PL_n'} \leq Q_{L_n'} \leq B_{PL_n'})$.
- **ancestor:** $(B_{R_n'} - B_{L_n'} < Q_{R_n'} - Q_{L_n'}) \&$
 $(B_{L_n'} < Q_{L_n'}) \& (B_{R_n'} < Q_{R_n'})$.
- **parent:** $(B_{R_n'} - B_{L_n'} < Q_{R_n'} - Q_{L_n'}) \&$
 $(B_{L_n'} < Q_{L_n'}) \& (B_{R_n'} < Q_{R_n'}) \&$
 $(B_{L_n'} \leq Q_{PL_n'} \leq B_{PL_n'})$.
- **preceding:** $(B_{R_n'} - B_{L_n'} > Q_{R_n'} - Q_{L_n'}) \&$
 $(B_{L_n'} < Q_{L_n'})$.
- **following:** $(B_{R_n'} - B_{L_n'} < Q_{R_n'} - Q_{L_n'}) \&$
 $(B_{L_n'} > Q_{L_n'})$.
- **preceding-sibling:** $(B_{R_n'} - B_{L_n'} > Q_{R_n'} - Q_{L_n'}) \&$
 $(B_{L_n'} < Q_{L_n'}) \& (B_{PL_n'} \leq Q_{PL_n'} \leq B_{PL_n'})$.
- **following-sibling:** $(B_{R_n'} - B_{L_n'} < Q_{R_n'} - Q_{L_n'}) \&$
 $(B_{L_n'} > Q_{L_n'}) \& (B_{PL_n'} \leq Q_{PL_n'} \leq B_{PL_n'})$.

5. USING HILBERT R-TREE

In this section, we present the Hilbert R-tree for XML siblings. The Hilbert curve is a space filling curve that visits all the points in k -dimensional space exactly once and never crosses itself. The order has been used to arrange data elements in many applications such as image processing, CAD, and etc. The reason that the Hilbert ordering preserves spatial locality is to map points which are close together in k -dimensional space into points that also close together in one-dimensional space. In [7], the Hilbert R-tree is constructed in manner to index smaller regions of space so that search can be focused on the relevant regions. It thus minimizes the resulting page boundaries.

In order to build Hilbert R-tree for XML data, we first generate the Hilbert values of data nodes in (L_n, R_n, PL_n) dimensions and then load the data into the index using the Hilbert ordering. The same index lookup algorithm for XPath location steps, described in Section 2.2, is used. Consequently, from our experiment, the resulting costs across all XPath axes (except for the following and preceding axes) are almost even (this result is explained in next section).

α	l-region	u-region
N	16,447	152,757
$N-L/2$	9,439	159,765
$N-L$	5,613	163,591
$N-2L$	2,203	163,591
$N-4L$	472	168,732
$N-8L$	82	169,122

Table 2: Node counts at each region

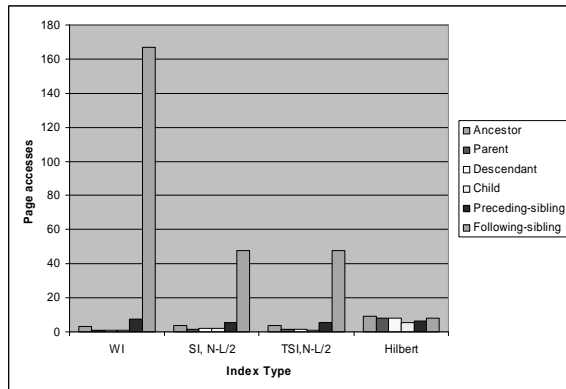
6. EXPERIMENTS

In this section, we present the result of the evaluation of our index. We conducted experiment using the XMark benchmark dataset. The size of the XMark is about 10 MBytes and the packing size is 100 nodes. In our experiment, we compare WI, SI, TSI, and Hilbert indexes.

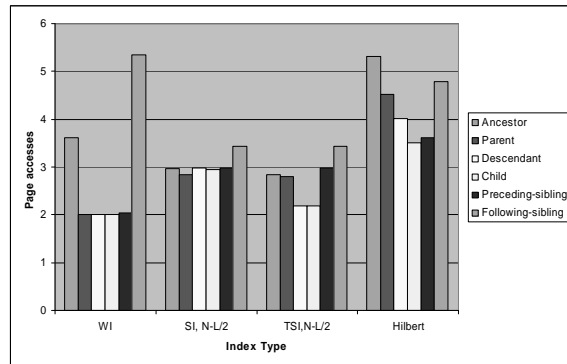
We used six XPath navigation axes for query, i.e., child, parent, ancestor, descendant, following-sibling and preceding-sibling axes. The results of following and preceding are excluded because the costs of those axes are the same on WI, SI, TSI, and Hilbert indexes. We also observed our evaluation with varying the value of α (the split value of a tree).

The leaf and non-leaf page access results for six axes over WI, SI, TSI, and Hilbert are given in Figure 5 (a) and (b), respectively, in which the value of α is chosen as $N - L/2$ for SI and TSI. From the plot (a), the SI, TSI and Hilbert indexes outperform the WI for the following-sibling and preceding-sibling axes. The number of leaf page accesses is roughly the same for the following-sibling and preceding-sibling axes between SI and TSI. When compared to WI, the following-sibling and preceding axes are about 3.5 times and 1.5 times cheaper respectively in the SI and TSI, whereas those are about 20 times and 1.1 times cheaper respectively in the Hilbert. More importantly the Hilbert has the symmetric results between the preceding-sibling and following-sibling axes due to its packing property. As for other axes, the number of page access in the SI and TSI is slightly higher than that in WI index (about 1.2 times larger for ancestor, child, descendant and 1.8 times for parent). This is expected in the split R-tree, since nodes that are close in the hierarchy, may be split and then packed separately. The Hilbert is between 3 and 8 times larger than the WI for other axes. From our results, the overlap affects the results of the descendant and child axes, but it nearly does for other axes. The TSI has a saving of 50% over the SI for the descendant and child axes. The number of non-leaf page accesses between WI and TSI is roughly the same, except for the following-sibling which is 36% cheaper in the TSI.

Next we measured the cost of executing XPath navigation axes with varying the value of α in the TSI. Associating with α , the number of nodes in the lower and upper regions is given in Table 2, where N is the total number of nodes and L is the maximum level of the tree. As one decreases the value of α , the number of leaf page accesses of the preceding-sibling and following-sibling axes increases, but that of the parent, ancestor, child, descendant axes decreases. For example, in our experiment, when α is $N - 8L$, in the TSI the ancestor, parent, descendant and child axes are up to 1.6 times smaller than when α is $N - L/2$. However, the preceding-sibling and following-



(a) Leaf



(b) Non-Leaf

Figure 4: WI, SI, TSI, and Hilbert

sibling axes are about 1.5 times and 3.5 times larger respectively with α of $N - 8L$. The result graph is shown in Figure 5 (a). The number of non-leaf page accesses is approximately the same with varying α , which is given in Figure 5 (b).

7. RELATED WORK

In this section, we primarily discuss the core problems in tree structured data. Research on indices for a tree-structured data is mainly divided into two classes: (i) a numbering based index, which assigns meaningful numbers to tree nodes as identifiers; (ii) a prefix based index for paths.

Recently index techniques using tree labeling have become a focus of research in efficiently answering XPath queries. One classical numbering based index is the level-based index (LBI) structure in [11], which decomposes the data into several levels indicating their nesting (i.e., descendant nodes are nested) and indexes the data at each level separately. Li et al. [9] proposed more flexible numbering scheme using a pair of pre-order and postorder to efficiently process regular path expression queries. Wang et al. [13] developed ViST, a dynamic indexing method for XML documents, by representing both XML data and queries in structure encoded sequences. Cohen et al. [3] proposed a dynamic labeling scheme, which is useful for maintenance of index.

There are several proposals for prefix based indexing [8, 4, 16]. In [8], an identifier of an ancestor is a prefix of the identifiers of its descendants. The problem with this method is that as the length of identifiers increases, the cost increases. In [4], paths which are sequences of element tags are encoded as strings and are indexed. Deschler et al. [16] proposed MASS (a multiple axis storage structure) indexing structure that supports XPath querying and XML document updates.

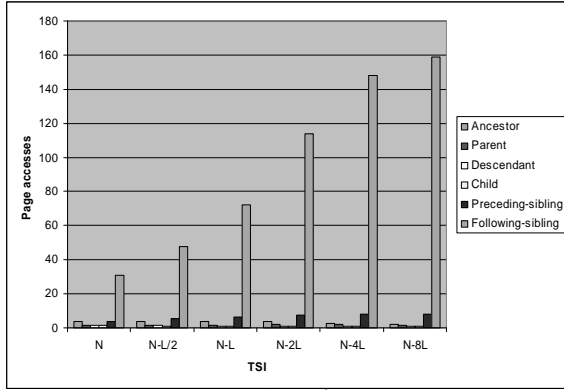
8. CONCLUSION

In this paper we have considered the problem of constructing an efficient R-tree index for XPath siblings. The main idea has been to group together XML data which are close to each other in the hierarchy as well as to contain less dead space. In this context, we developed TSI, which is constructed over a horizontally split tree. We also considered an existing R-tree

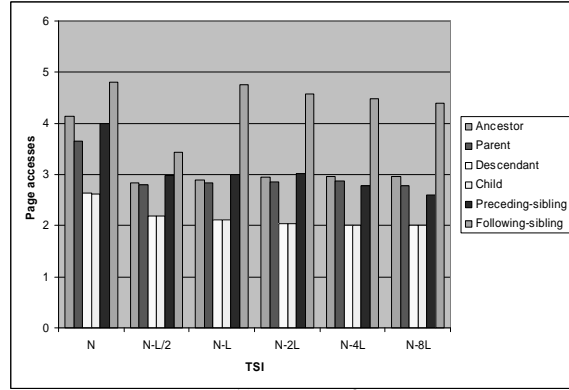
technique, the Hilbert tree, in order to take advantage of its clustering method for siblings. Our preliminary experiment results demonstrate the benefits of our techniques for siblings over the WI.

9. REFERENCES

- [1] M. Altinel and M. Franklin. Efficient filtering of XML documents for selective dissemination of information. In *Proc. of VLDB*, Cairo, Egypt, 53–64, 2000.
- [2] J. Clark and S. DeRose. XML path language (XPath) version 1.0 w3c recommendation, Technical Report REC-xpath-19991116, World Wide Web Consortium, 1999.
- [3] E. Cohen, H. Kaplan and T. Milo. Labeling dynamic XML trees, In *Proc. of PODS*, 271–281, 2002.
- [4] B.F. Copper, N. Sample, M.J. Franklin, G.R. Hjaltason and M. Shadmon. A fast index for semistructured data, In *Proc. of VLDB*, Rome, Italy, 341–350, 2001.
- [5] T. Grust. Accelerating XPath location steps, In *Proc. of SIGMOD*, 2002.
- [6] A. Guttman. R-trees: a dynamic index structure for spatial searching, In *Proc. of SIGMOD*, 45–47, 1984.
- [7] I. Kamel and C. Faloutsos, Hilbert r-tree: an improved r-tree using fractals, In *Proc. of VLDB*, Santiago, Chile, 500–509, 1994.
- [8] W.E. Kimber. HyTime and SGML: understanding the HyTime HYQ query language, Technical Report Version 1.1 IBM Corporation, 1993.
- [9] Q. Li and B. Moon. Indexing and querying XML data for regular path expressions, In *Proc. of VLDB*, Rome, Italy, 361–370, 2001.
- [10] T. Milo and D. Suciu. Index structure for path expressions, In *Proc. of ICDT*, Jerusalem, Israel, 271–295, 1999.
- [11] K.V. Ravikanth, D. Agrawal, A.E. Abbadi, A.K. Singh and T. Smith. Indexing hierarchical data, Univ. of California, CS-Tr-9514, 1995.
- [12] H. Wang and X. Meng. On the sequencing of tree structures for XML indexing, Preceding of the



(a) Leaf



(b) Non-Leaf

Figure 5: TSI

International Conference on Data Engineering, Tokyo, Japan, 2005.

- [13] H. Wang, S. Park, W. Fan and P. Yu. ViST: a dynamic index method for querying XML data by tree structures, In *Proc. of SIGMOD*, San Diego, USA, 2003.
- [14] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler. Extensible Markup Language (XML) 1.0 second edition W3C recommendation. Technical Report REC-xml-20001006 WWW Consortium, October 2000.
- [15] J. Zobel, A. Moffat and R. Sacks-Davis. An efficient indexing technique for full text database systems, In *Proc. of VLDB*, Vancouver, Canada, 352–362, 1992.
- [16] K. Deschler and E. Rundensteiner. MASS: A multi-axis storage structure for large XML documents, In *Proc. of CIKM*, Louisiana, USA, 2003.

APPENDIX

The constraints with page boundary are:

$$B_Ln'_{low} \leq Ln' \leq B_Ln'_{high}, B_Rn'_{low} \leq Rn' \leq B_Rn'_{high}. \quad (1)$$

The following equations are obtained from equation 1.

$$B_Rn'_{low} + B_Ln'_{low} \leq Rn' + Ln' \leq B_Rn'_{high} + B_Ln'_{high},$$

$$B_Rn'_{low} - B_Ln'_{high} \leq Rn' - Ln' \leq B_Rn'_{high} - B_Ln'_{low}. \quad (2)$$

Condition for descendant:

- Combined with query:

$$Ln' > Q_{Ln'}, Rn' > Q_{Rn'} \equiv Ln' > Q_{Ln'}, Rn' - Ln' > Q_{Rn'} - Q_{Ln'}. \quad (3)$$

- Combined with bounding box (by Equation 1, 2, and 3):

$$-(B_Rn'_{high} - B_Ln'_{low} > Q_{Rn'} - Q_{Ln'}) \& (B_Ln'_{high} > Q_{Ln'}).$$

$$-Ln' > Q_{Ln'}, Rn' - Q_{Rn'} + Q_{Ln'} > Ln' \Rightarrow Q_{Rn'} < Rn'.$$

- The result condition is:

$$(B_Rn'_{high} - B_Ln'_{low} > Q_{Rn'} - Q_{Ln'}) \& (B_Ln'_{high} > Q_{Ln'}) \& (B_Rn'_{high} > Q_{Rn'}).$$

Condition for ancestor:

- Combined with query:

$$Ln' < Q_{Ln'}, Rn' < Q_{Rn'} \equiv Ln' > Q_{Ln'}, Rn' - Ln' < Q_{Rn'} - Q_{Ln'}. \quad (4)$$

- Combined with bounding box (by Equation 1, 2, and 4):

$$-(B_Rn'_{low} - B_Ln'_{high} < Q_{Rn'} - Q_{Ln'}) \& (B_Ln'_{low} < Q_{Ln'}).$$

$$-Ln' < Q_{Ln'}, Rn' - Q_{Rn'} + Q_{Ln'} < Ln' \Rightarrow Q_{Rn'} > Rn'.$$

- The result condition is:

$$(B_Rn'_{low} - B_Ln'_{high} < Q_{Rn'} - Q_{Ln'}) \& (B_Ln'_{low} < Q_{Ln'}) \& (B_Rn'_{low} < Q_{Rn'}).$$

Condition for preceding:

- Combined with query:

$$Ln' < Q_{Ln'}, Rn' > Q_{Rn'} \equiv Ln' < Q_{Ln'}, Rn' - Ln' > Q_{Rn'} - Q_{Ln'}. \quad (5)$$

- The result condition after combined with bounding box (by Equation 1, 2, and 4) is:

$$(B_Rn'_{high} - B_Ln'_{low} > Q_{Rn'} - Q_{Ln'}) \& (B_Ln'_{low} < Q_{Ln'}).$$

Condition for following:

- Combined with query:

$$Ln' > Q_{Ln'}, Rn' < Q_{Rn'} \equiv Ln' > Q_{Ln'}, Rn' - Ln' < Q_{Rn'} - Q_{Ln'}. \quad (6)$$

- The result condition after combined with bounding box (by Equation 1, 2, and 6) is:

$$(B_Rn'_{low} - B_Ln'_{high} < Q_{Rn'} - Q_{Ln'}) \& (B_Ln'_{high} > Q_{Ln'}).$$