An XML Subtree Segmentation Method Based on Syntactic Segmentation Rate

Wenxin Liang
Japan Science and Technology Agency
Tokyo Institute of Technology
wxliang@de.cs.titech.ac.jp

Xiangyong Ouyang
Tokyo Institute of Technology
ouyang@de.cs.titech.ac.jp

Haruo Yokota
Tokyo Institute of Technology
yokota@cs.titech.ac.jp

Abstract

In this paper, we propose an effective method for segmenting large XML documents into independent meaningful subtrees based on two syntactic segmentation rates: vertical segmentation rate and horizontal segmentation rate. In the proposed method, we use DO-VLEI code to calculate the required parameters for the subtree segmentation. We conduct experiments to observe the effectiveness of the proposed subtree segmentation method using real bibliography XML documents stored in RDBs. We apply our previously proposed subtree matching algorithm SLAX to match the segmented subtrees and evaluate how the matching threshold impacts the precision and recall of subtree matching. Besides, we also integrate the matched subtrees determined by SLAX by our previously proposed subtree integration algorithm. The experimental results indicate that the proposed subtree segmentation method is effective for segmenting XML documents into independent meaningful subtrees and our previously proposed subtree matching algorithm achieves reasonable matching precision and recall using the segmented subtrees.

1 Introduction

XML has rapidly become the *de facto* standard for representing and exchanging data on the Web, because it is portable for representing different types of data from multiple sources. Recently, more and more data represented by XML are disseminated and exchanged on the Internet. However, XML documents from different sources may contain exactly or nearly the same information but may be different on structures.

Example 1 *Figure 1 shows two example XML document trees with different DTDs. Although the two document trees are different on structures, they represent very similar information. Besides, each document has some information what the other does not do. For example,* `volume` *in Figure 1 (a); and* `pages` *in Figure 1 (b).*

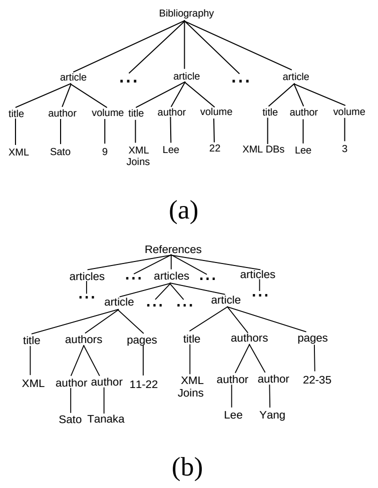


Figure 1. Example XML documents

Along with the increasing size of XML documents, parallelly storing and processing large XML documents over advanced distributed storage systems such as Autonomous Disks [16] becomes important. To store large XML documents into different nodes of a distributed storage system, we need to segment them into small fragments. XML Query results are often represented by meaningful subtrees that contain the query keywords. Therefore, it is important for segmenting the documents into meaningful subtrees and treating each subtree as a minimum unit for data placement and migration in the distributed storage system.

In our previous work [7], we have proposed a method for segmenting XML documents into independent meaningful subtrees using DOM Parser. However, for large-scale XML documents stored in RDBs, an imprecise segmentation problem may occur by using our previous method. In order to solve this problem, in this paper we propose a new subtree segmentation method for segmenting XML documents stored in RDBs into independent meaningful subtrees based on two syntactic segmentation rates: vertical segmentation rate and horizontal segmentation rate. In the proposed subtree segmentation method, we use the DO-VLEI code to calculate the required parameters for the subtree segmentation. We apply our previously proposed subtree matching algorithm SLAX to match the segmented subtrees and evaluate how the matching threshold impacts the precision and recall of subtree matching. Besides, we also apply our previously proposed subtree integration algorithm for integrating the matched subtrees selected by SLAX.

The main contributions of this paper are as follows:

1. We propose a new method for segmenting XML documents into independent meaningful subtrees based on the vertical and horizontal segmentation rates using our previously proposed DO-VLEI code. We experiment with real bibliography XML documents stored in RDBs to compare the effectiveness of subtree segmentation using the proposed method with that using the original one. The experimental results show that the proposed method is effective for segmenting XML documents into meaningful subtrees.

2. We apply our previously proposed leaf-clustering based XML join algorithm *SLAX* to match the segmented subtrees and integrate them at each hit subtree. We conduct experiments using SIGMOD Record and DBLP XML documents to observe how the matching threshold impacts the precision and recall of the subtree matching. Our experimental results indicate that *SLAX* is effective for matching the segmented subtrees. And we derive from the results that the matching threshold around 0.4 achieves the best performance for the bibliography XML documents used in our experiments.

3. We also perform experiments to apply our previous proposed subtree integration algorithm to integrate the matched subtree determined by *SLAX*. The experimental results show that our previously proposed subtree integration algorithm is effective and applicable for the XML documents segmented by the proposed method.

The remainder of this paper is organized as follows. Section 2 briefly describes related work and our previous work. In Section 3, we discuss the imprecise segmentation problem of our previously proposed subtree segmentation method and propose the new method. Section 4 describes the XML data preprocessing: parsing XML data into relational tables, labeling them by DO-VLEI codes and calculating the required parameters for subtree segmentation. Section 5 briefly describes the subtree matching based on the clustered leaves of segmented subtrees and the subtree integration method based on the insertion operation. In Section 6, we conduct experiments and show the experimental results. Finally, Section 7 concludes this paper and outlines our future work.

2 Related and Previous Work

Since XML is rapidly becoming a popular data format on the Internet, the size of the XML documents becomes larger and larger. This has led to researches focusing on how to store and handle XML into *Relational Databases* (RDBs). Recently, many researches are focusing on storing and querying XML data into RDBs [3, 4, 13, 14]. In this paper, we treat the XML documents stored in RDBs as the application object. We simply parse the original XML documents into three relational tables: element table, attribute table and text table.

When the XML documents are stored into RDBs, the structural information such as containment relationships (child-parent and ancestor-descendant relationships) of the nodes will be lost. An effective labeling method are helpful for reconstructing and detecting the containment relationships between nodes of XML documents stored in RDBs. Many effective labeling methods have been presented [2, 6, 10, 11]. In this paper, we use the DO-VLEI code [6,10] to calculate the required parameters for the subtree segmentation.

It is necessary to segment large XML documents into small fragments for the data placement and migration over distributed storage systems. Qin et al. [12] used graph partition algorithm to divide large XML data into small parts. Yu et al. [17] presented a data placement strategy based on path instances. Kido et al. [5] proposed a segmentation method based on DataGuide. However, these work do not address the problem of segmenting XML data into independent meaningful subtrees.

In our previous work, we have proposed *SLAX* [8] for matching the segmented subtrees based on the *Subtree Similarity Degree (SSD)* which is defined as follows:

Definition 1 (Subtree Similarity Degree (SSD)) *For a base subtree t_{bi} and a target one t_{tj}, the subtree similarity degree between them, $SSD(t_{bi}, t_{tj})$ is defined by Equation (1) as the percentage of the number of matched leaf nodes (the pair of leaf nodes that has the same PCDATA value) out of the number of leaf nodes in the base subtree t_{bi}, where N and N_{bi} denote the number of matched leaf nodes and the number of leaf nodes in the base subtree t_{bi}.*

$$SSD(t_{bi}, t_{tj}) = \frac{N}{N_{bi}} \times 100\ (\%) \tag{1}$$

In the ith join loop, the *matched subtree* and the *hit subtree* for the base subtree t_{bi} are defined as follows:

Definition 2 (Matched Subtree) *The matched subtree $t_m[i]$ for the base subtree t_{bi} is defined as the pair of* *subtrees that has the maximum subtree similarity degree in the join loop; that is, the subtree similarity degree between t_{bi} and $t_m[i]$, $SSD_m[i]$ can be calculated as follows, where k_t is number of subtrees in the target document tree.*

$$SSD_m[i] = Max_{j=1}^{k_t}(SSD(t_{bi}, t_{tj})) \tag{2}$$

Definition 3 (Hit Subtree) *In the ith join loop, the matched subtree $t_m[i]$ is a hit subtree $t_h[i]$, iff $SSD_m[i] \geq \mathcal{T}$ ($0 < \mathcal{T} \leq 1$), where $\mathcal{T}$ is the user defined threshold.*

For the base subtree t_{bi} and its hit subtree $t_h[i]$, the integration of them can be simply implemented by inserting the branches of unmatched leaf nodes in the hit subtree $t_h[i]$ into the base one t_{bi} [9]. Assume the number of leaf nodes in the base subtree t_b and the target one t_t is N_b and N_t, and the number of matched leaves in t_t is N_m, the total number of leaf nodes N in the integrated subtree and the number of insertions N_i can be calculated by the following two equations.

$$N = N_b + N_t - N_m \tag{3}$$

$$N_i = N - N_b = N_t - N_m \tag{4}$$

3 Subtree Segmentation Method

3.1 Basic Definition

In [7], we have presented a method for segmenting XML documents into subtrees representing independent meaningful items. The basic definitions for the subtree segmentation method are listed as follows:

Definition 4 (Candidate Element) *An element is a candidate element, if it has at least 2 children, or the distance to its furthest descendant is at least 3.*

Definition 5 (Link Branch) *A branch between two candidate elements is a link branch.*

Definition 6 (Segmentation Path) *A segmentation path P is a top-down path from the top candidate element to the bottom one via link branches.*

Definition 7 (Weighting Factor) *For a candidate element $E(n, d)$, let n denote the number of link branches below it, and d denote the distance to its furthest descendant. The weighting factor W is defined as follows:*

$$W = n \times d^{\phi} \qquad (0 < \phi \leq 1) \tag{5}$$

where ϕ is an adjustable constant.

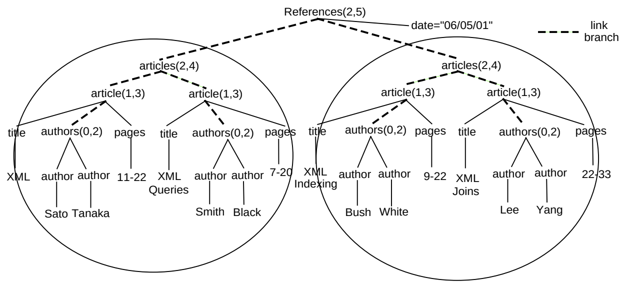


Figure 2. Example of imprecise segmentation

3.2 Imprecise Segmentation Problem

In our previous subtree segmentation method [7], the subtree segmentation was based on the maximum weighting factor in each segmentation path, which was effective for small XML documents with simple structures. However, for large XML documents stored in RDBs, an imprecise segmentation problem might occur; that is, the segmented subtrees sometimes do not correctly represent independent meaningful items.

Example 2 *Figure 2 shows an example of imprecise segmentation using our previously proposed subtree segmentation method based on the maximum weighting factor. Because the element* `Reference(2,5)` *gets the maximum weighting factor, the document is segmented into two subtrees as circled in Figure 2. However, here we can see that the two segmented subtrees do not represent independent meaningful items, because each of the them includes two independent articles.*

3.3 Proposed Segmentation Method

In order to solve the imprecise segmentation problem and hence improve the precision of subtree segmentation for XML documents in RDBs, we define the *vertical segmentation rate* and *horizontal segmentation rate* as follows.

Definition 8 (Vertical Segmentation Rate) *For an XML document tree T, assume M subtrees are segmented out of total N segmentation paths, the vertical segmentation rate $R_v(T)$ is defined as the following equation.*

$$R_v(T) = \frac{M}{N} \times 100\ (\%) \tag{6}$$

Definition 9 (Horizontal Segmentation Rate) *For an XML document tree T, the horizontal segmentation rate $R_h(T)$ is defined as percentage of the number of segmentation spots (n_s) out of all their siblings (n).*

```
Input: XML document tree $T$
Output: Segmented subtrees

begin{
Let $M$ be the number of segmentation path, $N$ be the number of elements in the segmentation path, and $\tau$ be the segmentation threshold.
    $max = 0$;
    for ($depth = 1$ to $N$){
        for ($i = 1$ to $M$){
            Segment subtrees into array $segSub$;
        }
        Calculate $R_v(T)$;
        Calculate $R_h(T)$;
        Calculate $R(T)$;
        if ($R(T) \geq max$){
            $max = R(T)$;
            $result = segSub$;
        }
    }
    Return ($result$);
}end
```

Figure 3. Subtree segmentation algorithm

$$R_h(T) = \frac{n_s}{n} \times 100\ (\%) \tag{7}$$

Then, the overall segmentation rate, $R(T)$ can be defined as follows.

Definition 10 (Segmentation Rate) *For an XML document tree T, the segmentation rate can be determined by the following equation, where θ is an adjustable constant*[1].

$$R(T) = R_v(T) \times R_h(T)^{\theta} \qquad (0 < \theta \leq 1) \tag{8}$$

The larger the segmentation rate $R(T)$ is, the more independent the meaningful items represented by the segmented subtrees are. Therefore, for each segmentation path, we can segment the document at all the candidate elements from the top to the bottom in the segmentation path. Then, we can calculate both the vertical and horizontal segmentation rates at each segmentation level. Finally, the best subtree segmentation can be determined by the segmentation that achieves the maximum segmentation rate. The details of the subtree segmentation algorithm based on the segmentation rate is illustrated in Figure 3

Example 3 *Figure 4 shows an example of subtree segmentation by using the proposed segmentation method. There are four segmentation paths in the document. Firstly, the subtree will be segmented at the two candidate elements* `articles`. *In this case, the vertical segmentation rate $R_v(T) = \frac{2}{4} \times 100\% = 50\%$, horizontal segmentation rate $R_h(T) = \frac{2}{2} \times 100\% = 100\%$, and hence the segmentation rate $R(T) = 50\% \times 100\% = 50\%$. For the second segmentation at the four candidate elements* `article`*,*

[1]For the sake of simplicity, we set $\theta = 1$ in this paper.

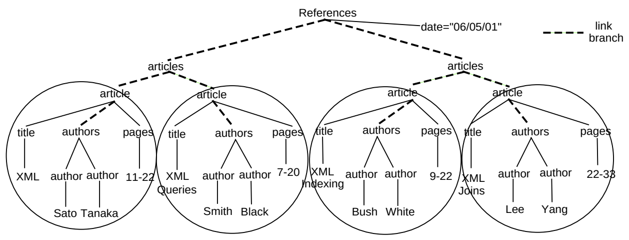


Figure 4. Example of subtree segmentation

$R_v(T) = \frac{4}{4} \times 100\% = 100\%$, $R_h(T) = \frac{4}{4} \times 100\% = 100\%$ and $R(T) = 100\% \times 100\% = 100\%$. For the third segmentation at the four candidate elements `authors`*, $R_v(T) = \frac{4}{4} \times 100\% = 100\%$, $R_h(T) = \frac{4}{12} \times 100\% = 33.3\%$ and $R(T) = 100\% \times 33.3\% = 33.3\%$. Therefore, the four subtrees as circled in Figure 4, which are segmented by the second segmentation that gets the maximum $R(T)$, will be output as the final results. From the segmentation results, we can see that each segmented subtree represents the complete information of an independent article.*

4 Data Preprocessing

In this paper, we take XML documents stored in RDBs as the application object. In store and handle XML documents stored in RDBs, we perform the following data preprocessing: parsing XML documents into relational tables, labeling them by Dewey Order labeling methods and calculating the required parameters for the subtree segmentation using the Dewey Order labels. The schema of each relational table used in this paper is shown in Table 1.

4.1 Parsing XML Data into Relational Table

The original XML document is parsed into three relational tables: element table, text table and attribute table. In the parsing phase, preorder numbers are assigned to the element and text nodes as their IDs, and the element name, text value, attribute name and value are stored into corresponding attributes of each relational table.

4.2 DO-VLEI Code

In [6], we have proposed the *VLEI (Variable Length Endless Insertable)* code which is defined as follows.

Definition 11 (VLEI Code) *A bit sequence $v = 1 \cdot \{0|1\}^*$ is a VLEI code, if the following condition is satisfied.*

$$v \cdot 0 \cdot \{0|1\}^* < v < v \cdot 1 \cdot \{0|1\}^*$$

Table 1. Schema of each relational table

Element Table

Attribute	Type of value	Description
ID	INTEGER	Preorder ID of element node
ParentID	INTEGER	ID of its parent node
Label	STRING	Dewey Order label
CE	BOOLEAN	1 for candidate element, 0 for non-candidate element
ChildNum	INTEGER	Number of children
CECNum	INTEGER	Number of candidate elements among its children
Depth	INTEGER	Depth of element node
Dist	INTEGER	Distance to its furthest descendant
Name	STRING	Element name

Text Table

Attribute	Type of value	Description
ID	INTEGER	Preorder ID of text node
ParentID	INTEGER	ID of its parent node
Label	STRING	Dewey Order label
Value	STRING	PCDATA value

Attribute Table

Attribute	Type of value	Description
ElementID	INTEGER	ID of the element it belongs to
Name	STRING	Attribute name
Value	STRING	Attribute value

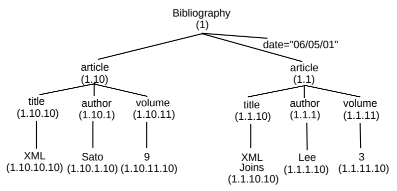


Figure 5. Example of labeling by using DO-VLEI code

For example, $10 < 1 < 11$ and $100 < 10 < 101 < 1 < 110 < 11 < 111$. In [10], we have proposed the DO-VLEI (Dewey-Order VLEI) code, as defined as follows.

Definition 12 (DO-VLEI Code)
1. The DO-VLEI code of the root node, $C_{root} = 1$.
2. The DO-VLEI code of a non-root node, $C = C_{parent} +$ "." $+ C_{child}$, where C_{parent} denotes the DO-VLEI code of its parent and C_{child} denotes the VLEI code satisfying the order of sibling.

An example XML document tree labeled by DO-VLEI code is shown in Figure 5.

4.3 Calculating Required Parameter for Segmentation

After parsing and labeling the XML document in RDBs, we have already had the information of *ID*, *ParentID*, *Label* and *Depth* for the element table. The parameters required for segmentation: *ChildNum*, *Dist*, *CE* and *CECNum* can be calculated by the SQL shown in Figure 6 using the Dewey Order labels.

```
// Number of children
SELECT COUNT(*) AS ChildNum FROM element, text
WHERE trim(start '.' from substring ('child.Label' from length('parent.Label'))) not
LIKE '%.%'

// Distance to its furthest descendant
SELECT MAX(parent.Depth-child.Depth) AS Dist FROM element, text
WHERE child.Label LIKE parent.Label

// Whether it is a candidate element or not
UPDATE element SET CE = 1
WHERE ChildNum > 1 OR Dist > 2

// Number of candidate elements among its children
SELECT COUNT(*) AS CECNum FROM element
WHERE trim(start '.' from substring ('child.Label' from length('parent.Label'))) not
LIKE '%.%'
AND child.CE = 1
```

Figure 6. SQL for calculating required parameters for segmentation

Example 4 *Table 2, Table 3 and Table 4 show the created element table, text table and attribute table, respectively for the XML document in Figure 4.*

5 Join and Integration

After the data preprocessing, the subtree segmentation can be implemented by the proposed subtree segmentation method based on the segmentation rate, and then the join process will be executed based on the clustered leaves

Table 2. Element table for the XML document in Figure 4

ID	ParentID	Label	CE	CECNum	Depth	Name
1	1	1	1	2	6	Reference
2	1	1.1	1	2	5	articles
3	2	1.1.1	1	1	3	article
4	3	1.1.1.1	0	0	1	title
6	3	1.1.1.2	1	0	2	authors
7	6	1.1.1.2.1	0	0	1	author
9	6	1.1.1.2.2	0	0	1	author
11	3	1.1.1.3	0	0	1	pages
13	2	1.1.2	1	1	3	article
14	13	1.1.2.1	0	0	1	title
16	13	1.1.2.2	1	0	2	authors
17	16	1.1.2.2.1	0	0	1	author
19	16	1.1.2.2.2	0	0	1	author
21	13	1.1.2.3	0	0	1	pages
23	1	1.2	1	2	5	articles
24	23	1.2.1	1	1	3	article
25	24	1.2.1.1	0	0	1	title
27	24	1.2.1.2	1	0	2	authors
28	27	1.2.1.2.1	0	0	1	author
30	27	1.2.1.2.2	0	0	1	author
32	24	1.2.1.3	0	0	1	pages
34	23	1.2.2	1	1	3	article
35	34	1.2.2.1	0	0	1	title
37	34	1.2.2.2	1	0	2	authors
38	37	1.2.2.2.1	0	0	1	author
40	37	1.2.2.2.2	0	0	1	author
42	34	1.2.2.3	0	0	1	pages

of each segmented subtree using SSD[2]. Given a matching threshold $\mathcal{T}$, the integration of the base subtree and the hit subtree can be executed by inserting the branch of unmatched leaves in the hit subtree into the base subtree[3].

Example 5 *For the base subtree t_b, and the target one t_t shown in Figure 7 (a) and (b), let the matching threshold $\mathcal{T} = 0.5$. Because $SSD(t_b, t_t) = 66.7\% > \mathcal{T}$, the t_t is the hit subtree for t_b and should be integrated with t_b. Figure 7 (c) shows the result of integration. We can also learn that the insertion operations does not cause any relabeling of any node by using the DO-VLEI code.*

6 Experiment and Evaluation

6.1 Experiment Setup

The experiments have been done under the environment shown in Table 5. In our experiments, we use the XML document of SIGMOD Record [1], named sigmod.xml (482KB, about 20,000 nodes) and we divide DBLP.xml [15] into 955 fragment documents, named dblp1∼955.xml. The size of each fragment is 300KB, about 15,000 nodes.

[2]The details of the leaf-clustering based join are available in [7, 8]

[3]The details of the subtree-based integration method are available in [9]

Table 3. Text table for the XML document in Figure 4

ID	ParentID	Label	Value
5	4	1.1.1.1.1	XML
8	7	1.1.1.2.1.1	Sato
10	9	1.1.1.2.2.1	Tanaka
12	11	1.1.1.3.1	11-22
15	14	1.1.2.1.1	XML Queries
18	17	1.1.2.2.1.1	Smith
20	19	1.1.2.2.2.1	Black
22	21	1.1.2.3.1	7-20
26	25	1.2.1.1.1	XML Indexing
29	28	1.2.1.2.1.1	Bush
31	30	1.2.1.2.1.2	White
33	32	1.2.1.3.1	9-22
36	35	1.2.2.1.1	XML Joins
39	38	1.2.2.2.1.1	Lee
41	40	1.2.2.2.1.2	Yang
43	42	1.2.2.3.1	22-33

Table 4. Attribute table for the XML document in Figure 4

ElementID	Name	Value
1	date	06/05/01

6.2 Evaluation of Subtree Segmentation

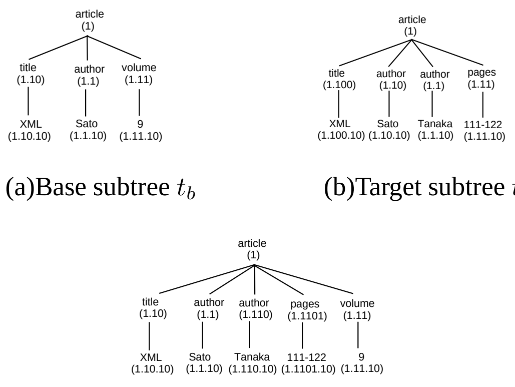
In order to observe the effectiveness of the subtree segmentation, we apply the new method and the original one to segment subtrees from sigmod.xml and dblp290∼292.xml. The experimental results of using the new method are shown in Table. 6. For the sigmod.xml, the horizontal segmentation rate using the previous proposed method R_h is 100%. However, the vertical segmentation rate R_v is only 4.45%. For the DBLP fragment files, the segmentation rates using the previous method are the same as those using the new method. Therefore, we consider that the proposed method is more effective for segmenting XML documents into independent meaningful subtrees than the previous proposed method.

6.3 Evaluation of Subtree Matching and Integration

In order to evaluate the effectiveness of subtree matching, we define *precision* and *recall* as follows.

Table 5. Experimental environment

CPU	AMD Opteron Processor 248×2
Memory	6.0 GB
Hard Disk	Seagate ST336607LC 37GB
OS	Linux 2.6.9
DBMS	PostgreSQL 8.1.3
Java	Sun JDK 1.5.0



(a)Base subtree t_b (b)Target subtree t_t

(c) Integrated subtree

Figure 7. Example subtree integration using DO-VLEI code

Table 6. Result of subtree segmentation using the proposed method

File name	N_p	N_s	$R_s(\%)$	$R_h(\%)$	$R(\%)$
sigmod.xml	1504	1504	100%	100%	100%
dblp290.xml	698	698	100%	100%	100%
dblp291.xml	569	569	100%	100%	100%
dblp292.xml	653	653	100%	100%	100%

N_p: Number of segmentation paths
N_s: Number of segmented subtrees, R_v: Vertical segmentation rate
T_{sd}: Horizontal segmentation rate, R: Segmentation rate

Definition 13 (Precision) *The precision of subtree matching ($\mathcal{P}$) is the percentage of the number of correctly selected hit subtrees ($\mathcal{N}_s$) out of the total number of hit subtrees ($\mathcal{N}_h$).*

$$\mathcal{P} = \frac{\mathcal{N}_s}{\mathcal{N}_h} \times 100\ (\%) \tag{9}$$

Definition 14 (Recall) *The recall of subtree matching ($\mathcal{P}$) is the percentage of the number of correctly selected hit subtrees ($\mathcal{N}_s$) out of the total number of correct answer($\mathcal{N}_c$).*

$$\mathcal{P} = \frac{\mathcal{N}_s}{\mathcal{N}_c} \times 100\ (\%) \tag{10}$$

Table 7. Result of subtree matching

$\mathcal{T}$	N_h	N_c	$\mathcal{P}$	$\mathcal{R}$
0.1	916	128	13.97%	90.14%
0.2	624	127	20.35%	89.44%
0.3	226	122	53.98%	85.92%
0.4	214	120	56.07%	84.51%
0.5	184	117	63.59%	82.39%
0.6	81	77	95.06%	54.23%
0.7	39	36	92.31%	25.35%
0.8	22	21	95.45%	14.79%
0.9	11	11	100%	7.75%

$\mathcal{T}$: Matching threshold, N_h: Number of hit subtrees
N_c: Number of correctly matched subtrees, $\mathcal{P}$: Precision, $\mathcal{R}$: Recall

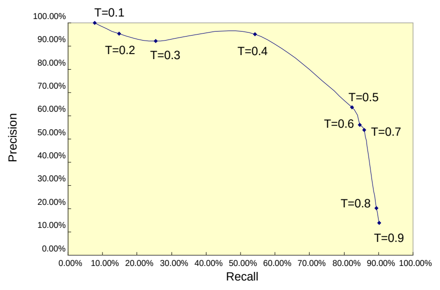


Figure 8. Precision and Recall of subtree matching

Table 8. Result of subtree integration

$\mathcal{T}$	N_h	N_i	$T_i(s)$
0.1	916	9994	331.48
0.2	624	6509	233.78
0.3	226	2149	79.11
0.4	214	1989	66.46
0.5	184	1649	59.89
0.6	81	723	25.69
0.7	39	343	12.39
0.8	22	175	6.50
0.9	11	77	2.49

$\mathcal{T}$: Matching threshold, N_h: Number of hit subtrees
N_i: Number of branch insertions, T_i: Integration time

We experiment with sigmod.xml and dblp290.xml to observe how the matching threshold impacts the precision and recall of the subtree matching. Table 7 shows the results of subtree matching. Figure 8 indicates that the precision of subtree matching using the proposed method is almost directly proportional to the matching threshold, while the recall is nearly inversely proportional to the matching threshold. We can also learn from the results that the matching threshold around 0.4 achieves the best performance for the documents used in our experiments.

We also integrate the matched subtrees based on the insertion of the unmatched leaves. Table 8 shows the results of subtree integration. The integration of sigmod.xml and DBLP290.xml requires 2623.1 insertions on the average. Figure 9 indicates that the integration time is proportional to the number of insertions.

7 Conclusion and Future Work

In this paper, we have proposed a new method for segmenting XML documents into independent meaningful subtrees based on two syntactic segmentation rates: vertical

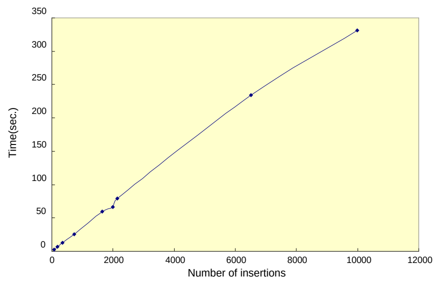


Figure 9. Time for subtree integration

segmentation rate and horizontal segmentation rate. In the proposed subtree segmentation method, we use our previously proposed the DO-VLEI code to calculate the required parameters for the subtree segmentation. We have performed experiments with real bibliography XML documents stored in RDBs. The experimental results show that the proposed subtree segmentation method is effective for segmenting XML documents into independent meaningful subtrees.

We have also performed experiments to evaluate how the matching threshold impacts the precision and recall of the subtree matching by using our previously proposed subtree matching algorithm *SLAX*. The experimental results indicate that the precision is almost directly proportional to the matching threshold, while the recall is nearly inversely proportional to the matching threshold. We have learned from the results that our previously proposed subtree matching algorithm achieves reasonable matching precision and recall using the segmented subtrees. Besides, we have performed experiments to integrate the matched subtrees. The experimental results indicate that our previously proposed subtree integration algorithm is effective and applicable for XML documents segmented by our proposed method.

In the future, we plan to do further experiments with large-scale XML documents over distributed storage systems. In addition, path-based and semantics-based subtree matching are to be taken into consideration to improve the precision and recall of subtree matching.

Acknowledgments

This work was supported in part by the Grant-in-Aid for Scientific Research of MEXT Japan (grant number 16016232), by CREST of JST (Japan Science and Technology Agency), and by the TokyoTech 21COE Program “Framework for Systematization and Application of Large-Scale Knowledge Resources”.

References

[1] ACM SIGMOD Record in XML. Available at http://www.acm.org/sigmod/record/xml/.

[2] T. Amagasa, M. Yoshikawa, and S. Uemura. QRS: A Robust Numbering Scheme for XML Documents. In *Proc. of ICDE 2003*, pages 705–707, 2003.

[3] J. E. Funderburk, G. Kiernan, J. Shanmugasundaram, E. J. Shekita, and C. Wei. XTABLES: Bridging Relational Technology and XML. *IBM Systems Journal*, 41(4):616–641, 2002.

[4] H. V. Jagadish, S. Al-Khalifa, A. Chapman, L. V. S. Lakshmanan, A. Nierman, S. Paparizos, J. M. Patel, D. Srivastava, N. Wiwatwattana, Y. Wu, and C. Yu. TIMBER: A Native XML Database. *VLDB Journal*, 11(4):274–291, 2002.

[5] K. Kido, T. Amagasa, and H. Kitagawa. An Improved Parallel Processing of XML Data Using PC Clusters and Performance Evaluation (in Japanese). In *Proc. of DEWS (The 18th IEICE Data Engineering Workshops)*, 2007.

[6] K. Kobayashi, W. Liang, D. Kobayashi, A. Watanabe, and H. Yokota. VLEI code: An Efficient Labeling Method for Handling XML Documents in an RDB. In *Proc. of ICDE 2005*, pages 386–387, Tokyo, Japan, 2005.

[7] W. Liang and H. Yokota. *LAX*: An Efficient Approximate XML Join Based on Clustered Leaf Nodes for XML Data Integration. In *Proc. of BNCOD 2005*, pages 82–97, Sunderland, UK, 2005.

[8] W. Liang and H. Yokota. *SLAX*: An Improved Leaf-Clustering Based Aproximate XML Join Algorithm for Integrating XML Data at Subtree Classes. *IPSJ Transactions on Databases*, 47(SIG8(TOD30)):47–57, June 2006.

[9] W. Liang and H. Yokota. Subtree-based XML Data Integration Using Leaf-Clustering Based Aproximate XML Join Algorithms. *DBSJ Letters*, 4(4):21–24, March 2006.

[10] K. Nagara, K. Kobayashi, D. Kobayashi, and H. Yokota. Evaluation of XML Labeling Methods Using VLEI Code. In *Proc. of IEICE DEWS2005 (in Japanese)*, 2005.

[11] P. E. O’Neil, E. J. O’Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury. ORDPATHs: Insert-Friendly XML Node Labels. In *Proc. of ACM SIGMOD Conference 2004*, pages 903–908, 2004.

[12] J. Qin, S. Yang, and W. Dou. Parallel Storing and Querying XML Documents Using Relational DBMS. In *Proc. of APPT*, pages 629–633, 2003.

[13] T. Shimura, M. Yoshikawa, and S. Uemura. Storage and Retrieval of XML Documents Using Object-Relational Databases. In *Proc. of DEXA’99*, pages 206–217, 1999.

[14] I. Tatarinov, S. Viglas, K. S. Beyer, J. Shanmugasundaram, E. J. Shekita, and C. Zhang. Storing and Querying Ordered XML Using a Relational Database System. In *Proc. of ACM SIGMOD Conference 2002*, pages 204–215, 2002.

[15] XML Version of DBLP. Available at http://dblp.uni-trier.de/xml/.

[16] H. Yokota. Autonomous Disks for Advanced Database Applications. In *Proc. of DANTE*, pages 435–442, 1999.

[17] Y. Yu, G. Wang, G. Wu, J. Hu, and N. Tang. Data Placement and Query Processing Based on RPE Parallelisms. In *Proc. of COMPASC*, 2003.