論文 / 著書情報
Article / Book Information

Title	LAX: An Efficient Approximate XML Join Based on Clustered Leaf Nodes for XML Data Integration
Author	Wenxin Liang, Haruo Yokota
Journal/Book name	22nd British National Confernece on Databases (BNCOD22), Lecture Notes in Computer Science, Vol. LNCS, No. 3567, pp. 82-97
発行日 / Issue date	2005, 7
DOI	10.1007/11511854_7
権利情報 / Copyright	The original publication is available at www.springerlink.com.
Note	このファイルは著者（最終）版です。 This file is author (final) version.

LAX: An Efficient Approximate XML Join Based on Clustered Leaf Nodes for XML Data Integration

Wenxin Liang[1] and Haruo Yokota[2]

[1,2] Department of Computer Science
[2] Global Scientific Information and Computer Center
Tokyo Institute of Technology
2-12-1 Oh-okayama, Meguro-ku, Tokyo 152-8552, Japan
[1] wxliang@de.cs.titech.ac.jp,[2] yokota@cs.titech.ac.jp

Abstract. Recently, more and more data are published and exchanged by XML on the Internet. However, different XML data sources might contain the same data but have different structures. Therefore, it requires an efficient method to integrate such XML data sources so that more complete and useful information can be conveniently accessed and acquired by users.
The tree edit distance is regarded as an effective metric for evaluating the structural similarity in XML documents. However, its computational cost is extremely expensive and the traditional wisdom in join algorithms cannot be applied easily. In this paper, we propose *LAX* (**L**eaf-clustering based **A**pproximate **X**ML join algorithm), in which the two XML document trees are clustered into subtrees representing independent items and the similarity between them is determined by calculating the similarity degree based on the leaf nodes of each pair of subtrees. We also propose an effective algorithm for clustering the XML document for *LAX*. We show that it is easily to apply the traditional wisdom in join algorithms to *LAX* and the join result contains complete information of the two documents. We then do experiments to compare *LAX* with the tree edit distance and evaluate its performance using both synthetic and real data sets. Our experimental results show that *LAX* is more effcient in performance and more effective for measuring the approximate similarity between XML documents than the tree edit distance.

1 Introduction

The eXtensible Markup Language (XML) is increasingly recognized as the *de facto* standard for representing and exchanging data on the Internet, because it can represent different kinds of data from multiple sources. Recently, more and more data, especially bioinformatics and bibliography data such as MAGE [12] , DBLP [19] and ACM SIGMOD Record [1], are published by XML on the Internet. However, the same data might have different structures and contents in different XML data sources. Thus, it is paramount to integrate such data

sources so that users can conveniently access and acquire more complete and useful information. However, the integration of XML data from multiple sources is not an easy task, because XML documents from different sources might have different structures even though they represent the same information.

The Document Type Descriptor (DTD) is regarded as a useful tool to obtain the structural information from XML documents [2, 7]. However, even if XML data sources have the same DTDs, they may not have identical tree structures due to the repeating and optional elements and attributes [8, 9, 14]. Therefore, an effective approximate XML join algorithm, which is able to measure the similarity between XML documents without considering DTDs, becomes of great importance to solve the problem of integrating multiple XML data sources.

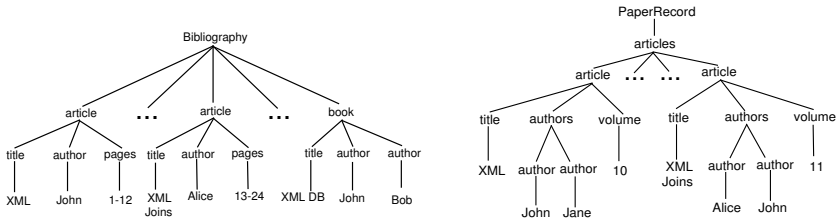
Example 1. Fig. 1 shows an example of two XML documents with different DTDs[1]. Although these two documents are structurally different, they represent the similar data. Moreover, in terms of the related data items of the two documents (i.e. "article" here in this example), one document may have some information what the other does not have. For instance, `pages` in (a); and `volume` in (b).

The tree edit distance is currently verified as an effective metric for measuring the structural similarity in XML documents [8, 14]. However, the computational cost of the tree edit distance is extremely high. Besides, the traditional wisdom in join algorithms (sort merge, hash joins etc) is of difficulty to be applied to this area [8].

The main contributions of this paper are as follows:

- Computing tree edit distance between two XML documents is a very expensive operation. To solve this problem, we propose an efficient join algorithm *LAX* (**L**eaf-clustering based **A**pproximate **X**ML join algorithm), in which the two XML document trees are clustered into subtrees representing independent items and the similarity between them is determined by calculating the similarity degree based on the leaf nodes of each pair of subtrees. We also present an algorithm for effectively clustering the XML document into independent items for *LAX*.
- The traditional wisdom in join algorithms can be easily applied to *LAX*, because the join operation of *LAX* is the same as traditional joins in RDBs. Besides, the integration of the hit subtrees can make the join results contain complete information from the two XML documents been joined.
- We do experiments to evaluate *LAX* using both synthetic and real data sets, investigating how the number of leaf nodes and the number of clustered subtrees affect the performance of *LAX*. We also do experiments to compare *LAX* with the tree edit distance. The experimental results show that our

[1] Associating to our interested real bibliography XML data, we make the DTD in Fig. 1(a) similar to that of DBLP, and the DTD in Fig. 1(b) similar to that of ACM SIGMOD Record.



(a) (b)

Fig. 1. Example XML document trees

algorithm is more efficient in performance and more effecitive for measuring the approximate similarity between XML documents than the tree edit distance.

The rest of this paper is arranged as follows: Section 2 briefly introduces the work related to the issues addressed in this paper. In Section 3, we briefly introduce the tree edit distance, and we provide basic definitions necessary for the proposed algorithm and state the problem considered in this paper. In Section 4, we propose and discuss *LAX*. In Section 5, we compare *LAX* with the tree edit distance evaluate its performance by experiments. In the end, Section 6 concludes the paper and outlines the future work.

2 Related Work

An XML document can be modeled as an ordered labeled tree [18]. Each element in the XML document corresponds to a node in the ordered labeled tree labeled with the element tag name. A lot of work has been done to solve the problem of measuring the edit distance between such trees [3, 4, 13, 16, 17, 21, 22]. A general definition of the distance between ordered labeled trees is presented by using the tree edit distance that is defined as the minimum cost edit operations (insertions, deletions and substitutions) required to transform one tree to another [22]. The tree edit distance is considered to be an effective metric for calculating the structural similarity in XML documents [8, 14]. However, the tree edit distance is a very expensive operation and the traditional wisdom in join algorithms (sort merge, hash joins etc) is not easy to be extended to this application field [8].

To avoid the expensive tree edit distance operation as much as possible, S. Guha, et al. [8] developed lower and upper bounds as inexpensive filters for the tree edit distance operation. However, when the upper bound is greater than the threshold distance τ and, at the same time, the lower bound is less than τ , the expensive tree edit distance still must be calculated.

Besides, XML and its schema languages do not provide any semantic information. A number of work related to XML schema matching and integration

has been studied by many researchers [5, 6, 11, 15, 20]. Generally, schema matching is an important and difficult problem for many database applications such as schema integration, data warehousing, and E-business [15]. From the XML data integration point of view, the problem of semantic heterogeneities is still a pervasive and paramount issue. However, many real XML documents contain repeating elements, `articlesTuple` in `SigmodRecord.xml` [1] for example. Taking such XML documents as the target, the approximate similarity degree between them can be effectively determined by computing the similarity degree of clustered subtrees (rooted at the repeating elements) even without considering the semantic heterogeneity. In Section 4, we will mention this problem associated with our algorithm.

3 Preliminaries

3.1 Tree Edit Distance

A well formed XML document can be parsed into an ordered labeled tree, in which the tree structure represents nesting of the elements and node labels records the contents of the elements by element tags, attribute names, attribute values and PCDATA values.

Definition 1 (XML Document Tree). *An XML document tree T is an ordered labeled tree parsed from an XML document.*

Let T_1 and T_2 be two XML document trees, the tree edit distance between them is defined as follows:

Definition 2 (Tree Edit Distance). *Given two XML document trees T_1 and T_2, the tree edit distance, TEDist(T_1,T_2), is defined as the minimum cost edit operations (insertions, deletions and substitutions) that transforms one tree to the other.*

Assume each node label is a symbol chosen from an alphabet Σ of size $|\Sigma|$. Let $\lambda \notin \Sigma$ denote the null symbol. An edit operation can be represented as $\gamma(a \rightarrow b)$. $\gamma(a \rightarrow b)$ is an *insert* operation if $a = \lambda$, a *delete* operation if $b = \lambda$, and a *substitute* operation if $a \neq \lambda$ and $b \neq \lambda$.

The tree edit distance $TEDist(T_1, T_2)$ can be figured out by a mapping M between the nodes of the two trees. Formal description of the mapping and algorithms for computing the tree edit distance are available in [22].

Given an XML document tree T, let $d(T)$ denote its depth. For two document trees T_b and T_t, and let t_b and t_t be any pair of subtree. Then the time complexity of the computation of the tree edit distance can be bounded by the following equation [22]:

$$O(\sum_{i=1}^{|T_1|}\sum_{j=1}^{|T_2|}|t_{1i}| \times |t_{2j}|) = O(\sum_{i=1}^{|T_1|}|t_{1i}| \times \sum_{j=1}^{|T_2|}|t_{2j}|) = O(|T_1| \times |T_2| \times d(T_1) \times d(T_2)) \quad (1)$$

For document trees of size $O(n)$, in the worst case, it is an $O(n^4)$ operation.

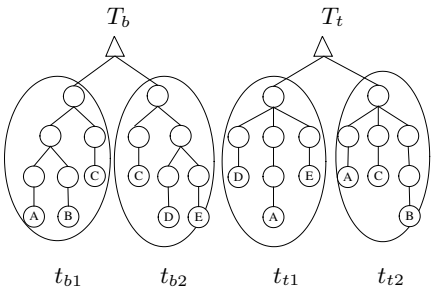


Fig. 2. Example clustering of XML document trees

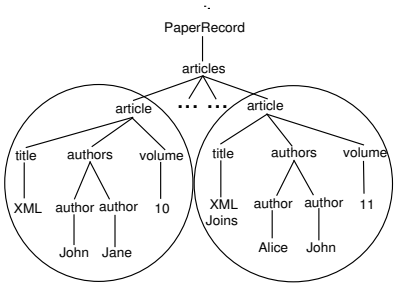


Fig. 3. Example of a well-clustered document

3.2 Basic Definitions for LAX

Notation. Let T_b and T_t be two XML document trees, where b denotes *base*, and t denotes *target*. Assume T_b and T_t are clustered into k_b and k_t sub-trees $t_{bi}(1 \leq i \leq k_b)$ and $t_{tj}(1 \leq j \leq k_t)$, as shown in Fig. 2, respectively.

Definition 3 (Subtree Similarity Degree). *For each pair of subtrees t_{bi} and t_{tj}, let t_{bi} be the base subtree, and t_{tj} be the target one. Let n_{bi} and n_{tj} represent the number of leaf nodes of t_{bi} and t_{tj}. If there are n pairs of leaf nodes of the two subtrees having the same PCDATA values, then the similarity degree of subtrees t_{bi} and t_{tj}, $S(t_{bi}, t_{tj})$ is defined as follows:*

$$S(t_{bi}, t_{tj}) = \frac{n}{n_{bi}} \times 100\ (\%) \tag{2}$$

Definition 4 (Matched Subtree). *In each join loop i, for the base subtree t_{bi} and each target subtree t_{tj} $(1 \leq j \leq k_t)$, the subtree similarity degree $S(t_{bi}, t_{tj})$ is computed one by one. The matched subtree T_{Mi} is defined as the pair of subtrees t_{bi} and t_{tj} that has the maximum subtree similarity degree in that join loop.*

Definition 5 (Tree Similarity Degree). *Let the base document tree T_b that has the less number of subtrees be the outer loop and the target one T_t be the inner loop. In each join loop i, let the similarity degree of each matched subtree be recorded into an array $S_M[i]$. The tree similarity degree $S(T_b, T_t)$ is defined as follows:*

$$S(T_b, T_t) = \frac{\sum_{i=1}^{k_b} S_M[i]}{k_b} \times 100\ (\%) \tag{3}$$

3.3 Problem Statement

Let S_b and S_t be two XML data sources. We are pursuing an algorithm to execute join operations, based on the leaf nodes of each pair of clustered subtrees of the XML documents, using similarity degree as a join predicate. The main problem addressed in this paper is formally defined as follows:

Problem 1 (Leaf-clustering based Approximate XML Joins). Given two XML data sources, S_b and S_t, a user defined threshold τ, and the tree similarity degree $S(T_b, T_t)$ accessing the distance between pairs of XML documents trees T_b and T_t parsed from two documents $d_b \in S_b$ and $d_t \in S_t$. The leaf-clustering based approximate join operation outputs all pairs of documents $(d_b, d_t) \in S_b \times S_t$, such that $S(T_b, T_t) \geq \tau$.

In the tree edit distance, for any two XML documents with different DTDs that have the same number of nodes, the tree edit distances of them do not change a lot when the PCDATA values of the leaf nodes change. However, in *LAX*, the change of the values of the leaf nodes might change the values of the tree similarity degrees in a large scale. Therefore, pairs of XML documents that have the same tree edit distance might have different tree similarity degrees. Besides, because in *LAX* the XML document is clustered into subtrees representing independent items, the matched subtrees that have large enough subtree similarity degrees still can be integrated even though the tree edit distance of the two whole documents exceeds the threshold.

4 LAX

4.1 Clustering

An XML document can be generally divided into many independent items by clustering it into subtrees at some specific element nodes. However, it is not easy to cluster an XML document tree into subtrees representing independent items. As a matter of fact, a well-clustered document requires that each clustered subtree meets the following conditions.

1. Each subtree represents only one independent item; that is, a subtree does not include any information of other items.
2. One independent item is clustered into one subtree; that is, one item does not have more than one corresponding subtrees.
3. Each subtree includes the information of an item as much as possible. In other words, the leaf nodes belonging to that item should be included in the subtree as much as possible.

Example 2. Fig. 3 shows an example of a well-clustered document. The document tree is clustered into two subtrees at the element nodes `article` so that each subtree represents complete information of an independent article.

In order to include more information of an independent item, an element is not supposed to be selected as the spot for clustering, if 1) it has only one child, and 2) the distance to its furthest child is less than 3. Before we treat of the algorithm for clustering XML document trees, we give the following definitions.

Definition 6 (Candidate Element). *An element is a candidate element, if it has at least 2 children, or the distance to its furthest child is at least 3.*

```
Algorithm ClusterXMLDoc(T) {
Input: XML document tree T
Output: Clustering spots for T
Let N be the number of top-down paths, and M[i] be the
number of candidate elements in the i-th path.
    for (i = 1 to N) {
        ClusteringSpot[i] = null;
        w_max[i]=0;
        for (j = 1 to M[i]) {
            w=n[j] × d[j]^φ;
            if ( w_max[i] < w) {
                w_max[i] = w;
                ClusteringSpot[i] = E(n[j], d[j]);
            }
        }
        return CluteringSpot[i];
    }
}
```

Fig. 4. Algorithm ClusterXMLDoc

Definition 7 (Link Branch). *A branch between two candidate elements is a link branch.*

Definition 8 (Top-down Path). *A top-down path is defined as a path from the top candidate element to the bottom one via link branches.*

Only one candidate element should be selected as the place for clustering in one top-down path. Generally, we consider a candidate element as a proper spot for clustering, if it has more link branches (i.e. there are more candidate elements among its children), and it is at higher level of the document tree (i.e. it is far from its furthest child). To effectively find the most appropriate spot for clustering, we define the weighting factor for evaluating each candidate element in a top-down path as follows.

Definition 9 (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 child. The weighting factor w is defined as follows:*

$$w = n \times d^{\phi} \qquad (0 < \phi \leq 1) \tag{4}$$

where ϕ is an adjustable constant[2]*.*

Then we define the `clustering spot` that indicates the place for clustering using the weighting factor w as follows.

[2] For the sake of simplicity, we set $\phi = 1$ for the examples in this paper. In fact, documents from different sources may require different ϕ to achieve better clusterings. In the real application, ϕ can be dynamically optimized by experiments.

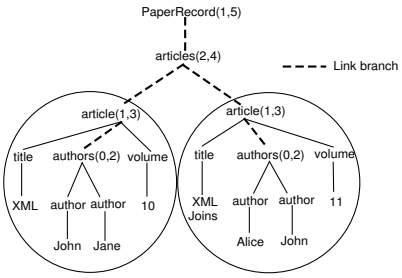
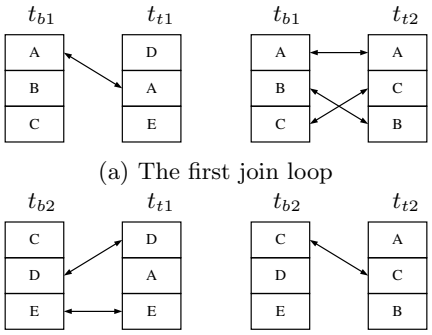


Fig. 5. Example of clustering using Algorithm *ClusterXMLDoc*



(a) The first join loop

(b) The second join loop

Fig. 6. Example of calculating subtree similarity degrees

Definition 10 (Clustering Spot). *In each top-down path of an XML document tree T, the clustering spot, indicating the place for clustering, is the candidate element $E(n, d)$ that has the maximum w in that top-down path. If two or more candidate elements have the same value of w in the same path, the one who has the maximum d is chosen as the clustering spot.*

In a top-down path, the subtree can be simply generated by deleting the link branch below the clustering spot; that is, the root of the subtree is the child element of the clustering spot in that top-down path. The algorithm for determining the clustering spots for an XML document tree is shown in Fig. 4 .

Example 3. Fig. 5 shows a simple example of clustering an XML document tree by Algorithm ClusterXMLDoc. There are two top-down paths in the document tree. In the left path, {`PaperRecord(1,5)`, `articles(2,4)`, `article(1,3)`, `authors(0,2)`}, the clustering spot is the candidate element `articles(2,4)` because of the maximum $w = 2 \times 4 = 8$. Similarly, the clustering spot in the right path is the same element, `articles(2,4)`. Therefore, the document tree can be clustered into the two circled subtrees shown in Fig. 5.

4.2 Join Algorithm

Let S_b and S_t be two XML data sources, and each $d_b \in S_b$ and $d_t \in S_t$ be parsed into XML document trees T_b and T_t. Assume T_b and T_t are clustered into k_b and k_t subtrees t_{bi} and t_{tj} by using Algorithm *ClusterXMLDoc*. Given a user-defined threshold τ, the **L**eaf-clustering based **A**pproximate **X**ML join algorithm (LAX) is illustrated by Fig. 7.

Example 4. Fig. 6 shows the join process by LAX for the two XML documents trees T_b and T_t in Fig. 2. Let T_b be the outer loop and T_t be the inner loop for the join operation. In the first join loop shown in Fig. 6 (a), the similarity

```
Algorithm LAX {
Input: XML data source S_b and S_t
Output: Pairs of XML documents (d_b, d_t)
    for each d_b ∈ S_b {
        Parse d_b into T_b;
        ClusterXMLDoc(T_b);
        for each d_t ∈ S_t {
            Parse d_t into T_t;
            ClusterXMLDoc(T_t);
            Sum = 0;
            for (i = 1 to k_b) {
                S_M[i] = 0;
                for (j = 1 to k_t) {
                    Calculate S(t_bi, t_tj);
                    S_M[i] = Max(S_M[i], S(t_bi, t_tj));
                }
                Sum = Sum + S_M[i];
            }
            if(Sum/k_b ≥ τ) {
                return (d_b, d_t);
            }
        }
    }
}
```

Fig. 7. Algorithm *LAX*

degrees of each pair of subtrees can be calculated as follows:

$$s(t_{b1}, t_{t1}) = \frac{1}{3} \times 100\% = 33.3\%$$

$$s(t_{b1}, t_{t2}) = \frac{3}{3} \times 100\% = 100\%$$

Then the similarity of the matched subtree,
$S_M[1] = Max\{S(t_{b1}, t_{t1}), S(t_{b1}, t_{t2})\} = 100\%$. In the same way, we have $S_M[2] = 66.7\%$ for the second join loop. Finally, the tree similarity degree $S(T_b, T_t)$ can be calculated by equation (3), i.e., $S(T_b, T_t) = \frac{S_M[1]+S_M[2]}{2} \times 100\% = \frac{1+0.667}{2} \times 100\% = 83.4\%$. If $S(T_b, T_t) \geq \tau$, the two documents should be output as the final result.

4.3 Discussion

Cost. Let two XML document trees T_b and T_t be clustered into k_b and k_t subtrees, respectively. For $i = 1$ to k_b, assume each subtree t_{bi} has α_i leaf nodes, and for $j = 1$ to k_t, each subtree t_{tj} has β_j leaf nodes. Then the total computational

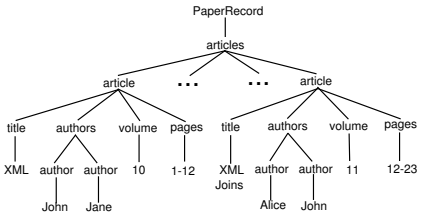


Fig. 8. Example of an output of LAX

cost of LAX can be figured out by the following equation:

$$C = \sum_{i=1}^{k_b} \sum_{j=1}^{k_t} \alpha_i \times \beta_j \tag{5}$$

If the sizes of the two XML document trees are both $O(n)$, in the worst case, LAX is an $O(n^2)$ operation.

Traditional Wisdom in Join Algorithms. The traditional wisdom in join algorithm can be easily applied to LAX, because the join operations based on the clustered leaf nodes in LAX are just the same as the traditional joins in an RDB. Therefore, LAX may achieve more efficiency by using traditional techniques for join algorithms. For example, the total cost of LAX using hash joins can be calculated as follows:

$$C_{HASH} = \sum_{i=1}^{k_b} \sum_{j=1}^{k_t} (C_{gen}(\alpha_i) + C_{hash}(\alpha_i + \beta_j) + C_{comp}(\beta_j)) \tag{6}$$

where, C_{gen} represents the cost of making entries for subtree t_{bi}; C_{hash} stands for the cost of using the hash function to the two subtrees; and C_{comp} means the cost of comparisons in the probe phase.

Output of LAX.

Definition 11 (Hit Subtree). *In the ith join loop, let the similarity degree of the matched subtree T_{Mi} be $S_M[i]$. Given a threshold $\mathcal{T}$ ($0 < \mathcal{T} \leq 1$), the matched subtree is a hit subtree, if $S_M[i] \geq \mathcal{T}$.*

Given two XML document trees T_1 and T_2, if the tree similarity degree of T_1 and T_2, $S(T_1, T_2) \geq \tau$, the two XML document trees can be integrated at each hit subtrees. Fig. 8 shows an example of the output XML document from joining the two XML documents in Fig. 1 using LAX, in which the whole information of the articles from the two documents been joined is included. Thus, users can conveniently acquire more complete and useful information of the articles by accessing the output document.

Table 1. Experimental Environment

CPU	Intel Pentium IV 2.80GHz
Memory	1.0 GB
OS	MS Windows XP Professional
Programming Environment	Sun JDK 1.4.2

Issues to be Considered. In our algorithm, we just compare the PCDATA values of the leaf nodes without considering their semantic similarities. The more precise join can be achieved by using techniques of semantic matching. Another issue is that in case the similarity degrees of one subtree in the outer loop and several subtrees in the inner loop happen to be the same, how to choose the right pair? In this case, one effective solution is to compare the common parents of the leaf nodes to decide which subtree is the right one. However, there still exists semantic problem when comparing the common parents.

5 Experimental Evaluation

In this section, we conduct experiments to observe the efficiency and effectiveness of our algorithm comparing with the tree edit distance. We also perform experiments to investigate how the number of leaf nodes and the number of clustered subtrees affect the performance of our algorithm.

5.1 Data Set Used

We used both real and synthetic data sets to perform our experiments. For a synthetic data set, we used IBM XML generator available through AlphaWorks [10]. The XML generator can randomly generate XML documents by inputting DTDs. In our experiments, we utilized SigmodRecord.dtd [1] to randomly generate XML documents of different sizes by changing the two parameters: `MaxLevels` and `MaxRepeats`. The size range of the generated XML documents was from 1 to 150 KB (about 0 to 5000 nodes).

For the real data set, we made use of the XML documents of `OrdinaryIssuePage`, `ProceedingsPage` and `SigmodRecord` from the XML version of ACM SIGMOD record [1], and the XML document of the DBLP database [19].

5.2 Experimental Environment

Our experiments were done under the environment shown in Table 1.

5.3 Comparing LAX with Tree Edit Distance

Efficiency. To evaluate the efficiency of our algorithm, we compared the time to computer the tree similarity degree for a pair of XML documents by our

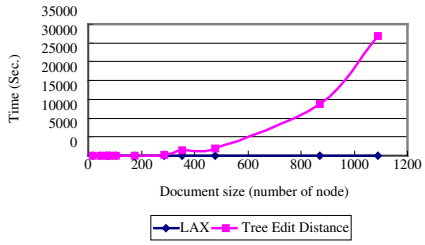


Fig. 9. Time for computing tree edit distance and tree similarity degree

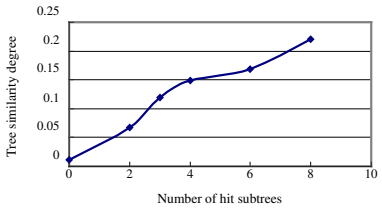


Fig. 10. Tree similarity degree increases proportionally to the number of hit subtrees

Table 2. The number of nodes and clustered subtrees included in each fragment

	sigmod.xml	dblp1.xml	dblp2.xml	dblp3.xml	dblp4.xml	dblp5.xml	dblp6.xml
No. of nodes	194	196	196	193	202	198	202
No. of subtrees	17	9	9	9	9	9	9

algorithm with that of tree edit distance using synthetic data sets. Because the tree edit distance is extremely time-consuming, in this paper we only used the pair of documents whose total number of nodes is less than 1200.

From Fig. 9, we observe that our algorithm is overwhelmingly faster comparing to the tree edit distance when the number of nodes is more than 500, corresponding with our analytical expectations. Therefore, we can consider that our algorithm is more efficient than the tree edit distance for measuring the similarity between XML documents.

Effectiveness. In our algorithm, the similarity degree is defined as the quantitative measurement for calculating the subtree similarity degree. The larger the similarity degree is, the higher the probability of the subtrees being the same is, even though the element nodes above the leaf nodes have different structures or values.

To verify the effectiveness of our algorithm for determining the similarity between XML documents, we calculated the tree similarity degrees using LAX and compared them with the tree edit distances of the same pairs of XML docu-

Table 3. Result of each pair of fragments

	Tree edit distance	Tree similarity degree	No. of hit subtrees
$sigmod.xml \times dblp1.xml$	216	0.149	4
$sigmod.xml \times dblp2.xml$	216	0.120	3
$sigmod.xml \times dblp3.xml$	210	0.067	2
$sigmod.xml \times dblp4.xml$	219	0.011	0
$sigmod.xml \times dblp5.xml$	216	0.220	8
$sigmod.xml \times dblp6.xml$	220	0.169	6

Table 4. Result of the matched subtrees of *sigmod.xml* × *dblp*6.*xml*

	$T_M[1]$	$T_M[2]^*$	$T_M[3]^*$	$T_M[4]$	$T_M[5]^*$	$T_M[6]^*$	$T_M[7]^*$	$T_M[8]$	$T_M[9]^*$
N_{sigmod}	25	21	21	23	23	21	23	23	21
N_{dblp}	12	10	10	12	12	10	12	14	10
S_M	0.083	0.2	0.2	0.091	0.273	0.2	0.273	0.0	0.2
TEDist	24	20	20	22	22	20	22	23	20

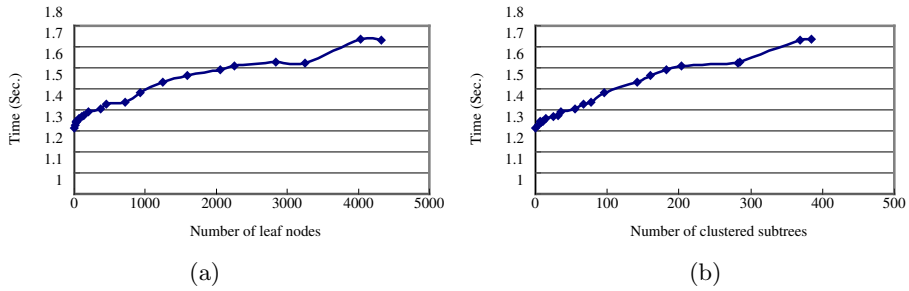
ments. In our experiments, we utilized the real XML documents, `DBLP.xml` [19] and `SigmodRecord.xml` [1]. Because the calculation of the tree edit distance is extremely time-consuming, we divided the `SigmodRecord.xml` into small fragments. Each fragment contains the entire articles of one issue. In the same way, we divided the `DBLP.xml` into fragments contains almost the same number of nodes as those of `SigmodRecord.xml`. Here we show the result of an example using one fragment of `SigmodRecord.xml`[3] and six fragments of `DBLP.xml`[4]. Table 2 shows the number of nodes and clustered subtrees (each subtree contains complete information of an article) included in each fragment. Table 3 shows the results of the tree edit distance and tree similarity degree of each pair of fragments. From the results, we can observe that the tree edit distance of each pair of fragments is almost the same. However, the tree similarity degree increases proportionally to the number of hit subtrees as shown in Fig. 10. That is to say, our algorithm can effectively distinguish the similarity differences between pairs of XML documents even they have the same tree edit distance. Table 4 shows the detailed results of each matched subtree of *sigmod.xml* × *dblp*6.*xml*, where $T_M[i]$ denotes the matched subtree, $*$ indicates the hit subtree, N_{sigmod} and N_{dblp} represent the number of nodes in each subtree of the matched subtree of *sigmod.xml* and *dblp*6.*xml*, respectively, and S_M and *TEDist* denote the similarity degree and the edit distance of each matched subtree, respectively. From the results, we can see that it is difficult for the tree edit distance to determine the hit subtree. However, our algorithm can effectively determine the hit subtree by setting an appropriate threshold $\mathcal{T}$. Therefore, by integrating the hit subtrees, the XML document that contains more complete information can be output.

5.4 Evaluating LAX

In our experiments, we took two XML documents from synthetic or real data sets as the input for our algorithm. And then we investigated how the number of leaf nodes and the number of clustered subtrees impacted the performance of our algorithm. The time for computing the tree edit distance using synthetic data sets are shown in Fig. 11. The X-axis in Fig. 11 (a) represents the total number of leaf nodes of the two documents to be joined, and the X-axis in (b) denotes

[3] Vol.20, No.3, SIGMOD Record 1991

[4] To obtain different number of hit subtrees, in this paper we specially chose the fragments that contain different number of articles from Vol.20, No.3, SIGMOD Record 1991



(a) (b)

Fig. 11. Time for computing the tree similarity degree using synthetic data sets

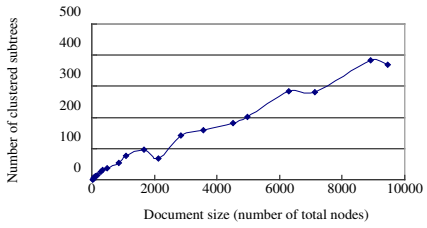


Fig. 12. Number of clustered subtrees of synthetic XML documents

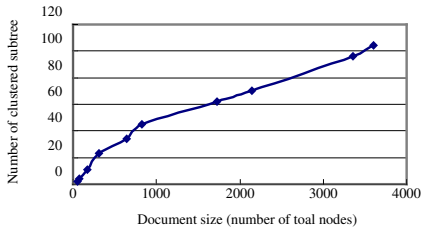
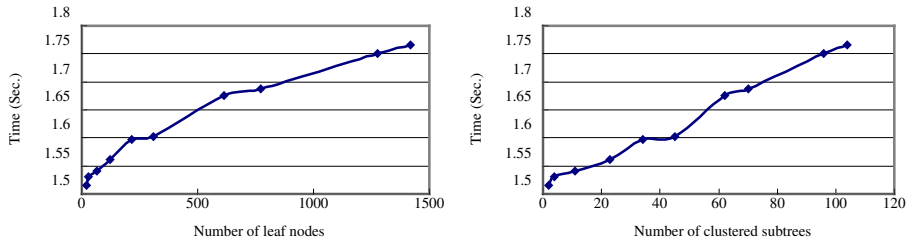


Fig. 13. Number of clustered subtrees of real XML documents

the total number of clustered subtrees in the two documents. From Fig. 11, we observe that the runtime of our algorithm increases almost proportionally to the number of leaf nodes or the number of clustered subtrees, and the impacts on the time to computer the tree similarity degree by the two factors are almost the same. Fig. 11 also shows that for the total number of leaf nodes of the two documents less than 5000 (document size less than 300KB) or the total number of clustered subtrees less than 400, the computation of the tree similarity degree can be accomplished within 2 seconds.

We also investigated how the number of clustered subtrees changed when the document size increased. Fig. 12 indicates that the number of clustered subtrees generally increases, when the size of document becomes larger. However, the number of clustered subtrees does not always increase monotonously, because the clustered subtrees might contain different number of nodes due to different DTDs.

The results using real XML data sets are shown in Fig. 13 and 14. The runtime using real data sets increases faster than the one using synthetic data does under the same scale number of leaf nodes. Because the length of the PCDATA of real data is generally longer than that of synthetic data made by the XML generator.



(a) (b)

Fig. 14. Time for computing the tree similarity degree using real data sets

6 Conclusions and Future Work

It becomes more important to measure the approximate similarity between XML documents for integrating multiple XML data sources. Tree edit distance is currently recognized as a general metric for computing the structural similarity between XML documents. However, its computational cost is too expensive. Recognizing this problem, in this paper we have proposed *LAX* (**L**eaf-clustering based **A**pproximate **X**ML join algorithm), in which the two XML document trees are clustered into many subtrees representing independent items and the approximate similarity between them are determined by calculating the similarity degree based on the leaf nodes of each pair of subtrees. We have also proposed an effective algorithm for clustering the XML document for *LAX*.

The proposed algorithm has the following advantages: 1) it is an inexpensive and effective algorithm to determine the approximate similarity between XML documents; 2) the traditional wisdom in join algorithms can be applied to it without any difficulties; and 3) its output document contains complete information of the two documents been joined.

We have done experiments to compare our algorithm with the tree edit distance and evaluate its performance using both synthetic and real data sets. Our experimental results show that *LAX*, comparing with the tree edit distance, is more efficient in performance and more effective for measuring the approximate similarity between XML documents.

In our experiments, we just used XML data of small size generated by the DOM Parser. In the future, we plan to do further experiments with the real bioinformatics and large-scale knowledge-based XML data stored in RDBs.

Acknowledgements. We thank the anonymous reviewers for their valuable comments. We are also grateful to Mr. Xiangyong Ouyang for his assistance on programming. This work is 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. M. Arenas and L. Libkin. A Normal Form for XML Documents. *ACM Transactions on Database Systems*, 29(1):195-232, March 2004.
3. S. Chawathe and H. Garacia-Molina. Meaningful Change Detection in Structured Data. In *Proc. of ACM SIGMOD 1997*, pages 26-37, 1997.
4. S. Chawathe, A. Tajaraman, H. Garacia-Molina and J. Widom. Change Detection in Hierarchically Structured Information. In *Proc. of ACM SIGMOD 1996*, pages 493-504, 1996.
5. I. F. Cruz, H. Xiao and F. Hsu. An Ontology-Based Framework for XML Semantic Integration. In *Proc. of IDEAS 2004*, pages 217-226, 2004.
6. A. Doan, P. Domingos and A. Halevy. Reconciling Schemas of Disparate Data Sources: A Machine-learning Approch. In *Proc. of ACM SIGMOD 2001*, pages 509–520, 2001.
7. W. Fan and L. Libkin. On XML Integrity Constraints in the Presence of DTDs. In *Proc. of PODS'01*, pages 114-125, 2001.
8. S. Guha, H.V. Jagadish, N. Koudas, D. Srivastava and T. Yu. Approximate XML Joins. In *Proc. of ACM SIGMOD 2002*, pages 287-298, 2002.
9. S. Guha, N. Koudas, D. Srivastava and T. Yu. Index-Based Approximate XML Joins. In *Proc. of ICDE 2003*, pages 708-710, 2003.
10. IBM XML Generator. Available at http://www.alphaworks.ibm.com/xml/
11. M. Lee, L. Yang, W. Hus and X. Yang. XClust: Clustering XML Schemas for Effective Integration. In *Proc. of CIKM'02*, pages 292-299, 2002.
12. MAGE (MicroArray and Gene Expression). Available at http://www.mged.org/Workgroups/MAGE/mage.html
13. A. Marian, S. Abiteboul, G. Cobena and L. Mignet. Change-Centric Management of Versions in an XML Warehouse. In *Proc. of 27th VLDB*, pages 581-590, 2001.
14. A. Nierman and H. V. Jagadish. Evaluating Structural Similarity in XML Documents. In *Proc. of WebDB 2002*, pages 61-66, 2002.
15. E. Rahm and P. A. Bernstein. A Survey of approaches to automatic schema matching. *the VLDB Journal*, 10(1):334-350, 2001.
16. S. Selkow. The Tree-to-tree Editing Problem. *Information Processing Letters*, 6(6):184-186, December 1977.
17. Y. Wang, D. J. DeWitt and J. Cai. X-Diff: An Effective Change Detection Algorithm for XML Documents. In *Proc. of ICDE 2003*, pages 519-530, March 2003.
18. World Wide Web Consortium (W3C). The Document Object Model (DOM). http://www.w3.org/DOM/.
19. XML Version of DBLP. Available at http://dblp.uni-trier.de/xml/
20. X. Yang, M. Lee and T. Ling. Resolving Structural Conflicts in the Integration of XML Schemas: A Semantic Approach. In *Proc. of ER2003*, papges 520-533, 2003.
21. K. Zhang and D. Shasha. Simple Fast Algorithm for the Editing Distance Between Trees and Related Problems. *SIAM Journal of Computing*, 18(6):1245-1262, December 1989.
22. K. Zhang and D. Shasha. Tree Pattern Matching. *Pattern Matching Algorithms*, chapter 11. Oxford University Press, 1997.