Abstract—In the paper, we addressed the problem of finding the common structures in a collection of XML documents. Since an XML document can be represented as a tree structure, the problem how to cluster a collection of XML documents can be considered as how to cluster a collection of tree-structured documents. First, we used SOM (Self-Organizing Map) with the Jaccard coefficient to cluster XML documents. Then, an efficient sequential mining method called GST was applied to find maximum frequent sequences. Finally, we merged the maximum frequent sequences to produce the common structures in a cluster.

Keywords- XML document; tree-structured; clustering; sequential pattern mining; common structure

I. INTRODUCTION

Recently, XML (eXtensible Markup Language) has become the standard of data representation and exchange in many applications. With the characteristics of including structural information in the document itself and being easy to use and understand the information, it becomes very useful in many applications such as databases and web. Since more and more data tend to represent itself in the XML format, it inspired a challenge for the integration between different kinds of data sources. In general, integrating different XML documents is usually through their DTD (Document Type Definition) [1] or XSD (XML Schema Definition) [2], but it cannot guarantee that both XML documents and their DTD or XSD could be obtained. Therefore, some research has been directed at trying to infer the DTD for a collection of XML documents [1].

In the past, there have been many studies focusing on XML document clustering. Some of them were based on calculating the editing distances between XML documents [3], whereas some of them were based on VSM (Vector Space Model) [4]. In this paper, we adopted the popular clustering method SOM (Self-Organizing Map) [5] with the Jaccard coefficient as the distance measure in order to get better results. In other words, we considered clustering a collection of XML documents according to their structures, and further found their common structures. Within each cluster, we employed a sequential pattern mining technique called GST [6] to extract maximum frequent sequences, and then merged them to produce common structures.

The remainder of the paper is organized as follows. In Section 2, we review the different variations of tree inclusion and related work. The system framework is described and explained using an example in Section 3. Then, we conduct some experiments and observe the performance in Section 4. Finally, we make conclusions in Section 5.

II. RELATED WORK

Since an XML document can be represented as a tree structure, the problem how to cluster a collection of XML documents can be considered as how to cluster a collection of tree-structured documents. Then, we addressed the issue of finding common tree structures from a collection of tree-structured XML documents in a cluster.

A. XML Tree Structures

An XML document can be mapped to a tree structure, as shown in Fig. 1. Here, we do not consider the IDREFS and hyperlinks, and elements and attributes are viewed as the same; i.e., they are all mapped to nodes using element or attribute names as labels. For clustering XML documents and then finding the common tree structures from XML documents in the same cluster, we only need to consider the structure of XML documents, but not their contents.

![Figure 1. XML document and its corresponding tree.](image-url)
B. Tree Inclusion

According to [7], the tree inclusion of XML documents has three different variations of notions; i.e., subtree inclusion, tree embedding, and tree subsumption.

1) Subtree Inclusion: The subtree inclusion is the most exclusive (or the strongest) definition. Suppose that there are two trees X and Y, and we say that Y is included in X iff X has a subtree absolutely the same as Y. For the example as shown in Fig. 2, T' is included in T1, but not in T2 and T3 according to the subtree inclusion definition.

![Figure 2. Various tree-structured XML documents.](image)

2) Tree Embedding: The tree embedding definition is a little different from the subtree inclusion definition. Suppose that there are two trees X and Y, and we say that Y is included in X if all the nodes of Y are also in X, and all the ancestor relations of Y are preserved in X and vice versa. For the example as shown in Fig. 2, T' is included in T2 according to the tree embedding definition. However, T' is not included in T3 since although all the ancestor relations of T' are preserved in T3, there exists an ancestor relation (c, e) in T3, but not in T'. Definitely, T' is also included in T1 according to the tree embedding definition.

3) Tree Subsumption: The tree subsumption definition is the weakest one among three variations. Suppose that there are two trees X and Y, and we say that Y is included in X if all the nodes of Y are also in X, and all the ancestor relations of Y are preserved in X. For the example as shown in Fig. 2, T' is included in T3 according to the tree subsumption definition. Definitely, T' is also included in T1 and T2 according to the tree subsumption definition. In the paper, finding the common tree structures from XML documents in the same cluster only needs to conform to the tree subsumption definition.

C. Sequential Pattern Mining in Tree Structures

The sequential pattern mining was introduced by Rakesh and Ramakrishnan in 1995 [8]. Given a transactional database with the transaction time in each record, they mined out the sequential patterns with the supports more than the minimum support.

Let I={i1, i2, i3, ..., im} be a set of items and S=<s1, s2, s3, ..., sn> be a sequence which is an ordered list of itemsets. Each sj in S is an itemset whose items are purchased by a customer at the same moment. For example, given the purchases S=<<(1),(23),(4)> of customer C, it means that C bought item 1, then bought item 2 and 3 at the same moment, and finally bought item 4.

A sequence <a1,a2,a3,...,an> is a subsequence of another sequence <b1,b2,b3,...,bn> if there exist integers i1<i2<i3<...<in such that a1≤bi1, a2≤bi2, a3≤bi3, ..., and an≤bin. For example, sequence <(1),(2),(4)> is a subsequence of sequence <(1),(23),(4)>, but sequence <(1),(2),(3)> is not because item 2 and item 3 must be bought together. A sequence with k items is called a k-sequence, and it may have three types, such as ordered, non-ordered, and mixed. For example, <(1),(2),(4)> is a 3-sequence with order, <(124)> is a 3-sequence with no order, and <(1),(2),(4)> is a 3-sequence with mixed order.

In the paper, we used the sequential mining technique to find common tree structures from XML documents. Here, an XML node (or element) is considered as an item, the level of nodes is as transaction time, and a path from the root to a leaf is as a sequence. Since only one node is allowed in each level of a path, a sequence like <(ab)> exists in our work. In other words, each sj in S=<s1, s2, s3, ..., sn> is a 1-sequence. For the example as shown in Fig. 2, T1 has five paths (or sequences); i.e., <a,b,c>, <a,b,d>, <a,b,e>, <a,a,c>, and <a,a,e>.

D. Graph Search Techniques

Generally, a sequential mining algorithm like the Apriori-like algorithm always finds large k-sequences through large (k-1)-sequences. Since the graph search techniques (GST) can out of order find large k-sequences (k=>3) [6], the performance is considered to be better than the Apriori-like algorithm. First, the GST finds large k-sequences (L2), and then uses L2 to construct an item relation graph (IRG). Finally, it employs the graph search techniques to find all other large k-sequences (k=>3) based on the item relation information built in the graph.

III. SYSTEM FRAMEWORK

As shown in Fig. 3, the system framework can be divided into two parts; i.e., 1) clustering phase and 2) mining phase. In the clustering phase, we used a matrix translation method and SOM approach to cluster a collection of XML documents. Then, in the mining phase, we used GST (Graph Search Technique) to find maximum frequent sequences and produced common structures for each cluster.

A. Clustering Phase

In this phase, we first translated each XML document into multiple label-pairs which form a feature matrix. Then, we utilized the SOM approach to cluster these XML documents.

1) Matrix Translation: An XML document can be represented as a tree structure consisting of nodes and edges.
For a tree structure, we can express the reachable relation between nodes \( x \) and \( y \) as \( x \prec y \) if there is a path from \( x \) to \( y \). Thus, we can translate each XML document into a set of label-pairs \( x \prec y \) such that \( x \) is the ancestor of \( y \). For the example as shown in Fig. 2, \( T_1 \) can be translated into \( T_1 = \{a \prec b, a \prec c, a \prec d, a \prec e, a \prec a, b \prec c, b \prec d, b \prec e\} \). To determine the similarity among XML documents, label-pairs act as major and basic roles, not only considering labels but also the connectivity between them. After translating all XML documents, we can merge all sets of label-pairs into a document matrix \( DM(n, m) \), where \( n \) is the number of XML documents and \( m \) is the number of distinct label-pairs. Here, the label-pairs can be viewed as the features of XML documents, and used to determine the similarity among XML documents in clustering methods.

### 2) Clustering Method

For the produced document matrix, we employed SOM (Self-Organizing Map) method to cluster XML documents. SOM was proposed by Kohonen in 1980, and is an unsupervised clustering method which organizes a topological map from a random starting point, so it is also called Kohonen’s self-organizing feature map. In general, the Euclidean norm is usually adopted as the distance measure in SOM; however, our goal is to find the similar structure, so we ignored those features (or label-pairs) not existing in both documents while calculating the similarity.

Here, we adopted the Jaccard coefficient proposed by Guha et al. in 1998 to be the distance measure in SOM. The Jaccard coefficient also called Intersection Over Union measure is usually used to measure the similarity in datasets as follows:

\[
J(X, Y) = \frac{|X \cap Y|}{|X \cup Y|}
\]

The value of the Jaccard coefficient is in the range \([0, 1]\) where 0 represents two objects are absolutely different, and 1 represents two objects are absolutely the same. For example, five sets \( A, B, C, D, E \) and their features are shown in TABLE I, and the Jaccard similarity of all pairs is calculated as shown in TABLE II.

#### B. Mining Phase

In this phase, we have two stages to obtain the common structures of XML documents in a cluster. First, we used GST (Graph Search Technique) to retrieve maximum frequent sequences (or longest common paths) [6]. Then, we merged them into common structures for each cluster.

### 1) Retrieving Paths

To retrieve all longest common paths in one cluster, we have four steps; i.e., 1) translating into XML table, 2) joining with itself, 3) constructing IRG (Item Relation Graph), and 4) using GST to produce maximum frequent sequences.

#### a) Translating into XML Table to Produce Large 1-sequences

First, we translated an XML document into XML_table with the format as shown in Fig. 4. XML is a convenient and flexible way for human to describe structured data, and the XML grammar allows that the same label appears repeatedly in a document. In other words, it allows the same label to appear in one path repeatedly. This flexibility would bring us a problem; i.e., the problem whether the path with repeated labels is a common path or not is hard to decide without any tricky handling. In order to solve this problem, when there is a label re-appearing in one path, we would rename it and use MapTable to recall its original label. The rename operation is independent for each XML document.

![Figure 4. Structure of XML_table.](image)

For the example as shown in Fig. 2, \( T_1 \) has a path \( a \prec a \prec c \), and the second label \( a \) would be renamed as \( a_1 \) and stored in MapTable, as shown in Fig. 5. By traversing a tree from left to right, all paths in the document could be found and numbered from 1. Finally, they would be translated into XML_table, as shown in TABLE III.
After translating each XML document in a cluster into XML_table, we have candidate 1-sequences C1. While calculating their support counts, we must consider renamed labels as original labels by referring to MapTable, and then we can get large 1-sequences L1 used to further construct C2 and L2. Here, since we would like to retrieve the common structures of XML documents in a cluster, the minimum support is defined as the number of documents in a cluster. Besides, when a sequence appears in one document more than once, we considered it only appears once.

b) Joining L1 with Itself to Produce Large 2-sequences: In the step, we utilized Doc_Name, Path_No, and Level_No to join L1 with itself to produce L2. In a tree, it is impossible for more than one node to appear in the same level of one path; i.e., there should not be a sequence such as <(AB)>. Therefore, if there are three large 1-sequences A, B, and C, we can get six candidate 2-sequences C2={<(A)(B)>, <(A)(C)>, <(B)(C)>, <(B)(A)>, <(C)(A)>, <(C)(B)>} after joining. The format of C2 and L2 is shown in Fig. 6. Each 2-sequence contains two labels in order. The first label has smaller Level_No than the second one, and their Level_Nos are stored in Level_Start and Level_End, respectively. After filtering out the 2-sequences not satisfying the minimum support, we can produce L2. As well, we must consider renamed labels as original labels while calculating their support counts.

c) Constructing IRG: After producing L2, we can construct an IRG where the nodes are the labels of the 2-sequences in L2, and the edges record the orders between labels. For a 2-sequence <(A)(B)>, we draw a directional edge from A to B and each edge contains the attached information such as Doc_Name, Path_NO, Level_Start, and Level_End, as shown in Fig. 7. While constructing an IRG, a renamed node is viewed as a normal node. For the path a-a1-c as shown in Fig. 5, we construct the edges <(a)(a1)> and <(a1)(c)> in the IRG.

d) Using GST to Produce Maximum Frequent Sequences: After constructing an IRG, we can employ the DFS (Depth-First-Search) method to find large sequences from each node of the IRG, and further find maximum frequent sequences. The processing is almost the same as the original GST [6], except that we must check whether the next edge in the IRG is really the next one in a path before searching down. For example, if there are two edges e and w and e is in front of w, three conditions should be checked according to the attached information; i.e., 1) e.(Doc_Name, Path_No)=w.(Doc_Name, Path_No), 2) e.Level_End=w.Level_start, and 3) whether the path satisfies the minimum support. If these conditions are satisfied, go forward; otherwise, go back.

Finally, we found all large sequences from the IRG. Besides, in order to find maximum frequent sequences, we restored renamed labels to original labels by referring to MapTable, and filtered out those large sequences contained in other large sequences. The algorithm of finding maximal frequent sequences is showed in Fig. 8.

2) Merging Paths: In this stage, we merged the maximum frequent sequences to produce the common structures in a cluster. The algorithm of merging paths is shown in Fig. 9. During merging paths, when including a new node in the common structure, we checked whether the common structure conforms to the tree subsumption definition; i.e., one-to-one node mapping between the common structure and the original trees must be preserved, although the original tree subsumption definition did not

![Figure 5. Map table.](image)

![Figure 6. Structure of C2 and L2.](image)

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Doc_Name</th>
<th>Path_No</th>
<th>Level_Start</th>
<th>Level_End</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>T1</td>
<td>1</td>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>B</td>
<td>T1</td>
<td>1</td>
<td>2</td>
<td>b</td>
</tr>
<tr>
<td>C</td>
<td>T1</td>
<td>1</td>
<td>3</td>
<td>c</td>
</tr>
<tr>
<td>A</td>
<td>T1</td>
<td>2</td>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>B</td>
<td>T1</td>
<td>2</td>
<td>2</td>
<td>b</td>
</tr>
<tr>
<td>C</td>
<td>T1</td>
<td>2</td>
<td>3</td>
<td>d</td>
</tr>
<tr>
<td>A</td>
<td>T1</td>
<td>3</td>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>B</td>
<td>T1</td>
<td>3</td>
<td>2</td>
<td>b</td>
</tr>
<tr>
<td>C</td>
<td>T1</td>
<td>3</td>
<td>3</td>
<td>e</td>
</tr>
<tr>
<td>A</td>
<td>T1</td>
<td>4</td>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>B</td>
<td>T1</td>
<td>4</td>
<td>2</td>
<td>a1</td>
</tr>
<tr>
<td>C</td>
<td>T1</td>
<td>4</td>
<td>3</td>
<td>c</td>
</tr>
<tr>
<td>A</td>
<td>T1</td>
<td>5</td>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>B</td>
<td>T1</td>
<td>5</td>
<td>2</td>
<td>a1</td>
</tr>
<tr>
<td>C</td>
<td>T1</td>
<td>5</td>
<td>3</td>
<td>e</td>
</tr>
</tbody>
</table>

![Figure 7. IRG with sequences <(A)(B)>, <(A)(C)>, and <(C)(B)> and edge information.](image)

![Figure 8. Maximal frequent sequences algorithm.](image)
enforce this constraint yet. If the constraint cannot be conformed, we split the common structure into two parts which would be done respectively.

\[\text{For each group with the same first label}\]
\[\text{n: the number of paths in the group}\]
\[\text{x: the first label in these paths}\]
\[\text{For(k=1;k<n;k++)}\]
\[\text{If(x does not exist)}\]
\[\text{Create the root x;}\]
\[\text{AddNode(x,k);}\]
\[\text{Output these common structures;}\]
\[\text{AddNode(x,k);}\]
\[\text{If(x is immediately followed by any y in path k)}\]
\[\text{Create y linked from x;}\]
\[\text{If the partial graph does not conform to the tree subsumption definition}\]
\[\text{Split the partial graph into path k and graph G;}\]
\[\text{For path k, AddNode(y,k);}\]
\[\text{For graph G, return;}\]
\[\text{else}\]
\[\text{AddNode(y,k);}\]
\[\text{}}\]

C. Example

In this section, we used the example as shown in Fig. 2 to illustrate how to produce the common structures in a cluster. Suppose these three XML documents have been clustered in the same cluster, and we would like to find the common structures. Since the number of XML documents is 3, the minimum support is defined as 3.

In the step of translating into XML table, we detected the labels re-appearing in one path, then renamed and recorded them in MapTable as shown in Fig. 10. Then, we translated all XML documents into XML_table(C1). After calculating the support count of each node, we filtered node f not satisfying the minimum support out from XML_table(L1), as shown in Fig. 11.

In the step of constructing IRG, we used 12 distinct large 2-sequences in L2, such as <(a)(b)>, <(a)(c)>, <(a)(d)>, <(a)(e)>, <(a)(a1)>, <(b)(c)>, <(b)(d)>, <(b)(e)>, <(a)(b)>, <(a)(c)>, <(a)(d)>, and <(a)(e)> to construct the IRG and their edge information, as shown in Fig. 13.

In the step of using GST, we found all large sequences from the IRG according to three conditions. Finally, through restoring and filtering, we found four maximum frequent sequences as shown in Fig. 14.

In the merging paths stage, we merged four maximum frequent sequences to produce two common structures, as shown in Fig. 15.

In the joining step, we joined L1 with itself, and produced C2. After calculating the support count of each 2-sequence, we filtered <(b)(a)> and <(c)(e)> not satisfying the minimum support out from L2, as shown in Fig. 12.
IV. EXPERIMENTAL RESULTS

A. Test Environment and Data Sets

We implemented the system using an SOM tool for the clustering and using JAVA 6 for the mining. Then, the experiments were conducted on the platform with Intel(R) Core(TM)2 Quad CPU Q8200, 2G memory, and running Windows XP Professional sp3. Two kinds of data sets were used to evaluate the performance; i.e., 1) synthetic data sets and 2) real data sets. The synthetic data sets used in the experiments were generated by the XML Generator [9]. As shown in TABLE IV, two sets of 500 synthetic XML documents were generated from 10 heterogeneous DTDs respectively [10], denoted by DB500DTD10MR3 and DB500DTD10MR6, and one set of 150 synthetic XML documents was generated from 3 homogeneous DTDs [10], denoted by DB150DTD3MR6. Here, DB means the number of generated documents, DTD means the number of DTDs, and MR means MaxRepeats (i.e., a parameter of the XML Generator). Varying the parameter MaxRepeats can determine the maximum number of times that a node appears as a child of its parent node. Besides, the parameter NumLevels means the maximum number of levels in the generated document trees. For the real data sets, we used the SIGMOD Record [11] which has 100 XML documents generated from two DTDs (i.e., 50 documents from IndexTermsPage.dtd and 50 documents from OrdinaryIssuePage.dtd).

<table>
<thead>
<tr>
<th>DB500</th>
<th>DB500</th>
<th>DB150</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTD10</td>
<td>DTD10</td>
<td>DTD3</td>
</tr>
<tr>
<td>MR3</td>
<td>MR6</td>
<td>MR6</td>
</tr>
<tr>
<td>10*50=500</td>
<td>10*50=500</td>
<td>3*50=150</td>
</tr>
<tr>
<td>MaxRepeats</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>NumLevels</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

Here, we used precision and recall to evaluate the clustering quality. They are expressed as follows.

\[
\text{Precision} = \frac{\sum a_i}{\sum a_i + b_i}, \quad \text{Recall} = \frac{\sum a_i}{\sum a_i + e_i}
\]

where ai is the number of documents in cluster Ci, stemmed from dominant DTD Di, bi is the number of documents in Ci, not stemmed from Di, and ei is the number of documents not in Ci, but stemmed from Di.

B. Result Analyses

The experimental results are shown in TABLE V, where all values are the averages for 10 times of experiments. Whatever data sets were used, their results were good enough for precision, recall, and running time. The value of the recall for data set DB150DTD3MR6 is a little less than those of other data sets; this might be resulted from that the XML documents were generated from the homogeneous DTDs, and this made SOM distinguish them not easily. For the real data sets, the precision and recall reach 100%, since it is easy to distinguish the features (i.e., labelpair) within the documents.

TABLE IV. SYNTHETIC DATA SETS.
TABLE V. EXPERIMENTAL RESULTS.

<table>
<thead>
<tr>
<th></th>
<th>DB500 DTD10 MR3</th>
<th>DB500 DTD10 MR6</th>
<th>DB150 DTD3 MR6</th>
<th>SIGMOD Record</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10*50=500</td>
<td>10*50=500</td>
<td>3*50=150</td>
<td>2*50=100</td>
</tr>
<tr>
<td>Cluster number</td>
<td>10.2</td>
<td>10</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Precision</td>
<td>0.98</td>
<td>0.98</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Recall</td>
<td>0.98</td>
<td>0.99</td>
<td>0.92</td>
<td>1</td>
</tr>
<tr>
<td>Runtime (sec.)</td>
<td>280</td>
<td>281</td>
<td>78</td>
<td>34</td>
</tr>
</tbody>
</table>

In addition to the clustering result analyses, we also tried to analyze the produced common structures and to compare them with the original DTDs; in other words, we calculated their similarity according to the label-pairs. In the past, researchers only focused on how to make the clustering better and/or how to find common structures, but they never did similarity comparisons. As shown in TABLE VI, the average similarity between the produced common structures and original DTDs reaches more than 50%. As we know, DTDs may have optional elements, and no methods can find the common structures with option elements, without given DTDs. This explains why the achieved similarity is not high enough.

Table VI. SIMILARITY ANALYSES.

<table>
<thead>
<tr>
<th></th>
<th>DTD10 MR3</th>
<th>DTD10 MR6</th>
<th>DTD3 MR6</th>
<th>SIGMOD Record</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average similarity</td>
<td>0.52</td>
<td>0.60</td>
<td>0.88</td>
<td>0.61</td>
</tr>
</tbody>
</table>

C. Scalability Test

In this experiment, we also tested the system scalability for different numbers of documents. As shown in Fig. 16, the label-pair time and clustering time increase expectably and there is no much difference for data sets DTD10MR3 and DTD10MR6. In general, GST time would increase progressively with the number of documents, especially for DTD10MR6, since the more the documents are, the more the edge information must be checked.

V. CONCLUSIONS

In this paper, we proposed an efficient method to extract the common structures from a collection of XML documents. First, the SOM method was used to cluster XML documents. Then, a sequential pattern mining technique called GST was applied to find the maximum frequent sequences in each cluster. Finally, we merged them to produce the common structures in a cluster. Through the experiments, we found that our method has high precision, high recall, and acceptable running time. Besides, the system can be also scaled up to very large data sets.

ACKNOWLEDGMENT

This work was supported by National Science Council of R.O.C.

REFERENCES


