Encoding of XML Elements for Mining Association Rules

Gongzhu Hu[1], Yan Liu, Qiong Huang
Central Michigan University, USA

Abstract

Mining of association rules is to find associations among data items that appear together in some transactions or business activities. As of today, algorithms for association rule mining, as well as for other data mining tasks, are mostly applied to relational databases. As XML being adopted as the universal format for data storage and exchange, mining associations from XML data becomes an area of attention for researchers and developers. The challenge is that the semi-structured data format in XML is not directly suitable for traditional data mining algorithms and tools. In this paper we present an encoding method to encode XML tree-nodes. This method is used to store the XML data in Value Table and Transaction Table that can be easily accessed via indexing. The hierarchical relationship in the original XML tree structure is embedded in the encoding. We applied this method to association rules mining of XML data that may have missing data.

Keywords: XML tree, node encoding, XML indexing, data mining, association rules.

1. Introduction

XML is a language specifying semi-structured data. It is rapidly emerging as a new standard for data representation and exchange on the Web. It is also a set of the rules and guidelines describing semi-structured data in plain text rather than proprietary binary representations. Since its standardization by the W3C in 1998, XML has been the driving force behind numerous other standards and vocabularies that provide services to a wide range of industries. In today's business world, enormous amount of data are stored in various formats to be processed and integrated to support decision/policy making. The vast majority of the data reside in relational databases, the most widely used data storage and management format. Relational databases are well-defined and applications based on relational databases are in general very robust and efficient. Many algorithms and methodologies have been developed to apply to relational databases. The very basic feature of relational databases is that the data are stored in *tables*, or 2-dimensional form, in which rows represent data objects and columns specify the attributes of the objects. That is, the values of an object's attributes are "encapsulated" in a row (or a line for plain text data) that can be retrieved by column-access methods (or separated by some "delimiters" such as space, comma, tab, etc. for plain text data). This table-based data representation is very convenient for algorithms to access and process the data. For example, most data mining algorithms [9] work on relational databases. In particular, mining of association rules works on transactions, each of which consists of one or more items. A transaction is represented by one row of data values, each of which is an item of the transaction. These algorithms all assume that the data items are arranged in this way, either in relational tables or in plain text.

XML data, however, are organized and stored in tree-like structures of multiple-level hierarchies. This format is very different from 2-dimensional table, and has presented some challenges to the algorithms and applications that were developed based on relational tables. Considering the task of mining association rules, the main difficulty is to find what constitute a transaction and how to find the items in the transactions that are the same. First, let's review the basic concept of association rule mining.

Mining of association rules, also called market basket analysis in business, is to find interesting association or correlation among data items that often appear together in the given data set. This is a loose definition. To be more precise, we include a formal definition [4, 9] here to make this paper self-contained. A transaction $T = \{X_1, \ldots, X_n, n > 0\}$ is a finite set of items X_i in an application domain, indicating that these items are involved in the same transaction T. An association rule is of the form

$$\{X_1, \ldots, X_m\} \Rightarrow \{Y_1, \ldots, Y_k\}, m, k \geq 1$$

where X_i and Y_j are items involved in a set of transactions. Each association rule is associated with two measures, *support* and *confidence*. Support is the probability that a transaction contains item set $\{X_1,\ldots,X_m,Y_1,\ldots,Y_k\}$, and confidence is the conditional probability that a transaction containing $\{X_1,\ldots,X_m\}$ also contains $\{Y_1,\ldots,Y_k\}$. The association rule says that whenever $X_1,\ldots,X_m$ appear in a transaction T, $Y_1, \ldots, Y_k$ also appear in T with probabilities given in *support* and *confidence*. In practice, the data is given as a set of transactions $\boldsymbol{T} = \{T_1,\ldots,T_N\}$ of size N. Let the left-hand side of an association rule be $\vartheta_L = \{X_1,\ldots,X_m\}$ and the right-hand side be $\vartheta_R = \{Y_1,\ldots,Y_k\}$. The support of the item set $\vartheta = \vartheta_L + \vartheta_R = \{X_1,\ldots,X_m,Y_1,\ldots,Y_k\}$ can be

[1] Department of Computer Science
Central Michigan University
Mount Pleasant, MI 48859, USA
hu1g@cmich.edu

calculated by counting the number of occurrences of ϑ in T. That is,

$$\text{support}(\vartheta) = F(\vartheta) / N$$

where $F(\vartheta) = \Sigma_{i=1,N}\delta_i$, $\delta_i = 1$ if $\vartheta \subseteq T_i$, 0 otherwise. That is, $F(\vartheta)$ is the "frequency" of ϑ in T. Similarly,

$$\text{confidence}(X_1,\ldots,X_m \Rightarrow Y_1,\ldots,Y_k) = F_{\vartheta L}(\vartheta_R) / F(\vartheta_L)$$

where $F_{\vartheta L}(\vartheta_R)$ is the frequency (number of occurrences) of ϑ_R in those transactions that contains ϑ_L.

There are several algorithms for finding association rules, for example, the *Apriori Algorithm* [20]. These algorithms iteratively calculate frequent itemsets of progressively increasing sizes, starting from singleton sets, that satisfy the minimum support and confidence levels. The iteration steps are quite straightforward, largely due to the simplicity of the way the input data are structured as a 2-dimensional table. If the data are given in XML format, however, the algorithms may be complicated in locating transactions and the itemsets in the data. If the XML data have only two levels, there won't be any complications because the XML file can be easily converted to a 2-D table. However, XML documents are not always so simple; they often contain hierarchies of more than two levels and have arbitrary user-defined tags for representing document elements, and allow the elements to be organized in a nested structure regardless of the existence of an explicitly specified document, the descriptor or schema. Consider the example given below.

```
<?xml version="1.0" ?>
<CATALOG>
  <CD>
    <TITLE>Moonlight</TITLE>
    <ARTIST>Sam Brown</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>A and M</COMPANY>
  </CD>
  <CD>
    <TITLE>Flying</TITLE>
    <ARTIST>
      <firstname>Savage</firstnam>
      <lastname>Rose</lastname>
    </ARTIST>
    <COUNTRY>EU</COUNTRY>
    <COMPANY>Polydor</COMPANY>
  </CD>
  <CD>
    <TITLE>Highland</TITLE>
    <ARTIST>Sam Brown</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>A and M</COMPANY>
  </CD>
  ....
</CATALOG>
```

In this example, the transactions and the items in the transactions are not given as clearly as in relational tables. But when we take a close look at the file, we can find that there exists some correlation among the data elements. For example, the association rule $\{Sam\ Brown,\ A\ and\ M\} \Rightarrow \{UK\}$ holds with support 0.67 and confidence 1.0. The question is how to identify the items that are considered in the same "transactions" and can be easily retrieved. Finding the "same" items in XML is much more complex than that in 2-D tables where the items are primary types (strings and numbers). Because of the nature of the tree structure, an XML element is an object composed of sub-elements as its children, except those leaf elements that are of primary types. This is conceptually similar to what is considered in multilevel data mining tasks where data items are composite objects. However, the levels in traditional multilevel data mining tasks are "predetermined" based on the schema defined on relational databases or data cubes, whereas the "levels" in XML data are not well-defined because they are semi-structured and there may not be a schema defined on the XML documents. Even a schema exists, there may be missing elements and varying number of sub-elements of the same tag name, among other flexible features of XML.

We present in this paper a solution to the problem of mining association rules on XML data by an encoding method that assigns a unique code to each different tag name and using 2-dimensional array (or vector of vectors) to store the data with the tag codes to index the collection. The code of an element embeds the codes of its children, so that the parent-child relationship can be easily determined by simple operations on the codes. With this coding scheme, we can also easily determine if two elements are the same because their children's codes are embedded in the parents' codes. This coding method is in fact equivalent to the recursive definition of structure-equivalence in programming languages. And, we give a new method that organizes the XML data into 2-dimensional arrays for easy operations.

The data mining method we use is based on the *Apriori* Algorithm – a basic algorithm for association rule mining. We use bottom-up algorithms to get all the rules for any level including cross-levels.

2. Related Work

A lot of work have been done on association rule mining since it was first introduced by Agrawal, Imielinski, and Swami [1]. Shortly after, Agrawal and Srikant, and Mannila et al. published the Apriori algorithm, independently [3, 20] and jointly [2]. Several variations of the Apriori algorithm were proposed, including use of hash table to improve the mining efficiency [22], partitioning the data to find candidate itemsets [24], reducing the number of transactions [3, 10, 22], sampling [27], and dynamic itemset

counting [5]. Various extensions of association rule mining were also studied [8, 12, 13, 19, 23] and is still an active research area.

Han and Fu [10, 11], and Srikant and Agrawal [26] proposed methods for *multilevel association rule mining* that is to certain degree similar to working on a tree structure like the one in XML. Páircéir el. at. extended multilevel mining to distributed systems [21]. Multilevel association rule mining finds rules at different abstraction levels of the concept hierarchy of the data items. For example, there may not be a strong association between "Gateway E4100 computer" and "HP 960 inkjet printer" because the items are too specific and hence spread in the multidimensional space too thin to have a strong support, but there would be such association at a higher abstraction level, such as between "Gateway computer" and "HP printer", or an even higher level as between "computer" and "printer." Mining association rules at different levels requires that the data items are categorized in a concept hierarchy before the algorithms can be applied. For example, the data should contain the information "HP 960 inkjet printer is an HP printer, which is a printer." This information needs to be provided by the applications in question.

Data in XML format are naturally form a hierarchy. However, it in general does not reflect the multilevel abstraction of the data items. The hierarchy in an XML file mostly represents the "has-a" relationship between a higher level item and its children rather than the "is-a" relationship as used in multilevel association rule mining. Hence the multilevel mining methods do not directly apply to XML data, or at least not easily.

Although applying data mining algorithms to XML data has not been extensively studied as of today, there are some research work reported. Some commercial products such as XMLMiner [7] are also available. Buchner et al. [6] outlined the XML mining and pointed out research issues and problems in the area. Hu [14] also discussed the problems of mining XML documents, particularly with respect to the tree structure (vs. tables in relational databases) and semantics of XML. Lee and her colleagues [17, 18] proposed a method for preparing XML documents for quantitative determination of the similarity between XML documents based on the XML semantics. Their method may be used to find items that appear together for discovering association rules.

The XML technologies related to the work presented in this paper include XML parsers (Document Object Model or DOM [28], Simple Access XML or SAX [25]). We do not use any specific query language to query the XML data for association rules; rather, we simply apply the Apriori algorithm to find all rules (of sizes from 2 to the maximum size allowed by the data) using user-provided parameters (minimum support and minimum confidence). This paper is an extension of our preliminary work [15] where the basic XML tree encoding scheme was established. This paper extends that work by adding more experimental results and shows the encoding method can help mining association rules involving missing data.

3. Encoding of XML Elements

Because XML data can be abstracted to a tree structure by an XML parser, we use the tree terminologies (root, level, sub-tree, node, parent, child, sibling, leaf, path, etc.) throughout this paper. An XML element starts with a start-tag (e.g. <name>) and ends with an end-tag (e.g. </name>). Between the two tags is the "actual data" of the element, which may in turn contain sub-elements. For the purpose of association rule mining, we consider the first level elements as "transactions." The sub-level elements are considered items in the transactions.

To keep track of the relationships between the nodes on different levels, we use several data structures to hold some information coming from the DTD and the DOM tree produced by an XML parser (e.g. IBM parser [16]). First, the nodes in the tree are encoded so that each node is assigned a unique number that embeds in it the location information of the node as well as the relationship between the node and its parent and siblings. Two "2-dimensional arrays" (vector of vectors) are used to hold the values of the nodes and information about the "transactions." These 2-D arrays are indexed by tag names and elements' values for fast access. First, let's introduce the encoding of the nodes.

3.1 Coding for XML elements

Let an element E be a node in the DOM tree constructed from a given XML file. E and its siblings (all nodes having the same parent of E) are *ordered* based on the order in which the elements appear in the DTD file under the same parent tag. That is, each node is assigned a unique ordinal number, d, among its siblings, starting from 1. Nodes under different parents may have the same ordinal number but they are on different paths. For example, in example 1 given before, the tags <TITLE>, <ARTIST>, <COUNTRY>, and <COMPANY> have ordinal numbers 1, 2, 3, and 4, respectively under their parent tag <CD>. The tags <firstname> and <lastname> are assigned ordinal numbers 1 and 2 under parent <ARTIST>.

Now, we can encode E with an integer C of n digits where n is the number of nesting levels of the XML as

$$C = d_1 \ldots d_n$$

where d_i is the ordinal number of the node along the path from the root to E, or 0 if the level of E in the tree is less than i. That is, the code C is 0-filled on the right if E is not a leaf node. Here we do not count the outmost tag (e.g.

<CATELOG>). For example, there are three nesting levels in the example XML file and hence each node is encoded using a 3-digit number (3-integer number, in fact). The tag <CD> will be encoded as 100, <TITLE> 110, <ARTIST> 120, <firstname> 121, <lastname> 122, <COUNTRY> 130, <COMPANY> 140, etc. The tree nodes and their codes are shown in Figure 1.

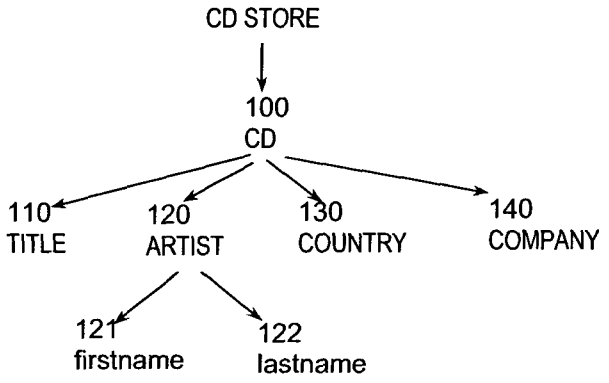


Figure 1. Tree nodes encoding of the example XML.

This encoding method is also easy to apply for node insertion and deletion. Once the nodes are encoded, we can store the values of the elements in a 2-D array (lengths of the rows may differ, as vector of vectors), called the *Value Table*. We can also store information about the "transactions" (considering the first-level tag as a transaction) in a *Transaction Table*.

3.2 Value Table

Each XML element has a "value" as given in the XML file and its code as discussed above. Only the codes of the elements are used during the mining of association rules. However, we need to show the values of the elements in the association rules when the rules are found. The *Value Table* serves the purpose of code-value mapping by storing the "values" of the elements in an array (again, can be considered a vector of vectors) with the elements' encodings to index the rows.

We get the structure of XML file from its DTD and make an index (i.e. encoding) for each tag name. The indexes are stored in a hash table. All occurrences of the same tag name are stored in a vector pointed to by the index. When a new element is encountered while parsing the XML file, it is appended at the end of the row indicated by the indexing code. There are two kinds of elements: leaf nodes that have actual values and intermediate nodes that have "composite values." For a leaf node, we simply put the actual value (character string, for example) in the row. For an intermediate node, the composite value stored in the row cell is represented by a string of its children's positions in respective rows. To be more precise, let V denote the Value Table. The value stored in a cell of the table

V[r][c] = actual value if it is a leaf node,

Otherwise

$$V[r][c] = .L_1.L_2 \ldots L_t$$

where L_i is the column number of the i-th child of the node whose tag's code is r. Here, c is the column number of the row where the value is stored. Note that we put a dot before each L_i just to distinguish the sequence from a numeric value that may be an actual value of a leaf element and to make the process (decomposition of the sequence) simple.

Consider the previous example, the Value Table with the hash header will look like the following shown in Figure 2. Note that the "lengths" of the rows are different because there may be different number of values for each tag. That is the reason we have mentioned the "equivalence" between array and vector of vectors.

Hash header		Value Table		
<CD>	100	.1.1.1.1	.2.2.2.2	.3.1.1.1
<TITLE>	110	Moonlight	Flying	Highland
<ARTIST>	120	Sam Brown	.1.1	
<firstname>	121	Savage		
<lastname>	122	Rose		
<COUNTRY>	130	UK	EU	
<COMPANY>	140	A and M	Polydor	

Figure 2. Value Table with hash header.

Here, each element (tag name) is encoded in a 3-digit number as discussed before and shown in the hash header. All distinct values of the same tag name are stored in the same row pointed to by the code of the tag (e.g. 100 for <TITLE>). If a node is a composite element (e.g. <CD>, or <ARTIST> with sub-elements <firstname> and <lastname>), the "value" of the element is the combination of the location of the "values" of all its children. For example, the third <CD> element is stored in column 3 of the 1st row, which stores for all CD's. Its value is ".3.1.1.1," meaning that it has 4 children (of codes 110, 120, 130, and 140), in columns 3, 1, 1, and 1, respectively. Namely, "Highland," "Sam Brown," "UK," and "A and M."

For the same token, the second <CD> element is stored in column 2 of the row for all CD's. Its value is ".2.2.2.2" indicating that the values of its 4 children are all in column 2 of their respective rows. The second child (an <ARTIST> element) is itself a composite element with value ".1.1" indicating that it has two children in column 1 of the row for <firstname> (first child of <ARTIST>) and column 1 of the row for <lastname> (second child of <ARTIST>). This is shown in the shaded cells in Fig. 2.

3.3 Transaction Table

The *Transaction Table* holds the information of the transaction sets. Because XML is of a multilevel tree

structure, it is a primary task to organize the transaction table in such a way that it will support easy retrieval of information at all levels related to association rules without storing redundant data. The transaction table is information-encoded instead of the traditional transaction table as classical association rule mining algorithms would apply to. Each row in the Transaction Table represents a "transaction," although not in the traditional sense. Columns of the table are the tag names. A cell in the table is an encoded string, which is a concatenation of the position of the element in the hierarchy and its value index. That is, let T be the Transaction Table. T[r][c] represents the item of transaction r with tag name c. The value stored in the cell is

$$A = T[r][c] = t \cdot v$$

where t is the code of the tag and A represents the value of the item in the Value Table cell V[t][v]. The advantage of this representation is the easy cross-reference between V and T and the simplicity of decomposition of the digit sequence to get the indexes. In our implementation, we encode the information into bits, combine them together and convert to an integer just to reduce space. This requires fewer bits than using the object ID or barcode methods. The Transaction Table for the example in our previous discussion is shown in Table 1.

Table 1. Transaction Table for the example XML

tid	Tag name index						
	100	110	120	121	122	130	140
0	1001	1101	1201			1301	1401
1	1002	1102	1202	1211	1221	1302	1402
2	1003	1103	1201			1301	1401

In the table, each cell is a 4-digit number, which is a concatenation of a 3-digit number and a 1-digit number. For example, T[1][120]=1202 is concatenation of 120 and 2, meaning that the code of the item's tag name is 120 (which is <ARTIST>) and the value of the item is stored in V[120][2] (which is .1.1 indicating a composite value with <firstname> and <lastname>).

Using this organization of the Transaction Table and Value Table, we have avoided storing redundant data (same value but with different tag names) and still kept the hierarchical relationships between the elements intact.
Another problem is about missing data (DTD may define an element that can appear zero or more times). When processing the XML data, we should consider the different structures of the transactions including the missing and disordering of its sub-elements appearing in the XML file. The approach we use is to use an array to represent the pattern of a "right" element whose sub-elements are all present and in the order given in the DTD. The actual XML data are checked against the pattern. If a sub-element is missing or not in the "right" order according to the DTD, the "digit" for the sub-element in the encoding would be filled with a special symbol.

Furthermore, we used the term "digit" above in the encoding, but in fact it is an integer so that it can represent a number larger than 9. Hence, the code of an element is a sequence of integers separated by a dot. Each integer is of $\log_{10}(N)+1$ digits where N is the number of children of the node. Decomposing the sequence is just a matter of going through the sequence, only a bit more work than if they were digits.

3.4 Algorithms

Here are the algorithms for encoding of the XML elements and for building the Value Table and Transaction Table.

Encoding

```
get structure information from DTD;
get children from root into a collection of vectors;
for each vector in the collection
  for each element of this vector
    while the element has children
        get its children into a new vector;
        add it to the collection;
        N = number of children;
        encoding each child (i.e. using log10(N)+1),
        and push it into a hash map;
```

Building Value Table and Transaction Table

```
parse XML document to get the DOM tree;
trans = 0;
for each sibling of the first child of root
    trans ++;
    col = V[trans].size;
    if this node is a leaf element
       V[trans][col++] = attribute name
    else
       value = concatenation of the "values" along
               the path from root;
        V[trans][col++] = value;
    T[trans][tag code] = concatenation of code of
                tag name and col of the value in V;
```

4. Mining Association Rules

We use the Apriori algorithm in this paper. It is the very basic and an influential algorithm for mining frequent itemsets for association rules dealing with the presence/absence of items. We shall briefly describe the basic idea of the Apriori algorithm below, and then discuss some modifications so that it becomes more flexible.

4.1 Apriori Algorithm

A set of items is called an *itemset*. An itemset of size k is called a *k-itemset*. An itemset that satisfies the minimum support level is called a *frequent k-itemset*, denoted L_k. The

task of association rule mining is to find frequent k-itemsets of a given k. The Apriori algorithm starts with frequent 1-itemsets, L_1, and then iteratively finds L_{i+1} from L_i until i+1=k. Each iteration step consists of a join operation and a prune operation. The join operation joins L_i with itself to produce a candidate set C_i, which is a superset of L_{i+1} and in general quite large. The prune operation deletes the members in C_i that are not frequent (i.e. those that do not meet the minimum support requirement) to produce L_{i+1}. This pruning significantly reduces the size of C_i and at the same time still guarantees that nothing useful is removed. This is possible because of the Apriori property that says all nonempty subsets of a frequent itemset must also be frequent.

Once the frequent k-itemsets are found, it is straightforward to generate association rules that also satisfy the minimum confidence level:

1. For each frequent itemset L, generate all nonempty subsets of L.
2. For each nonempty subset S of L, output the rule $S \Rightarrow L - S$ if the rule satisfies min-confidence.

It is clear that all association rules generated by this algorithm are of size |L|.

4.2 Modifications to the Algorithm

As mentioned above, the traditional Apriori algorithm is to find the k-frequent itemsets for a given k. That is, all association rules resulted from the algorithm contains k items. However, if we want to find all association rules of sizes from 2 to k, we should have kept those frequent i-itemsets in L_i that were pruned while calculating L_{i+1} but they did satisfy the minimum support in the previous iteration. These i-itemsets can generate association rules of size i although they won't be in rules of size i+1. We added, therefore, an additional data structure (a Collection) to store the frequent i-itemsets during the iteration so that smaller sized association rules will also be found, rather than just the longest possible rules.

Another modification is to allow the user to select the elements he or she is interested in for finding associations. This may not be desirable as far as data mining is concerned (DM is supposed to find something unexpected), but sometimes the user does know for sure some items are not interesting. By selecting only certain items would cut the sizes of the itemsets and result in much faster computation. This can be done by presenting to the user the list of items extracted from the DTD. For a data set given in the next section that has four levels in the hierarchy, it will find over one hundred rules (many of which are not at all interesting to the user), but by selecting three items, the number of rules found is reduced to only 11, and the time spent on the computation is cut by 2/3.

5. Experiments

We have tested the mining algorithm with the encoding method on several small XML files, which may not be very meaningful in the real world, but just to show the feasibility of our approach.

5.1 Example of Music CDs

Here we show an example that is an extension of the example used in the previous sections. In this example, there are nine CDs, the element <ARTIST> many contain the PCDATA (name of the artist) or sub-elements <firstname> and <lastname>, and some elements (like <PRICE> and <YEAR> may be missing from a CD.

```
<?xml version="1.0" ?>
<CATALOG>
   <CD>
      <TITLE>Moonlight</TITLE>
      <ARTIST>Sam Brown</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>A and M</COMPANY>
   </CD>
   <CD>
      <TITLE>Flying</TITLE>
      <ARTIST>
         <firstname>Savage</firstname>
         <lastname>Rose</lastname>
      </ARTIST>
      <COUNTRY>EU</COUNTRY>
      <COMPANY>Polydor</COMPANY>
   </CD>
   <CD>
      <TITLE>Highland</TITLE>
      <ARTIST>Sam Brown</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>A and M</COMPANY>
   </CD>
   <CD>
      <TITLE>Romanza</TITLE>
      <ARTIST>Andrea Bocelli</ARTIST>
      <COUNTRY>EU</COUNTRY>
      <COMPANY>Polydor</COMPANY>
      <PRICE>10.80</PRICE>
      <YEAR>1996</YEAR>
   </CD>
   <CD>
      <TITLE>
        When a man loves a woman
      </TITLE>
      <ARTIST>
         <firstname>Savage</firstname>
         <lastname>Rose</lastname>
      </ARTIST>
      <COUNTRY>EU</COUNTRY>
      <COMPANY>Polydor</COMPANY>
      <PRICE>8.70</PRICE>
      <YEAR>1987</YEAR>
   </CD>
   <CD>
      <TITLE>Romanza2</TITLE>
      <ARTIST>Andrea Bocelli</ARTIST>
```

```
    <COUNTRY>EU</COUNTRY>
    <COMPANY>Polydor</COMPANY>
    <PRICE>10.80</PRICE>
    <YEAR>1996</YEAR>
</CD>
<CD>
    <TITLE>Black angel</TITLE>
    <ARTIST>
        <firstname>Savage</firstname>
        <lastname>Rose</lastname>
    </ARTIST>
    <COUNTRY>EU</COUNTRY>
    <COMPANY>Mega</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1995</YEAR>
</CD>
<CD>
    <TITLE>1999 Grammy Nominees</TITLE>
    <ARTIST>Many</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Grammy</COMPANY>
    <PRICE>10.20</PRICE>
    <YEAR>1999</YEAR>
</CD>
<CD>
    <TITLE>Romanza3</TITLE>
    <ARTIST>
        <firstname>Savage</firstname>
        <lastname>Rose</lastname>
    </ARTIST>
    <COUNTRY>EU</COUNTRY>
    <COMPANY>Polydor</COMPANY>
    <PRICE>10.80</PRICE>
    <YEAR>1996</YEAR>
  </CD>
</CATALOG>
```

The Value Table and Transaction Table for this XML are computed and shown below in Table 2 and Table 3.

If the user did not select any specific tags, the algorithm just finds all association rules satisfying a given support and confidence level. For example, for min-support = 0.3 and min-confidence = 0.7, a total of 38 rules were found. Note that the individual rules may have different support and confidence measures, all above the minimum required. Also note that the total number rules found with our algorithm is more than that found by the traditional Apriori algorithm. The reason is, as mentioned before, that the traditional Apriori algorithm throws away those that satisfy the min-support at iteration i but failed the test at iteration i+1 due to the increasing size of the frequent itemsets.

The 38 rules that were obtained are shown on the next page. Note that they are only for a given min-support and min-confidence requirements. The numbers of rules mined differ for different support and confidence levels.

Table 2. Value Table

Tag index									
100	.1.1.1.1	.2.2.2.2	.3.1.1.1	.4.3.2.2.1.1	.5.2.2.2.2.2	.6.3.2.2.1.1	.7.2.2.3.3.3	.8.4.3.4.4.4	.9.2.2.2.1.1
110	Moonlight	Flying	Highland	Romanza	When a man loves a woman	Romanza2	Black angel	1999 Grammy Nominees	Romanza3
120	Sam Brown	1.1	Andrea Bocelli	Many					
121	Savage								
122	Rose								
130	UK	EU	USA						
140	A and M	Polydor	Mega	Grammy					
150	10.80	8.70	10.90	10.20					
160	1996	1987	1995	1999					

Table 3. Transaction Table

TID	Tag Name Index								
	100	110	120	121	122	130	140	150	160
0	1001	1101	1201			1301	1401		
1	1002	1102	1202	1211	1221	1302	1402		
2	1003	1103	1201			1301	1401		
3	1004	1104	1203			1302	1402	1501	1601
4	1005	1105	1202	1211	1221	1302	1402	1502	1602
5	1006	1106	1203			1302	1402	1501	1601
6	1007	1107	1202	1211	1221	1302	1403	1503	1603
7	1008	1108	1204			1303	1404	1504	1604
8	1009	1109	1202	1211	1221	1302	1402	1501	1601

```
rule  0:  {1996} ⇒ {Polydor}
rule  1:  {1996, 10.80} ⇒ {Polydor}
rule  2:  {Polydor} ⇒ {1996, 10.80}
rule  3:  {1996} ⇒ {10.80}
rule  4:  {10.80} ⇒ {1996}
rule  5:  {Polydor} ⇒ {10.80}
rule  6:  {10.80} ⇒ {Polydor}
rule  7:  {1996, Polydor} ⇒ {10.80}
rule  8:  {10.80} ⇒ {1996, Polydor}
rule  9:  {1996, EU} ⇒ {Polydor}
rule 10:  {Polydor} ⇒ {1996,EU}
rule 11:  {1996,10.80,EU} ⇒ {Polydor}
rule 12:  {Polydor} ⇒ {1996,10.80,EU}
rule 13:  {Polydor, EU} ⇒ {1996, 10.80}
rule 14:  {1996, 10.80} ⇒ {Polydor, EU}
rule 15:  {1996, EU} ⇒ {10.80}
rule 16:  {10.80} ⇒ {1996, EU}
rule 17:  {10.80,EU} ⇒ {1996}
rule 18:  {1996} ⇒ {10.80, EU}
rule 19:  {Polydor, EU} ⇒ {10.80}
rule 20:  {10.80} ⇒ {Polydor, EU}
rule 21:  {10.80, EU} ⇒ {Polydor}
rule 22:  {Polydor} ⇒ {10.80, EU}
rule 23:  {1996,Polydor,EU} ⇒ {10.80}
rule 24:  {10.80} ⇒ {1996, Polydor, EU}
rule 25:  {10.80, EU} ⇒ {1996, Polydor}
rule 26:  {1996, Polydor} ⇒ {10.80, EU}
rule 27:  {10.80} ⇒ {EU}
rule 28:  {EU} ⇒ {10.80}
rule 29:  {1996} ⇒ {EU}
rule 30:  {EU} ⇒ {1996}
rule 31:  {Polydor} ⇒ {EU}
rule 32:  {EU} ⇒ {Polydor}
rule 33:  {10.80, 1996, Polydor} ⇒ {EU}
rule 34:  {EU} ⇒ {10.80, 1996, Polydor}
rule 35:  {<ARTIST>
             <firstname>Savage</firstname>
             <lastname>Rose</lastname>
           </ARTIST>} ⇒ {EU}
rule 36:  {<ARTIST>
             <firstname>Savage</firstname>
             <lastname>Rose</lastname>
           </ARTIST>} ⇒ {Polydor}
rule 37:  {<ARTIST>
             <firstname>Savage</firstname>
             <lastname>Rose</lastname>
           </ARTIST>} ⇒ {EU, Polydor}
```

As mentioned before, we applied different min-support and min-confidence values for this XML data to mine association rules. For these experiments, we did not select any specific tag(s) to restrict the mining process for certain target, rather we let the algorithm to find all rules that satisfy the support and confidence requirements.

The aggregate summery of some other results is shown in Table 4 below.

Table 4. Summary of the experiments

min-support	min-confidence	total # rules	# rules with traditional Apriori algorithm	size of largest rule
0.2	0.3	166	142	5
0.2	0.5	139	115	5
0.2	0.7	16	0	
0.3	0.3	56	44	4
0.3	0.5	56	44	4
0.3	0.7	38	35	4

If the user chose COMPANY, ARTIST, YEAR as the tags of interest, only 11 rules are found out of the total of 166 rules, with min-support = 0.2 and min-confidence = 0.3.

5.2 Mining Involving Missing Data

This XML document contains information about criminal records. The XML data is given below.

```
<crime record>
  <crime No= "0001">
    <criminal>
      <age>23</age>
      <height>6.8</height>
      <weight>140</weight>
    </criminal >
    <time>00:40</time>
    <location>AA</location>
    <tool>pistol</tool>
  </crime>
  <crime No= "0002">
    <criminal>
      <height>6.5</height>
    </criminal >
    <time>13:00</time>
    <location>AB</location>
  </crime>
  <crime No= "0003">
    <criminal>
      <age>35</age>
      <weight>150</weight>
    </criminal >
    <time>13:00</time>
    <location>AQ</location >
  </crime>
  <crime No= "0003">
    <criminal>
      <height>5.5</height>
      <weight>138</weight>
    </criminal >
    <time>2:00</time>
    <location>BB</location>
    <tool>rope</tool>
  </crime>
  <crime No= "0004">
    <criminal>
      <age>30</age>
      <height>6.0</height>
```

```
    <weight>145</weight>
  </criminal >
  <time>13:00</time>
</crime>
<crime No= "0005">
  <criminal>
    <age>35</age>
    <height>6.2</height>
    <weight>150</weight>
  </criminal >
  <time>00:30</time>
  <tool>knife</tool>
</crime>
<crime No= "0006">
  <criminal>
    <weight>145</weight>
  </criminal >
  <time>13:00</time>
  <location>CM</location>
</crime>
<crime No= "0007">
  <criminal>
    <height>6</height>
  </criminal>
  <time>22:00</time>
  <location>ED</location>
  <tool> pistol</tool>
</crime>
<crime No= "0008">
  <criminal>
    <age>32</age>
  </criminal>
  <time>1:00</time>
  <location>ED</location>
    <tool>bomb/tool>
  </crime>
</crime record>
```

There are a lot of missing data in this XML document. Unlike the "transactions" represented in relational format, missing data in our method are also encoded. During the tree node encoding process, if a child node is missing the corresponding digit is filled with 0. Any missing data is encoded as its tag name code followed by 0. Thus different missing data are distinguished with different code. The encoding tree of this example is shown in Figure 3. The Value Table and Transaction Table are shown in Table 5 and Table 6.

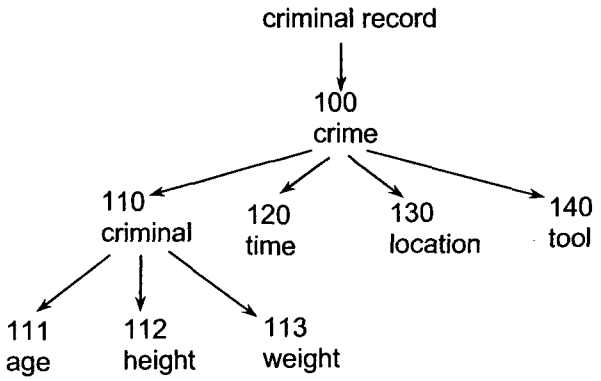


Figure 3. Encoding tree of the criminal record example.

Table 5. Value Table of the Criminal Data

100	.1.1.1.1	.2.2.2.0	.3.2.3.0	.4.3.4.2	.5.2.0.0	.6.1.0.3	.7.2.5.0	.8.2.6.1	.9.5.6.4
110	.1.1.1	.0.2.0	.2.0.2	.0.3.3	.3.4.4	.2.5.2	.0.0.4	.0.4.0	.4.0.0
111	23	35	30	32					
112	6.8	6.5	5.5	6.0	6.2				
113	140	150	138	145					
120	00:40	13:00	2:00	22:00	1:00				
130	AA	AB	AQ	BB	CM	ED			
140	pistol	rope	knife	bomb					

Table 6. Transaction Table for the Criminal Data

TID	Tag Name Index							
	100	110	111	112	113	120	130	140
1	1001	1101	1111	1121	1131	1201	1301	1401
2	1002	1102	1110	1122	1130	1202	1302	1400
3	1003	1103	1112	1120	1132	1202	303	1400
4	1004	1104	1110	1123	1133	1203	1304	1402
5	1005	1105	1113	1124	1134	1202	1300	1400
6	1006	1106	1112	1125	1132	1201	1300	1403
7	1007	1107	1110	1120	1134	1202	1305	1400
8	1008	1108	1110	1124	1130	1204	1306	1401
9	1009	1109	1114	1120	1130	1205	1306	1404

With a minimal support 0.3 and confidence level 0.9, the Apriori algorithm generated association rules that include

<time>13:00</time> ⇒ <tool> </tool>

This rule indicates that when the time of the criminal act is 13:00, the information about the tool used in the criminal act is unknown. Probably the same unknown criminal uses same unknown weapon which is always available around 13:00. Without the capability of processing missing data it would be much harder to find the connection between timing and the missing tool, which may alert for further inspection on a certain feature of the tool. Another interesting issue is to find if two items are always missing together. Without some information attached to the missing items, such the encoding approach we use in this paper, finding the connection between these two missing items would be a challenge, if not impossible.

As mentioned before, we use arrays in our implementation for the Value Table and the Transaction Table. There is a time-space trade-off between using arrays and using vector of vectors. Using arrays is much faster but seems wasting a lot of space as transaction and value holders. However, when we make indexes for tag names, we first get the number of nodes at each level, so we can fix the rage of the index numbers for each level. That is, we can decide how many bits to use to express the tag names. If the XML DOM tree is close to be balanced, using arrays won't waste too much space. Of course, DOM trees are rarely balanced. Using vector of vectors is cleaner conceptually but a lot slower.

6. Summary and Future Work

In this paper we have discussed a method for mining association rules from XML data. Although both data mining and XML are by themselves known technologies if considered alone, the combination of the two is still a research area that is attracting more and more attentions. The major difficult lies on the fact that the XML format is semi-structured and does not particularly suitable for the existing data mining algorithms. Several problems we have to deal with, including (a) items in "transactions" may be cross-levels and spread in different sub-trees, (b) there may be missing data as, and (c) duplicate tag names. An encoding scheme is used to assign each XML element a unique code that is used to index to the Value Table and the Transaction Table. The two tables, as "2-dimensional arrays," store the values of the elements and keep the hierarchical relationships among the elements. Using this internal data structure and indexing method, it is easy to retrieve XML data involved in "transactions." We used a modified Apriori algorithm to find frequent itemsets for mining association rules. In addition to mining rules with data present in the XML documents, our encoding method also allows mining with missing data. The method was implemented and tested on some simple XML files and produced correct results. We are currently testing on large XML files to study the performance of the method.

For future work, we plan to study ways of applying other data mining algorithms to XML data using the encoding approach and data structures. We will also be working on formulating DTD/Schema as an XML "standard" for data mining.

References

[1] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases," in *Proceedings of the ACM-SIGMOD Intl. Conf. on Management of Data (SIGMOD'93)*, 207-216, Washington, DC, July 1993.

[2] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo, "Fast discovery of association rules," in Fayyad, Piatesky-Shapori, Smyth, and Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, 307-328, AAAI/MIT Press, 1996.

[3] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proceedings of the Intl. Conf. on Very Large Databases (VLDB'94)*, 487-499, Santiago, Chile, Sept. 1994.

[4] Y. Aumann and Y. Lindell, "A statistical theory for quantitative association rules." *Proc. 1999 Int. Conf. Knowledge Discovery and Data Mining (KDD'99)*, San Diego, CA, 261-270, Aug. 1999.

[5] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur, "Dynamic itemset counting and implication rules for market basket analysis," in Proceedings *of the Intl. Conf. on Very Large Databases (VLDB'97)*, 265-276, Tucson, AZ, May 1997.

[6] A. G. Buchner, M. Baumgarten, M. D. Mulvenna, S. S. Anand, "Data mining and XML: current and future issues," *Proceedings of the First International Conference on Web Information Systems Engineering (WISE'00)*, 127-131, Hong Kong, 2000.

[7] Edmond, "XMLMiner, XMLRule and metarules white paper," Sciento Inc. April 2002.

[8] H. Lu., L. Feng, and J. Han, "Beyond intra-transaction association analysis: mining multi-dimensional inter-transaction association rules," *ACM Transactions on Information Systems*, 423-454, Vol. 18, No. 4, October 2000.

[9] J. Han, and M. Kamber, *Data Mining, Concepts and Techniques*, Morgan Kaufmann, CA, USA, 2001.

[10] J. Han and Y. Fu, "Discovery of multiple-level association rules from large databases," in Proceedings *of the Intl. Conf. on Very Large Databases (VLDB'95)*, 420-431, Zürich, Switzerland, Sept. 1995.

[11] J. Han and Y. Fu, "Mining multiple-level association rules in large database," *IEEE Transactions on Knowledge and Data Engineering*, 11:798-804, 1999.

[12] J. Han, L. V. S. Lakshmanan, and R. T. Ng, "Constraint-based, multidimensional data mining." *COMPUTER*, 32(8): 46-50, 1999.

[13] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *Proc. ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD'00)*, Dallas, TX, May 2000.

[14] G. Hu, "Mining XML documents," in *Proceedings of the 2001 IEEE International Conference on Systems, Man, and Cybernetics* (SMC 2001), Tucson, Arizona, Oct. 7-10, 2001.

[15] G. Hu, Y. Liu and Q. Huang, "Mining association rules from XML data," *Proceedings of the 31st Intl. Conf. on Computers and Industrial Engineering,* pp. 44-49, San Francisco, CA, Feb. 2-4, 2002.

[16] IBM, "XML Parser for Jave," in *IBM Alphaworks* [online]. Available: http://www.alphaworks.ibm.com/tech/xml4j.

[17] J. W. Lee, W. Lee, and W. Kim, "Preparation of semantics-based XML mining," In *Proceedings of the IEEE Conference on Data Mining (ICDM 2001)*, San Jose, Dec. 2001.

[18] J. W. Lee, "XML Document Analysis for Semantics-Based XML Mining," *Journal of the Korea Information Science Society (KISS)*, 2001.

[19] Marek, W., Maciej, Z, "Hash-Mine: a new framework for discovery of frequent itemsets," *Proc. Of 2000 ADBIS-DASFAA Conference*, Prague, Czech Republic, 2000.

[20] H. Minnalia, H. Toivonen, and A. I. Verkamo, "Efficient algorithms for discovering association rules," *Data Mining and Knowledge Discovery (KDD94)*, 181-192, Seattle, WA, July 1994.

[21] R. Páircéir, S. McClean, and B. Scotney, "Discovery of multi-level rules and exceptions from a distributed database," *ACM Conference on KDD 2000*, Bonston, MA, USA, 2000.

[22] J. S. Park, M. S. Chen, and P. S. Yu, "An effective hash-based algorithm for mining association rules," in *Proceedings of the ACM-SIGMOD Intl. Conf. on Management of Data (SIGMOD'95)*, 175-186, San Jose, CA, May 1995.

[23] J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, and D. Yang, "H-Mine: hyper-structure mining of frequent patterns in large databases," in *Proc. Int. Conf. on Data Mining (ICDM'01)*, San Jose, CA, Nov. 2001.

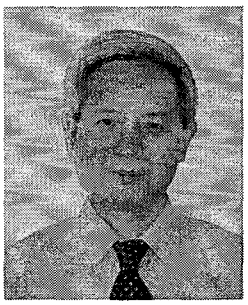
[24] A. Savasere, E. Omiecinski, and S. Navathe, "An efficient algorithm for mining association rules in large databases," in Proceedings *of the Intl. Conf. on Very Large Databases (VLDB'95)*, 432-443, Zürich, Switzerland, Sept. 1995.

[25] "SAX – Simple API for XML," SAX web site, online: http://www.saxproject.org.

[26] R. Srikant and R. Agrawal, "Mining generalized association rules." In *VLDB'95*, pp. 407-419, Zürich, Switzerland, Sept. 1995.

[27] H. Toivonen, "Sampling large databases for association rules," in Proceedings *of the Intl. Conf. on Very Large Databases (VLDB'96)*, 134-145, Bombay, India, Sept. 1996.

[28] XML Working Group of W3C, "Document Object Model (DOM)," online: http://www.w3.org/dom.



Gongzhu Hu received Bachelor of Science in Numeric Analysis from Tsinghua University, China, Master of Science degree in Computer Science from University of Wisconsin, Madison, and Ph. D. in Computer Science from Michigan State University. He is currently a professor and chair of Department of Computer Science, Central Michigan University. His research interests include databases, data mining, programming languages, distributed systems, and image processing. He has published over 80 technical articles in journals and conference proceedings. He has chaired two international conferences. Dr. Hu is a member of IEEE and the IEEE Computer Society, the Association for Computing Machinery (ACM), International Society of Computers and Their Applications (ISCA), and International Association of Computer and Information Science (ACIS). He serves on the Editorial Board of the IJCIS Journal.

Yan Liu received Bachelor of Science degree in computer science from Sichuan University, China, and Master of Science degree in computer science from Central Michigan University. Her research in interests are in databases, data mining, and bioinformatics.

Qiong Huang received Bachelor of Science degree in Physics from Sichuan University, China, Master of Science degree in Physics from Central Michigan University, and is pursuing his Ph.D. degree in physics at the University of Michigan, Ann Arbor. His research interests include optics, optical instrumentation, computational physics, and computer applications in sciences.