ISSN:0975-9646

Application of Web Mining with XML Data using XQuery

Roop Ranjan,Ritu Yadav ,Jaya Verma

Department of MCA,ITS Engineering College,Plot no-43, Knowledge Park 3,Greater Noida

***Abstract*-In recent years XML has become very popular for representing semi structured data and a standard for data exchange over the web. Mining XML data from the web is becoming increasingly important. Several encouraging attempts at developing methods for mining XML data have been proposed. However, efficiency and simplicity are still a barrier for further development. Normally, pre-processing or post-processing are required for mining XML data, such as transforming the data from XML format to relational format. In this paper, we show that extracting association rules from XML documents without any pre-processing or post-processing using XQuery is possible and analyze the XQuery implementation of the well-known Apriori algorithm. In addition, we suggest features that need to be added into XQuery in order to make the implementation of the Apriori algorithm more efficient.**

***Keywords*: XQuery, XML, Association Rule mining, Web mining, Apriori algorithm**

I. INTRODUCTION

The web is rich with information. However, the data contained in the web is not well organized which makes obtaining useful information from the web a difficult task. The successful development of eXtensible Markup Language (XML) [1] as a standard to represent semi structured data makes the data contained in the web more readable and the task of mining useful information from the web becomes feasible. Although tools for mining information from XML data are still in their infancy, they are starting to emerge. As mentioned in [13], the fast growing amount of available XML data, raises a pressing need for languages and tools to manage collections of XML documents, as well as to mine interesting information from them. There are developments like Xyleme [2, 3] which is a huge warehouse integrating XML data from the web, and also vendors of data management tools such as Microsoft, Oracle and IBM, who have focused on incorporating XML technologies in their products. Therefore, it is essential that direct techniques for mining XML data are developed. The query language XQuery [4] was proposed by the W3C [5] and is currently in “last call” status. The purpose of XQuery is to provide a flexible way to extract XML data and provide the necessary interaction between the web world and database world. XQuery is expected to become the standard query language for extracting XML data from XML documents. Therefore, if we can mine XML data using XQuery, then we can integrate the data mining technique into XML native databases. So, we are interested to know whether XQuery is expressive enough to mine XML data. One data mining technique that has proved popular is association rule mining[11, 12] It finds associations between items in a database. In this paper, we show that XML data can be mined using XQuery and discuss the XQuery implementation of the well-known Apriori algorithm. Moreover, we discuss other useful capabilities that need to be added into XQuery to make association rule mining efficient. The outline of this paper is as follows. In section II, we discuss the related work. In section III, we discuss the basic concept of association rule mining. In section IV, we describe the XQuery implementation of the Apriori algorithm that is used to mine XML data in order to discover association rules. In section V, we analyze the performance of the XQuery implementation. In section VI, we discuss the features that can be added into XQuery in order to make the implementation of the Apriori algorithm more efficient. We conclude this paper and discuss the future direction of our research in section 7.

II. RELATED WORK

Algorithms for mining association rules from relational data have been well developed. Several query languages have been proposed, to assist association rule mining such as [15, 16]. The topic of mining XML data has received little attention, as the data mining community has focused on the development of techniques for extracting common structure from heterogeneous XML data. For instance, [19] has proposed an algorithm to construct a frequent tree by finding common sub trees embedded in the heterogeneous XML data. On the other hand, some researchers focus on developing a standard model to represent the knowledge extracted from the data using XML. For example, the Predictive Model Markup Language (PMML) [6] is an XML-based language, which provides a way for applications to define statistical and data mining models and to share models between PMML compliant applications. To date, mining XML documents requires mapping the data to the relational data model and using techniques designed for relational databases to do the mining. For instance, the XMINE operator has been introduced by Braga et al. [13] for extracting association rules from XML documents, where mapping the XML data to a relational structure is required before mining is performed.

III. ASSOCIATION RULES

In this section we overview the basic concepts of association rule mining. We refer the reader to [11, 12] for further details. Association rule mining was first introduced by Agrawal et al.[11], and was used for market basket analysis. The problem of mining association rules can be explained as follows: There is the itemset I = i1, i2,. . ., in, where I is a set of n distinct items, and a set of transactions D, where each transaction T is a set of items such that T _ I. Table 1 gives an example where a database D contains a set of transactions T, and each

TABLE I AN EXAMPLE DATABASE

tid	items
1	{bread, butter, milk}
2	{bread, butter, milk, ice cream}
3	{ice cream, coke}
4	{battery, bread, butter, milk}
5	{bread, butter, milk}
6	{battery, ice cream, bread, butter}

An association rule is an implication of the form X) Y, where X, Y _ I and X \ Y = ;.The rule X) Y has supports in the transaction set D if s% of transactions in D contain X [Y. The support for a rule is defined as support(X[Y). The rule X) Y holds in the transaction set D with confidence c if c% of transactions in D that contain X also contain Y. The confidence for a rule is defined as support(X[Y)/support(X). For example, consider the database in table 1. When people buy bread and butter, they also buy milk in 66% of the cases and 80% of the transactions with bread and butter also contain milk. Such a rule can represented as "bread,butter)milk|support=0.66,confidence=0.8" Not all the rules found are useful and the number of rules generated maybe enormous. Therefore, the task of mining association rules is to generate all association rules that have support and confidence greater than the user-defined minimum support (minsup) and minimum confidence (minconf) respectively. An itemset with minimum support is called the large (or frequent) itemset. The rule X) Y is a strong rule iff X[Y is in the large itemset and its confidence is greater than or equal to minconf. Normally, the task of mining association rules can be decomposed into two sub-tasks:

1. Discover all large itemsets in the set of transactions D. In section 2.1, we give the algorithm Apriori for solving this problem.

2. Use the large itemsets to generate the strong rules. The algorithm for this task is simple. For every large itemset l1, find all large itemsets l2 such that l2 _ l1 where support(l1 [l2) / support(l2) _ minconf. For every such large itemset l2, output a rule of the form l2) (l1 - l2). The performance of mining association rules is mainly dependent on the large itemsets discovery process (step 1), since the cost of the entire process comes from reading the database (I/O time) to generate the support of candidates (CPU time) and the generation of new candidates (CPU time). Therefore, it is important to have an efficient algorithm for large itemsets discovery.

A. Algorithm Apriori

The Apriori algorithm [12] uses a bottom-up breadth first approach to finding the large itemsets. It starts from large 1-itemsets and then extends one level up in every pass until all large itemsets are found. For each pass, say pass k, there are three operations. First, append the large (k-1)-itemsets to L. Next, generate the potential large k-itemsets using the (k-1)-itemsets. Such potential large itemsets are called candidate itemsets C. The candidate generation procedure consists of two steps:

1. Join step – generate k-itemsets by joining lk−1 with itself.

2. Prune step – remove the itemset X generated from the join step, if any of the subsets of X is not large. Since any subset of a large itemset must be large. This can be written formally as follows:

C0 = {X[Y | X, Y 2 lk−1, |X[Y|=k+1 }

C = {X 2 C0, 8 Y _ X | |Y| = k-1 and Y 2 lk−1}

In the last operation, we select the itemset X from the candidate itemsets where support(X) _ minsup.

Figure 1 gives the general Apriori algorithm and table 2 summarizes the notation used in the algorithm.

Algorithm *Apriory Algorithm*

Input *A Database D and minimum support minsup*

Output *All large itemsets*

```
$L_k$=Ø; k=0;
$C_{1=}$ All distinct items in D
$L_{1=}$ large itemsets in $C_1$
While $L_{k+1}$ is not empty
$C_{k+1}$=Candidate-gen($L_k$)
$L_{k+1}$=large itemsets in $C_{k+1}$
k++
return U L
```

Fig. 1: Algorithm Apriori

TABLE II NOTATION

Notation	Definition
k-itemset	An itemset having k items
C_k	Set of candidate k-itemsets
L_k	Set of large k-itemsets

We should mention here that there exists other algorithms for generating large itemsets such as [9, 10,14, 17, 21]. We have chosen this one because it is easyto understand. We are in the process of implementing the others but this work is outside the scope of this paper.

IV. XQUERY EXPRESSION FOR MINING ASSOCIATION RULES FROM XML DATA

In this section, we introduce association rules from XML data and give an example of association rule mining using XQuery. For the purpose of the following discussion, we assume that the reader has some knowledge of XQuery and refer the reader to [4] for further details. We refer to the sample XML document, depicted in Fig. 2 where information about the items purchased in each transaction are represented. For example, the set of transactions are identified by the tag <transactions> and each transaction in the transactions set is identified by the tag <transaction>. The set of items in each transaction

```
<transactions>
      <transaction id="1">
                  <items>
                      <item>a</item>
                      <item>d</item>
                      <item>e</item>
                  </items>
     </transaction>
     <transaction id="2">
```

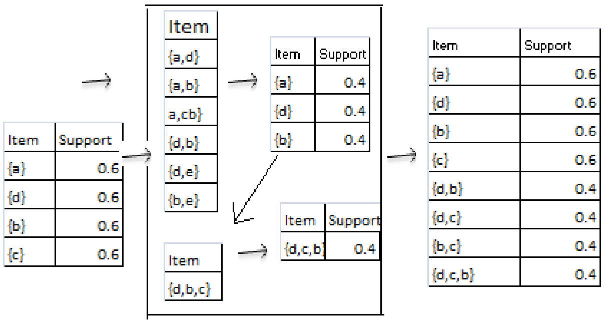
```
                <items>
                     <item>b</item>
                     <item>c</item>
                     <item>d</item>
                </items>
     </transaction>
     <transaction id="3">

                <items>
                     <item>a</item>
                     <item>c</item>

                </items>
     </transaction>
     <transaction id="4">
                <items>
                     <item>b</item>
                     <item>c</item>
                     <item>d</item>
                </items>
     </transaction>
     <transaction id="5">
                <items>
                     <item>a</item>
                     <item>b</item>
                </items>
     </transaction>

</transactions>
```

Fig. 2: Transaction document (transactions.xml)

are identified by the tag <items> and an item is identified by the tag <item>. Consider the problem of mining all association rules among items that appear in the transactions document as shown in Figure 2. With the understanding of traditional association rule mining we expect to obtain the large itemsets document and association rules document from the source document. Let us set the minimum support (minsup) = 40% and minimum confidence (minconf) = 100%. We now present the XQuery expression which computes the association rules.

```
let $src := document("/transactions.xml")//items
let $minsup := 0.4
let $total := count($src) * 1.00
let $C := distinct-values($src/*)
let $l :=(for $itemset in $C
          let $items := (for $item in $src/*
                         where $itemset = $item
                         return $item)
           let $sup := (count($items) * 1.00) div $total
where $sup >= $minsup
return <largeItemset>
                      <items> {$itemset} </items>
                      <support> {$sup} </support>
            </largeItemset>)
let $L := $l
return <largeItemsets> {apriori($l, $L,$minsup, $total, $src)}
</largeItemsets>
```

The computation of the above expressions begins with several let clauses where we specify the data source $src, the support threshold $minsup and total number of transaction $total in the data source (transactions.xml). Next, it starts to generate the candidate 1-itemsets $C. Once the candidate 1-itemsets are available, we are ready to generate the large 1-itemset $l by scanning through the transactions document to obtain the support value of each candidate 1-itemset and remove the one which is less than minsup. The variable $L is used to collect all the large itemsets in the transaction document. Finally, it will pass the information (e.g. $l, $L, $minsup, $total) to the recursive function apriori in the return clause to generate the other large itemsets. The XQuery expressions for the user-defined function *apriori* are as follows:



Item	Support
{a}	0.6
{d}	0.6
{b}	0.6
{c}	0.6

Item
{a,d}
{a,b}
a,cb}
{d,b}
{d,e}
{b,e}

Item	Support
{a}	0.4
{d}	0.4
{b}	0.4

Item
{d,b,c}

Item	Support
{d,c,b}	0.4

Item	Support
{a}	0.6
{d}	0.6
{b}	0.6
{c}	0.6
{d,b}	0.4
{d,c}	0.4
{b,c}	0.4
{d,c,b}	0.4

Fig. 3: Process of generating the large itemsets document in relational representation

```
define function apriori(element $l, element $L, element
$minsup, element $total, element $src) returns element {
      let $C := removeDuplicate(candidateGen($l))
      let $l := getLargeItemsets($C, $minsup, $total, $src)
      let $L := $l union $L
      return if (empty($l)) then
                  $L
             else
               apriori($l, $L, $minsup, $total, $src)
}
```

The function apriori is called once in each level,it generates the candidate set C in the current level by joining the large itemsets in the previous level. It then removes the unnecessary itemsets from C and obtains the large itemsets by reading the database to calculate the support. Figure 3 illustrates the process of generating large itemsets. We refer the interested reader to [20] for the details of the XQuery implementation of Apriori algorithm. Fig. 4 shows all the large itemsets generated by our XQuery queries.

```
<largeitemsets>
          <largeitemset>
                <items>
                     <item>a</item>
                </items>
                <support>0.6</support>
          </largeitemset>
          <largeitemset>
                <items>
                     <item>d</item>
                </items>
                <support>0.6</support>
          </largeitemset>
          <largeitemset>
                <items>
                <item>b</item>
            </items>
            <support>0.6</support>
        </largeitemset>
        <largeitemset>
            <items>
                <item>c</item>
            </items>
            <support>0.6</support>
        </largeitemset>
        <largeitemset>
            <items>
                <item>d</item>
                <item>b</item>

            </items>
            <support>0.4</support>
        </largeitemset>
        <largeitemset>
            <items>
                <item>d</item>
                <item>c</item>
            </items>
            <support>0.4</support>
        </largeitemset>
        <largeitemset>
            <items>
                <item>b</item>
                <item>c</item>
            </items>
            <support>0.4</support>
        </largeitemset>
        <largeitemset>
            <items>
                <item>d</item>
                <item>c</item>
                <item>b</item>
            </items>
            <support>0.4</support>
        </largeitemset>
</largeitemsets>
```

Fig. 4: Large Itemsets document (large.xml)

Now that we have explained how to generate the large itemset document, we can move on to discuss how to compute the association rules from the large itemsets. We present the following XQuery expression that computes the rules document.

```
let $minconf := 1.00
let $src := document(“/large.xml”)//largeItemset
for $itemset1 in $src
let $items1 := $itemset1/items/*
    for $itemset2 in $src
        let $items2 := $itemset2/items/*
        where count($items1) > count($items2) and
            count(commonIts($items1, $items2)) =
            count($items2) and $itemset1/support div
            $itemset2/support _ $minconf
        return <rule support =“{$itemset1/support}”
                confidence = “{($itemset1/support*1.0) div
                        ($itemset2/support*1.0)}”>
            <antecedent> {$items2} </antecedent>
            <consequent>
                {removeItems($items1,$items2)}
            </consequent>
            </rule>
```

The above expression can be explained as follows.
For each large itemset X in the large itemsets document, we look for other itemsets Y in the large itemsets document such that Y _ X and support(X[Y) / support(Y) _ minconf. The association rules generated by the above queries are shown in Fig. 5.

```
<rules>
    <rule support=0.4 confidence=”1.0”>
        <antecedent>
                <item>d</item>
                <item>b</item>
            </ antecedent>
            <consequent>
                <item>c</item>
            </consequent>
    </rule>
    <rule support=0.4 confidence=”1.0”>
        <antecedent>
                <item>d</item>
                <item>c</item>
            </ antecedent>
            <consequent>
                <item>b</item>
            </consequent>
    </rule>
    <rule support=0.4 confidence=”1.0”>
        <antecedent>
                <item>b</item>
                <item>c</item>
            </ antecedent>
            <consequent>
                <item>d</item>
            </consequent>
    </rule>
</rules>
```

Fig. 5: Association Rules document(rules.xml)

As we can see from Fig. 5, the data inside the rules document is self describing. For example, the set of rules are identified by the tag <rules> and each rule is identified by the tag <rule> with two attributes support and confidence to describe the strength of the rule. Inside the tag <rule>, there are two sub-tags <antecedent> and <consequent> which are used to identify the items, antecedent or consequent of the rule.

V. PERFORMANCE ANALYSIS

TABLE III DESCRIPTION OF DIFFERENT DATASETS

Datasets	Number of transactions	Average number of items per transaction
dataset-1	100	11
dataset-2	500	10
dataset-3	1000	10

*30 distinct items and maximum items in each transaction is 20

We study the performance for the XQuery implementation of the Apriori algorithm using the datasets summarized in table 3. The datasets are generated randomly depending on the number of distinct items, the maximum number of items in each transaction and the number of transactions. Our experiment is performed on the XQuery engine inside the native XML database X-Hive/DB 4.1 [7] installed on an Intel Pentium 4, 1.8MHz system running Window XP Professional with 256 MB main memory. Fig. 4. shows the results of our experiment.

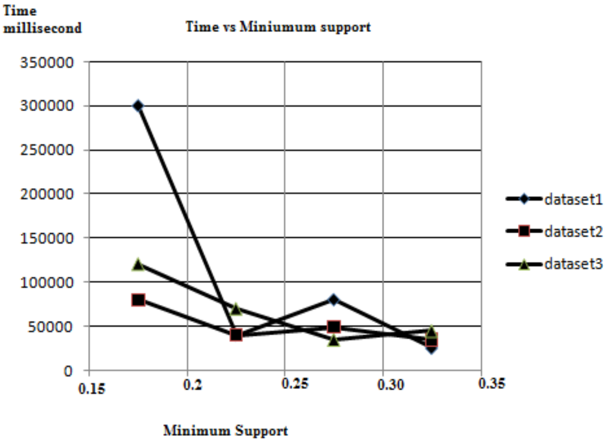


Fig. 4a: Time vs Minimum Support

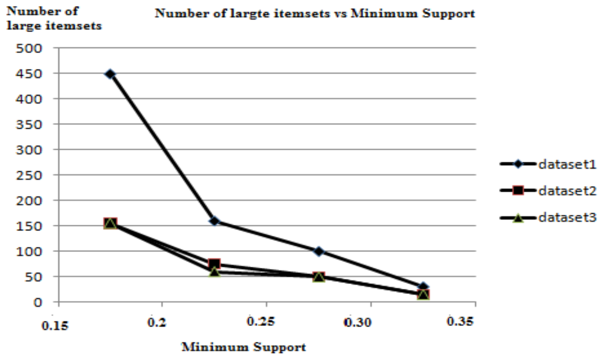


Fig. 4b: Number of large itemsets vs Minimum Support

We see that the performance of the XQuery implementation is dependent on the number of large itemsets found and the size of the dataset. For example, the number of large itemsets found from dataset-1 is much more than dataset-2 and dataset-3 with minimum support 20%. The running time for dataset-1 with minimum support 20% is much higher than the running time of dataset-2 and dataset-3, since the number of large itemsets found for dataset-1 is about 2.4 times more than the other datasets.

We also notice that the majority of the time for the XQuery implementation is spent in counting support to find large itemsets. The XQuery implementation requires that for each itemset in the Candidate set, it will read the database once to obtain the support value. Therefore, the number of times needed to scan the database to obtain the support count for finding large itemsets is $O(2^{l})$, where l is the length of the maximal large itemset. This is very inefficient compared with the implementation in other languages. For instance, the C++ implementation of the Apriori algorithm by Goethals [8] scans the database to obtain the support count for finding large itemsets l times, so only needs to scan the database once in each pass. The reason why the number of times needed to scan the database for the XQuery implementation is significantly higher than the C++ implementation is that, in XQuery we cannot store the support value for the itemsets but instead return it in a query. Therefore, it is difficult to read a record in the database and update the support count for the itemsets using XQuery. This means that the whole database needs to be read for each candidate itemset. The results of our experiments shown that XQuery is more suitable for mining data from small datasets.

VI. EXTENSION TO XQUERY

As discussed in the last section, the number of times needed to read the database for the XQuery implementation of the Aprori algorithm is much more than the C++ implementation. To improve the performance of the XQuery implementation of the Apriori algorithms, operations such as update and insert must be added into XQuery. Tatarinov et al. [18] also highlight the need for update capabilities for modifying XML documents, and also for propagating changes through XML views and for expressing and transmitting changes to documents. With the update capabilities added into XQuery, it is possible to reduce the number of times needed to read the database in the large itemset discovery step. For example, assume that there is an update operation in XQuery. For each level of computing large itemsets, we create a XML document which contains all the candidate itemsets with the corresponding support at the current level. Then when reading each record t from the source XML document, find itemsets from the candidate set document which are the subset of the itemsets in t and increment their corresponding support count. After reading the whole source XML document, large itemsets in the current level are found and candidate itemsets in the next level can be generated. We can see that the number of times needed to read the database is reduced significantly, that is it is now equal to l, rather than $O(2^{l})$, where l is the length of the maximal large itemset.

VII. CONCLUSION

In this paper, we show how XQuery can be used to extract association rules from XML data and we analyze the performance of the XQuery implementation of the Apriori algorithm. Our results show that the XQuery implementation of the Apriori algorithm is not efficient, and the number of times it needs to read the database (i.e., I/O time) is much more than a C++ implementation. In order to improve the efficiency of the XQuery implementation, we suggest an update operation is added into XQuery. With the update operation added into XQuery, not only the performance of the XQuery implementation of the Apriori algorithm can be improved, but it is also possible to propagate changes through XML views and express and transmit changes to documents [18].

Although this research is a good starting point, there are still many issues that remain open. One of the issues concerns the structure of the XML data. Our algorithm currently mines any set of items that we can write a path expression for. However, the structure of the XML data can be more complex or irregular than this, so identifying the mining context of such XML data becomes difficult. Therefore, to simplify the task of identifying the context, a set of transformations of the XML data might be required. Another direction is to extend our problem by mining association rules from more than one XML document, where documents have different structure.

Our current and future research in this area focuses on investigating how much the XQuery implementation of the Apriori algorithm can be improved with the update operation Also, we are interested to know if it is possible to implement algorithms like FPgrowth [14] where candidate generation is not needed.

REFERENCES

[1] World Wide Web Consortium. Extensible Markup Language (XML) 1.0 (Second Edition) W3C Recommendation. http://www.w3.org/XML.

[2] Xyleme. http://www.xyleme.com.

[3] Lucie Xyleme. A dynamic warehouse for XML data of the web. IEEE Data Engineering Bulletin, 2001.

[4] World Wide Web Consortium. XQuery 1.0: An XML Query Language (W3C Working Draft). http://www.w3.org/TR/2002/WDxquery-20020816, Aug. 2002.

[5] World Wide Web Consortium. http://www.w3.org.

[6] PMML 2.1 Predictive Model Markup Language. http://www.dmg.org, March 2003.

[7] X-Hive/DB. http://www.x-hive.com.

[8] Frequent Pattern Mining Implementations. http://www.cs.helsinki.fi/u/goethals/software.

[9] R. C. Agarwal, C. C. Aggarwal, and V. V. V. Prasad. Depth first generation of long patterns. In Proceedings of the ACM SIGKDD Conference,2000.

[10] R. C. Agarwal, C. C. Aggarwal, and V. V. V. Prasad. A tree projection algorithm for Generation of frequent itemsets. volume 61, pages 350–371, 2001.

[11] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In P. Buneman and S. Jajodia,
editors,SIGMOD93, pages 207–216, Washington, D.C., USA, May 1993.

[12] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In J. B. Bocca, M. Jarke, and C. Zaniolo, editors,
Proceedings of 20th International Conference on Very Large Data Bases, pages 487–499, Santiago,Chile, September 12-15 1994.

[13] D. Braga, A. Campi, M. Klemettinen, and P. L. Lanzi. Mining association rules from xml data. In Proceedings of the 4th International
Conference on Data Warehousing and Knowledge Discovery (DaWaK 2002), September 4-6, Aixen-Provence,France 2002.

[14] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In W. Chen, J. Naughton, and P. A. Bernstein, editors, 2000 ACM SIGMOD Intl. Conference on Managemen of Data, pages 1–12. ACM Press, 05 2000.

[15] T. Imielinski and A. Virmani. MSQL: A query language for database mining. 1999.[16] R. Meo, G. Psaila, and S. Ceri. A new SQLlike operator for mining association rules. In The VLDB Journal, pages 122–133, 1996.

[17] J. Pei, J. Han, and R. Mao. CLOSET: An Efficient algorithm for mining frequent closed itemsets. In ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery,pages 21–30, 2000.

[18] I. Tatarinov, Z. G. Ives, A. Y. Halevy, and D. S. Weld. Updating XML. In SIGMOD Conference, 2001.[19] A. Termier, M.-C. Rousset, and M. Sebag. Mining XML data with frequent trees. In DBFusion Workshop'02, pages 87–96.

[20] J. W. W. Wan and G. Dobbie. Extracting association rules from XML documents using XQuery. In Proceedings of Fifth International Workshop on Web Information and Data Management,New Orleans, LA, USA.

[21] M. J. Zaki. Generating non-redundant Association rules. In Knowledge Discovery and Data Mining, pages 3443, 2000.