

Association Rule Mining from XML Data

Qin Ding and Gnanasekaran Sundarraj

Computer Science Program

The Pennsylvania State University at Harrisburg

Middletown, PA 17057, USA

Email: {qding, gxs241}@psu.edu

Abstract - With the growing usage of XML in the World Wide Web and elsewhere as a standard for the exchange of data and to represent semistructured data, there is an imminent need for tools and techniques to perform data mining on XML documents and XML repositories. In this paper we look at the various approaches for association rule mining from XML data. We present a Java-based implementation of the Apriori and the FP-Growth algorithms for this task and compare their performances. We also compare the performance of our implementation with an XQuery-based implementation.

Keywords: Data Mining, Association Rule Mining, XML Data.

I. INTRODUCTION

Advances in data collection and storage technologies have led organizations to store vast amounts of data pertaining to their business activities. Extracting “useful” information from such huge data collections is of importance in many business decision making processes. Such an activity is referred to as *Data Mining* or *Knowledge Discovery in Databases (KDD)* [7]. Data mining includes tasks such as *Classification*, *Similarity Analysis*, *Summarization*, *Association Rule Mining*, *Sequential Pattern Discovery*, and so forth [7].

The task of association rule mining is to find correlation relationships among different data attributes in a large set of data items, and this has gained lot of attention since its introduction [1]. Such relationships observed between data attributes are called *association rules* [1]. A typical example of association rule mining is the *market basket analysis*. Consider a retail store that has a large collection of items to sell. Business decision regarding discount, cross-selling, grouping of items in different aisles, and so on needs to be made often in order to increase the sales and hence the profit. This inevitably requires knowledge about past transaction data that gives the buying habits of customers. The association rules in this case will be of the form “customers who bought item *A* also bought

item *B*,” and association rule mining is to extract such rules from the given transaction data history.

Explosive use of World Wide Web to buy and sell items over the Internet has led to similar data mining requirements from online transaction data. In an attempt to standardize the format of data exchanged over the Web and to achieve interoperability between the different technologies and tools involved, World Wide Web Consortium (W3C) introduced *eXtensible Markup Language (XML)* [6]. XML is a simple but very flexible format derived from *Standard Generalized Markup Language (SGML)* [6], and has been playing an increasingly important role in the exchange of wide variety of data over the Web. Even though it is a markup language much like the *Hyper Text Markup Language (HTML)* [6], XML was designed to describe data and to focus on what the data is, whereas HTML was designed to display data and to focus on how the data looks on the Web browser. A data object described in XML is called an *XML Document*. XML also plays the role of a metalanguage, and allows document authors to create customized markup languages for limitless different types of documents, making it a standard data format for online data exchange. This growing usage of XML has naturally resulted in the increasing amount of available XML data which raises the pressing need for more suitable tools and techniques to perform data mining on XML documents and XML repositories.

In this paper we study the various approaches that have been proposed for association rule mining from XML data, and present a Java-based implementation for the two well-known algorithms for association rule mining: *Apriori* [2] and *Frequent Pattern Growth (FP-Growth)* [8]. The rest of this paper is organized as follows. In Section II we describe the basic concepts and definitions for association rule mining. In this section we also explain the above two algorithms briefly. In Section III we detail the various approaches to association rule mining from XML data, and in Section IV we present our Java-based implementation for this task. Finally we give the experimental results in Section V before concluding in Section VI.

II. ASSOCIATION RULE MINING

The first and the foremost step in association rule mining is to identify *frequent sets*, the sets of items that occur together often enough to investigate further. Because of the exponential scale of the search space, this step is undoubtedly the most demanding in terms of computational power and in the need for the use of efficient algorithms and data structures. These factors really become important when dealing with real time data.

Next we give the basic concepts and definitions for association rule mining, and then briefly explain the Apriori and FP-Growth algorithms. Note that when describing the data, we use the terms “transaction” and “item” in the rest of the paper just to be in line with our examples.

A. Basic Concepts and Definitions

Let $\mathcal{I} = \{i_1, i_2, i_3, \dots, i_m\}$ be a set of items. Let D be the set of transactions where each transaction $T \in D$ is a set of items such that $T \subseteq \mathcal{I}$. An association rule is of the form $A \Rightarrow B$ where $A \subseteq \mathcal{I}$, $B \subseteq \mathcal{I}$, and $A \cap B = \emptyset$. The set of items A is called *antecedent* and the set B the *consequent*. Such rules are considered to be interesting if they satisfy some additional properties, and two properties have been mainly used in association rule mining: *Support* and *Confidence*. Though other measures have been proposed for this task in the literature, we will consider only the above two in this paper [3], [11]. Support s for a rule $A \Rightarrow B$, denoted by $s(A \Rightarrow B)$, is the ratio of the number of transactions in D that contain all the items in $A \cup B$ to the total number of transactions in D . That is,

$$s(A \Rightarrow B) = \frac{\sigma(A \cup B)}{|D|}, \quad (1)$$

where the function σ of a set of items A denotes the number of transactions in D that contain all the items in A . $\sigma(A)$ is also called the *support count* of A . Confidence c for a rule $A \Rightarrow B$, denoted by $c(A \Rightarrow B)$, is the ratio of the support count of $A \cup B$ to that of the antecedent A . That is,

$$c(A \Rightarrow B) = \frac{\sigma(A \cup B)}{\sigma(A)}. \quad (2)$$

For a user-specified minimum support s_{min} and minimum confidence c_{min} , the task of association rule mining is to extract, from the given data set D , the association rules that have support and confidence greater than or equal to the user-specified values. Formal definition of this problem is given below.

Input: A non-empty set D of transaction data where each transaction $T \in D$

is a non-empty subset of items set $\mathcal{I} = \{i_1, i_2, i_3, \dots, i_m\}$, minimum support s_{min} , and minimum confidence c_{min} .

Output: Association rules of the form “ $A \Rightarrow B$ with support s and confidence c ” where $A \subseteq \mathcal{I}$, $B \subseteq \mathcal{I}$, $A \cap B = \emptyset$, $s \geq s_{min}$, and $c \geq c_{min}$.

A set of items is referred to as an *itemset*. An itemset that contains k items is a k -*itemset*. A k -itemset L_k is called *frequent* if $\sigma(L_k) \geq s_{min} \times |D|$. Such a k -itemset is also referred to as a *frequent k -itemset*. A frequent 1-itemset is simply called *frequent item*.

Consider the sample transaction data given in table I. Let us assume that $s_{min} = 0.4$ and $c_{min} = 0.6$. It can be seen that the rule $\{i_2, i_4\} \Rightarrow \{i_3\}$ has support 0.4 and confidence 0.66. This is a valid association rule satisfying the given s_{min} and c_{min} values.

TABLE I
SAMPLE TRANSACTION DATA

Transaction	Items
T_1	$\{i_1, i_2\}$
T_2	$\{i_1, i_3, i_4, i_5\}$
T_3	$\{i_2, i_3, i_4, i_6\}$
T_4	$\{i_1, i_2, i_3, i_4\}$
T_5	$\{i_1, i_2, i_4, i_6\}$

The task of mining for association rules from a given large collection of data is a two-step process:

- 1) Find all frequent itemsets satisfying s_{min} .
- 2) Generate association rules from the frequent itemsets satisfying s_{min} and c_{min} .

The second step is straight-forward, and in this paper we will be concentrating only on the first step.

B. Apriori Algorithm

As the name implies, this algorithm uses prior knowledge about frequent itemset properties. It employs an iterative approach where k -itemsets are used to explore $(k+1)$ -itemsets. To improve the efficiency of the generation of frequent itemsets, it uses an important property called the apriori property which says that all nonempty subsets of a frequent itemset must also be frequent. In other words if an itemset A does not satisfy the minimum support, then for any item $i_i \in \mathcal{I}$, the set $A \cup i_i$ cannot satisfy the minimum support either.

The Apriori algorithm first computes the frequent 1-itemsets, L_1 . To find frequent 2-itemsets L_2 , a set of *candidate* 2-itemsets, C_2 , is generated by joining L_1 with itself, i.e., $C_2 = L_1 \bowtie L_1$. The join is performed in such a way that for $L_k \bowtie L_k$, the k -itemsets l_1 and l_2 , where $l_1 \in L_k$ and $l_2 \in L_k$, must have $k-1$

items in common. Once C_2 is computed, for every 2-itemset c_2 in C_2 , all possible 1-subsets of c_2 are checked to make sure that all of them are frequent. If any one of them is not a frequent itemset, then c_2 is removed from C_2 . Once all the 2-itemsets in C_2 are checked, the set now becomes L_2 from which L_3 can be computed. This process is continued until, for some value k , L_{k+1} becomes an empty set. The algorithm is shown in Fig. 1.

```

APRIORI( $D, s_{min}$ )
1   $C_1 \leftarrow \{\{i\} | i \in \mathcal{I}\}$ 
2   $k \leftarrow 1$ 
3  while  $C_k \neq \emptyset$  do
4    for each transaction  $T \in D$  do
5      for each candidate itemset  $c_k \in C_k$  do
6        if  $c_k \subseteq T$  then
7           $s(c_k) \leftarrow s(c_k) + 1$ 
8       $L_k \leftarrow \{c_k | c_k \in C_k, s(c_k) \geq s_{min}\}$ 
9       $C_{k+1} \leftarrow \emptyset$ 
10     for each  $l_1, l_2 \in L_k, |l_1 \cap l_2| = k - 1$  do
11        $I \leftarrow l_1 \cup l_2$ 
12       if  $\forall J \subset I, |J| = k$  and  $J \in L_k$  then
13          $C_{k+1} \leftarrow C_{k+1} \cup I$ 
14      $k \leftarrow k + 1$ 
15 return  $L_k$ 

```

Fig. 1. Apriori Algorithm

In order to generate the frequent k -itemset L_k , this algorithm scans the input dataset k times. Also during the beginning stages the number of candidate itemsets generated could be so large. These factors greatly affect the running time of the algorithm. In the next subsection we describe the FP-Growth algorithm which is normally faster than the Apriori algorithm.

C. FP-Growth Algorithm

FP-Growth algorithm adopts *divide-and-conquer* strategy. First it computes the frequent items and represents the frequent items as a compressed database in the form of a tree called *frequent-pattern tree*, or *FP-tree*. The rule mining is performed on this tree. This means the dataset D needs to be scanned only once. Also this algorithm does not require the candidate itemset generation. So it is normally many times faster than the Apriori algorithm.

The frequent items are computed as in the Apriori algorithm and represented in a table called *header table*. Each record in the header table will contain the frequent item and a link to a node in the FP-Tree that has the same item name. Following this link from the header table, one can reach all nodes in the tree having the same item name. Each node in the FP-Tree, other

than the root node, will contain the item name, support count, and a pointer to link to a node in the tree that has the same item name. The steps for creating the FP-Tree are given below.

- Scan the transaction data D once and create L_1 along with the support count for each frequent item in L_1 . Sort L_1 in the descending order of support count and create the header table L .
- Create the FP-tree with an empty root node M . For each transaction $T \in D$, perform the following.

Select and sort the frequent items in T to the order of L . Let the sorted frequent items in T be $p|P$, where p is the first element and P is the remaining list. Let INSERT_TREE be the function that is called recursively to construct the tree. Call INSERT_TREE($p|P, M$), which does the following. If M has a child N such that $N.item-name = p.item-name$, then increment N 's support count by 1; else create a new node N with support count 1, let M be its parent, and link N to other node in M with the same *item-name*. If P is not empty, call INSERT_TREE(P, N) recursively.

Once the header table and the FP-Tree are constructed, then for each frequent item in the header table, the *conditional pattern base*, which is a list of nodes that link the frequent item's node in the FP-Tree to the root node, is formed. Each pattern base is assigned a support count which is the minimum of the support counts for the items in the pattern base. If the support count of a pattern base is less than s_{min} , then it is ignored. So if a frequent item appears n times in the FP-Tree, then it can have at most n conditional pattern bases. For each included pattern base of a frequent item, an FP-Tree called *conditional FP-Tree* is constructed, and the mining process is repeated until the conditional FP-Tree is empty or there is only one conditional pattern base. The set of items in such single pattern bases form the frequent itemsets. Finally association rules are extracted from these frequent itemsets.

In the Apriori and the FP-Growth algorithms described above, once the frequent itemsets are computed, association rules can be extracted using that information. The algorithm to perform this is the same for both Apriori and FP-Growth, and as mentioned before it is not in the scope of this paper.

III. VARIOUS APPROACHES TO XML RULE MINING

Association rule mining from XML data has gained momentum over the last few years and is still in its nascent stage. Several techniques have been proposed

to solve this problem. The straightforward approach is to map the XML documents to relational data model and to store them in a relational database. This allows us to apply the standard tools that are in use to perform rule mining from relational databases. Even though it makes use of the existing technology, this approach is often time consuming and involves manual intervention because of the mapping process. Due to these factors it is not quite suitable for XML data streams.

Recently World Wide Web consortium introduced an XML query language called *XQuery* [4]. This query language addresses the need for the ability to intelligently query XML data sources. It is also flexible enough to query a broad spectrum of XML information sources, including both databases and documents. Naturally this led to the use of XQuery to perform the association rule mining directly from XML documents. Since XQuery is designed to be a general purpose XML query language, it is often very difficult to implement complicated algorithms. So far only the Apriori algorithm has been implemented by using XQuery [12]. It has been raised as an open question in [12] whether or not FP-Growth algorithm can be implemented by using XQuery, and there is no such implementation available at this point.

The other approach is to use programs written in a high level programming language for this task. Most of such implementations require the input to be in a custom text format and do not work with XML documents directly. In order to adopt this approach to XML rule mining, it requires an additional step to convert the XML documents into the custom text files and apply these tools. This step often affects the overall performance of this approach.

Our approach for XML rule mining is to use programs written in Java to work directly with XML documents. This offers more flexibility and performs well compared to other techniques. The implementation details and experimental results of our approach are given in the following sections.

IV. IMPLEMENTATION DETAILS

Java provides excellent support for handling XML documents. Programs written in Java can access XML documents in one of the following two ways.

- 1) *Document Object Model (DOM)*: This allows programs to randomly access any node in the XML document, and requires that the entire document is loaded into memory.
- 2) *Simple API for XML (SAX)*: This approach follows event-driven model and allows programs to perform only sequential access on the XML doc-

ument. This does not load the entire document into memory.

Since Apriori algorithm needs to scan the input data many times, we used DOM for implementing this algorithm. Similarly SAX is the natural choice for FP-Growth, since it needs to scan the input data only once and works with the FP-Tree constructed in memory for further processing.

An XML document contains one root level element with the corresponding opening and closing tags. These tags surround all other data content within the XML document. The format of a sample XML data used to test our algorithm is shown in Fig. 2. The *transactions* tag is the root element that contains many *transaction* elements. Each *transaction* element is uniquely identified by its *id* attribute. Each *transaction* element contains one *items* element which in turn contains many *item* elements. An *item* element has the name of the particular item in the given transaction.

Note that the input XML document can have a very complicated structure, containing the transaction data at different depths. We assume in this case that the input document is preprocessed by using an XML style sheet language, like XSLT, to convert it into a simply structured document format as shown in Fig. 2 [5]. This preprocessing can be done quickly and easily, and is not in the scope of this paper.

```

<transactions>
  <transaction id="1">
    <items>
      <item>Bread</item>
      <item>Milk</item>
    </items>
  </transaction>
  <transaction id="2">
    <items>
      .
      .
      .
    </items>
  </transaction>
  .
  .
  .
</transactions>

```

Fig. 2. XML Input File Format

The configuration settings for our implementation is given in Fig. 3. These configurations are stored in a Java property file as property name-value pairs. The first four properties are self-explanatory. In order to make our implementation more generic in being able

to work with any XML tag names, we allow the user to pass the name of these tags through the properties in lines 5 through 9.

```

INPUT_FILE_NAME=data.xml
OUTPUT_FILE_NAME=rules.xml
MINIMUM_SUPPORT=0.6
MINIMUM_CONFIDENCE=1.0
FIRST_LEVEL_ELEMENT_NAME=transactions
SECOND_LEVEL_ELEMENT_NAME=transaction
THIRD_LEVEL_ELEMENT_NAME=items
FOURTH_LEVEL_ELEMENT_NAME=item
SECOND_LEVEL_ELEMENT_UNIQUE_
ATTRIBUTE_NAME=id

```

Fig. 3. XML Input File Format

Our implementation outputs the association rules in XML format as shown in Fig. 4. The root level element name is *rules* which contains zero or more *rule* elements. Each *rule* element has one *antecedent* and one *consequent* elements, and each rule has two attributes: *support* and *confidence*.

```

<rules>
  <rule support="0.6" confidence="1.0">
    <antecedent>
      <item>Bread</item>
    </antecedent>
    <consequent>
      <item>Milk</item>
    </consequent>
  </rule>
  .
  .
  .
</rules>

```

Fig. 4. XML Output File Format

Our implementation includes several optimization strategies outlined in the previous literatures [9]–[11]. Also we used custom-built data structures to improve the performance instead of using the ones provided in *Java Software Development Kit (JSDK)* library. In our FP-Growth implementation, we stored the FP-Tree in the form of an XML document. This allowed us to use XPath [5] expressions to quickly query any node in the FP-Tree.

V. EXPERIMENTAL RESULTS

We studied the performance of our implementation on three transaction datasets created randomly. The details of these datasets are given in Table II. We used 30 distinct items and a maximum of 20 items in each

transaction for all the datasets. We created our own datasets due to the fact that there are no benchmark data available for this problem. The experiments were performed on a Pentium 4, 3.2 GHz system running Windows XP Professional with 1 GB of main memory. Fig. 5 shows the running time comparison between the Apriori algorithm and the FP-Growth algorithm for the Dataset 1. It can be seen that the FP-Growth always outperforms the Apriori for all values of minimum support. This was the case in all the three datasets tested.

TABLE II
TEST DATASETS

Datasets	Number of Transactions
Dataset 1	100
Dataset 2	500
Dataset 3	1000

Fig. 6 shows the running time comparison between the Java-based Apriori and the XQuery-based Apriori. We used the XQuery implementation from [12] for this comparison. We observed that the Java-based Apriori outperforms the XQuery implementation on all three datasets. But the gap between the two narrows as the number of transactions increases. All these graphs were obtained for a minimum confidence of 0.6.

It can be observed that the performance of these algorithms largely depends on the number of frequent itemsets. For lower values of minimum support, it is expected to have many frequent itemsets, and this number will decrease as the minimum support increases. So the running time decreases as the minimum support increases. The large gap between the Apriori and the FP-Growth at lower values of minimum support was caused by the large number of candidate itemsets created in Apriori. It is our opinion that the data structure overhead in the XQuery implementation is what led to the performance difference between the Java-based Apriori and XQuery-based Apriori. The performance graphs for the remaining two datasets resembled the ones shown here except for the numerical values on the time axis.

VI. CONCLUSION

In this paper we studied the association rule mining problem and various approaches for mining association rules from XML documents and XML repositories. We presented a Java-based approach to this problem and compared ours with an XQuery-based implementation. Our approach performed very well against the one that we compared. There are several modifications that have been proposed to both Apriori and FP-Growth algorithms which include modifications to

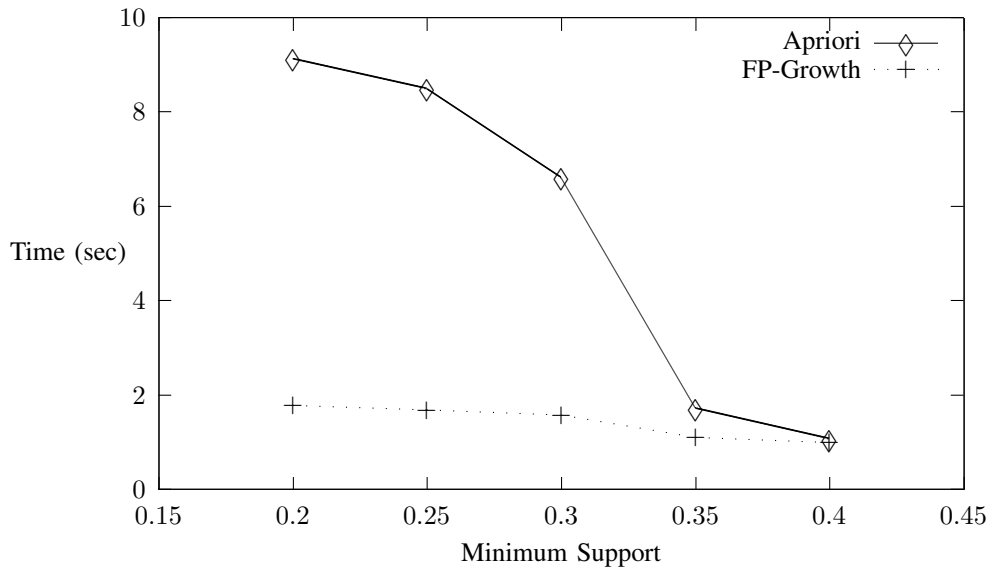


Fig. 5. Apriori Vs FP-Growth on Dataset 1

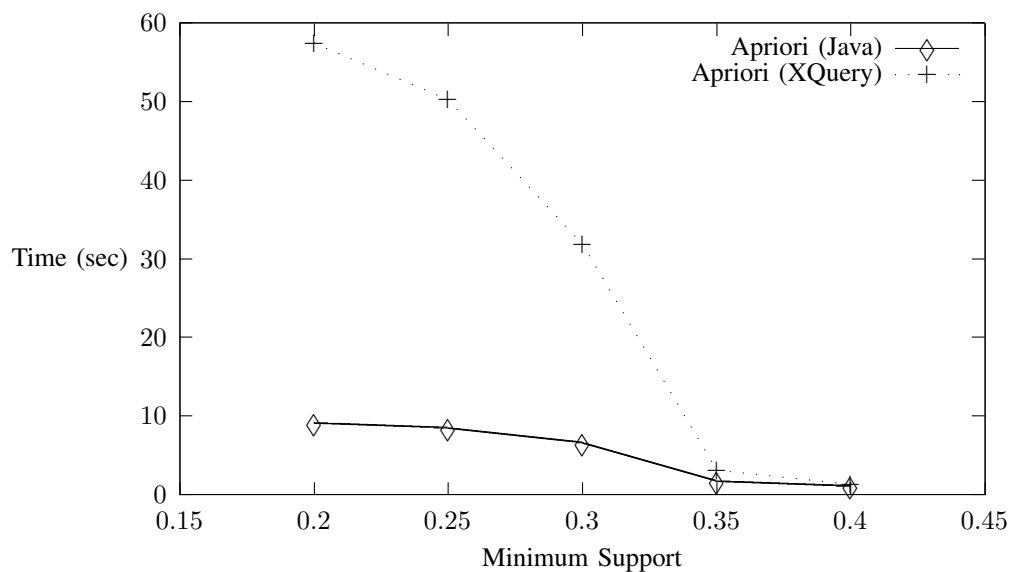


Fig. 6. Java-based Apriori Vs XQuery-based Apriori on Dataset 1

the data structures used in the implementation and modifications to the algorithm itself [9]–[11]. Though our implementation includes many such techniques, we did not have the time to try them all. We plan to do more analysis on this front.

Though FP-Growth algorithm is normally faster than the Apriori algorithm, it is harder to implement the first one. We plan to try XQuery to implement the FP-Growth algorithm and compare its results with our current Java-based implementation.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their valuable comments.

REFERENCES

- [1] R. Agrawal, T. Imielinski, and A. Swami, "Mining Association Rules between Sets of Items in Large Databases," Proceedings of the ACM SIGMOD International Conference on Management of Data, Washington, D.C., USA, May 1993, pp. 207–216.

- [2] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," Proceedings of the Twentieth International Conference on Very Large Data Bases, Santiago, Chile, September 1994, pp. 487–499.
- [3] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur, "Dynamic Itemset Counting and Implication Rules for Market Basket Data," Proceedings of the ACM SIGMOD International Conference on Management of Data, Tucson, AZ, USA, May 1997, pp. 255–264.
- [4] M. Brundage, *XQuery: The XML Query Language*, Addison-Wesley Professional, 2004.
- [5] J. R. Gardner and Z. L. Rendon, *XSLT and XPATH: A Guide to XML Transformations*, Prentice Hall, 2001.
- [6] G. F. Goldfarb, *XML Handbook*, Prentice Hall, 2003.
- [7] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufmann, 2006.
- [8] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach," *Data Mining and Knowledge Discovery*, 8(1), 2004, pp. 53–87.
- [9] J. S. Park, M. S. Chen, and P. S. Yu, "An effective Hash-Based Algorithm for Mining Association Rules," Proceedings of the ACM SIGMOD International Conference on Management of Data, San Jose, CA, USA, May 1995, pp. 175–186.
- [10] A. Savasere, E. Omiecinski, and S. Navathe, "An Efficient Algorithm for Mining Association Rules in Large Databases," Proceedings of the Twenty First International Conference on Very Large Databases, Zurich, Switzerland, September 1995, pp. 432–444.
- [11] C. Silverstein, S. Brin, and R. Motwani, "Beyond Market Baskets: Generalizing Association Rules to Dependence Rules," *Data Mining and Knowledge Discovery*, 2(1), 1998, pp 39–68.
- [12] J. W. W. Wan and G. Dobbie, "Extracting association rules from XML documents using XQuery," Proceedings of the Fifth ACM International Workshop on Web Information and Data Management, New Orleans, LA, USA, November 2003, pp 94–97.