A Framework for Efficient Association Rule Mining in XML Data

Ji Zhang, University of Toronto, Canada
Han Liu, University of Toronto, Canada
Tok Wang Ling, National University of Singapore, Singapore
Robert M. Bruckner, Microsoft, USA
A Min Tjoa, Vienna University of Technology, Austria

ABSTRACT

In this paper, we propose a framework, called XAR-Miner, for mining ARs from XML documents efficiently. In XAR-Miner, raw data in the XML document are first preprocessed to transform to either an Indexed XML Tree (IX-tree) or Multi-relational Databases (Multi-DB), depending on the size of XML document and memory constraint of the system, for efficient data selection and AR mining. Concepts that are relevant to the AR mining task are generalized to produce generalized meta-patterns. A suitable metric is devised for measuring the degree of concept generalization in order to prevent under-generalization or over-generalization. Resulting generalized meta-patterns are used to generate large ARs that meet the support and confidence levels. A greedy algorithm is also presented to integrate data selection and large itemset generation to enhance the efficiency of the AR mining process. The experiments conducted show that XAR-Miner is more efficient in performing a large number of AR mining tasks from XML documents than the state-of-the-art method of repetitively scanning through XML documents in order to perform each of the mining tasks.

Keywords: Association Rule Mining, XML Data, Data Transformation and Indexing, Concept Generalization, Meta Patterns

INTRODUCTION

The eXtensive Markup Language (XML) has become a standard for representing and exchanging information on the Internet. The fast-growing amount of XML-based information on the web has made it desirable to develop new techniques to discover patterns and knowledge from XML data. Association Rule (AR) mining is frequently used in data mining to reveal interesting trends, patterns, and rules in large datasets. Mining ARs from XML documents thus has become a new and interesting research topic.

Association rules have been first introduced in the context of retail transaction databases (Agrawal, 1994). Formally, an AR is an implication in the form of *X*→*Y*, where *X* and *Y* are sets of items in database *D* that satisfy $X \cap Y = \phi$ and $X \cup Y \subseteq I$. *D* is a set of data cases, and *I* is the complete set of distinct items appear in *D*. *X* and *Y* are called the antecedent and subsequent of the rule. The rule *X*→*Y* has support of *s* in *D* if *s%* of the data cases in *D* contain both *X* and *Y*, and the rule has a confidence of *c* in *D* if *c%* of the data cases contain *Y* if they also contain *X*. Association rule mining aims to discover all large rules whose support and confidence exceed user-specified minimum support and confidence thresholds: *minsup* and *minconf*.

A recent trend of mining ARs is the development of query language to facilitate association rule mining. The MINE RULE operator is introduced by Meo (1996) and extended by Meo (1998). In addition, Imielinski (1999) introduced a SQL extension, called MSQL, which is equipped with various operators over association rules such as generation, selection, and pruning. DMSQL is another query language used for association rule specification (Han, 2000). There has already been research in mining ARs from semi-structured documents (Amir, 1997, Feldman, 1996, Singh, 1997, Singh, 1999), among which the works in Amir (1997) and Feldman (1996) use unsupervised procedures and Singh (1997) and Singh (1999) use single and multi-constraints in mining ARs. The constraint-based methods, compared to the non-constraint ones, are able to achieve better efficiency in the mining process and generate a manageable number of rules.

Though we have witnessed intensive research work in AR mining in the past years, there has been very little work in the domain of AR mining from XML documents. For instance, the work by Braga (2002) uses the MINE RULE operator introduced by Meo (1996) for AR mining purposes in native XML documents. The Predicative Model Markup Language (PMML) is proposed to present various patterns such as association rules and decision trees extracted from XML documents (PMML, 2000). Feng (2003) presents a XML-enable AR mining framework, but does not give any details on how to implement this framework efficiently. Wan (2003) claims that XML AR can be simply accomplished using XQuery language.

However, the major problems with the state-of-the-art methods are two-fold: (i) These approaches select data from native XML data, thus the efficiency of these approaches is low because of the normally huge volume of XML data that need to be scanned in the process of AR mining. Each AR mining task involves a scan of XML data sources, which is not practical when performing a large number of AR mining tasks. Data organized in XML format inherently render the data selection inefficient, thus it is motivated that the XML data should be extracted and organized in a proper way before a computationally intensive AR mining can be performed efficiently; (ii) These approaches do utilize generalization (e.g. group or cluster operations) of data. However, they lack the mechanism to control the degree to which generalization is performed. Under-generalization or over-generalization will seriously affect the effectiveness of the AR mining.

To address the above problems, we will propose a framework, called XAR-Miner, to efficiently mine ARs from XML documents in this paper. In this framework, XML data are extracted and organized in a way that is suitable for efficient data selection in the AR mining. Concepts relevant to the AR mining task are generalized, if necessary, to a proper degree in order to generate meaningful yet nontrivial ARs. XAR-Miner has been experimentally shown to be more efficient in mining a large number of ARs from XML documents than the naive method that has to scan through XML documents repetitively in each of the mining tasks.

The remainder of this paper is structured as follows. Section 2 presents an overview of our framework for AR mining of XML data. Section 3 discusses the construction of IX-tree and

Multi-DB. The details of AR mining from XML documents are discussed in Section 4. In Section 5, an efficient algorithm of integrating data selection and large itemset generation is presented. Section 6 gives the empirical complexity analysis of using the proposed data structure in AR mining. The experimental results are reported in Section 7. The final section concludes this paper.

OVERVIEW OF THE FRAMEWORK

The framework of AR mining of XML data consists of the following major parts: (1) Pre-processing (i.e., construction of the Indexed XML Tree (IX-tree) or Multiple-relational Databases (Multi-DB)); (2) Generation of generalized meta-patterns; and (3) Generation of large ARs of generalized meta-patterns. The overview of the framework is shown in Figure 1.

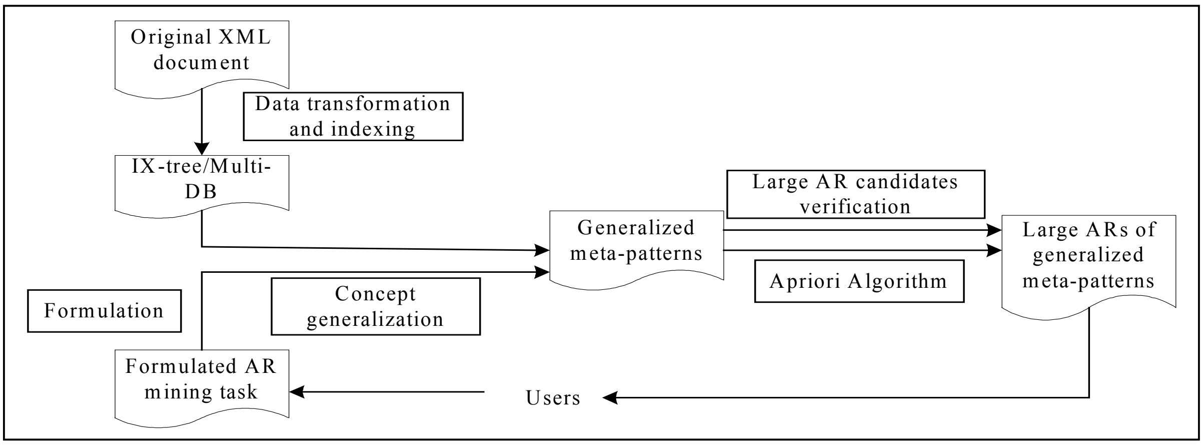
In the pre-processing module, data in the XML document are extracted and the IX-tree or Multi-DB will be built. Choice of IX-tree or Multi-DB is made based on the size of XML data and the main memory constraint of system. IX-tree will be used when the XML data can be fully loaded into the main memory of the system while Multi-DB will be used otherwise. The transformed XML data will be indexed using B-tree to facilitate efficient data selections during the mining process.

Secondly, the AR mining task issued by user is analyzed and properly formulated by the system. The AR mining task formulation involves specifying the format (antecedent and subsequent) of the AR rules, the data source from which the AR rules are to be mined, the conditional expression that dictates the scope of the data used for the AR mining and finally the minimum support and confident thresholds, *minsup* and *minconf*.

The concepts in the AR mining are the conceptual notions involved in the antecedent and subsequent of AR mining. For instance, in the AR mining task of finding large ARs between a person's highest degree and the kind of jobs he/she is engaged in, the concepts are highest degree and job. Because of the sparsity of the XML data, especially in the AR mining tasks that involve multiple concepts, concept generalization will usually be performed to generate generalized meta-patterns. For example, the concept of juice can be generalized as drink and the AR mining task of finding large ARs between the meat and the juice the customs buy in supermarket can be generalized to finding large ARs between the meat and the drink the customs buy. Two constraints, called *Min_gen* and *Max_gen*, are used to specify the allowed degree of generalization, which will avoid under-generalization or over-generalization of the meta-patterns.

Large ARs of the generalized meta-patterns that meet the support and confidence requirements are then mined using Apriori algorithm (Agrawal, 1994), a typical AR mining algorithm, in the AR Mining Module and returned to the users. The Apriori algorithm mines large ARs in two steps. First, the large itemsets whose supports exceed *minsup* are found. Second, select from the large itemsets obtained in the first step the itemsets whose confidences exceed *minconf*.

Figure 1. The overview of the framework for AR mining


Original XML document
Data transformation and indexing
IX-tree/Multi-DB
Large AR candidates verification
Generalized meta-patterns
Large ARs of generalized meta-patterns
Apriori Algorithm
Formulation
Concept generalization
Formulated AR mining task
Users

In this paper, a sample XML document about employee information, EmployeeInfo.xml (shown in Figure 4), is used to illustrate the ideas, definitions and algorithms of XAR-Miner. Figure 2 and 3 illustrate the DTD and the hierarchical structure of this sample XML document.

Figure 2. DTD of EmployeeInfo.xml

```
<!ELEMENT EmployeeInfo (Employee+)>
<!ELEMENT Employee (Name, Education, TypeOfJob)>
<!ELEMENT Education (Major, Degree)+>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT TypeOfJob (#PCDATA)>
<!ELEMENT Major (#PCDATA)>
<!ELEMENT Degree (#PCDATA)>
```

Figure 3. Hierarchical structure of EmployeeInfo.xml

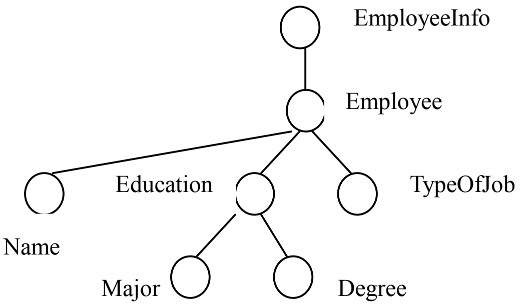


Figure 4. Sample XML document (EmployeeInfo.xml)

```
<EmployeeInfo>
     <Employee>
          <Name> James Wang </Name>
          <Education>
                    <Major> Computer Science </major>
                    <Degree> B.S </Degree>
                 <Major> Computer Science </major>
                    <Degree> M.Sc </Degree>
           </Education>
          <TypeOfJob> System Analyst </TypeOfJob>
     </Employee>
     <Employee>
          <Name> Ghai Vandana </Name>
          <Education>
                    <Major> Accounting </major>
                    <Degree> B.A </Degree>
           </Education>
          <TypeOfJob> Accountant </TypeOfJob>
     </Employee>
     <Employee>
          <Name> Linda Lee </Name>
          <Education>
                    <Major> Mathematics </major>
                    <Degree> B.S </Degree>
                  <Major> Management </major>
                    <Degree> M.B.A </Degree>
                 <Major> Economics </major>
                    <Degree> Ph.D </Degree>
           </Education>
          <TypeOfJob> Project Manager </TypeOfJob>
     </Employee>
</EmployeeInfo>
```

XML DATA EXTRACTION AND TRANSFORMATION

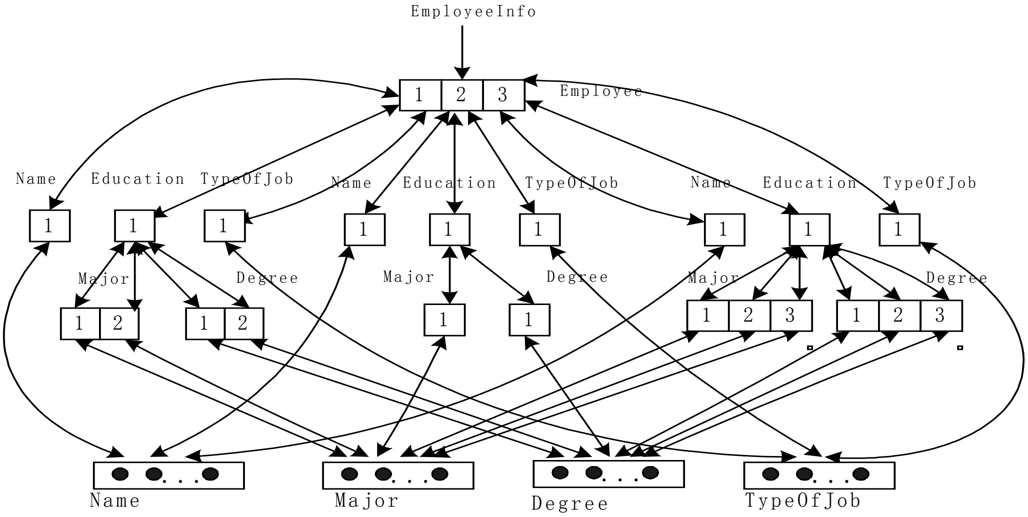
The preprocessing work of XAR-Miner is to extract information from the original XML document and transform them into a way that is suitable for efficient AR mining. Specifically, we build *Indexed XML Tree* (IX-tree) when all the XML data can be loaded into main memory, and build

Multi-relational Databases (Multi-DB) otherwise. We will evaluate the size of the XML data involved and the main memory available to select the proper strategy for XML data transformation and storage before AR mining tasks are preformed.

Construction of Indexed XML Tree (IX-tree)

We first discuss the construction of the Indexed XML Tree (IX-tree). A formal definition of the IX-tree is given as follows.

Figure 5. IX-tree of EmployeeInfo.xml



Definition 1. ***Indexed XML Tree (IX-tree)*****:** An Indexed XML Tree (IX-tree in short) is a rooted tree IX-tree=<*V, E, A*>, where *V* is the vertex set, *E* is the edge set and *A* is the indexed array set. *V* is set of nodes appearing in the XML document. The intermediate nodes in the IX-tree store the addresses of its immediate parent and children. An edge $e(v_1,v_2)$ in the IX-tree connects the two vertices v_1 and v_2 using a bi-directional link. The set of indexed arrays *A* positioned at the bottom level within the IX-tree stores the data in the leaf element or attribute nodes in the original XML document.

The IX-tree of EmployeeInfo.xml is shown in Figure 5. The IX-tree can be seen as a sort of hybrid of hierarchical and array-like structure. The hierarchical structure maintains the inherent parent-child and sibling relationships of the extracted XML data while the arrays store their indexed structural values. The construction of the IX-tree consists of two steps:

Hierarchical structure building of IX-tree. The XML document will be scanned to get hierarchical information of the document and structural values in the leaf nodes of the document. The moment an element or attribute is scanned in the XML document, a new node will be created in the corresponding IX-tree. Two pointers, *ParentNode* and *ChildNode* are used in this scanning process. If *ParentNode* is currently not a leaf node, the next node (its child node) will be read and these two nodes will be connected using a bi-directional link in the IX-tree. When the leaf node has been reached, the data in this leaf node will be retrieved and stored in an array. This process will be terminated when the end of the XML document is reached.

Indexing of data in the arrays. The arrays storing the extracted data from the XML document are indexed using B-tree to facilitate fast selection of relevant data for the AR mining task.

Construction of Multi-relational Databases of XML Data

Under the circumstance that the size of XML data exceeds the main memory available, it is impossible to build an in-memory IX-tree. Therefore, we alternatively choose to construct Multi-relational Databases to accommodate the extracted XML data. In this architecture, an XML document will be transformed to a few relational databases, each of which will store the indexed structural values of a leaf node in the XML document. Unlike using hierarchical structure to maintain the inherent parent-child and sibling relationships of XML data, we use the notion of Serial Xpath String (SXS) to identify each XML data in relational databases.

Definition 2. ***Serial Xpath String (SXS) of XML data***: The Serial Xpath String of an XML data x, denoted as SXS(x), is a string that gives the ordinal numbers of concepts of different levels along the path from the root to the leaf node of x. Any SXS will start with the root node and the ordinal number of concepts along the path will be delimited by dashed lines.

For example, the SXS of the data "Economics" in Figure 4 is "R-3-1-3". R in the SXS denotes the root node of the hierarchy. 3, 1 and 3 are the ordinal number of the concepts along the path, i.e. Employee, Education and Major. Please note an SXS with in a relational database is unique but not so across different databases since an SXS itself does not hold any information regarding the concepts along the path. To uniquely identify XML data stored in the multi-relational databases, we use the complete path in the XML tree starting from the root to the leaf node of the XML data, called *Xpath*, and associate it with each of the relational databases. In this particular example, the complete path of "Economics' which is associated with Major.db is Employeeinfo/Employee/Education/Major, thus the SXS of "R-3-1-3" is equivalent to Employinfo-Employee(3)-Education(1)-Major(3), meaning that Economics is the third major (i.e. PhD) in the education information of the third employee in the XML document. The salient feature of SXS is that it is compact and, together with the Xpath associated with each relational database, it is able to uniquely identify an XML data in the whole XML document using the ordinal occurrence number of concepts in SXS. In addition, by using a string format, it is able to well maintain the hierarchical relationships of XML data. The Multi-relational databases transformation of the Employeeinfo.xml is given in Figure 6.

Similar to the construction of IX-tree, the building of the Multi-relational Databases of extracted XML data will follow two steps as follows.

Feed Multi-relational databases. In building the Multi-relational Databases of the XML data, the XML document will also be scanned once to get the data values in the leaf nodes of the document. We create a few variables to record the ordinal number of concepts appearing in the DTD tree. When a mediate element or attribute is scanned, the corresponding variables will be augmented by 1. When the leaf node has been reached, the SXS of the XML data in this leaf node will be generated. Both data in this leaf node and its SXS will be stored in the corresponding relational database. This process will be terminated when the end of the XML document is reached.

Indexing of data in the databases. The multi-relational databases storing the extracted XML data are indexed using B+-trees to facilitate fast retrieval of relevant data for the AR mining task.

Figure 6. Multi-relational database transformation of Employeeinfo.xml

Name.db (XPath: Employeeinfo/Employee/Name)

Name	**SXS**
James Wang	R-1-1
Ghai Vandana	R-2-1
Linda Lee	R-3-1

TypeOfJob.db (XPath: Employeeinfo/Employee/TypeOfJob)

TypeOfJob	**SXS**
System Analyst	R-1-1
Accountant	R-2-1
Project Manager	R-3-1

Major.db(XPath: Employeeinfo/Employee/Education/Major)

Major	**SXS**
Computer Science	R-1-1-1
Computer Science	R-1-1-2
Accounting	R-2-1-1
Mathematics	R-3-1-1
Management	R-3-1-2
Economics	R-3-1-3

Degree.db (XPath: Employeeinfo/Employee/Education/Degree)

Degree	**SXS**
B.S	R-1-1-1
M.Sc	R-1-1-2
B.A	R-2-1-1
B.S	R-3-1-1
M.B.A	R-3-1-2
Ph.D	R-3-1-3

Advantages of Relational Transformation of XML Data

The major advantage of performing XML Transformation into either IX-tree or Multi-DB architecture is that this transformation can ideally satisfy the requirements of AR mining as follows: (a) AR mining often involves selecting data that are only relevant to the AR mining task. These data are usually only a fragment of the whole XML document. Information indexing in each of the arrays or relational databases helps to greatly facilitate the retrieval of relevant data from the relational databases for AR mining. The selection of data from these indexed structures is much more efficient than selecting data from the original XML document that usually requires a scan through the whole XML document; (b) The data in the XML document has relationships with each other such as a parent-child relationship or a sibling relationship in the hierarchy of the XML document. These relationships among data can be perfectly maintained in the transformation by either using hierarchical structure in IX-tree or SXSs in each of the relational databases.

GENERATE LARGE AR RULES

Formulation of AR mining task

There are two basic kinds of AR mining tasks performed on XML data, i.e. single-concept AR mining and multi-concepts AR mining. Single-concept AR mining aims to find large ARs among instances of the same concept (i.e. the antecedents and subsequents of a rule are of the same domain) while multi-concept AR mining tries to find large ARs among instances of a few different concepts (i.e. the antecedents and subsequents of the rule are of different domains). For example, in a customer shopping XML document, a single-concept AR mining might be "*find large ARs among the products customers purchase in the supermarket*" and a multi-concept AR mining might be *"find* *large ARs among the age of the customer and the products they purchase"*. Single and multi-AR mining from the IX-tree or Multi-DB architecture can be generically formulated using the statements in Figure 7 and 8.

Figure 7. Formulation of single concept AR mining

MINE RULE *"ARsObtained.db"* ***AS***
SELECT *(A[1,n_1])* ***AS ANTECEDENT****, (A[1,n_2])* ***AS SUBSEQUENT***
FROM IX-TREE: *"Treename"* ***ARRAYS:*** *A.array,|* ***FROM Multi-DB****: A$_{.}$.db,*
WHERE *……*
CONSTRAINTS SUPPORT*: minsup* ***CONFIDENCE****: minconf*

Figure 8. Formulation of multiple concepts AR mining

MINE RULE *"ARsObtained.db"* ***AS***
SELECT *(A_1[1,n_1], A_2[1,n_2]…A_i[1,n_i])* ***AS ANTECEDENT****, (S_1[1,n_1], S_2[1,n_2]…S_j[1,n_j])* ***AS SUBSEQUENT***
FROM IX-TREE: *"Treename"* ***ARRAYS:*** *$A_{1.}$.array, A_2. array,…, A_i. array, $S_{1.}$.array, S_2. array,…, S_i. array |**FROM Multi-DB**: $A_{1.}$.db, $A_{2.}$.db,…, $A_{i.}$.db, S_1.db, S_2.db…, S_j.db*
WHERE *……*
CONSTRAINTS SUPPORT*: minsup* ***CONFIDENCE****: minconf*

In the above two formulations, "ARsObtained.db" specified by the Mine Rule clause is the location to store the large ARs mined. The SELECT clause specifies the antecedent and subsequent of the ARs, where the two kinds of AR mining tasks mainly differ. $A_i[1, n_i]$ denotes the number of instances of the concept A_i appearing in the AR rules. The FROM clause specifies the data sources from which data are used for performing this AR mining task, either from the arrays in the bottom level of IX-tree (consisting of the arrays *$A_{1.}$.array, A_2. array,…, A_i. array, $S_{1.}$.array, S_2. array,…,* and *S_i. Array*) or from the multi-relational databases (consisting of *A_1.db, $A_{2.}$.db,…, $A_{i.}$.db, S_1.db, S_2.db…, S_j.db*). The WHERE clause determines the scope for data selection, and finally the CONSTRAINTS clause specifies the *minsup* and *minconf* constraints input by users.

Data Selection from IX-tree and Multi-DB

One of the key problems we have to deal with in the AR mining from XML document is the selection of data that are relevant to the AR mining task. In this step, we will select all the instances of involved concepts that meet the WHERE specification. This step resembles the answering of queries. Since all the data in IX-tree and the Multi-relational databases have been indexed, the retrieval of qualified data from them can be very efficient. This efficiency is very significant when the number of AR mining tasks to be performed is large. If the WHERE statement involve multiple concepts, then the query results of multiple database will be joined to generate the final result of this step. We will discuss in details how to select data efficiently from IX-tree and Multi-relational databases.

Recall that in IX-tree, we take advantage of the bi-directional linking between parent and child nodes in the tree hierarchy to realize fast top-down and bottom-up traversing. This fast traversing can facilitate the retrieval of the value of the concepts involved in AR mining. The path between the instances of related concepts is needed for retrieving data from IX-tree or Multi-DB. To create such a path, the Nearest Common Ancestor Node (NCAN) of these concepts must first be found.

Definition 3. ***Nearest Common Ancestor Node (NCAN)***: The NCAN of elements $e_1,e_2,...,e_n$ in the hierarchy of an XML document *H*, denoted as $\text{NCAN}_H(e_1,e_2,...,e_n)$, is defined as the common ancestor element node in *H* that is the closest to e_i ($1 \le i \le$ n).

The NCAN in the hierarchy serves as the common information bridge semantically linking together data of the elements involved in an AR mining task. It is also called the *most informative subsumer* of nodes in the tree structure (Resnik, 1999). For example, the NCANs of the various nodes in the hierarchy in Figure 3 are as follows:

$$NCAN_H(Education, TypeOfJob)=Employee$$
$$NCAN_H(Major, Degree)=Education$$
$$NCAN_H(Major, Degree, TypeOfJob)= Employee$$

Searching a NCAN of concepts in a XML hierarchy involves path comparison for these concepts. The searching of NCAN can be divided into two sub-steps. The first step is to find the common nodes of all the paths and the second step is to find the node from these common nodes that is closet to the leaf nodes of all these paths.

For example, suppose we would like to find the NCAN of Major, Degree and TypeOfJob in EmployeeInfo.xml. The Xpaths of Major, Degree and TypeOfJob are Employee.Education.Major, Employee.Education.Degree and Employee.TypeOfJob, respectively. The common node that is closest to the leaf nodes of Major, Degree and TypeOfJob is Employee, which serves as the NCAN for these three nodes.

After the NCAN has been determined, instances of non-queried concepts will be retrieved by first performing a bottom-up traversing from the queried concepts to their NCAN and then a top-down traversing from their NCAN to the non-queried concepts in IX-tree. For example, to retrieve the value of Major and TypeOfJob if the Degree value is known, we only have to first traverse the paths of Degree→Education→Major and then Degree→Education→Employee→TypeOfJob.

Unlike IX-tree, Multi-DB architecture does not have the bi-directional link among nodes for traversing. However, we have created SXS for each XML data and Xpath for each relational database when the XML data are transformed to Multi-DB system that perfectly maintains the hierarchical information of concepts in the original XML document. It is noticed that the related XML data have the identical substring of varied length in their SXSs. This identical substring is the Xpath from the root to the NCAN of these related concepts. The ordinal number of the NCAN of the concepts can be used to identify data uniquely. This observation can help to retrieve the values/instances of non-queried concepts easily. The system finds data values of non-queried concepts in two substeps: (1) The format of SXS of the to-be-retrieved concepts will be decided by using the Xpath associated with the databases of the concepts; (2) Search in the relevant databases for the data whose SXS meets the format specified in the first step. Because SXS of the data has also been indexed, thus the SXS searching in step 2 is efficient.

Consider, for example, if we want to find the name of employees whose has B.S degree. The SXSs of the data valued “B.S” are “R-1-1-1” and “R-3-1-1” in Major.db. The NCAN of Name and Degree are Employee, so the SXS of name value of the employee who has B.S degree should be in the format of “R-1-*” and “R-3-*” in the Name.db, where * denotes the string after the employee node in the complete SXS. We then search in the Name.db for the name value whose SXS matches the above two formats, which are “James Wang” and “Linda Lee”. Thus, the employees who have B.S degree are James Wang and Linda Lee.

Generate Generalized Meta-Patterns

The raw XML data that are relevant to the AR mining task will usually be generalized in order to generate ARs that are significant enough. The generalization of XML data is necessitated by the sparsity of the XML data involved in the AR mining task. Generally speaking, the sparsity of the XML data usually results from three factors: (1) A large number distinct instances of concepts involved in the AR mining; (2) A large number of empty values of concepts involved in the AR mining; (3) A large number of concepts in multiple concepts AR mining task. These factors cause the interesting ARs have low support levels.

It is worthwhile mentioning that the data should be generalized properly in order to find significant yet nontrivial ARs. On one hand, under-generalization may not render the data dense enough for finding patterns which extract significant ARs that can meet support or confidence level. On the other hand, over-generalization may lead to patterns which extract trivial ARs that are not depended on the minimum support and confidence. To well control the generalization process, we will propose a metric to measure the degree of generalization a generalization strategy will result in and two constraints, *Min_gen* and *Max_gen*, to help the generalization process avoid under-generalization and over-generalization.

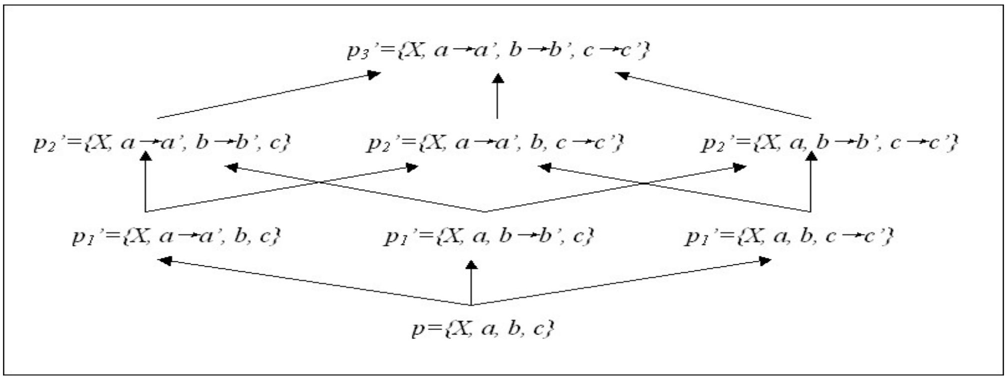
Psaila (2000) proposed the definition of the meta-patterns of ARs. This definition is limited in that it is only applied to the single-concept AR mining. In this paper, we extend the definition of meta-patterns to cater for the need of multi-concept AR mining.

Definition 4. *Primitive meta-patterns*: A *primitive meta-pattern* for single-concept AR mining is a tuple p: $\{X, a\}$, where X is the XML data source (IX-tree or Multi-DB) and a is the concept involved in the AR mining. A *primitive meta-pattern* for multi-concept AR mining is a tuple p: $\{X, a_1,...,a_i, s_1,...,s_j\}$, where X is the XML data source (IX-tree or Multi-DB), $a_1,...,a_i$ are the concepts for the antecedent of the AR, and $s_1,...,s_j$ are the concepts for the subsequent of the AR.

Definition 5. *Generalized primitive meta-patterns*: Single-concept generalization involves the generalization of only a concept while in multi-concept generalization more than one concept will be generalized. A *generalized primitive meta-pattern* for single-concept generalization is a tuple p': $\{X, a \rightarrow a'\}$, meaning that concept a is generalized to concept a'. A *k-generalized primitive meta-pattern* for multi-concept generalization ($1 \leqslant k \leqslant n$), denoted as p_k', is a pattern in which k concepts are generalized. n is the number of concepts in this meta-pattern.

Note that there are a combinatorial number of k-generalized primitive meta-patterns for a primitive meta-pattern. To depict all the generalized primitive meta-patterns, we use a structure of generalization lattice. Consider, for example, a primitive meta-pattern in the form of p: $\{X, a, b, c\}$. The corresponding generalization lattice of p is shown in Figure 9.

Figure 9. Generalization lattice of a primitive meta-pattern



Definition 6. Generalization Reference Table (GRT): A Generalization Reference Table (GRT) is a table specifying the concept generalization schemes of different levels. A generic form of the table is given in Table 1.

In the GRT as shown in the Table 1, C_{i0} denotes the i^{th} original concept appearing in the XML document and C_{i1}, C_{i2}… C_{im} are the 1st, 2nd,…, m^{th} level generalized concept for C_{i0}, respectively. GRT is created offline by human users and provides guidance regarding the concept generalization performed in the AR mining.

Table. 1. Generalization Reference Table (GRT)

Original concept	1st level generalized concept	2nd level generalized concept	…	m^{th} level generalized concept
C_{10}	C_{11}	C_{12}	…	C_{1m}
C_{20}	C_{21}	C_{22}	…	C_{2m}
C_{30}	C_{31}	C_{32}	…	C_{3m}
…	…	…	…	…
C_{n0}	C_{n1}	C_{n2}	…	C_{nm}

Now, we will present the quantitative generalization metric for measuring the degree of single-concept and multi-concept generalization. We denote the generalization degree of a generalized meta-patterns p' as $Gen(p')$ and is defined as follows:

(a) Single-concept generalization

$$Gen\ (p') = \overline{D} / D$$

Where $\overline{D}$ denotes the average number of distinct values in each record of the concept and D denotes the total number of distinct values of the concept.

(b) Multi-concept generalization

$$Gen(p') = \prod_{i=1}^{k} \overline{D_i} / D_i$$

Where $\overline{D}i$ denotes the average number of distinct values in each record of the i^{th} concept, D_i denotes the total number of distinct values of the i^{th} concept, and k denotes the number of concepts that have been generalized.

Intuitively, this generalization metric *Gen* computes the fraction of the average number of distinct values in each record of concept against the total number of distinct values. The higher value of *Gen*, the higher degree of generalization will be. Obverse that, in the case of multi-concept generalization, the generalization degree will be decreased exponentially when the number of concepts involved in the AR mining increases. This is because that, as the number of concepts goes up, the joint occurrence frequency of a specific multi-dimensional itemset will be decreased in an exponential rate.

At the end of this part, we will propose the algorithm of finding all the generalized meta- patterns whose generalization degree falls into the range of [*Min_gen*, *Max_gen*] in multi-concept generalization (the AR mining of single-concept generalization is relatively quite simple since there will be only one generalized meta-pattern). It is known that if p_2 is a derived generalized meta-pattern of p_1, then $Gen(p_1) \leqslant Gen(p_2)$. This theorem can be applied to facilitate the pruning of the excessively generalized meta-patterns: if the generalization degree of a meta-pattern exceeds *Max_gen*, then all its derived generalized meta-patterns are definitely excessively generalized and are therefore pruned away.

```
Algorithm Find_Generalized_Meta_Patterns (p, k, Max_gen, Min_gen)
p—a primitive meta-pattern;
k—the number of concepts in p.
Max_gen and Min_gen—the maximum and minimum generalization degree
Step 1: Initialize a full lattice of p. Compute the generalization degree Gen of the primitive
        meta-pattern p.
        IF $Gen(p) \leqslant Min\_gen$, THEN p is pruned from the lattice;
        ELSE IF $Gen(p) \geqslant Max\_gen$, THEN return an empty set of generalized meta-patterns and
                whole algorithm is terminated.
Step 2: FOR i= 1 to k DO
        Compute generalization degree Gen of i-generalized meta-patterns $p_i'$ of p that has yet been
        pruned from the lattice.
        IF $Gen(p_i') \leqslant Min\_gen$, THEN $p_i'$ is pruned from the lattice;
        ELSE IF $Gen(p_i') \geqslant Max\_gen$, THEN $p_i'$ and all its derived generalized meta-patterns in the
```

lattice are pruned;

Step 3: Output all the meta-patterns that are not pruned in the lattice.

In the algorithm of *Find_Generalized_Meta_Patterns*()**,** Step 1 mainly performs the initialization of the lattice and evaluation of generalization degree *Gen* of the primitive meta-pattern *p*. A bottom-up level-wise pruning process is carried out in Step 2. The final step returns all the meta-patterns that are not pruned in the lattice.

Generate Large AR Rules

After the generalized meta-patterns have been obtained, XAR-Miner will generalize the raw XML data based on the meta-patterns and generate large ARs w.r.t. the user-specified minimum support (*minsup*) and confidence (*minconf*) requirements using Apriori algorithm. The Apriori algorithm finds the large ARs that meet the support and confidence constraints by step-wise candidate generation; namely, the large *k*-itemsets are generated using (*k*-1)-itemset candidates. The two general steps of AR mining algorithm are (1) find all the itemsets that meet the minimum support constraint and (2) find all the itemsets that meet the minimum confidence constraint from itemsets obtain in Step 1.

In this step, we can adopt two modes in mining large ARs of generalized meta-patterns: the manual mode and the automatic mode. In the manual mode, the user has the freedom to choose, according to their preference on the granularity of concepts, one or more output generalized meta-patterns of Algorithm *Find_Generalized_Meta_Patterns* to generate large ARs. While in the automatic mode, system will generate large ARs of all output generalized meta-patterns. We will now focus on the automatic mode and propose an efficient algorithm to perform the automatic AR generation.

Theorem 1: Consider two ARs R_1 and R_2. If R_2 is the generalized AR of R_1, then Support(R_1)$\leqslant$ Support(R_2).

Proof: Let Freq(R_1) and Freq(R_2) be the occurrence frequency of itemsets of R_1 and R_2, and *N* be the number of records. Obviously, Freq(R_1)$\leqslant$Freq(R_2). We assume that the number of records will not be changed after the generalization, thus we have Freq(R_1)/*N*$\leqslant$Freq(R_2)/*N*, namely Support(R_1)$\leqslant$ Support(R_2). (End of the proof.)

Based on the above theorem, we can infer that if R_2 is not a large AR, then it is definite that R_1 is not a large AR either. This observation helps us devise an efficient algorithm to perform AR generation of generalized meta-patterns, which adopts a top-down strategy to traverse the generalization meta-patterns. The basic idea is that, instead of directly working on the raw generalized data to generate large ARs for a certain meta-pattern, we can first draw on the large ARs of its higher generalized meta-pattern in order to obtain the large AR candidates and then verify the largeness of these large AR candidates. Obviously, verifying the largeness of ARs is much cheaper than mining all the large AR directly from the data.

Algorithm Find_Large_ARs (*M, n*)

M—the set of generalized meta-patterns obtained;

n—the number of meta-patterns in *M*.

Step1: Sort *M* in a descending order based on the number of generalized concepts of the meta-patterns;

Step 2: FOR *i* =1 to *n* DO
IF there exists the derived meta-pattern of the i^{th} meta-pattern in the sorted *M*, THEN large AR candidates of this i^{th} meta-pattern are generated based on the large ARs of its derived meta-pattern and the largeness of large AR candidates is verified;
ELSE mine large ARs of this i^{th} meta-pattern using Apriori algorithm;

Step 3: Output the large ARs of all generalized meta-patterns.

The algorithm of *Find_Large_ARs()* first performs sorting of the set of generalized meta-patterns obtained by the algorithm of *Find_Generalized_Meta_Patterns*() in a descending order based on the number of generalized concepts of the meta-patterns. Such a sorting provides an easier way for the generalized meta-patterns with lower degree of generalization to generate its large AR candidates in Step 2 by using the large ARs of their derived mate-patterns that have already been mined. The large ARs of the generalized meta-patterns are retuned in the final step.

INTEGRATING DATA SELECTION AND AR MINING

In the section, we will integrate the process of data selection and AR mining so as to enhance the efficiency of these two steps. A greedy algorithm of data selection and AR mining will also be presented. Before presenting the algorithm, we will first discuss the cost of data selection from IX-tree or Multi-DB.

Retrieving instances of related concepts in the AR mining will have different cost. We differentiate the cases of IX-tree and Multi-DB in computing the cost involved. Specifically, the cost of retrieving the corresponding instance(s) of concept *y* from a single instance of concept *x* in the IX-tree *H* is computed as

$$Cost(x \rightarrow y) = dist(x, NCAN_H(x, y)) + dist(y, NCAN_H(x, y))$$

where $NCAN_H(x, y)$ denotes the Nearest Common Ancestor Node (NCAN) of *x* and *y* in the IX-tree *H*. dist() is the function computing the number of edges between two nodes in *H*, which indicates the distance between the two concepts in *H*. Intuitively, the further the two nodes are apart from each other in *H*, the higher cost will be in the data selection and vice versa.

In contrast to the IX-tree, the cost of retrieving the corresponding instance(s) of concept *y* from a single instance of concept *x* in the multi-DB architecture is computed as

$$Cost(x \rightarrow y) = dist(Root_H, NCAN_H(x, y))$$

where $Root_H$ denotes the root node of *H*. Recall that SXS is used in Multi-DB to maintain the inherent hierarchical structure of the original XML document. The related XML data have the identical substring of varied length in their SXSs and this identical substring is the Xpath from the root to the NCAN of these related concepts. For the sake of brevity, we will not deliberately differentiate $Cost(x \rightarrow y)$ in terms of IX-tree and Multi-DB in the following discussions. The appropriate cost function will be used according to the architecture utilized.

Now, we will give a classification of the concepts involved in the AR mining. The whole set of concepts T involved in an AR mining task can be generally classified into two subsets: Rule concept set (R), including the antecedent and subsequent concepts, and Queried concept set (Q) such that:

(a) $T = R \cup Q$, and

(b) $T \neq \phi$, $R \neq \phi$, but it is possible that $Q = \phi$, and

(c) $\phi \subseteq R \cap Q \subseteq T$

Potentially Qualified Data of a concept are the data of this concept that will potentially appear in the final large ARs. Given the above concept classification, we compute the initial number of Potentially Qualified Data of a concept $c \in T$, denoted as P_c, as follows:

$$\begin{cases} P_c = \sum freq(v_i) \mid freq(v_i) \geq \sup \text{, if } c \in R \text{ but } c \notin Q \\ P_c = \sum freq(v_i) \mid v_i \text{ satisfies the query in the AR mining, if } c \in Q \text{ but } c \notin R \\ P_c = \sum freq(v_i) \mid freq(v_i) \geq \sup \textit{ and } v_i \text{ satisfies the query in the AR mining, otherwise} \end{cases}$$

where $freq(v_i)$ denotes the frequency (number) of instance v_i appearing in the database. When $c \in R$ but $c \notin Q$, meaning that c is an antecedent or subsequent in the AR, thus the instances of c cannot become the item in the final large ARs unless $freq(v_i) \geq \sup$. When $c \in Q$ but $c \notin R$, meaning that c is a queried concept, thus the instances of c cannot be the qualified data unless v_i satisfies the query. The two requirements need to be met when c is not only an antecedent/subsequent in the AR but also a queried concept.

The total cost of retrieving corresponding instances of concept y from instances of concept x, denoted as $TCost(x \to y)$, is computed as follows:

$$TCost(x \to y) = P_x * Cost(x \to y)$$

Similarly, we have

$$TCost(y \to x) = P_y * Cost(y \to x)$$

From the definitions of $TCost(x \to y)$ and $TCost(y \to x)$, we can see that, though the order of retrieving related instances from a single instance does not make a difference, i.e. $Cost(x \to y) = Cost(y \to x)$, such order is crucial to the computational complexity of the data selection step which typically involves multiple instances. This is due to the fact that different concepts may have different number of Potentially Qualified Data, i.e. $P_x \neq P_y$. Therefore, we choose the order of retrieving data as from concept *x* to concept *y* iff $P_x \leq P_y$ and let

$$TCost(x, y) = \min(P_x, P_y) * Cost(x \rightarrow y)$$

A matrix *C*, called Cost Matrix, can be constructed with each entry $C_{i,j} = TCost(c_i, c_j)$, $c_i \in T$ and $c_j \in T$. It is observed that, as the data selection process evolves, the number of Potentially Qualified Data of concepts will be changed. Thus, the ordering of data selection can be greedily determined such that the cost of data selection can be minimized. To this end, we devise the following algorithm. The output of this algorithm will be the large itemsets based on which the second step of the Apriori algorithm, finding the large itemsets whose confidence exceed *conf*, can be performed.

Algorithm Data_selection (*C, sup*)
C—the initial Cost Matrix *C*;
minsup—the minimum support threshold.

```
Step 1. Select the concept x that has the smallest number of Potentially Qualified Data. IF $x \in R$,
        THEN retrieve the data of x and generate large itemsets using the retrieved data of x;
Step 2. Select c from the concepts in T that are yet evaluated such that $c=argmin_c\{min(C(c, c_i))\}$ for
        all concepts $c_i$ that have been evaluated, where $C(c, c_i)$ denotes the entry value of c and $c_i$
        in the cost matrix C.
        IF $c \in R$, THEN retrieve the data of c, generate new large itemset candidates using the
        retrieved data of c and the large itemsets of the concepts $c_j \in R$ whose data have been
        retrieved previously, and update the number of Potentially Qualified Data of these concepts
        as:
            $P_{c_j} = P_c = \sum freq(V, v_k) \mid freq(V, v_k) \geq \min\sup$
        for all $c_j$ and c. V represents the large itemsets involving all the concepts $c_j$ and $v_k$ is the
        instance of concept c. The cost matrix C will also be updated accordingly;
Step 3. Repeat Step 2 and 3 until all the concepts in T have been evaluated.
```

One of the unique features of the algorithm is that *it incorporates the first step of Apriori algorithm, which is, generating large itemsets whose support exceeds the support threshold sup, into the data selection process.* The step-wise generation of large itemsets and data selection interact with each other in that: (1) The data selection process determines, in each step, which data should be retrieved to produce the new large itemsets by means of the cost matrix *C*; (2) The large itemset generation process produces new large itemsets by including the newly chosen concept in each step and the new large itemsets generated will in return used to update the cost matrix *C*. The step-wise generation of large itemsets and data selection are characterized by the following advantages: (1) By progressively updating the cost matrix, we are able to greedily choose the optimal data retrieval scheme in each step of the data selection process, which contributes to enhancing the efficiency; (2) By adopting the large itemset generation in the data selection process, we can step-wisely refine the large itemset candidates by pruning away the itemset candidates that impossibly appear in the final ARs, which is also able to speedup the AR mining process.

COMPLEXITY ANALYSIS

In this part, we will empirically analyze the complexity of utilizing the IX-Tree/Multi-DB for AR mining. Specifically, the cost of using IX-Tree/Multi-DB involves the construction cost, data selection cost and maintenance cost. The relevant notions to be used in the analysis are first given as follows.

Let N be the total number of instances in the whole XML document
N_d be the total number of distinct instances for all concepts
C be the number of concepts in the XML tree
H be the averaged height of the XML tree (the averaged length of all the paths from the root to the leaf node)
C_q be the number of queried concepts in the AR mining specification
N_q be the total number of qualified instances of the queried concepts
α be the average ratio of the number of instances in a certain non-leaf level of IX-tree versus that of its immediate lower non-leaf level of IX-tree, $0 < \alpha \leq 1$

Construction Cost
In constructing the hierarchical structure of IX-Tree/Multi-DB, the whole XML tree will be traversed and each instance of concepts will be read, which requires a cost of O(H*N). The distinct instances of each concept stored in the array of the IX-Tree or Multi-DB will be indexed which requires a cost of O(C*N_d/C*$log(N_d/C)$)=O(N_d*$log(N_d/C)$). In sum, the cost for constructing the data structure will be O(H*N+N_d*$log(N_d/C)$). Since N_d is bounded by N, thus the worst case complexity for constructing the data structure is O($NlogN$).

Data Selection Cost
Selecting the qualified instances for AR mining involves searching each of the indexed arrays/databases of the queried concepts, with a cost of O(C_q*$log(N_d/C)$). The upper bound of the greedy data selection algorithm is O(N_q*H), which is equivalent to the complexity of traversing the hierarchical structure for each of N_q data without the step-wise data selection. Thus, the total cost for data selection will be O(C_q*$log(N_d/C)$+N_q*H). In the worst case where N_q=N, meaning that the whole dataset will be used in the AR mining, the complexity will become O(C_q*$log(N_d/C)$+H*N). Given C_q<<N and $logN_d$<<N, this complexity can be approximated as O(H*N).

Maintenance Cost
Maintenance cost denotes the work to maintain the data structure for dealing with data updates (insertions and deletions). Specifically, for a data insertion/deletion, we have to add/delete the instance of each of its concepts into or from the corresponding array/database. Finding the

appropriate place for inserting/deleting the instance amounts to the Nearest Neighbor (NN) search problem which can be accomplished with a cost of O($log(N_d/C)$) for each concept. Thus, inserting/deleting the whole data into the structure requires O($C*log(N_d/C)$). We also have to update the hierarchical structure, i.e. create inter-mediate nodes and bi-directionally connect them for insertion or delete the inter-mediate nodes and the edges connecting them for data deletion in IX-tree or create new or delete old SXSs for this update in Multi-DB. This cost will be O($C*H$). Thus, the total cost for handling a data insertion will be O($C*(log(N_d/C)+H)$).

Remarks: Note that the time complexities of IX-Tree/Multi-DB construction and data selection presented above are both for the worse cases. In most cases, however, the practical costs for IX-Tree/Multi-DB construction and data selection are much lower since we have $N_d<<N$ and $N_q<<N$.

Memory/Storage Overhead Analysis

Finally, we will analyze the space overhead of the XAR-Miner. In the architecture of IX-tree, its bottom level stores the distinct instances of various concepts in the XML database, thus the space requirement will be N_d. It is known that the number of nodes in the last non-leaf level (that is, the level H-1, if we call the root the level 1 and the leaf level the level H) of IX-tree is equal to the total number of instances (i.e. N), and the ratio α is defined as be the average ratio of the number of instances in a certain non-leaf level of IX-tree versus that of its immediate lower non-leaf level of IX-tree, thus we can obtain the approximate number of nodes of all non-leaf levels of IX-tree as $N+\alpha N+\alpha^2 N+...+\alpha^{H-2}N=\frac{N(1-\alpha^{H-2})}{1-\alpha}, 0<\alpha\le 1$. Because each node in the non-leaf levels uses two-way address referencing (the addresses of a node's parent and children nodes will be stored besides its own instance ID), thus the space requirement for non-leaf levels of IX-tree will be $\frac{3N(1-\alpha^{H-2})}{1-\alpha}$. Therefore, the space overhead of IX-tree will approximately be $\frac{3N(1-\alpha^{H-2})}{1-\alpha}+N_d$. Since the each instance will be associated with its corresponding SXS in Multi-DB, hence the space overhead of Multi-DB will be $2N$. From this space overhead analysis, we can see that IX-tree and Multi-DB feature larger space (memory or second storage) consumption than storing only the naïve XML data, which is N. However, being an efficiency-space trade-off, they are able to give the AR mining process a noticeable efficiency boost.

EXPERIMENTAL RESULTS

In order to evaluate the performance of XAR-Miner, a comparative study is conducted between XAR-Miner and the method using the MINE RULE operator for AR mining purposes in native XML documents (Braga, 2002) (called MineRule Operator Method), which has to scan the whole XML document in order to select the qualified data for AR mining. These two algorithms are executed on the same platform (on 1.8 GHz PC with 256 megabytes of main memory running Windows 2000) and are written in C++. Two sets of experiments are carried out. The first set of experiments demonstrates the efficiency of the IX-tree and Multi-DB construction. The second one investigates the efficiency of mining ARs from XML documents. We mainly look at the execution time of performing a AR mining task from XML documents with varied sizes and the time of performing small number of AR mining (1-10) and large number of AR mining (10-100). The IBM XML Generator is used to generate synthetic XML documents with varying sizes specified by user in our experiments.

Efficiency of the IX-tree and Multi-DB Construction

The execution time of constructing IX-tree and Multi-DB with respect to the size of XML documents is given in Figure 10. Note that the IX-tree and Multi-DB are used based on the total size of XML data and main memory available. This experiment aims to compare the time spent in constructing them from XML document with same size. Recall that the construction of IX-tree and Multi-DB involves the building of hierarchical structure of IX-tree or SXS of XML data in Multi-DB, and indexing of the data in the arrays or the multi-relational databases. Because all the operations of IX-tree construction are performed in main memory, its time is less than that required in the construction of Multi-DB. This analysis is justified by the result shown in Figure 10.

Efficiency of XAR-Miner in AR Mining

To study the efficiency of XAR-Miner in AR mining, we first investigate the execution time of performing an AR mining task from XML documents with varied sizes. From the observation of the experimental results as shown in Figure 11, we can see that the execution time of XAR-Miner using IX-tree and Multi-DB is much smaller than that of the MineRule Operator method. The main reason for this is that each AR mining task in the MineRule Operator method entails a full scan of the XML document while the AR mining in XAR-Miner only requires retrieval of relevant data directly from the IX-tree or Multi-DB, which is obviously more efficient.

Because efficient AR mining in XAR-Miner benefits from the construction of the IX-tree or Multi-DB, the comparison will be more fair if we take into account the cost in this pre-processing step and investigate the total execution time of performing 1-10 and 10-100 AR mining tasks. The total time for performing *n* AR mining is $T_{average}*n+ T_{pre\text{-}processing}$, where $T_{average}$ denotes the average time spent in each of the AR mining and $T_{pre\text{-}processing}$ denotes the time spent in the pre-processing step, i.e. construction of IX-tree or Multi-DB.

The results are presented in Figures 12 and 13. We can see that the execution time of XAR-Miner using IX-tree or Multi-DB is larger than that of MineRule Operator method when the number of AR mining tasks to be performed is small (1-3). But as the number of AR mining tasks continues to increase, MineRule Operator begins to require more time than XAR-Miner. This is because that the speedup of XAR-Miner by using IX-tree and Multi-DB overwhelms the cost of constructing IX-tree and Multi-DB when the number of AR mining tasks becomes large. Figure 13 shows that result comparison when performing larger number of AR mining tasks (10-100), in which the efficiency of XAR-Miner is much better than MineRule Operator method.

Figure10. Construction of IX-tree and Multi-DB

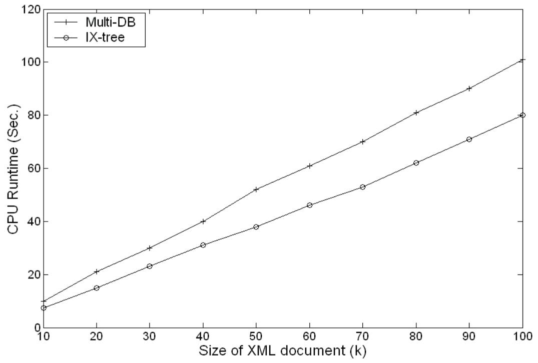


Figure11. Mining large ARs from XML document

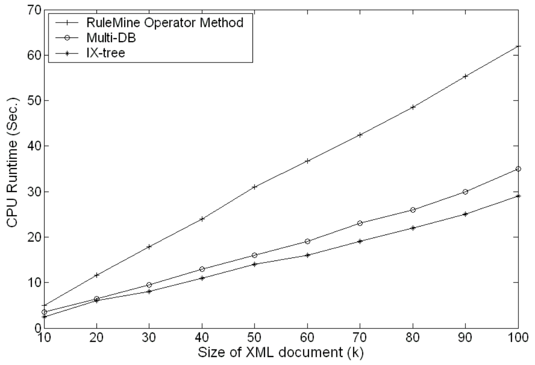


Figure12. Performing small number of AR mining tasks (1-10)

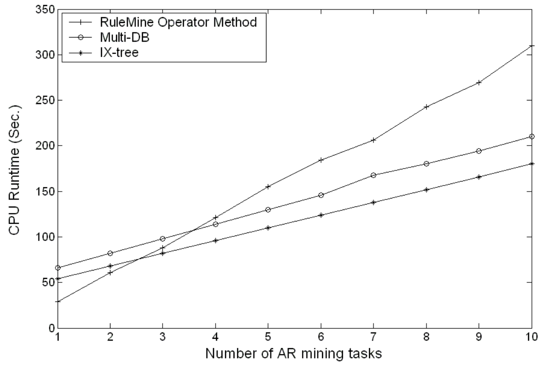
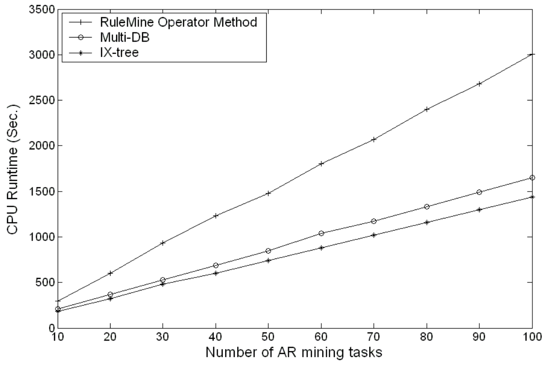


Figure13. Performing large number of AR mining tasks (10-100)



DISCUSSIONS AND CONCLUSIONS

In this paper, we investigate the problem of AR mining from XML documents. We propose a framework, called XAR-Miner, to mine ARs from XML documents efficiently. XAR-Miner transforms data in the XML document and constructs an Indexed XML Tree (IX-tree) if the XML data can be fully loaded into main memory or Multi-relational databases (Multi-DB) otherwise. The IX-tree and Multi-DB perfectly maintain the hierarchical information of XML data and perform indexing of the data to realize efficient retrieval of data in AR mining. Concepts that are relevant to the AR mining task are generalized to produce generalized meta-patterns. A suitable quantitative metric is devised for measuring the degree of concept generalization in order to prevent under-generalization or over-generalization. Resultant generalized meta-patterns are used to generate large ARs that meet the support and confidence levels. Experimental results justify that XAR-Miner is indeed much more efficient in performing large number of AR mining tasks than the state-of-the-art method that has to scan the XML document repetitively in the mining process.

The original XML data source is subject to regular updates: insertions of new XML data, deletion or changes of old XML data. Based on the old AR mining results, it is desirable to incrementally update the old AR mining results in order to produce the new ones, rather than generating all the rules again based on the whole data. The update of support and confidence of large rules when a data is inserted or deleted is easy to be accomplished without even using IX-tree or Multi-DB. In contrast, the update of small rules is more complex. The reason is that the support and confidence information of small ARs are unknown and therefore cannot update quickly. XAR-Miner will use IX-tree or Multi-DB to only evaluate some small ARs that possibly become large ones when the data are updated. After support and confidence of old large ARs are updated and support and confidence of some small ARs are computed, the new large ARs will be obtained by picking up from them the ARs that meet the minimum support and confidence requirements.

The current limitation of the XAR-Miner is that the thresholds controlling the concept generalization, i.e. *Max_gen* and *Min_gen*, can only be tuned through experiments and the possible future work involves exploring a more automated mechanism to precisely specify the values of the thresholds to achieve good effectiveness of the ARs mined.

REFERENCES

Agrawal, R. & Srikant, R. (1994). Fast Algorithms for Mining Association Rules in Large Databases, *the 20th International Conference of Very Large Data Bases (VLDB'94)*, 487-499.

Amir, A., Feldman, R., & Kashi, R. (1997). A New and Versatile Method for Association Generation, *Information Systems*, Vol. 22(6/7), 333-347.

Braga, D., Campi, A., Klemettinen, M., & Lanzi, P. (2002). Mining Association Rules from XML Data, in Proceedings of *the 4th International Conference on Data Warehousing and Knowledge Discovery (DaWaK'02)*, 21-30.

Feldman, R., & Hirsh, H. (1996). Mining Associations in the Presence of Background Knowledge, in Proceedings of *the 2nd International Conference on Knowledge Discovery in Databases*, 343-346.

Han, J. , & Kamber, M. (2000). *Data Mining Concepts and Techniques*, Morgan Kaufmann.

IBM XML Generator. http://www.alphaworks.ibm.com/tech/xmlgenerator.

Imielinski, T., & Virmani, A. (1999). MSQL: A Query Language for Database Mining, *Data Mining and Knowledge Discovery*, 3(4), 373-408.

Meo, R., Psaila, G., & Ceri, S. (1996). A New Operator for Mining Association Rules, in Proceeding of *the 22nd International Conference of Very Large Data Bases (VLDB'96)*, 122-133.

Meo, R., Psaila, G., & Ceri, S. (1998). A Tightly-coupled Architecture for Data Mining, in Proceedings of *the 14th International Conference on Data Engineering (ICDE'98)*, 316-323.

PMML 2.0: Predicative Model Makeup Language. (2000). Available at http://www.dmg.org.

Resnik, P. (1999). Semantic Similarity in A Taxonomy: An Information-based Measure as its Application to Problems of Ambiguity in Natural Language, in *Journal of Artificial Intelligence Research*, 11, 95-130.

Singh, L., Chen, B., Haight, R., & Scheuermann, P. (1999). An Algorithm for Constrained Association Rule Mining in Semi-structured Data, in Proceedings of *the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'99)*, 148-158.

Singh, L., Scheuermann, P., & Chen, B. (1997). Generating Association Rules from Semi-structured Documents Using an Extended Concept Hierarchy, in Proceedings of *the International Conference on Information and Knowledge Management (CIKM'97)*, 193-200.

Psaila, G., & Lanzi, P. L. (2000). Hierarchy-based Mining of Association Rules in Data Warehouses, in Proceedings of *2000 Symposium of Applied Computing (SAC'00)*, 307-312.

Feng, L., Dillon, T. S., Weigand, H., & Chang, E. (2003). An XML-Enabled Association Rule Framework. In Proceedings of *International Conference on Database and Expert Systems Applications (DEXA'03*), 88-97.

Wan, W. W., & Dobbie, G. (2003). Extracting association rules from XML documents using XQuery. In Proceedings of *Fifth International Workshop on Web Information and Data Management (WIDM'03)*, 94-97.