Mining tree-based association rules from XML documents

Mirjana Mazuran  
Dipartimento di Elettronica e Informazione  
Politecnico di Milano  
Italy  
mazuran@elet.polimi.it

Elisa Quintarelli  
Dipartimento di Elettronica e Informazione  
Politecnico di Milano  
Italy  
quintarelli@elet.polimi.it

Letizia Tanca  
Dipartimento di Elettronica e Informazione  
Politecnico di Milano  
Italy  
tanca@elet.polimi.it

ABSTRACT
The increasing amount of XML datasets available to casual users increases the necessity of investigating techniques to extract knowledge from these data. Data mining is widely applied in the database research area in order to extract frequent correlations of values from both structured and semi-structured datasets. In this work we describe an approach to mine tree-based association rules from XML documents. Such rules provide information on both the structure and the content of XML documents; moreover, they can be stored in XML format to be queried later on. The mined knowledge is approximate, intensional knowledge used to provide: (i) quick, approximate answers to queries and (ii) information about structural regularities that can be used as dataguides for document querying. A prototype of the proposed system is also briefly described.

1. INTRODUCTION
In the recent years the database research field has concentrated on XML (eXtensible Markup Language [29]) as an expressive and flexible hierarchical model suitable to represent huge amounts of data with no absolute and fixed schema, and with a possibly irregular and incomplete structure. Despite its impressive growth in popularity, XML is still lacking appropriate techniques to retrieve datasets available to casual users; such datasets, on one hand, have a limited or absent structure, and on the other hand contain a huge amount of data.

Together with intrinsically unstructured documents, there is a significant portion of XML documents which have only an implicit structure, that is, their structure has not been declared in advance, for example via a DTD or an XML-Schema [26]. Querying such documents is quite difficult for users for two main reasons: 1) they are not able to specify a reasonably probable structure in the query conditions and 2) they are very often confused by the large amount of information available.

This limitation of XML is a crucial problem, which did not emerge in the past years in the context of traditional (relational) database management systems, and must be addressed in order to provide access to these data to a wider set of users.

The application of data mining techniques to extract useful knowledge from XML datasets has received a lot of attention in the recent years due to the wide availability of these datasets. In particular, the process of mining association rules to provide summarized representations of XML documents has been investigated in many proposals and in particular either by using languages (e.g. XQuery) and techniques developed in the XML context, or by implementing graph/tree-based algorithms.

By mining frequent patterns from XML documents, we provide the users with partial, and often approximate, information both on the document structure and on the content. Such patterns can be useful for the users to obtain information and implicit knowledge on the documents and thus be more effective in query formulation. Moreover, this information is also useful for the system, which is provided with discovered information, like hidden integrity constraints, which can be used for semantic optimization.

Goal and contributions
The goal of this paper is to provide a method for mining intensional knowledge from XML datasets expressed by means of association rules. In particular, we propose Tree-based association rules as a means to represent such an intensional knowledge in native XML language. We introduce a proposal for mining, and also storing tree-based association rules for two main purposes: 1) to get a concise view of both the structure and the content of XML documents, and 2) to use them for intensional query answering. Our mining procedure is characterized by the following key aspects: 1) it works directly on the XML documents, without transforming the data into relational or any other intermediate format, 2) it looks for general association rules, without the need to impose what should be contained in the antecedent and consequent of the rule, and 3) it stores association rules in XML format.

(This research is partially supported by the Italian MIUR project ARTDECO)
Structure of the paper
The paper is organized as follows. Section 2 introduces other work related to XML association rule mining and usage. Section 3 explains what tree-based association rules are and two possible applications of such rules then Section 4 presents how tree-based rules are extracted from XML documents and how they are used to respond to intensional queries. Section 5 describes a prototype that implements our proposal whereas Section 6 explains the experimental results obtained by testing our prototype on real XML datasets. Section 7 at last, states the possible follow-ups to this work.

2. RELATED WORK
The problem of association rule mining was initially proposed in Agrawal 2 and successively many implementations of the algorithms, downloadable from [14], were developed and described in the database literature. Weka 1 being a known framework. More recently the problem has been investigated also in the XML context [7,30,24]. In [30] the authors use XQuery 28 to extract association rules from simple XML documents. They propose a set of functions written only in XQuery which implement together the Apriori algorithms. 2. The same authors show in [30] that their approach performs well on simple XML documents; however it is very difficult to apply this proposal to complex XML documents with an irregular structure. This limitation has been overcome in [24], where the authors introduce a proposal to enrich XQuery with data mining and knowledge discovery capabilities, by introducing XMINE RULE, a specific operator for mining association rules for native XML documents. They formalize the syntax and an intuitive semantics for the operator and propose some examples of complex association rules.

However, the operator proposed in [7] uses the MINE RULE operator, which works on relational data only. This means that, after a step of pruning of unnecessary information, the XML document is translated into the relational format. Moreover, both [7] and [30] force the designer to specify the structure of the rule to be extracted and then to mine it, if possible. This means that the designer has to specify what should be contained in the body and head of the rule, i.e. the designer has to know the structure of the XML document in advance, and this is an unreasonable requirement when the document has not an explicit DTD. Another limitation of these approaches is that the extracted rules have a fixed root, thus once the root node of the rules to mine has been fixed, only its descendants are analyzed. Let us consider the dataset in Figure 1 to explain this consideration. In order to infer the co-author relationship among authors of conference articles, it is necessary to fix the root node of the rules in the article element, the body and head in author. In such way it is possible to learn that “John Black” and “Mark Green” frequently write papers together. However, it is not possible to mine itemsets stating that frequently, during “2008” conferences have been held in “Milan”. Indeed, to mine such property the body of the rules should be fixed in the year element, which is not contained in the subtree of the article node, and the head in place.

Our idea is to take a more general approach to the problem of extracting association rules from XML documents, i.e. to mine all frequent rules, without having any a-priori knowledge of the XML dataset. A similar idea was presented in [24] where the authors introduced HoPS, an algorithm for extracting association rules in a set of XML documents. Such rules are called XML association rules and are implications of the form $X \Rightarrow Y$, where $X$ and $Y$ are fragments of an XML document. In particular the two trees $X$ and $Y$ have to be disjunct.

The limitation of the proposal in [24] is that it does not contemplate the possibility to mine general association rules within a single XML dataset, while achieving this feature is one of our goals.

The idea of using association rules as summarized representations of XML documents was also introduced in [5] where the XML summary is based on the extraction of association rules both on the structure (schema patterns) and on content values (instance patterns) from XML datasets. The limitation of such an approach is that the so-called schema patterns, used to describe general properties of the schema applying to all instances, are not mined, but derived as an abstraction of similar instance patterns.

In our work, XML association rules are mined starting from frequent subtrees of the tree-based representation of a document. In the database literature it is possible to find many proposals of algorithms to extract frequent structures from tree/graph-based data structures. Just to cite some of them, Tree Miner [29], Path Join [31], CloseGraph [32] propose algorithms to directly mine frequent itemsets -not association rules- from XML documents. Tree Miner and CloseGraph do not preserve the exact structure of the itemsets extracted - only the "descendant-of" (and not the "child-of") relationship between nodes is preserved - whereas Path Join does.

In this work we propose an algorithm that extends Path Join to mine generic tree-based association rules directly from XML documents.

3. TREE-BASED ASSOCIATION RULES
Association rules $2$ describe the co-occurrence of data items in a large amount of collected data and are usually represented as implications in the form $X \Rightarrow Y$, where $X$ and $Y$ are two arbitrary sets of data items, such that $X \cup Y = \emptyset$. The quality of an association rule is usually measured by means of support and confidence. Support corresponds to the frequency of the set $X \cup Y$ in the dataset, while confidence corresponds to the conditional probability of finding $Y$, having found $X$ and is given by $sup(X \cup Y)/sup(X)$.

In this work we extend the notion of association rule originally introduced in the context of relational databases, in order to adapt it to the hierarchical nature of XML documents.

In particular, we consider the element-only Infoset content model [27], which allows an XML nonterminal tag to include only other elements and/or attributes, while the text is confined to terminal elements. Furthermore, without loss of generality, we do not consider some features of the Infoset that are not relevant to the present work, such as names-
paces, the ordering label, the referencing formalism through ID-IDREF attributes, URIs, and Links.

Following the Infoset conventions, we represent an XML document by a labeled treestructure \((N, E, r)\) where \(N\) is the set of nodes, \(r \in N\) is the root of the tree (i.e. the root of the XML document), \(E\) is the set of edges. Moreover, the following properties on nodes and edges hold: 1) Each node \(n_i\) has a tuple of labels \(N_i = (\text{Ntag}, \text{Ntype}, \text{Ncontent})\); the type label \(\text{Ntype}\) indicates whether the node is the root, an element, text, or attribute \(^2\) whereas the label \(\text{Ncontent}\) can assume as value a PCDATA or \(\perp\) (undefined, for nonterminals). 2) Each edge \(e_j = (n_i, n_k, EL_j)\), with \(n_i\) and \(n_k\) in \(N\), has a label \(EL_j = (\text{Etype})\), \(\text{Etype} \in \{\text{attribute, sub-element of}\}\). Note that edges represent the “containment” relationship between different items of an XML document, thus edges do not have names.

We are interested in finding relationships among subtrees of XML documents. Thus, we do not distinguish between textual content of leaf elements and value of attributes. As a consequence, in order to draw graphical concepts in a more readable way, we do not report the edge label and the node type label. Attributes and elements are characterized by empty circles, whereas the textual content of elements, or the value of attributes, is reported under the outgoing edge of the element or attribute it refers to.

### 3.1 Fundamental concepts

Given two labeled trees \(T = (\langle N_T, E_T, r_T \rangle)\) and \(S = (\langle N_S, E_S, r_S \rangle)\), \(S\) is said to be an induced subtree of \(T\) \(^3\) if and only if there exists a mapping \(\theta : N_i \rightarrow N_T\) such that \(\forall n_i \in N_S, NL_i = NL_j\), where \(\theta(n_i) = n_j\) and for each edge \(e_j = ((n_1, n_2), EL_j) \in E_S, ((\theta(n_1), \theta(n_2)), EL_j) \in E_T\).

Figure 2 shows an example of an XML document (Figure 2a)), its tree-based representation (Figure 2b)) and three induced subtrees of the document (Figure 2c)).

A **Tree-based Association Rule (TAR)** is a tuple of the form \(T_r = (S_B, S_H, s_T, c_T)\), where \(S_B = (\langle n_B, E_B, r_B \rangle)\) and \(S_H = (\langle n_H, E_H, r_H \rangle)\) are trees and \(s_T\) and \(c_T\) are real numbers representing the support and confidence of the rule respectively. Furthermore, \(S_B\) is an induced subtree of \(S_H\) with an additional property on the node labels. Indeed, the following properties must hold:

- \(N_B \subseteq N_H\)
- \(E_B \subseteq E_H\) and \(\forall n, m \in N_B, ((n, m), EL) \in E_B\) iff \(\langle(n, m), EL \rangle \in E_H\)
- the set of tags of \(S_B\) is equal to the set of tags of \(S_H\) with the addition of the empty label “\(\perp\)”.  

The empty label is introduced because the body of a rule may contain nodes with unspecified tags (a sort of placeholder nodes) and these tags will be declared in the head part of the rule (see the rule (4) of Figure 2). For the sake of clarity the label \(\epsilon\) is omitted in the figures and all nodes with empty labels do not present any label at all.

It is worth noticing that Tree-based association rules are different from XML association rules \(^4\) because the first require that \((X \not\subseteq Y) \land (Y \not\subseteq X)\), i.e. the two trees \(X\) and \(Y\) have to be disjunct, while Tree-based association rules require that \(X\) is an induced subtree of \(Y\).

A TAR represents intensional knowledge in the form \(S_B \Rightarrow \)
$S_H$, where $S_B$ is the body tree and $S_H$ the head tree of the rule. Indeed, the rule $S_B \Rightarrow S_H$ states that if the tree $S_H$ appears in an XML document $D$, it is likely that the “bigger”, or more “specific”, tree $S_H$ also appears. In our graphical representation, we will render the nodes of the body of a rule by black circles, and the nodes of the head by empty circles.

Thus, every tree-based association rule is characterized by two measures:

- $s_{Tr}$, support, measures the frequency of the tree $S_H$ in the XML document
- $c_{Tr}$, confidence, measures the reliability of a rule, that is the frequency of the tree $S_H$, once $S_B$ has already been found

Given function $count(S,D)$ denoting the number of occurrences of a subtree $S$ in the tree $D$ and function $cardinality(D)$ denoting the number of nodes of $D$, it is possible to define formally the two measures as:

$$support(S_B \Rightarrow S_H) = \frac{count(S_B, S_H, D)}{cardinality(D)}$$

$$confidence(S_B \Rightarrow S_H) = \frac{count(S_B, S_H, D)}{count(S_B, D)}$$

Given an XML document it is possible to extract two types of tree-based association rules:

- **iTARs**: instance TARs are association rules providing information both on the structure and on the PCDATA values contained in a target XML document (see Figure 3).
- **sTARs**: structure TARs are association rules on the structure of the XML document. An sTAR is a tuple $T_i = (S_B, S_H, s_T, c_T)$ where, for each node $n$ either in $S_B$ or in $S_H$, $n$ has as label a triple $\langle \text{TAG}, \text{TYPE}, \bot \rangle$, i.e. no PCDATA is present in an sTAR (see Figure 5 for some examples).

Table 1: Support and confidence of rules in Fig 5

<table>
<thead>
<tr>
<th>rule</th>
<th>rule support</th>
<th>body support</th>
<th>rule confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>3/28 = 0.10</td>
<td>3/28 = 0.10</td>
<td>3/3 = 1.00</td>
</tr>
<tr>
<td>(2)</td>
<td>2/28 = 0.07</td>
<td>3/28 = 0.10</td>
<td>2/3 = 0.66</td>
</tr>
<tr>
<td>(3)</td>
<td>3/28 = 0.07</td>
<td>3/28 = 0.10</td>
<td>3/3 = 1.00</td>
</tr>
</tbody>
</table>

Figure 4: Sample dataset

It is possible to notice, from the examples, that tree-based association rules, in addition to correlation of PCDATA values, provide information about the structure of frequent portions of XML documents; thus they are more expressive than classical association rules which only provide us with frequent correlations of flat values.

Similarly, Figure 5 shows some examples of sTARs referred to the XML document in Figure 3. Such dataset is intentionally more complex and irregular than the dataset in Figure 3 in order to better show the potential of sTARs. In particular, rule (1) states that if there is a node labeled $A$ in the document, probably that node has a child labeled $B$. Rule (2) states that if there is a node labeled $B$, probably its parent is labeled $A$. Rule (3) states that if a node $C$ has a child labeled $B$, it probably also has a child labeled $E$. To conclude, Rule (4) states that if a node $A$ is the grandparent of a node $C$ (note, in the body of the rule, the empty label of the parent of node $C$), probably the child of $A$ and parent of $C$, is labeled $B$. Table 2 shows, for each mined rule, its support and confidence.

3.2 The use of TARs

Tree-based association rules provide an approximate view of both the content and the structure of an XML document. Such features can be used for:

1. providing intensional, although approximate, answers

Table 2: Support and confidence of rules in Fig 5

<table>
<thead>
<tr>
<th>rule</th>
<th>rule support</th>
<th>body support</th>
<th>rule confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>4/34 = 0.11</td>
<td>7/34 = 0.2</td>
<td>4/7 = 0.57</td>
</tr>
<tr>
<td>(2)</td>
<td>4/34 = 0.11</td>
<td>12/34 = 0.35</td>
<td>4/12 = 0.33</td>
</tr>
<tr>
<td>(3)</td>
<td>3/34 = 0.08</td>
<td>5/34 = 0.14</td>
<td>3/5 = 0.6</td>
</tr>
<tr>
<td>(4)</td>
<td>2/34 = 0.05</td>
<td>6/34 = 0.17</td>
<td>2/6 = 0.33</td>
</tr>
</tbody>
</table>
to user queries;


3.2.1 Intensional answers to queries

The classes of queries that can be managed with our approach have been introduced in [13] and further analyzed in the relational database context in [5]. We show some simple examples for four classes of queries, discussed in the following.

Class 1: this kind of query is used to impose a simple, or complex (containing AND and OR operators), restriction on the value of an attribute or the content of a leaf node. An example is “Find the articles written by Mark Green”. The query imposes a constraint on the content of the author element; the XQuery expression, to be applied on the original XML dataset, is:

```xml
<result> {
  for $a in document("conferences.xml")//article
  where $a/author = 'Mark Green'
  return $a
}</result>
```

Class 2: this kind of query is used to retrieve some properties described in the subtrees rooted in a specified element, possibly ordering the result. An example is “Retrieve in an ascending order the titles of all articles”. The XQuery expression, which does not specify any constraint on the content of elements, is:

```xml
<result> {
  for $t in document("conference.xml")//title
  order by $t ascending
  return $t
}</result>
```

Class 3: this kind of query is used to count the number of elements with a specific content. To answer this queries we use an association rule whose body matches the query conditions, and obtain as answer \( \frac{sup(A)}{conf(A \Rightarrow B)} \), where \( A \Rightarrow B \) is an association rule. More specifically, to count the elements satisfying a condition on their content, we use the equality \( conf(A \Rightarrow B) = \frac{sup(A \Rightarrow B)}{sup(A)} \); here \( A \Rightarrow B \) is an association rule. For example, in the “conferences.xml” document, we can count the different articles written by “Mark Green” by using an association rule \( A \Rightarrow B \) that has the path article-author (with content “Mark Green”) in the body, thus the number of articles is given by \( sup(A) = \frac{sup(A \Rightarrow B)}{conf(A \Rightarrow B)} \).

Class 4: this kind of query is used to select the best \( k \) answers satisfying a counting and grouping condition, for example “Retrieve the \( k \) authors who wrote the highest number of articles”.

3.2.2 Provide a DataGuide

The extracted sTARs can be used as a sort of approximate, but also partial, DataGuide [15] of the original document; these rules can help the user to gain knowledge about the structure of the most frequent subtrees and to formulate queries on these subtrees. As an example, the sTAR mined from the XML dataset in Figure 1 with support 0.03...
Figure 6: A sSTAR mined from the XML dataset in Figure 1

and confidence 1, is shown in Figure 6 whenever there is a conference element with an articles sub-element (see the antecedent), then the structure of the XML document rooted in the conference node is the one represented by the tree in the consequent part of the rule (right-hand-side of the figure).

4. TAR EXTRACTION AND PROCESSING

In this section we show how Tree-based rules are extracted from an XML document. Moreover, we explain how they are stored and used to respond to the types of queries introduced in Section 3.2.1.

4.1 Association Rule Extraction

Mining tree-based association rules is a process composed of two steps:

1. mining frequent subtrees from the XML document;
2. computing interesting rules from the previously mined frequent subtrees.

As already said, the problem of finding frequent subtrees has been widely discussed in the literature [35, 4, 32, 34, 36]. The problem of computing interesting rules from frequent subtrees can be compared with the problem of extracting classical association rules from large sets of elements, initially introduced in [2].

Algorithm 1 shows how tree-based association rules are mined. The inputs of the algorithm are the set of frequent subtrees, $F_S$, and the minimal threshold for the confidence of the rules, $minconf$.

Algorithm 1 Get-Interesting-Rules ($F_S$, $minconf$)

1: ruleSet = $\emptyset$
2: for all $s \in F_S$ do
3:    tempSet = Compute-Rules($s$, $minconf$)
4:    ruleSet = ruleSet $\cup$ tempSet
5: end for
6: return ruleSet

Depending on the amount of frequent subtrees and their cardinality, the amount of generated rules may be very high. The explosion of the number of mined association rules w.r.t. the number of frequent itemsets occurs also in the relational context; an optimization of the basic algorithm has been proposed in [2]. Such optimization will be adapted to our XML context, for mining tree-based association rules.

In fact, given a frequent subtree $S$, all rules derived from that tree have the same support and different confidences. Since the confidence of a rule $S_B \Rightarrow S_H$ can be computed as $supp(S_H)/supp(S_B)$, the support of $S_B$ influences the confidence of rules having the same body tree; the higher the support of the body tree, the lower is the confidence of the rule.

Figure 7 shows a frequent subtree (Figure 7(a)) and three possible tree-based rules mined from the tree; all three rules have the same support $k$ and confidence to be determined.

Let the support of the body tree of rule (1) be $s$. Since the body trees of rules (2) and (3) are subtrees of the body tree of rule (1), their support is at least $s$, and possibly higher. This means that the confidences of rule (2) and (3) are equal, or lower, than the confidence of rule (1).

Starting from this consideration, it is possible to state the following property, that allows us to optimize Algorithm 2.

Property If the confidence of a rule $T$ is below the established threshold $c$ then the confidence of every other rule $T_i$, such that its body $S_{B_i}$ is an induced subtree of the body $S_{B_T}$, is no greater than $c$.

Algorithm 2 shows how tree-based rules are mined exploiting the introduced optimization. Note that it is advisable to first generate the rules with the highest number of nodes in the body tree. For example, let us consider two rules $T_1$ and $T_2$ whose body trees contain 1 and 3 nodes respectively, as shown in Figure 8(a). Suppose both rules have confidence below the fixed threshold. If the algorithm considers rule $T_2$ first, all rules that have the bodies shown in Figure 8(b) will be discarded when $T_2$ is eliminated. On the other hand, since the body tree of $T_1$ has only one node and therefore has no induced subtree different from itself, starting from $T_1$ will not produce any optimization. Therefore, it is more convenient to first generate rule $T_2$ and in general, to start the mining process from the rules with a larger body.

Once the mining process has been performed and frequent tree-based rules have been extracted, we store them in XML format. This decision has been taken to allow the use of the same language (e.g. XQuery) for both the original dataset, and the mined rules.
4.2 Intensional query processing

iTARs can be used to provide intensional information about the actual data contained in the mined XML documents, when these documents are no available or reachable any more, or when the user prefers to obtain a faster (possibly partial) answer. Of course, this choice is worthwhile only if intensional queries processing is faster than extensional query processing. Therefore we introduce indices on tree-based association rules in order to improve the access to mined trees and in general the process of intensional query answering. We consider indices on both structure TARs and instance TARs.

Given a set of rules, the index associates with every distinct node n in this set, the references to those rules which contain n. Therefore, an index is a set of tuples \( \langle n, rif \rangle \) where n is a node and rif is a list of references to the rules. Algorithm 3 shows how the data structure for the index, which is then translated in XML format, is constructed, given the set of rules \( R \).

Algorithm 3 Compute-Rules \((s, \text{minconf})\)

1: \text{ruleSet} = \emptyset
2: \text{blackList} = \emptyset
3: \text{for all} \ c_s, \text{subtrees of} \ s \text{ do}
4: \quad \text{if} \ c_s \text{ is not an induced subtree of any element in blackList \ then}
5: \quad \quad \text{conf} = \frac{\text{supp}(s)}{\text{supp}(c_s)}
6: \quad \quad \text{if} \ \text{conf} \geq \text{minconf} \text{ then}
7: \quad \quad \quad \text{newRule} = \langle c_s, s, \text{conf}, \text{supp}(s) \rangle
8: \quad \quad \quad \text{ruleSet} = \text{ruleSet} \cup \{\text{newRule}\}
9: \quad \quad \text{else}
10: \quad \quad \quad \text{blackList} = \text{blackList} \cup c_s
11: \quad \quad \text{end if}
12: \quad \text{end if}
13: \text{end for}
14: \text{return ruleSet}

Figure 7 shows the two DTDs representing the structure of iTARs (Figure 7(a)) and sTARs (Figure 7(b)). In order to save space, only the head tree of a rule is stored. We exploit the fact that the body of the rule is a subtree of the head and use two attributes in the Node element to identify 1) the nodes that are also part of the body tree (the role attribute), and 2) those that have an empty label in the body (the isGeneric attribute).

Algorithm 4 Create-Index \((R)\)

1: \text{Index} \leftarrow \emptyset
2: \text{for all} \ r_i \in R \text{ do}
3: \quad \text{for all} \ n_j \in r_i \text{ do}
4: \quad \quad \text{if} \ \exists \ n, rif \in \text{Index}: \ n = n_j \text{ then}
5: \quad \quad \text{Index} = \text{Index} \cup \{\langle n_j, \emptyset \rangle\}
6: \quad \quad \text{end if}
7: \quad \quad \langle n, rif \rangle = \langle n_j, rif \cup \{r_i\} \rangle
8: \quad \quad \text{end for}
9: \quad \text{end for}
10: \text{return Index}

Algorithm 3 is used to create both the index on sTARs and the index on iTARs. The difference between the two indices
Indices, like rules, are stored in XML format. Figure 10 shows the two DTDs for the index on instance rules (Figure 10(a)) and the index on structure rules (Figure 10(b)).

Given a query $q$ and the intensional data represented by the file containing instance TARs and the file containing the index on such rules, it is possible to obtain the intensional answer in two steps:

1. query the index file, with a rewriting of the query $q$, looking for the references to the rules which satisfy the conditions imposed by the query $q$;
2. query the TARs file in order to return the rules whose reference was found in the previous step;

In this section we show the process of obtaining an intensional answer when the user composes an XQuery expression. In order to perform the two steps described above we introduce the XQuery definition of the following functions:

- **references**: given a pair $(name, content)$, describing a node of an XML tree, the function returns a set of references of the rules containing the node. The XQuery implementation of the function is:

```xquery
define function references (element $name, element $content) returns element {
  for $node in document("index.xml")//Node
  where $node/@label = $name and
  $node/Value = $content
  return $node/Reference
}
```

- **setinclusion**: given a set of elements and a specific element $e$, the function returns true when the element $e$ is in the set, false otherwise. The XQuery implementation of the function is:

```xquery
define function setinclusion(element $elem, element $set) as xs:boolean {
  for $s in $set
  where $s = $elem
  return true;
}
```

- **union**: the following XQuery function returns the union of two sets of elements:

```xquery
define function union (element $rif1, element $rif2) returns element {
  let $ins := (for $r2 in $rif2
    where every $r1 in $rif1 satisfies
    $r1 != $r2
  ) return $r2)
  return $rif1 union $ins
}
```

- **intersection**: the following XQuery function returns the intersection of two sets of elements:

```xquery
define function intersection (element $rif1, element $rif2) returns element {
  for $r1 in $rif1
  where setinclusion($r1, $rif2)
  return $r1
}
```

- **ruleset**: given a set of references, the function returns the set of references of the rules associated with the input references. The XQuery implementation of the function is:

```xquery
define function ruleset(element $rif) returns element {
  for $r in document("rules.xml")//Rule
  where setinclusion($r/@ID, $rif//RuleID)
  return $r
}
```

These functions allow us to easily obtain the intensional answer to the classes of XQuery expressions we have analyzed in Section 3.2. Indeed the intensional answer for an XQuery $e$ is obtained by applying such functions to the XML document of previously mined tree-based rules.

As an example, the intensional answer to the considered Class I query “Find the articles written by Mark Green” of Section 3.2 is obtained by the following function call:

```xquery
query (‘author’, ‘Mark Green’)
```

where the function $query$ is defined as follows:
define function query (element $name, element $content) returns element {
    let $Rif = references($name, $content)
    let $Rule = rules ($Rif)
    return $Rule
}

This expression retrieves useful intensional information by performing the two steps 1 and 2 described above: first, the variable $Rif is initialized with the set of index references of the rules containing the element author with content “Mark Green”. Then, the variable $Rule is initialized with the set of rules whose reference is in $Rif. The answer is an XML document composed by the tree-based rules satisfying the constraints indicated in the original query.

5. THE TREERULER PROTOTYPE
TreeRuler is a prototype tool that integrates all the functionalities proposed in our approach. Given an XML document, the tool is able to extract intensional knowledge, and allows the user to compose traditional queries as well as queries over the intensional knowledge.

Figure 11 shows the architecture of the tool. In particular, given an XML document, it is possible to extract Tree-based rules and the corresponding index file. The user formulates XQuery expressions on the data, and these queries are automatically translated in order to be executed on the intensional knowledge. The answer is given in terms of the set of Tree-based rules which reflect the search criteria.

A screenshot of the tool is shown in Figure 12: it is composed by several tabs for performing different tasks. In particular, there are three tabs:

- **get the Gist** (Figure 12) allows intensional information extraction from an XML document, given the desired support, confidence and the files where the extracted tree-based rules and their index must be stored.
- **get the Idea** allows the visualization of the intensional information as well as the original document, in order to give the user the possibility to compare the two kinds of information.
- **get the Answers** (Figure 13) allows to query the intensional knowledge and the original XML document. The user has to write an extensional query in the box on the left; when the query belongs to the classes we have analyzed it is translated into the intensional form, shown to the user in the right part of the form. Finally, once the query is executed, the Tree-based rules that reflect the search criteria are shown in the box at the bottom of the form.

5.1 Technical aspects
TreeRuler is implemented in C++ with the use of the eXpat library for XML parsing, and of the wxWidgets tools for the GUI. At present, TreeRuler implements the PathJoin algorithm for the extraction of frequent subtrees from the XML document.

6. EXPERIMENTAL RESULTS
In this section we introduce three types of experiments we performed with TreeRuler to evaluate the proposed underlying approach:

- experiments performed in order to monitor the time required for the extraction of the intensional knowledge from an XML database;
Figure 13: TreeRuler “get the Answers” tab

- experiments performed in order to monitor the time needed to answer intensional and extensional queries over an XML file;
- a use case scenario on the DocBook XML database, in order to monitor extraction times given a specific support or confidence.

6.1 Extraction time
We have performed some experiments on real XML datasets. First we tried to execute TreeRuler on the datasets found at the XMLData Repository but all the documents were too structured and the extracted intensional knowledge was very poor. Moreover the DTD for all document was already provided. Then we used GCC_XML in order to create XML documents starting from C++ source code. The documents produced by GCC_XML were unstructured and without a DTD.

In particular, we used these XML datasets to monitor the time TreeRuler takes to extract intensional knowledge, i.e. tree-based association rules and the corresponding index, from the XML file. Figure 14 shows how extraction time growth depends on the number of nodes in the XML document. It is possible to notice that extraction time growth is almost linear with respect to the cardinality of an XML tree. Moreover, the compression factor provided by the XML representation of TARs, compared to the size of the original XML dataset, is significant (e.g. 264 KB w.r.t. 4 KB).

6.2 Answer time
We have performed some experiments to monitor the time needed to obtain both intensional and extensional answers to user queries. Figure 15 shows, for each XML document we considered, the time TreeRuler took to give an intensional and extensional answer to a simple selection query.

In particular we performed the following extensional query on the analyzed XML documents:

```
define function ExtQuery (element document, element node-name) returns element {
    for $x$ in doc//node-name
    order by $x$ ascending
    return $x$
}
```

where `document` is the name of the considered XML file, and `node-name` is the name of a node contained in the document (do note that it changes on the basis of the XML document). Such query was translated by the prototype into the following intensional form and was executed on the knowledge we mined from each XML document:

```
define function IntQuery (element node-name) returns element {
    for $x$ in document("rules.xml")//Node[@name=node-name]
    order by $x$ ascending
    return $x$
}
```

It is possible to notice that intensional times are always significantly smaller than extensional ones (actually almost constant), thus proving the effectiveness of our approach.

Recall and precision are commonly used to evaluate the accuracy of approaches which return approximate answers. Recall measures the fraction of data in the extensional query result which is actually returned by querying TARs. Precision measures the fraction of the returned data which correctly matches the query. TARs are characterized by 100% precision for all query classes described in Section 5.2.1.

[http://www.gccxml.org/HTML/Index.html](http://www.gccxml.org/HTML/Index.html)
6.3 Use case scenario

In this section we report the results we have obtained by applying TreeRuler on the DocBook XML database. In particular we show the relationships between support, confidence, and the extraction time of tree-based rules.

By setting the confidence at 0.95, Figure 16 shows how the extraction time changes with respect to the support. Similarly, by setting the support at 0.02, Figure 17 shows how the extraction time changes with respect to the confidence.

7. CONCLUSIONS AND FUTURE WORK

In this work we have proposed an algorithm that extends PathJoin and allows us to extract frequent tree-based association rules from XML documents. The main goals we have achieved are: 1) we mine frequent association rules without imposing any a-priori restriction on the structure and the content of the rules; 2) we store mined information in XML format; as a consequence, 3) we can effectively use the extracted knowledge to gain information, by using query languages for XML, about the original datasets where the mining algorithm has been applied.

We have developed a C++ prototype that has been used to test the effectiveness of our proposal. As an ongoing work, we are studying how to further optimize our mining algorithm; moreover, we would like to find the exact fragment of XQuery expressions we can manage with intensional knowledge.

8. REFERENCES


http://www.w3C.org/TR/xquery.


