Mining tree-based association rules from XML documents*

Mirjana Mazuran
Politecnico di Milano
Italy
mazuran@elet.polimi.it

Elisa Quintarelli
Politecnico di Milano
Italy
quintarelli@elet.polimi.it

Letizia Tanca
Politecnico di Milano
Italy
tanca@elet.polimi.it

February 6, 2009

Abstract

The increasing amount of very large XML datasets available to casual users is a most challenging problem for our community, and calls for an appropriate support to efficiently gather knowledge from these data. Data mining, already widely applied to extract frequent correlations of values from both structured and semi-structured datasets, is the appropriate tool for knowledge elicitation. In this work we describe an approach to extract *Tree-based association rules* from XML documents. Such rules provide approximate, intensional information on both the structure and the content of XML documents, and can be stored in XML format to be queried later on. The mined knowledge is used to provide: (i) quick, approximate answers to queries and (ii) information about structural regularities. A prototype system demonstrates the effectiveness of the approach.

1 Introduction

In the recent years the database research field has concentrated on XML (eXtensible Markup Language [25]) 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 efficient techniques to query the

*This research is partially supported by the Italian MIUR project ARTDECO and by the European Commission, Programme IDEAS-ERC, Project 227077-SMScom many datasets available to casual users, since 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 [22]. Querying such documents is quite difficult for users for two main reasons: 1) they are not able to specify a reasonably likely 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 thus 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 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 its content. Such patterns can be useful for the users to obtain information and implicit knowledge on the documents and to 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* (TAR) as a means to represent such an intensional knowledge in native XML language. 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_B appears in an XML document D, it is *likely* that the "wider", or more "detailed", 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.

We introduce a proposal for mining, and also storing TARs 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: *a*) it works directly on the XML documents, without transforming the data into relational or any other intermediate format, *b*) it looks for general association rules, without the need to impose what should be contained in the antecedent and consequent of the rule, and c) it stores association rules in XML format.

Structure of the paper

The paper is organized as follows. Section 2 explains what tree-based association rules are, then Section 3 presents how tree-based rules are extracted from XML documents. Section 4 presents two possible applications of tree-based association rules and shows how they are used to answer 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 introduces other work related to XML association rule mining and usage. Section 8, at last, states the possible follow-ups of this work.

2 Tree-based Association Rules

Association rules [1] 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 $X \cap Y = \emptyset$. 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 $sup(X \cup Y)/sup(X)$.

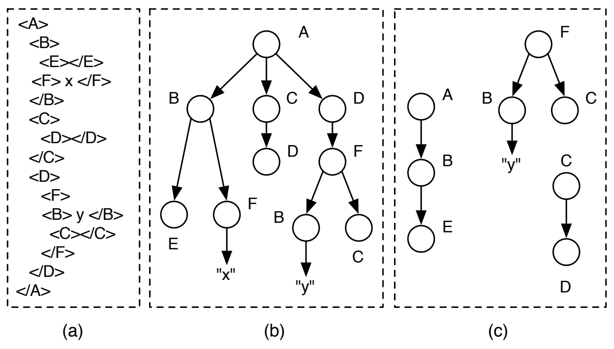
In this work we extend the notion of association rule originally introduced in the context of relational databases, in order to adapt it to the hierarchical nature of XML documents.

In particular, we consider the *element-only* Infoset content model [23], 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 namespaces, 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 tree[1] $\langle N, E, r\rangle$ 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 $NL_i = \langle Ntag_i, Ntype_i, Ncontent_i\rangle$; the type label $Ntype_i$ indicates whether the node is the root, an element, text, or attribute [2], whereas the label $Ncontent_i$ can assume as value a PCDATA or $\perp$ (undefined, for nonterminals).

[1]Note that XML documents are here tree-like structures (and not generic graphs) because we do not include referencing.

[2]XML documents may also contain `ENTITY` nodes (not unlike macro calls that must be expanded when parsing) or processing instructions or comments. We do not consider such elements in this paper.



```
<A>
  <B>
    <E></E>
   <F> x </F>
  </B>
  <C>
    <D></D>
  </C>
  <D>
    <F>
     <B> y </B>
      <C></C>
    </F>
  </D>
</A>
```

Figure 1: a) an example of XML document, b) its tree-based representation, and c) three induced subtrees

2) Each edge $e_j = \langle (n_h, n_k), EL_j \rangle$, with n_h and n_k in N, has a label $EL_j = \langle Etype_j \rangle$, $Etype_j \in \{\textit{attribute of}, \textit{sub-element of}\}$. 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.

2.1 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 [31] if and only if there exists a mapping $\theta : N_S \rightarrow N_T$ such that $\forall n_i \in N_S, NL_i = NL_j$, where $\theta(n_i) = n_j$ and for each edge $e_j = \langle (n_1, n_2), EL_j \rangle \in E_S, \langle (\theta(n_1), \theta(n_2)), EL_j \rangle \in E_T$.

Given a tree $T = \langle N_T, E_T, r_T \rangle$, a subtree of T $t = \langle N_t, E_t, r_t \rangle$ and a user fixed support threshold s_{min}:

- t is **frequent** if its support is greater or at least equal to s_{min};
- t is **maximal** if it is frequent and none of its proper supertrees is frequent.

Moreover, t is **closed** if none of its proper supertrees has support greater than t's

Figure 1 shows an example of an XML document (Figure 1(a)), its tree-based rappresentation (Figure 1(b)) and three induced subtrees of the document (Figure 1(c)).

A **Tree-based Association Rule (TAR)** is a tuple of the form $T_r = \langle S_B, S_H, s_{T_r}, c_{T_r} \rangle$, 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_r} and c_{T_r} 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. Therefore, altogether, the following properties must hold:

- $N_B \subseteq N_H$
- $E_B \subseteq E_H$ and $\forall n, m \in N_B, \langle (n,m), EL \rangle \in E_B$ 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 "ϵ".

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 5). For the sake of clarity the label ϵ is omitted in the figures and all nodes with empty labels do not present any label at all.

It is worth to point out that Tree-based association rules are different from XML association rules [18], because the latter require that $(X \not\subseteq Y) \wedge (Y \not\subseteq X)$, i.e. the two trees X and Y have to be disjunct; on the contrary Tree-based association rules require X to be an induced subtree of Y.

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

s_{T_r} *support*, measures the frequency of the tree S_H in the XML document

c_{T_r} *confidence*, measures the reliability of a rule, that is the frequency of the tree S_H, once S_B has already been found

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

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

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

Furthermore, 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 $T_i = \langle S_B, S_H, s_{T_i}, c_{T_i} \rangle$ where, for each node n either in S_B or in S_H, n has as label a triple $\langle \texttt{TAG}, \texttt{TYPE}, \perp \rangle$, i.e. no PCDATA is present in an sTAR (see Figure 5).

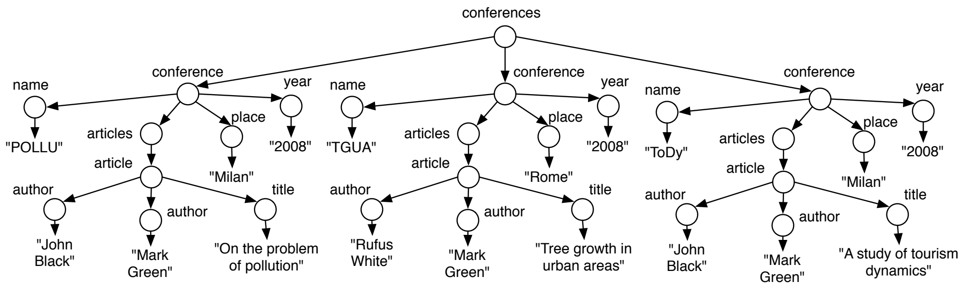


Figure 2: XML sample file: "conferences.xml"

rule	**rule support**	**body support**	**rule confidence**
(1)	$3/28 = 0.10$	$3/28 = 0.10$	$3/3 = 1.00$
(2)	$2/28 = 0.07$	$3/28 = 0.10$	$2/3 = 0.66$
(3)	$3/28 = 0.07$	$3/28 = 0.10$	$3/3 = 1.00$

Table 1: Support and confidence of rules in Fig 3

Figure 3 shows some examples of iTARs referred to the XML document in Figure 2. Rule (1) states that if there is a node labeled `conference` in the document, it probably has a child labeled `year` whose value is "2008". Rule (2) states that if there is a path composed by the following sequence of nodes: `conference/articles/article/author`, and the content of `author` is "Mark Green", then node `authors` probably has another child labeled `author` whose content is "John Black". Finally, rule (3), states that if there is a path composed by the following sequence of nodes: `conference/articles/article` and the node `article` has two children labeled `author` whose contents are "Mark Green" and "John Black", then node `conference` probably has two other children labeled `year` and `place` whose contents are respectively "2008" and "Milan". Table 1 shows, for each one of these rules, its support and confidence.

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 4. 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

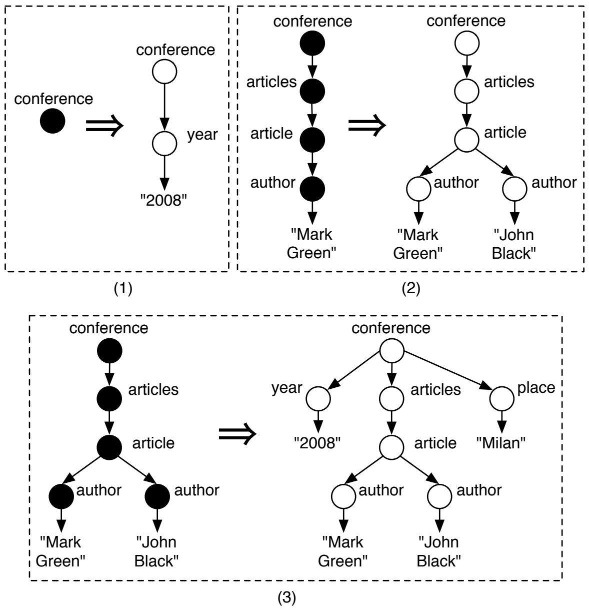


Figure 3: Sample iTARs (instance Tree-based Association Rules)

rule	**rule support**	**body support**	**rule confidence**
(1)	$4/34 = 0.11$	$7/34 = 0.2$	$4/7 = 0.57$
(2)	$4/34 = 0.11$	$12/34 = 0.35$	$4/12 = 0.33$
(3)	$3/34 = 0.08$	$5/34 = 0.14$	$3/5 = 0.6$
(4)	$2/34 = 0.05$	$6/34 = 0.17$	$2/6 = 0.33$

Table 2: Support and confidence of rules in Fig 5

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 TARs 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 [2, 27, 29–31]. The problem of computing interesting

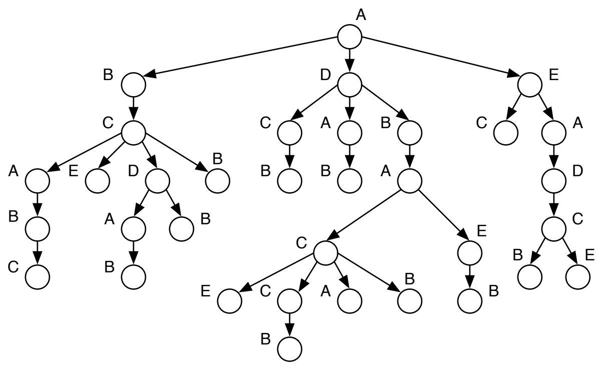


Figure 4: Sample dataset

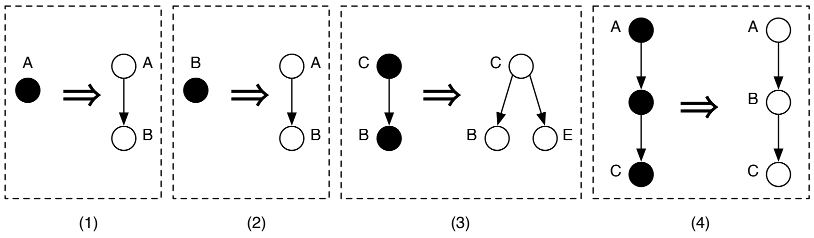


Figure 5: Sample sTARs (structure Tree-based Association Rules)

rules from frequent subtrees can be compared with the problem of extracting classical association rules from large sets of elements, initially introduced in [1].

Algorithm 1 presents the extension we introduce to CMTreeMiner in order to compute interesting TARs from frequent maximal subtrees. The authors of CMTreeMiner provide the C++ implementation for both ordered and unordered subtree extraction, which can be downoaded from http://www.nec-labs.com/ ychi/. In this work we provide a general extension which can be applied to both versions of CMTreeMiner but at this moment we focus on implementing such extension only to the ordered version.

In particular, 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
```

Algorithm 2 Compute-Rules (s, *minconf*)

```
1: ruleSet = $\emptyset$
2: for all $c_s$ subtrees of $s$ do
3:     conf = supp($s$) / supp($c_s$)
4:     if conf $\geq$ minconf then
5:         newRule = $\langle c_s, s, conf, supp(s) \rangle$
6:         ruleSet = ruleSet $\cup$ {newRule}
7:     end if
8: end for
9: 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 [1]. 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 $support(S_H)/support(S_B)$, 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.

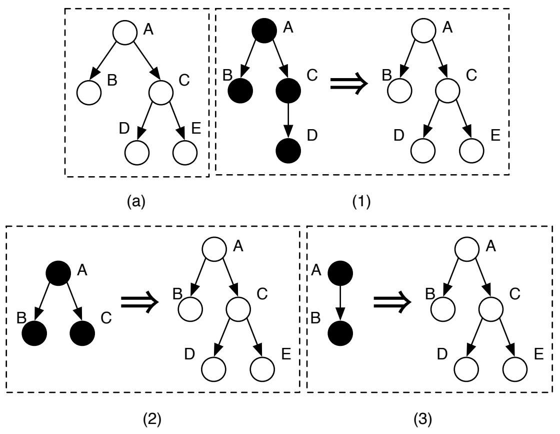


Figure 6: Rules optimization example

Figure 6 shows a frequent subtree (Figure 6 (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 $S_{B_{T_i}}$ is an induced subtree of the body S_{B_T}, is no greater than c.

Algorithm 3 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_{r1} and T_{r2} whose body trees contain 1 and 3 nodes respectively, as shown in Figure 7(a). Suppose both rules have confidence below the fixed threshold. If the algorithm considers rule T_{r2} first, all rules that have the bodies shown in Figure 7(b) will be discarded when T_{r2} is eliminated. On the other hand, since the body tree of T_{r1} has only one node and therefore has no induced subtree different from itself, starting from T_{r1} will not produce any optimization. Therefore, it is more convenient to first generate rule T_{r2} 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

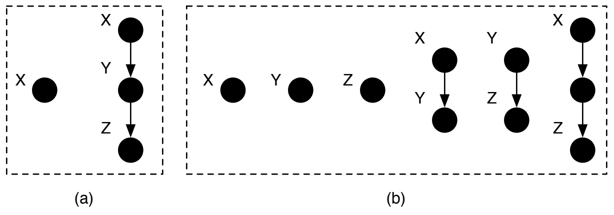


Figure 7: Rules optimization example

Algorithm 3 Compute-Rules $(s, minconf)$

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

dataset, and the mined rules.

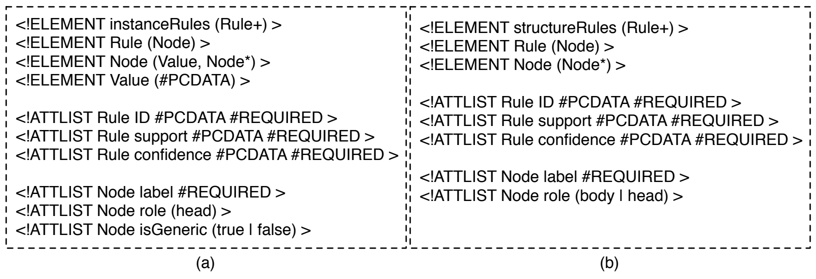
Figure 8 shows the two DTDs representing the structure of iTARs (Figure 8(a)) and sTARs (Figure 8(b)). In order to save space, only the head tree of a rule is stored. We exploit the fact that the body of the rule is a subtree of the head and use two attributes in the `Node` element to identify 1) the 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).

4 TARs usage

Tree-based association rules provide an approximate view of both the content and the structure of an XML document. In this section we explain how such features can be used for:

1. providing intensional, although approximate, answers to user queries;

2. providing an approximate DataGuide [10, 11] for the document.



```
<!ELEMENT instanceRules (Rule+) >
<!ELEMENT Rule (Node) >
<!ELEMENT Node (Value, Node*) >
<!ELEMENT Value (#PCDATA) >

<!ATTLIST Rule ID #PCDATA #REQUIRED >
<!ATTLIST Rule support #PCDATA #REQUIRED >
<!ATTLIST Rule confidence #PCDATA #REQUIRED >

<!ATTLIST Node label #REQUIRED >
<!ATTLIST Node role (head) >
<!ATTLIST Node isGeneric (true I false) >
```

(a)

```
<!ELEMENT structureRules (Rule+) >
<!ELEMENT Rule (Node) >
<!ELEMENT Node (Node*) >

<!ATTLIST Rule ID #PCDATA #REQUIRED >
<!ATTLIST Rule support #PCDATA #REQUIRED >
<!ATTLIST Rule confidence #PCDATA #REQUIRED >

<!ATTLIST Node label #REQUIRED >
<!ATTLIST Node role (body I head) >
```

(b)

Figure 8: DTD of the XML document containing tree-based rules

4.1 Intensional answers to queries

The classes of queries that can be managed with our approach have been introduced in [8] and further analyzed in the relational database context in [3]. We show some simple examples for four classes of queries, discussed in the following.

Class 1: 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 *"Retrieve the articles written by Mark Green"*. The query imposes a constraint on the content of the `author` element; the XQuery expression over the original XML dataset is:

```
for $a in document("conferences.xml")//article
where $a/author = ‘‘Mark Green’’
return $a
```

Class 2: 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:

```
for $t in document("conference.xml")//title
order by $t ascending
return $t
```

Class 3: used to *count the number of elements with a specific content.* An example is *"Retrieve the number of articles written by Mark Green"* and the XQuery expression is:

```
for $a in document("conference.xml")//article
where $a/author = ‘‘Mark Green’’
return count($a)
```

Class 4: 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"*. The XQuery expression is:

```
(for $t in document("conference.xml")//author
 order by score (count(
  for $s in document("conference.xml")//author
  where $s/text()[=$a/text()]
  return $s)) descending
 return $a
)[position()<=k]
```

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.

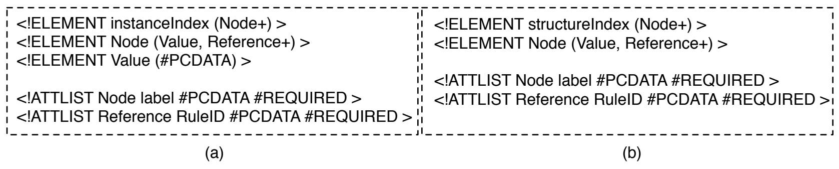
Given a set of rules, the index associates with every distinct node n in this set, the references to those rules which contain n. Therefore, an index is a set of tuples $\langle n, rif \rangle$ where n is a node and rif is a list of references to the rules. Algorithm 4 shows how the data structure for the index, which is then translated in XML format, is constructed, given the set of rules R.

Algorithm 4 Create-Index (R)

```
1: Index $\leftarrow \emptyset$
2: **for all** rule $r_i \in R$ **do**
3:   **for all** node $n_j \in r_i$ **do**
4:     **if** $\nexists \langle n, rif \rangle \in$ Index: $n = n_j$ **then**
5:       Index = Index $\cup \{\langle n_j, \emptyset \rangle\}$
6:     **end if**
7:     $\langle n_j, rif \rangle = \langle n_j, rif \cup \{r_i\} \rangle$
8:   **end for**
9: **end for**
10: **return** Index
```

Algorithm 4 is used to create both the index on sTARs and the index on iTARs. The difference between the two indices is that while for sTARs a node is identified only by its label, for iTARs a node is identified by the pair ⟨`label`, `value`⟩, where `value` is the content of the node labeled `label`.

Indices, like rules, are stored in XML format. Figure 9 shows the two DTDs for the index on instance rules (Figure 9(a)) and the index on structure rules (Figure 9(b)).



```
<!ELEMENT instanceIndex (Node+) >
<!ELEMENT Node (Value, Reference+) >
<!ELEMENT Value (#PCDATA) >

<!ATTLIST Node label #PCDATA #REQUIRED >
<!ATTLIST Reference RuleID #PCDATA #REQUIRED >
```

(a)

```
<!ELEMENT structureIndex (Node+) >
<!ELEMENT Node (Value, Reference+) >

<!ATTLIST Node label #PCDATA #REQUIRED >
<!ATTLIST Reference RuleID #PCDATA #REQUIRED >
```

(b)

Figure 9: DTD of the XML document containing indices

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:

```
define function references (element $name, element $content,
               bool $onlyName) returns element {
   for $node in document("index.xml")//Node
   where $node/@label = $name and
             ($node/Value = $content or $onlyName = true)
   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:

```
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:

```
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:

```
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:

```
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. Indeed the intensional answer for an XQuery e is obtained by applying such functions to the XML document of previously mined tree-based rules.

Class 1 the intensional answer to the query *"Retrieve the articles written by Mark Green"*, is obtained by the following generic function:

```
define function query (element $name, element $content)
       returns element {
   let $Rif=references($name, $content, false)
   let $Rule=ruleset ($Rif)
   return $Rule   }
```

where `$name` = “author” and `$content` = “Mark Green”. This expression retrieves the 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`.

Class 2 the intensional answer to the query *“Retrieve in an ascending order the titles of all articles”* is:

```
define function query (element $name) returns element {
   let $Rif = references($name, NULL, true)
   let $Rule = ruleset ($Rif)
   for $n in $Rule/Rule/Node[@label=$name]
   order by $n/Value ascending
   return $n/Value   }
```

where `$name` = “title”.

Class 3 the intensional answer to the query *“Retrieve the number of articles written by Mark Green”* is:

```
define function query (element $name, element $content)
       returns element {
   let $Rif = references($name, $content)
   let $Rule = ruleset ($Rif)
   return count($Rule)   }
```

where `$name` = “author” and `$content` = “Mark Green”. To answer these queries we use an association rule whose body matches the query conditions, and obtain as answer $\frac{sup}{conf}$ (where *sup* and *conf* are support and confidence of the rule). More specifically, to count the elements satisfying a condition on their content, we use the equality $conf(A \Rightarrow B) = \frac{sup(A \Rightarrow B)}{sup(A)}$, where $A \Rightarrow B$ is an association rule. With respect to the example 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 the intensional answer to the query *“Retrieve the k authors who wrote the highest number of articles”* is:

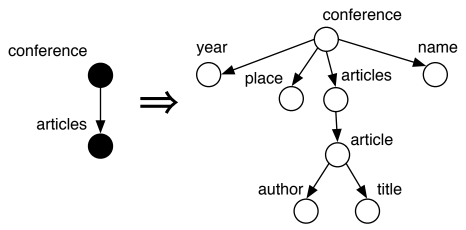


Figure 10: A sTAR mined from the XML dataset in Figure 2

```
define function query (element $name) returns element {
 let $Rif = references($name, NULL, true)
 let $Rule = ruleset ($Rif)
 ( for $a in $Rule//Node[@label=‘‘author’’]
   order by score (count(
       for $s in $Rule//Node[@label=‘‘article’’]
       where $s//Value[=$a/text()]
       return $s)) descending
   return $a
 )[position()<=k]   }
```

4.2 Provide a DataGuide

The extracted sTARs can be used as a sort of approximate, but also partial, DataGuide [10] 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 2 with support 0.03 and confidence 1, is shown in Figure 10: 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).

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.

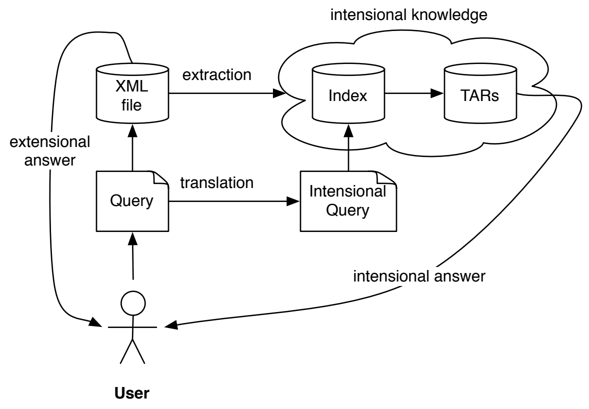


Figure 11: TreeRuler architecture

The answer is given in terms of the set of Tree-based rules which reflect the search criteria.

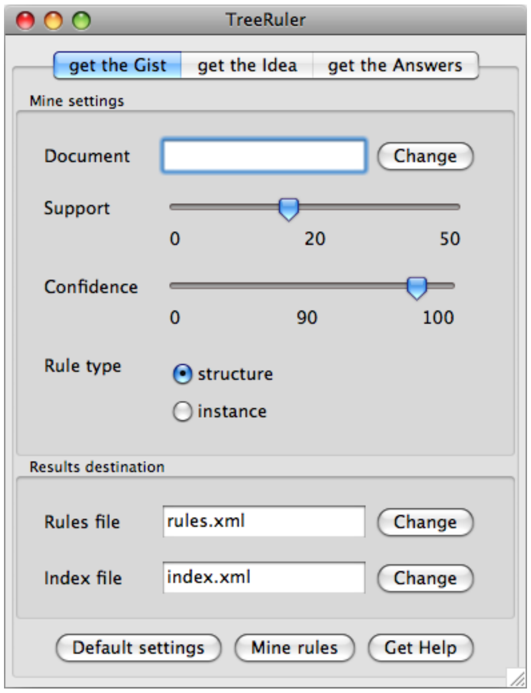
A schreenshot of the tool is shown in Figure 12: it is composed by several tabs for performing different tasks. In particular, there are three tabs:

- **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`[3] library for XML parsing, and of the `wxWidgets`[4] tools for the GUI. At present, TreeRuler implements the CMTTreeMiner [5] algorithm for the extraction of frequent subtrees from the XML document.

[3]http://expat.sourceforge.net
[4]http://www.wxwidgets.org/



Mine settings

Document

Support: 0 — 20 — 50

Confidence: 0 — 90 — 100

Rule type: (● structure) ○ instance

Results destination

Rules file: rules.xml

Index file: index.xml

Figure 12: TreeRuler “get the Gist” tab

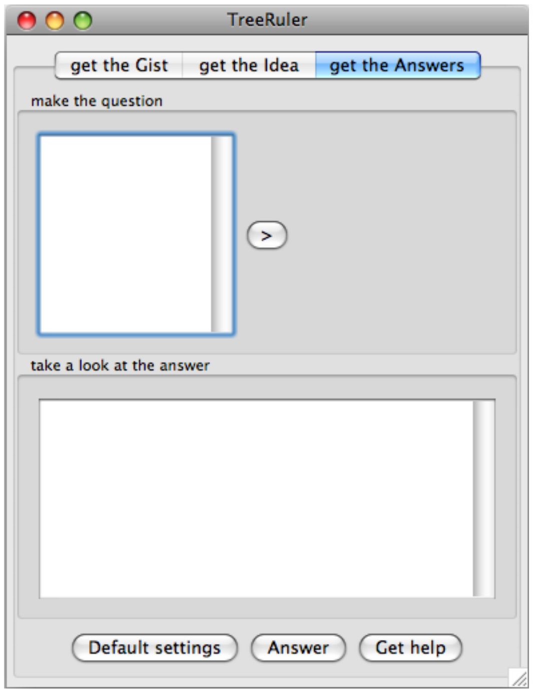


Figure 13: TreeRuler "get the Answers" tab

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;
- 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`[5] 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 Respository*[6] 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`[7] 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 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 the query of Class 2 introduced in Section 4. With respect to that query, which was evaluated on all XML datasets, `$name` is the name of a node contained in the document (such name changes on the basis of the XML document).

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.

[5]http://www.docbook.org/
[6]http://www.cs.washington.edu/research/xmldatasets/
[7]http://www.gccxml.org/HTML/Index.html

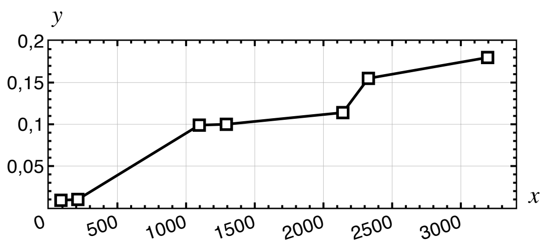


Figure 14: Extraction time growth (y = seconds), w.r.t. the number of nodes (x) in the XML tree

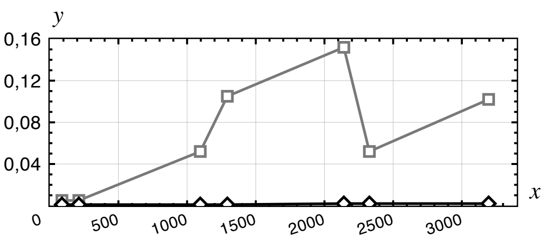


Figure 15: Extensional (curve with squares) and intensional (curve with diamonds) time answering (y = seconds) w.r.t. each document (x = number of nodes)

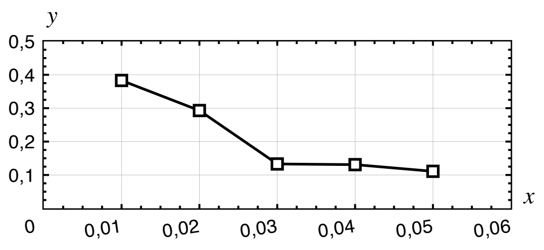


Figure 16: Extraction time growth (y = seconds), w.r.t. the support (x), given confidence = 0.95

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 4.1, while recall depends on the support threshold established before applying the mining algorithm. Indeed, the application of our mining algorithm returns only frequent subtrees and the number of such trees depends on the support threshold. Thus, the minimum support threshold strongly influences query recall, it is a relevant parameter for the intensional representation design.

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 Related work

The problem of association rule mining was initially proposed in [1] and successively many implementations of the algorithms, downloadable from [9], were developed and described in the database literature. More recently the problem has been investigated also in the XML context [4, 6, 7, 16, 18, 26, 28]. In [26] the authors use XQuery [24] to extract association rules from simple XML documents. They propose a set of functions written only in XQuery which implement together the Apriori algorithm [1]. The same authors show in [26] 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.

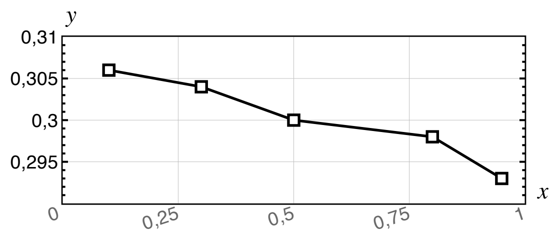


Figure 17: Extraction time growth (y = seconds), w.r.t. the confidence (x), given support = 0.02

This limitation has been overcome in [4], 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 [4] 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 [4] and [26] 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 2 to explain this consideration. In order to infer the co-author relationship among authors of conferences 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, once we fix the root of the rule in the `article` element, we can not 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 [18] 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.

Moreover both X and Y are embedded subtrees fo the XML documents which means that they do not always represent the actual structure of the data.

Another limitation of the proposal in [18] 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 [3] 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 are that: i) the root of the rule is established a-priori and ii) 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 and are therefore less precise and reliable.

In our work, XML association rules are mined starting from maximal 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 both from graph-based data representations [12,15,30] and tree-based data representations [5, 13, 14, 17, 19–21, 29, 31]. In this paper we focus on *tree mining* since XML documents are represented using a tree shaped structure.

Table 3 shows a brief overview of the most frequent tree mining algorithms with respect to the features of the input tree (ordered, unordered) and the features of the mined patterns (induced, embedded, maximal, closed).

We remark here that we are not interested in proposing yet another algorithm, but in extending an existing one in order to extract association rules within a single XML document. We choose to consider unorder XML trees, however, as desribed in Section 3, the algorithm at the basis of our work can mine also ordered trees.

Table 3: Tree mining algorithms overview

Algorithm	Ordered	Unordered	Induced	Embedded	Maximal	Closed
TreeMiner [31]	⋆			⋆		
PathJoin [29]		⋆	⋆		⋆	⋆
FREQT [19]	⋆		⋆			
DRYADE [20]		⋆		⋆		⋆
DRYADEPARENT [21]		⋆		⋆		⋆
CMTTreeMiner [5]	⋆	⋆	⋆		⋆	⋆
POTMiner [13]	⋆	⋆	⋆	⋆	⋆	⋆

In [21] the authors show that DRYADEPARENT is the current fastest tree mining algorithm and CMTTreeMiner is the second with respect to the efficiency. However, DRYADEPARENT extracts embedded subtrees wich means that the extracted tree maintains the ancestor relashionship between nodes and not the parent relashionship. In this work we are interested in extracting subtrees which maintain the parent-child relashionship. In particular we are interested in extracting induced, maximal (and therefore closed) frequent subtrees from unordered XML documents. Therefore, we propose an algorithm that extends CMTTreeMiner to mine generic tree-based association rules directly from XML documents.

8 Conclusions and future work

In this work we have proposed an algorithm that extends CMTTreeMiner [5] 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.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases*, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[2] T. Asai, H. Arimura, T. Uno, and S. Nakano. Discovering frequent substructures in large unordered trees, 2003.

[3] E. Baralis, P. Garza, E. Quintarelli, and L. Tanca. Answering xml queries by means of data summaries. *ACM Transactions of Information Systems*, 25(3):10, 2007.

[4] D. Braga, A. Campi, S. Ceri, M. Klemettinen, and P. Lanzi. Discovering interesting information in xml data with association rules. In *SAC '03: Proceedings of the 2003 ACM symposium on Applied computing*, pages 450–454, New York, NY, USA, 2003. ACM Press.

[5] Yun Chi, Yirong Yang, Yi Xia, and Richard R. Muntz. Cmtreeminer: Mining both closed and maximal frequent subtrees. In *PAKDD*, pages 63–73, 2004.

[6] C. Combi, B. Oliboni, and R. Rossato. Querying xml documents by using association rules. In *16th International Workshop on Database and Expert Systems Applications (DEXA'05)*, pages 1020–1024, 2005.

[7] L. Feng, T. S. Dillon, H. Weigand, and E. Chang. An xml-enabled association rule framework. In *14th International Conference on Database and Expert Systems Applications (DEXA '03)*, pages 88–97, 2003.

[8] S. Gasparini and E. Quintarelli. Intensional query answering to xquery expressions. In *16th International Conference on Database and Expert Systems Applications (DEXA '05)*, pages 544–553, 2005.

[9] B. Goethals and M. J. Zaki. Advances in frequent itemset mining implementations: report on FIMI'03. *SIGKDD Explor. Newsl.*, 6(1):109–117, 2004.

[10] R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 436–445, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[11] R. Goldman and J. Widom. Approximate DataGuides, 1999.

[12] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. Complete mining of frequent patterns from graphs: Mining graph data. *Machine Learning*, 50(3):321–354, 2003.

[13] Aída Jiménez, Fernando Berzal, and Juan C. Cubero. Mining induced and embedded subtrees in ordered, unordered, and partially-ordered trees. In *ISMIS*, pages 111–120, 2008.

[14] D. Katsaros, A. Nanopoulos, and Y. Manolopoulos. Fast mining of frequent tree structures by hashing and indexing. *Information & Software Technology*, 47(2):129–140, 2005.

[15] Michihiro Kuramochi and George Karypis. An efficient algorithm for discovering frequent subgraphs. *IEEE Trans. Knowl. Data Eng.*, 16(9):1038–1051, 2004.

[16] H. C. Liu and J. Zeleznikow. Relational computation for mining association rules from xml data. In *ACM 14th Conference on Information and Knowledge Management*, pages 253–254, 2005.

[17] S. Nijssen and J.N. Kok. Efficient discovery of frequent unordered trees. In *Proceedings of the first International Workshop on Mining Graphs, Trees and Sequences (MGTS'03)*, 2003.

[18] J. Paik, H. Y. Youn, and U. M. Kim. A new method for mining association rules from a collection of xml documents. In *Computational Science and Its Applications ICCSA 2005*, pages 936–945, 2005.

[19] et al. T. Asai. Efficient substructure discovery from large semi-structured data.

[20] Alexandre Termier, Marie-Christine Rousset, and Michèle Sebag. Dryade: A new approach for discovering closed frequent trees in heterogeneous tree databases. In *ICDM*, pages 543–546, 2004.

[21] Alexandre Termier, Marie-Christine Rousset, Michèle Sebag, Kouzou Ohara, Takashi Washio, and Hiroshi Motoda. Dryadeparent, an efficient and robust closed attribute tree mining algorithm. *IEEE Trans. Knowl. Data Eng.*, 20(3):300–320, 2008.

[22] World Wide Web Consortium. XML Schema, 2001. `http://www.w3C.org/TR/xmlschema-1/`.

[23] World Wide Web Consortium. XML Information Set, 2001. `http://www.w3C.org/xml-infoset/`.

[24] World Wide Web Consortium. XQuery 1.0: An XML query language, 2007. `http://www.w3C.org/TR/xquery`.

[25] World Wide Web Consortium. Extensible Markup Language (XML) 1.0, 1998. `http://www.w3C.org/TR/REC-xml/`.

[26] J. W. W. Wan and G. Dobbie. Extracting association rules from xml documents using xquery. In *WIDM '03: Proceedings of the 5th ACM international workshop on Web information and data management*, pages 94–97, New York, NY, USA, 2003. ACM Press.

[27] K. Wang and H. Liu. Discovering typical structures of documents: a road map approach. In *SIGIR '98: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 146–154. ACM Press, 1998.

[28] K. Wang and H. Liu. Discovering structural association of semistructured data. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):353–371, 2000.

[29] Y. Xiao, J. F. Yao, Z. Li, and M. H. Dunham. Efficient data mining for maximal frequent subtrees. In *ICDM '03: Proceedings of the Third IEEE International Conference on Data Mining*, page 379, Washington, DC, USA, 2003. IEEE Computer Society.

[30] X. Yan and J. Han. Closegraph: mining closed frequent graph patterns. In *KDD '03: Proceedings of the ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 286–295. ACM Press, 2003.

[31] M. J. Zaki. Efficiently mining frequent trees in a forest: algorithms and applications. *IEEE Transactions on Knowledge and Data Engineering*, 17(8):1021–1035, 2005.