Data Mining for XML Query-Answering Support

Mirjana Mazuran, Elisa Quintarelli, and Letizia Tanca

Abstract—Extracting information from semistructured documents is a very hard task, and is going to become more and more critical as the amount of digital information available on the Internet grows. Indeed, documents are often so large that the data set returned as answer to a query may be too big to convey interpretable knowledge. In this paper, we describe an approach based on Tree-Based Association Rules (TARs): mined rules, which provide approximate, intensional information on both the structure and the contents of Extensible Markup Language (XML) documents, and can be stored in XML format as well. This mined knowledge is later used to provide: 1) a concise idea—the gist—of both the structure and the content of the XML document and 2) quick, approximate answers to queries. In this paper, we focus on the second feature. A prototype system and experimental results demonstrate the effectiveness of the approach.

Index Terms—XML, approximate query-answering, data mining, intensional information, succinct answers.

1 INTRODUCTION

In recent years, the database research field has concentrated on the Extensible Markup Language (XML) [30] as a flexible hierarchical model suitable to represent huge amounts of data with no absolute and fixed schema, and a possibly irregular and incomplete structure. There are two main approaches to XML document access: keyword-based search and query-answering. The first one comes from the tradition of information retrieval [20], where most searches are performed on the textual content of the document; this means that no advantage is derived from the semantics conveyed by the document structure.

As for query-answering, since query languages for semistructured data rely on the document structure to convey its semantics, in order for query formulation to be effective users need to know this structure in advance, which is often not the case. In fact, it is not mandatory for an XML document to have a defined schema: 50 percent of the documents on the web do not possess one [5]. When users specify queries without knowing the document structure, they may fail to retrieve information which was there, but under a different structure. This limitation is a crucial problem which did not emerge in the context of relational database management systems.

Frequent, dramatic outcomes of this situation are either the information overload problem, where too much data are included in the answer because the set of keywords specified for the search captures too many meanings, or the information deprivation problem, where either the use of inappropriate keywords, or the wrong formulation of the query, prevent the user from receiving the correct answer. As a consequence, when accessing for the first time a large data set, gaining some general information about its main structural and semantic characteristics helps investigation on more specific details.

This paper addresses the need of getting the gist of the document before querying it, both in terms of content and structure. Discovering recurrent patterns inside XML documents provides high-quality knowledge about the document content: frequent patterns are in fact intensional information about the data contained in the document itself, that is, they specify the document in terms of a set of properties rather than by means of data. As opposed to the detailed and precise information conveyed by the data, this information is partial and often approximate, but synthetic, and concerns both the document structure and its content.

In particular, the idea of mining association rules [1] to provide summarized representations of XML documents has been investigated in many proposals either by using languages (e.g., XQuery [29]) and techniques developed in the XML context, or by implementing graph- or tree-based algorithms. In this paper, we introduce a proposal for mining and storing Tree-Based Association Rules (TARs) as a means to represent intensional knowledge in native XML. Intuitively, 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 and \( S_B \) is a subtree of \( S_H \). 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 \( D \). Graphically, we render the nodes of the body of a rule by means of black circles, and the nodes of the head by empty circles (to get the idea, see the rules in Fig. 4, mined from the data set of Fig. 1).

The intensional information embodied in TARs provides a valid support in several cases.

1. It allows to obtain and store implicit knowledge of the documents, useful in many respects: a) when a user faces a data set for the first time, she/he does not know its features and frequent patterns provide a way to quickly understand what is contained in the data set; b) besides intrinsically unstructured documents, there is a significant portion of XML...
documents which have some structure, but only implicitly, that is, their structure has not been declared via a DTD or an XML-Schema [27]. Since most work on XML query languages has focused on documents having a known structure, querying the above-mentioned documents is quite difficult because users have to guess the structure to specify the query conditions correctly. TARs represent a data guide that helps users to be more effective in query formulation; c) it supports query optimization design, first of all because recurrent structures can be used for physical query optimization, to support the construction of indexes and the design of efficient access methods for frequent queries, and also because frequent patterns allow to discover hidden integrity constraints, that can be used for semantic optimization; d) for privacy reasons, a document answer might expose a controlled set of TARs instead of the original document, as a summarized view that masks sensitive details [9].

2. TARs can be queried to obtain fast, although approximate, answers. This is particularly useful not only when quick answers are needed but also when the original documents are unavailable. In fact, once extracted, TARs can be stored in a (smaller) document and be accessed independently of the data set they were extracted from.

Summarizing, TARs are extracted for two main purposes: 1) to get a concise idea—the gist—of both the structure and the content of an XML document, and 2) to use them for intensional query-answering, that is, allowing the user to query the extracted TARs rather than the original document. In this paper, we concentrate mainly on the second task.

We have applied our techniques in the Odyssey EU Project, whose objective is to develop a platform for automated sharing, management, processing, analysis, and use of ballistic, and crime scene information across Europe. Frequent patterns, in the form of TARs, provide summaries of these integrated data sets shared by different EU Police Organizations. By querying such summaries, investigators obtain initial knowledge about specific entities in the vast data set(s), and are able to devise more specific queries for deeper investigation. An important side effect of using such a technique is that only the most promising specific queries are issued toward the integrated data, dramatically reducing time and cost.

1.1 Goal and Contributions

This paper provides a method for deriving intensional knowledge from XML documents in the form of TARs, and then storing these TARs as an alternative, synthetic data set to be queried for providing quick and summarized answers. Our procedure is characterized by the following key aspects:

1. It works directly on the XML documents, without transforming the data into any 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.
3. It stores association rules in XML format.
4. It translates the queries on the original data set into queries on the TARs set.

The aim of our proposal is to provide a way to use intensional knowledge as a substitute of the original document during querying and not to improve the execution time of the queries over the original XML data set, like in [34]. Accordingly, the paper’s contributions are

- An improved version of the TARs extraction algorithm introduced in [22], which was based on PathJoin [35]. The new version uses the better performing CMTreeMiner [7] to mine frequent subtrees from XML documents.
- Approach validation by means of experimental results, considering both the previous and the current algorithm and showing the improvements.
- Automatic user-query transformation into “equivalent” queries over the mined intensional knowledge. The notion of equivalence in this setting is given in Section 4.
- As a formal corroboration of the accuracy of the process, the proof that our intensional-answering process is sound and complete up to a frequency threshold.

1.2 Structure of the Paper

The paper is organized as follows. Section 2 defines tree-based association rules and introduces their usage, while Section 3 presents how these rules are extracted from XML documents. Section 4 presents the main interesting applications of TARs, that is, their use to provide intensional answers to queries. Section 5 describes a prototype that implements our proposal and the experimental results obtained on real XML data sets. Section 6 discusses related work and Section 7 states the possible follow-ups of this work and draws the conclusions.

2 TREE-BASED ASSOCIATION RULES

Association rules [1] describe the co-occurrence of data items in a large amount of collected data and are represented as implications of 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 measured by means of support and confidence. Support corresponds to the frequency of the set \( X \) in the data set, while confidence corresponds to the conditional probability of finding \( Y \), having found \( X \) and is given by \( \frac{\text{support}(X \cup Y)}{\text{support}(X)} \).

In this paper, we extend the notion of association rule introduced in the context of relational databases to adapt it to the hierarchical nature of XML documents. Following the Infoset conventions, we represent an XML document as a tree \( \langle N, E, r, \ell, c \rangle \), where \( N \) is the set of nodes, \( r \in N \) is the root of the tree, \( E \) is the set of edges, \( \ell : N \rightarrow \mathcal{L} \) is the label function which returns the tag of nodes (with \( \mathcal{L} \) the domain of all tags) and \( c : N \rightarrow C \cup \{ \bot \} \) is the content function which returns the content of nodes (with \( C \) the domain of all contents). We consider the element-only Infoset content model [28], where XML nonterminal tags include only elements and/or attributes, while the text is confined to terminal elements.

We are interested in finding relationships among subtrees of XML documents. Thus, since both textual content of leaf elements and values of attributes convey “content,” we do not distinguish between them. As a consequence, for the sake of readability, we do not report the edge label and the node type label in the figures. 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 (see Fig. 1).

2.1 Fundamental Concepts

Given two trees \( T = \langle N_T, E_T, r_T, \ell_T, c_T \rangle \) and \( S = \langle N_S, E_S, r_S, \ell_S, c_S \rangle \), \( S \) is an induced subtree of \( T \) if and only if there exists a mapping \( \theta : N_S \rightarrow N_T \) such that for each node \( n_i \in N_S \), \( \ell_T(\theta(n_i)) = \ell_S(n_i) \) and \( c_T(\theta(n_i)) = c_S(n_i) \), where \( \theta(n_i) = n_j \), and for each edge \( e = (n_1, n_2) \in E_S \), \( (\theta(n_1), \theta(n_2)) \in E_T \).

Moreover, \( S \) is a rooted subtree of \( T \) if and only if \( S \) is an induced subtree of \( T \) and \( r_S = r_T \).

Given a tree \( T = \langle N_T, E_T, r_T, \ell_T, c_T \rangle \), a subtree of \( T \), \( t = \langle N_t, E_t, r_t, \ell_t, c_t \rangle \) and a user-fixed support threshold \( s_{\text{min}} \); 1) \( t \) is frequent if its support is greater or at least equal to \( s_{\text{min}} \); 2) \( t \) is maximal if it is frequent and none of its proper supertrees is frequent; and 3) \( t \) is closed if none of its proper supertrees has support greater than that of \( t \).

A Tree-based Association Rule (TAR) is a tuple of the form \( T = \langle S_B, S_H, r_T, \ell_T, c_T \rangle \), where \( S_B = \langle N_B, E_B, r_B, \ell_B, c_B \rangle \) and \( S_H = \langle N_H, E_H, r_H, \ell_H, c_H \rangle \) are trees and \( r_T \) and \( c_T \) are real numbers in the interval \([0,1]\) representing the support and confidence of the rule, respectively, (defined below). A TAR describes the co-occurrence of the two trees \( S_B \) and \( S_H \) in an XML document.

A rooted TAR (RTAR) is a TAR such that \( S_B \) is a rooted subtree of \( S_H \), and an extended TAR (ETAR) is a TAR such that \( S_B \) is an induced subtree of \( S_H \).

Let \( \text{count}(S, D) \) denote the number of occurrences of a subtree \( S \) in the tree \( D \) and \( \text{cardinality}(D) \) denote the number of nodes of \( D \). We formally define the support of a TAR \( S_B \Rightarrow S_H \) as \( \text{count}(S_B, D) / \text{cardinality}(D) \) and its confidence as \( \text{count}(S_B, D) / \text{count}(S_B, D) \).

Notice that TARs, in addition to associations between data values, also provide information about the structure of frequent portions of XML documents; thus, they are more expressive than classical association rules which only provide frequent correlations of flat values.

It is worth pointing out that TARs are different from XML association rules as defined in [24], because, given a rule \( X \Rightarrow Y \), where both \( X \) and \( Y \) are subtrees of an XML document, that paper requires that \( X \nsubseteq Y \land (Y \nsubseteq X) \), i.e., the two trees \( X \) and \( Y \) have to be disjoint; on the contrary, TARs require \( X \) to be an induced subtree of \( Y \).

Given an XML document, we extract two types of TARs.

- A TAR is a structure TAR (STAR) iff, for each node \( n \) contained in \( S_H \), \( c_H(n) = \bot \), that is, no data value
is present in sTARs, i.e., they provide information only on the structure of the document (see Fig. 3).

- A TAR, $S_B \Rightarrow S_H$, is an instance TAR (iTAR) iff $S_H$ contains at least one node $n$ such that $c_H(n) \neq \perp$, that is, iTARs provide information both on the structure and on the data values contained in a document (Fig. 4).

According to the definitions above we have: structure-Rooted-TARs (sRTARs), structure-Extended-TARs (sETARs), instance-Rooted-TARs (iRTARs), and instance-Extended-TARs (iETARs).

Since TARs provide an approximate view of both the content and the structure of an XML document, 1) sTARs can be used as an approximate DataGuide [13], [14] of the original document, to help users formulate queries; 2) iTARs can be used to provide intensional, approximate answers to user queries. Fig. 3 shows a sample XML document and some sTARs. Rules (1) and (3) are rooted sTARs, rule (2) is an extended sTAR. Rule (1) states that, if there is a node labeled $A$ in the document, with 86 percent probability that node has a child labeled $B$. Rule (2) states that, if there is a node labeled $B$, with 75 percent probability its parent is labeled $A$. Finally, Rule (3) states that, if a node $A$ is the grandparent of a node $C$ (notice the empty node, parent of node $C$), with 75 percent probability the child of $A$ and parent of $C$, is labeled $B$.

By observing sTARs users can guess the structure of an XML document, and thus use this approximate schema to formulate a query when no DTD or schema is available: as DataGuides [13], sTARs represent a concise structural summary of XML documents. Consider a user, querying for the first time the document “incidents.xml.” The sTAR (4), in Fig. 3, allows users to understand that five times out of six an incident’s structure presents the fields type, ballistic item, country, and when reported, and that the ballistic item is a bullet which in turn comprises type, case type, diameter; thus for instance “the average diameter of the bullets” or “the number of incidents reported in Italy” are meaningful queries with respect to this document.

Differently from DataGuides, sTARs do not show all possible paths in the XML document but only the frequent paths. In particular, for each fragment, its support determines how frequent the substructure is. This means that sTARs provide a simple path index which supports path matching and can be used for the optimization of the query process. An index for an XML data set is a predefined structure whose performance is maximized when the query matches exactly the designed structure. Therefore, the goal, when designing an index, is to make it as similar as possible to the most frequent queries. For example, the sTAR 4 in Fig. 3 can suggest the index paths: incident/ballistic_item/bullet and incident/type.

By contrast, iTARs give an idea about the type of content of the different nodes. Fig. 4 shows some examples of iTARs referred to the XML document in Fig. 1. Rules (1) and (4) are rooted iTARs, while rules (2) and (3) are extended iTARs. Rule (1) states that, if there is a node labeled incident in the document, with confidence 0.8 it has a child labeled type whose value is “robbery.” That is, 80 percent of the incidents contained in the document are robberies. Rule (2) states that, if there is a path composed by the sequence of nodes bullet/type, and the content of type is “Full Metal Jacket,” then node bullet, with confidence 0.66, has another child labeled case_type whose content is “Rimless.” That
is, 66 percent of Full Metal Jacket bullets have rimless cases. Rule (3) states that, if there is a path composed by the sequence of nodes bullet/type, and the content of type is “Winchester,” then node bullet, with confidence 1.00, has two other children labeled diameter and case_type whose contents are, respectively, “7.8 mm” and “Belted.” That is, 100 percent of the Winchester bullets have a 7.8 mm diameter and a belted case. Note that, since Rule (3) has confidence 1, we understand that the data set satisfies the constraint “all Winchester bullets are Belted and measure 7.8 mm.” Finally, rule (4), states that, if there is a path composed by the following sequence of nodes: incident/ballistic_items/bullet and the node bullet has two children labeled type and case_type whose contents are, respectively, “Full Metal Jacket” and “Rimless,” then node incident, with confidence 0.75, has two more children labeled type and country whose contents are “robbery” and “Italy.” That is, 75 percent of incidents involving rimless Full Metal Jacket bullets are robberies happening in Italy.

### 3 TAR EXTRATION

TAR mining is a process composed of two steps: 1) mining frequent subtrees, that is, subtrees with a support above a user-defined threshold, from the XML document; 2) computing interesting rules, that is, rules with a confidence above a user-defined threshold, from the frequent subtrees.

As will be discussed in more detail in Section 6, the problem of finding frequent subtrees has been widely treated in the literature [36], [3], [32], [35], [37], [1].

Algorithm 1 presents our extension to a generic frequent-subtree mining algorithm in order to compute interesting TARs. The inputs of Algorithm 1 are the XML document $D$, the threshold for the support of the frequent subtrees $\text{minsupp}$, and the threshold for the confidence of the rules, $\text{minconf}$. Algorithm 1 finds frequent subtrees and then hands each of them over to a function that computes all the possible rules. Depending on the number of frequent subtrees and their cardinality, the amount of rules generated by a naive Compute-Rules function may be very high. Given a subtree with $n$ nodes, we could generate $2^n - 2$ rules, making the algorithm exponential. This explosion occurs in the relational context too, thus, based on similar considerations [1], it is possible to state the following property, that allows us to propose the optimized version of Compute-Rules shown in Function 2.

**Algorithm 1.** Get-Interesting-Rules ($D$, $\text{minsupp}$, $\text{minconf}$)

1: // frequent subtrees
2: $F_S = \text{FindFrequentSubtrees}(D, \text{minsupp})$
3: $\text{ruleSet} = \emptyset$

4: for all $s \in F_S$ do
5: // rules computed from $s$
6: $\text{tempSet} = \text{Compute-Rules}(s, \text{minconf})$
7: // all rules
8: $\text{ruleSet} = \text{ruleSet} \cup \text{tempSet}$
9: end for
10: return $\text{ruleSet}$

**Function 2.** Compute-Rules ($s$, $\text{minconf}$)

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

**Remark 1.** If the confidence of a rule $S_B \Rightarrow S_H$ is below the established threshold $\text{minconf}$ then the confidence of every other rule $S_B \Rightarrow S_H$, such that its body $S_B$ is an induced subtree of the body $S_B$, is no greater than $\text{minconf}$.

Consider Fig. 5, which shows a frequent subtree (Fig. 5a) and three possible TARs mined from the tree; all the 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 confidence of rules (2) and (3) are equal, or lower, than the confidence of rule (1).

In Function 2, TARs are mined exploiting Remark 1 by generating first the rules with the highest number of nodes in the body tree. Consider two rules $T_1$ and $T_2$ whose body trees contain one and three nodes, respectively; suppose both rules have confidence below the fixed threshold. If the algorithm considers rule $T_1$ first, all rules whose bodies are induced subtrees of $T_1$ will be discarded when $T_2$ is eliminated. 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. Using this solution, we can lower the complexity of the algorithm, though not enough to make it perform better than exponentially. However, notice that the process of deriving TARs from XML documents is only...
The algorithm has to be applied few times or only once (for documents that do not change).

Once the mining process has finished and frequent TARs have been extracted, they are stored in XML format. This decision has been taken to allow the use of the same language (XQuery in our case) for querying both the original data set and the mined rules. Each rule is saved inside a <rule> element which contains three attributes for the ID, support, and confidence of the rule. Follows the list of elements, one for each node in the rule head. We exploit the fact that the body of the rule is a subtree of the head, and use a Boolean attribute in each node to indicate if it also belongs to the body. Each blank node is described by an element <blank>. Finally, the rules in the XML file are sorted on the number of nodes of their antecedent; therefore, the TARs in Fig. 4 are stored in the following order: (1), (2), (3), and (4). This is an important feature that is used to optimize the answering of queries containing a count operator (see Section 4). Fig. 6 shows the XML file containing TARs (1) and (2) of Fig. 4.

One of the (obvious) reasons for using TARs instead of the original document is that processing iTARs for query-answering is faster than processing the document. To take full advantage of this, we introduce indexes on iTARs to further speed up the access to mined trees—and in general of full advantage of this, we introduce indexes on TARs to make XML query-answering faster by means of path-based intensional query-answering. In the literature, the problem of further speed up the access to mined trees—and in general of documents that do not change).

Before applying the algorithm, two sets A and C are constructed containing, respectively, the antecedent and consequent trees of all the TARs to be indexed. Each tree \( T_i \) in the index is annotated in a way that each node contains the reference to the ID of the rule it comes from; then trees are scanned looking for those that have the same root. After this step two sets \( P = \{ P_1, \ldots, P_m \} \) and \( D = \{ D_1, \ldots, D_m \} \) are obtained that are partitions of \( A \) and \( C \), respectively, where each \( P_i \) and \( D_i \) contains trees having the same root. Algorithm 3 is applied to merge the trees in each set using the same rationale behind the DataGuide construction procedure [13]. In particular, for each set, the first tree is merged together with the others, that means that the references of its root are added to the references of the roots of the other trees (line 3) and the same procedure is applied recursively to the children of the two roots (line 4).

For each node \( n \) of the resulting tree a set of references is stored, pointing to the TARs that contain the path from the root of the tree to node \( n \). Once the algorithm is applied to all TARs, the result is a set of trees whose nodes contain references to one or more rules and which are stored in an XML file to be queried later on. Notice that both antecedents and consequents of rules are indexed because we work on the assumption that an answer to a user query is a set of rules whose antecedents or consequents match user requests. However, they are indexed separately because for some categories of user queries we need both of them to provide the answer, while for others we need only antecedents. Detailed explanations will be given in Section 4.

### Algorithm 3. Create-Index (D)

1. for all \( D_i \in D \) do
2.     for all \( d_j \in D_i \) with \( j \in \{2,3,\ldots,n\} \) do
3.         references(root(\( d_j \))) = references(root(\( d_i \)))

4. return \( D \)

### Function 4. sumChildren \( (T_1, T_2) \)

1. for all \( x \in \text{children}(\text{root}(T_2)) \) do
2.     if \( \exists c \in \text{children}(\text{root}(T_1)) \quad |c = x\) then
3.         references(root(\( c \))) = references(root(\( c \)))
4.     else
5.         \( c = \text{sumChildren}(c, x) \)
6.     end if
7. end for
8. return \( T_1 \)

4 INTENSIONAL ANSWERS

iTARs provide an approximate intensional view of the content of an XML document, which is in general more
Concise than the extensional one because it describes the data in terms of its properties, and because only the properties that are verified by a high number of items are extracted. A user query over the original data set can be automatically transformed into a query over the extracted iTARs. The answer will be intensional, because, rather than providing the set of data satisfying the query, the system will answer with a set of properties that these data “frequently satisfy,” along with support and confidence.

There are two major advantages: 1) querying iTARs requires less time than querying the original XML document; 2) approximate, intensional answers are in some cases more useful than the extensional ones (see Section 1). For example, if a user asks for the incidents registered in the data set in Fig. 1, the extensional answer is the list of all incidents (possibly megabytes) to be inspected manually, while an intensional answer might be that “80 percent of incidents were robberies.”

Not all queries lend themselves to being transformed into queries on iTARs; we list three classes of queries that can be transformed, still preserving the soundness; moreover, we explain how such transformation can be automatically done.

The classes of queries that can be managed with our approach have been informally introduced in [11] and further analyzed in the relational database context in [4]. They include the main retrieval functionalities of XQuery, i.e., path expressions, FLOWR expressions, and the COUNT aggregate operator. We have not considered operators for adding new elements or attributes to the result of a query, because our purpose is to retrieve slender and approximate descriptions of the data satisfying the query, as opposed to modifying, or adding, new elements to the result. Moreover, since aggregate operators require an exact or approximate numeric value as answer, they do not admit intensional answers in the form of implications, thus queries containing aggregators other than COUNT are excluded. Note, however, that mined TARs allow us to provide exact answers to counting queries.

Table 1 shows each class with its syntax and an example. The emphasized objects are metaexpressions (queries or variables) which need to be replaced in the actual query.

- **Class 1: \( \sigma/\pi \)-queries.** 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, possibly ordering the result. The query imposes some conditions on a node’s content and on the content of its descendants, orders the results according to one of them and returns the node itself. For example “Retrieve all incidents where Full Metal Jacket types of bullets were used, ordered by the date the incident was reported.”

- **Class 2: count-queries.** Used to count the number of elements having a specific content. The query creates a set containing the elements which satisfy the conditions and then returns the number of elements in

---

**TABLE 1**

<table>
<thead>
<tr>
<th>Class</th>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>for variable in path [where condition [and/or condition]] [order by element [asc</td>
<td>desc]] return variable</td>
</tr>
<tr>
<td>2</td>
<td>let $set := (class 1 query) return count($set)</td>
<td>let $set := for $x$ in doc(&quot;incidents.xml&quot;)/incidents/incident/bullet where $x$/type/text()=&quot;Winchester&quot; return $x$</td>
</tr>
<tr>
<td>3</td>
<td>[for variable in distinct-values(path) let $set := (class 1 query) order by count($set) desc return variable</td>
<td>[for $x$ in distinct-values(doc(&quot;incidents.xml&quot;)/incidents/incident/bullet) let $set := for $x$ in doc(&quot;incidents.xml&quot;)/incidents/incident/bullet where $x$/type/text()=&quot;Winchester&quot; desc return $x$] [position() &lt;= k]</td>
</tr>
</tbody>
</table>
steps: 1) rewrite file, it is possible to obtain the intensional answer in two examples for Class 1 queries.

![Diagram](image)

Fig. 8. The intensional query answering commutative diagram and an example for Class 1 queries.

such set. For example “Retrieve the number of Winchester bullets.”

- **Class 3: top-k queries**. Used to select the best k answers satisfying a counting and grouping condition. The query counts the occurrences of each distinct value of a variable in a desired set; then orders the variables with respect to their occurrences and returns the most frequent k. For example “Retrieve the k most used types of bullets.”

Notice that, in all classes of queries, conditions can be imposed on the descendants of the element that is returned and not on its ancestors. That is, a query containing conditions on the contents of an element is supposed to be as depicted in Fig. 8b (where x is the element returned by the query).

Given query \( q_E \), a file containing iTARs and the index file, it is possible to obtain the intensional answer in two steps: 1) rewrite \( q_E \) into \( q_I \); 2) apply \( q_I \) on the intensional knowledge. That is: a) access the index retrieving the references to the rules satisfying the conditions in \( q_I \); b) access the iTARs file returning the rules whose references were found in Step a.

In Step 1, we start from the extensional query \( q_E \) and apply a rewriting algorithm to obtain the intensional query \( q_I \). We first extract from \( q_E \) the following variables and lists:

- \( v_F \), the path in the FOR clause of \( q_E \).
- \( v_{OB} \), the variable in the ORDER BY clause of \( q_E \).
- \( v_{DV} \), the variable in the distinct-values function of \( q_E \).
- \( VW = (vw_j) \) is a variable of the paths in the WHERE clause of \( q_E \) in the same order.
- \( CONN = (conn_k) \) is a connective in the WHERE clause of \( q_E \) in the same order.

These objects are the input of Algorithm 5 and its variants, whose output is the intensional query \( q_I \). In the following, we describe the algorithm for obtaining the rewritten query \( q_I \) for each class of queries. Each algorithm progressively builds the query \( q_I \) by concatenating pieces of the query. Notice that the operator “*” is used to denote concatenation of strings.

**Algorithm 5.** Class1-Query \((v_F,VW,CONN,v_{OB})\)

1: // the intensional query is empty
2: \( IQ = \epsilon \)
3: if \( VW \neq \emptyset \) then
4: // get instance rules for paths with a constraint
5: \( IQ = IQ \bullet get_iTARs(v_F,VW,CONN,\text{false}) \)
6: else
7: // structure rules for the path without constraint
8: \( IQ = IQ \bullet get_sTARs(v_F) \)
9: end if
10: // order the results
11: \( IQ = IQ \bullet \text{“for } $r$ \text{ in } $\text{Rules/Rule order by } $r/v_F/v_{OB}$ \text{ return } $r$” } \)
12: return \( IQ \)

**Function 6.** get_iTARs (for, variables, connectives, count)

1: \( Q = \epsilon \)
2: for all \( v_j \in \text{variables} \) do
3: if \( \text{count} = \text{true} \) then
4: // for count queries match only in the antecedent
5: \( Q = Q \bullet \text{“let } $\text{RefI}_j := \text{referencesA}(\text{for}, v_j)\” \)
6: else
7: // for queries without count match both in antecedent and consequent
8: \( Q = Q \bullet \text{“let } $\text{RefI}_j := \text{references}(\text{for}, v_j)\” \)
9: end if
10: end for
11: \( Q = Q \bullet \text{“let } $\text{Rules} := \” \)
12: for all \( v_j \in \text{variables}, j \in \{1, \ldots, n\} \) do
13: \( Q = Q \bullet \text{“ruleset($\text{RefI}_j) connective$} \)
14: end for
15: return \( Q \)

**Function 7.** get_sTARs (variable)

1: \( Q = \text{“let } $\text{RefS} := \text{references(\text{variable,”})} \text{ let } $\text{Rules} := \text{ruleset($\text{RefS})”} \)
2: return \( Q \)

1) **Class 1: \( \sigma/\pi \)-queries.** This class of queries is rewritten using Algorithm 5. The result is query \( q_I \) that looks for the rules which satisfy the conditions imposed in the where clause and returns those obtained by combining the previous sets using the logical connectives in the same order as in the where clause of \( q_E \), possibly ordered by the variable specified in the order by clause. The query \( q_I \) contains calls to the functions \( \text{references, referencesA} \)
1. the ones in $I_A$, because if they were extracted from $D_E$, they are extracted also from $E_A$; and
2. other iTARs, showing correlations between proper subtrees of the ones composing the extensional answer.

The application of $q_I$ to $R$ eliminates the iTARs in 2, that is, the ones that are not rooted in $x$. □

Notice that, when using iRTARs and iETARs satisfying the constraints of the query in the antecedent, we can only guarantee soundness; indeed the consequent of iETARs extends the antecedent and possibly contains some ancestors of the antecedent root. Thus, when using iETARs satisfying the query in their antecedent, we can obtain as result also rules containing knowledge about ancestors of the node $x$ specified in the original query (see Fig. 8), rules that cannot be mined from $E_A$, because they do not satisfy the constraint in $q_E$.

On the other hand, if we apply the intensional query $q_I$ to the set $E_A$—thus filtering only the interesting iRTARs w.r.t. the original query $q_E$—and do not impose any constraint on the support threshold, we obtain as result $I_A$, thus, our method is both sound and complete.

**Theorem 2.** Let $q_E$ be a Class 1 query on the XML document $D_E$, $q_I$ the intensional rewriting of $q_E$, $E_A$ the XML document obtained as result for $q_E$, and $I_A$ the intensional answer. If, in the mining process, the imposed support and confidence thresholds are 0, the procedure to obtain intensional answers is such that $q_I($Get - Interesting - Rules($E_A$)) = $I_A$, that is, the procedure is both sound and complete.

**Proof.** We have already proven the soundness, thus we have only to prove that, given a TAR $T_r$, if $T_r \in q_I($Get - Interesting - Rules($E_A$)) then $T_r \in I_A$.

If $T_r \in q_I($Get - Interesting - Rules($E_A$)) then 1) $T_r$ satisfies $q_I$ and 2) $T_r$ is frequent in $E_A$ (because $E_A$ is a collection of subtrees of $D$). From 2, it follows that $T_r$ is frequent also when we apply Get-Interesting-Rules to $D_E$, and from 1 we prove that $T_r \in I_A$. □

In the proof, we have highlighted the fact that the procedure is complete if we do not impose a threshold on the support and confidence values. If, during the mining process, we impose a threshold greater than 0 on the support and confidence, we cannot state that the described technique is complete because $q_I($Get - Interesting - Rules($E_A$)) may contain some rules whose support is greater when calculated on the extensional answer than the support of the same rule when calculated on the original data set $D_E$. Intuitively, the extensional answer is a tree with fewer nodes than the data set (or the same number of nodes when the answer coincides with the whole data set) which means that the support of a TAR is usually lower when calculated on the original data set w.r.t. the support of that TAR mined from the extensional answer.

**Class 2: count-queries.** This class of queries is rewritten using Algorithm 8. The result is a query $q_I$ that specifies the iTARs which satisfy the original query conditions and returns the support of the first rule which has been found, divided by its confidence. Notice that, since rules are ordered according to the number of nodes in their antecedent, the

and ruleset presented in [21], used to access the TARs and their index.

The sample Class 1 query in Table 1 is rewritten as $q_I$:

\[
\text{let } \$\text{Ref}_I := \text{references}([/\text{incidents/}
\text{incident}],
\text{"/\text{bullet/type[text()="Full Metal
\text{Jacket"]}"})
\]

\[
\text{let } \$\text{Rules} := \text{ruleset}($\text{Ref}_I_1)
\]

\[
\text{for } r \in \$\text{Rules}/\text{Rule}
\]

\[
\text{order by } r/\text{incidents/\text{incident}/when\text{ reported}
\text{ascending}
\]

\[
\text{return } r
\]

Applying $q_I$ to the index, the variable $\$\text{Ref}_I_1$ is initialized with the set of references to the iTARs containing the element /incidents/incident//bullet/type with content “Full Metal Jacket.” Then, variable $\$\text{Rules}$ will contain the set of iTARs whose reference is in $\$\text{Ref}_I_1$, that is, the iTAR (4) in Fig. 4. Finally, the iTARs in $\$\text{Rules}$ will be ordered according to the node when reported.

We consider as intensional answer to Class-1 queries iTARs that are

- iRTARs that match the constraints in the antecedent and/or consequent (see Fig. 8c); and
- iETARs that match the constraints in the consequent (see Fig. 8d).

Consider a Class-1 query $q_E$ applied to a document $D_E$, and its answer $E_A$. Consider the document $D_I$ containing the extracted TARs. We now show that, if we extract TARs from the answer to query $q_E$, we obtain a superset of the answers $I_A$ obtained by applying $q_I$ to the TARs $D_I$ extracted from $D_E$, i.e., the intensional answer constitutes a representation of the frequent properties of the extensional one. The commutative diagram is shown in Fig. 8a. Note that the set of iTARs extracted from $E_A$ includes the set $I_A$, obtained as intensional answer; the reason is that some frequent subtrees of $E_A$ are not given as result in $I_A$ because they do not represent an answer for the input query $q_A$. Thus, we can say that our procedure is sound. Note that the following theorems serve as the formal basis for the accuracy results presented in Section 5.

**Theorem 1.** Let $q_E$ be a Class-1 query on the XML document $D_E$, $q_I$ the intensional rewriting of $q_E$, $E_A$ the XML document obtained as result for $q_E$, and $I_A$ the intensional answer to $q_I$. The procedure to obtain intensional answers is sound, that is, if a TAR $T_r \in I_A$ then $T_r \in q_I($Get - Interesting - Rules($E_A$)).

**Proof.** Let us consider a Class 1 query $q_E$ specifying the subtrees rooted in a node $x$ and satisfying conditions cond$_1$, ..., cond$_n$ on the descendants of $x$, as shown in Fig. 8b. We have to prove that if a TAR $T_r \in I_A$ then $T_r \in q_I($Get - Interesting - Rules($E_A$)).

A possible extensional answer belonging to $E_A$ is sketched in Fig. 8e; the iTARs in $\mathcal{R} = \text{Get - Interesting - Rules($E_A$)}$ are

3. The set of XML trees obtained as answer of $q_E$ are collected in a unique document having a <result> tag as root.
first rule will be either the one which satisfies all and only the requested conditions or its best approximation (that is, a rule whose antecedent satisfies all the desired conditions and contains the least number of nodes).

Algorithm 8. Class2-Query (v_F, V_W, CONN)
1: // the intensional query is empty
2: IQ = ε
3: // get instance rules for paths with a constraint
4: IQ = IQ ∪ get_iTARs(v_F, V_W, CONN, true)
5: IQ = IQ ∪ get_count()
6: IQ = IQ ∪ “return $supp div $conf”
7: return IQ

Function 9. get_count()
1: Q = “let $supp := $Rules/Rule[1]@support
let $conf := $Rules/Rule[1]@confidence”
2: return Q

The sample Class 2-query in Table 1 is rewritten as q_I:

let $RefI_1 := referencesA(“*/incidents/incident/ballistic_items/bullet '*', 
"/type[()="Winchester"]")
let $Rules := ruleset ($RefI_1)
let $supp := $Rules/Rule[1]@support
let $conf := $Rules/Rule[1]@confidence
return $supp div $conf

To answer this query, an association rule is used (if it exists) whose body exactly matches the query conditions. Since for each association rule $A → B$, conf($A → B) = supp(A → B)/supp(A)$, it is possible to compute supp($A$) (that is, the number of elements satisfying the conditions in A) as supp($A → B)/conf($A → B). With respect to the example, it is possible to count the number of “Winchester” bullets using the iTAR (3) in Fig. 4, which contains exactly the path bullet/type (with content “Winchester”) in the body. By multiplying its support by the number of nodes in the document and dividing by the confidence, we obtain (0.03×62)/1 = 1.86 ≈ 2. Notice that the result of Class 2 queries is exact, up to the approximation introduced by the computation of the support and confidence. In general, the following theorem holds.

Theorem 3. Let $q_E$ be a Class 2 query on the XML document $D_E$, $q_I$ the intensional rewriting of $q_E$, count_E the extensional answer, and count_I the intensional answer. If we have at least a TAR exactly satisfying the antecedent of the constraints in $q_E$ then count_I = count_E, that is, the procedure is sound.

Proof. To answer Class-2 queries we use only iTARs which match the requested constraints in the antecedent and obtain as answer (supp × nodes)/conf (where supp and conf are support and confidence of the iTAR used to answer the query, and nodes is the number of nodes in $D_E$). The following two possibilities can be envisaged:

1. We use an iTAR whose body exactly matches the requested constraints, if it has been mined, and the answer will be exact up to approximation, that is, count_I ≈ count_E.
2. If the previous case cannot be applied, we use an iTAR whose body partially matches the requested constraints (that is, the body contains more constraints w.r.t. those required in the query) and the answer will be count_I ≤ count_E. For example, if we want to count the number of incidents involving “Full Metal Jacket” bullets and we use iTAR (4) in Fig. 4, we obtain an approximate answer; that iTAR describes a property of incidents involving “Rimless Full Metal Jacket.”

Class 3: top-k queries. This class of queries is rewritten using Algorithm 10. The result will be a query $q_I$ that, for each distinct value of a variable finds, the corresponding sTARs and uses them to compute the number of occurrences of each value; ranks the values according to the computed count and returns all the rules associated with the first $k$ ranked values.

Algorithm 10. Class3-Query (v_DY, v_F, V_W, CONN)
1: IQ = ε // the intensional query is empty
2: IQ = IQ ∪ get_sTARs(v_DY) // get instance rules for paths with a constraint
3: IQ = IQ ∪ “for $v$ in distinct-values ($Rules/Rule[1]”
4: IQ = IQ ∪ get_count()
5: IQ = IQ ∪ “order by $supp div $conf descending
return $Rules) [position()<=k]”
6: return IQ

The sample Class 3-query in Table 1 is rewritten as q_I:

let $RefS := references(“*/incidents/incident/ballistic_items/bullet/type”, “”)
let $RulesS := ruleset ($RefS)
(for $t in distinct-values($RulesS/incidents/incident/ballistic_items/bullet/type))
let $RefI_1 := referencesA(“*/incidents/incident/ballistic_items/bullet/*
"/type[()=$t"])
let $Rules := ruleset ($RefI_1)
let $supp := $Rules/Rule[1]@support
let $conf := $Rules/Rule[1]@confidence
order by $supp div $conf
return $Rules) [position() <= k]

The soundness of the intensional answering process for Class 3 queries can be easily derived from the proofs of Classes 1 and 2. Indeed, Class 3 queries only add a filter on the number of results which are first ordered according to a count condition. Notice that, with the type of index we chose to use, the translation of the constraints required in the original query are straightforward, that is, the index allows us to support XQuery and XPath constraints without any preprocessing.

The soundness and completeness results proven in this section testify for the general quality, and in particular accuracy, of the intensional answers. In particular, the soundness means that there are no “false positive” TARs, and thus TARs are characterized by 100 percent precision for the three query classes, while the completeness shows that the recall is strongly related to the support threshold imposed in the mining process and is guaranteed to be 100% only if we reduce the threshold to 0. In Section 5, we will reinforce this claim, analyzing, by means of experiments on real data sets, the recall of the intensional answers when varying the support threshold.
As a last remark, note that we have proposed an application scenario where iTARs are extracted from the original XML data set and then used to answer intensional queries. Another interesting use is a-posteriori, that is, mine iTARs from the extensional answer to a user query. Indeed, since extensional results may be too large to examine, the iTARs mined from them can provide the user with succinct and more interpretable—though approximate—knowledge.

5 EXPERIMENTAL RESULTS

5.1 The TreeRuler Prototype

TreeRuler is a tool that integrates the functionalities proposed in our approach. Given an XML document, it enables users to extract intensional knowledge and compose traditional queries as well as queries over the intensional knowledge, receiving both extensional and intensional answers. Users formulate XQueries over the original data, and queries are automatically translated and executed on the intensional knowledge. The answer is given in terms of the set of TARs which reflect the search criteria. TreeRuler interface offers three tabs.

- **Get the gist** allows intensional information extraction from an XML document, given the support, confidence and the files where the extracted TARs and their index are to be stored.
- **Get the idea** allows to show the intensional information as well as the original document, to give users the possibility to compare the two kinds of information.
- **Get the answers** allows to query the intensional knowledge and the original XML document. Users have to write an extensional query, when the query belongs to the classes we have analyzed it is translated and applied to the intensional knowledge. Finally, once it is executed, the TARs that reflect the search criteria are shown.

TreeRuler is implemented in C++ using the eXpat (http://expat.sourceforge.net) library for XML parsing, and wxWidgets (http://www.wxwidgets.org/) tools for the GUI. The tool implements the CMTTreeMiner [7] algorithm for the extraction of frequent subtrees from the XML document.

In our first proposal [22], we have used the PathJoin [35] algorithm to find frequent subtrees in XML documents. As will be shown in Section 5.2 such algorithm performed exponentially, therefore we have changed it and continued working with CMTTreeMiner [7]. The authors of CMTTreeM- iner provided (http://www.nec-labs.com/ychi/—Available in January 2008) the C++ implementation for both ordered and unordered subtree extraction. In this paper, we have provided a general extension which can be applied to both versions of CMTTreeMiner, but for the moment our prototype is focused on the ordered version.

5.2 Experiments

We performed four types of experiments.

1. Time required for the extraction of the intensional knowledge from an XML database.
2. Time needed to answer intensional and extensional queries over an XML file.

3. A use case scenario on the DocBook (http://www.docbook.org/) XML database, in order to monitor extraction time given a specific support or confidence.

5.2.1 Extraction Time

We have performed experiments using both PathJoin and CMTTreeMiner as algorithms for mining frequent subtrees from XML documents. We used PathJoin on both real and artificial XML data sets. First, we executed TreeRuler on the data sets found at the XMData Repository (http://www.cs.washington.edu/research/xmldatasets/), but they were too structured and the extracted intensional knowledge was not interesting. Moreover, the DTD for all documents was already provided thus the extraction of TARs did not give any advantages. Then, we applied TreeRuler to documents created by means of the GCC XML tool (http://www.gccxml.org/HTML/Default.html) because they were unstructured and without a DTD.

Fig. 9a shows how extraction time depends on the number of nodes in the XML document. In particular, 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 data set, is significant (e.g., 264 KB w.r.t. 4 KB).

After this first analysis, we used XMark (http://www.xml-benchmark.org/) to create large artificial XML documents and evaluate the extraction time with respect to such documents, producing the curve in Fig. 9b. Notice that extraction time growth is linear with respect to the cardinality of an XML tree only for documents with less than 2 billion nodes. After such threshold extraction time growth becomes exponential.

We took a step further in the analysis with the aim of understanding the parameters that influence extraction time growth. Therefore, we performed evaluation experiments tackling the depth and fan out of the XML documents. Fig. 10a shows that documents with the same number of nodes have different extraction times depending on the depth of the document. In particular, the greater is the cardinality of the
document, the more its depth influences the extraction time. Moreover, as shown in Fig. 10b, the greater the depth, the lower the threshold which separates linear from exponential time growth. Finally, Fig. 11a shows that the same observations about the depth, hold for the fan out as well.

We migrated to the CMTreeMiner algorithm, which does not suffer [7] from the exponential explosion affecting PathJoin. We performed experiments on real data sets and obtained the results shown in Fig. 11b. Note that the growth shown in the figure is almost linear because, since the documents are synthetic: 1) there is a significant initial pruning of the nodes and 2) the mined subtrees are very small.

5.2.2 Answer Time
Fig. 12 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 data sets, $\text{name}$ is the name of a node contained in the document (such name changes on the basis of the XML document). Notice that the time for processing queries with respect to extensional knowledge is always significantly greater than the time for processing queries with respect to intensional knowledge (actually almost constant), thus proving the effectiveness of our approach.

5.2.3 Use Case Scenario
We applied TreeRuler on the DocBook XML database by setting the confidence at 0.95, Fig. 13a shows how the extraction time changes w.r.t. support. Similarly, by setting the support at 0.02, Fig. 13b shows how the extraction time changes w.r.t. the confidence. Notice that the decrease of both support and confidence reflects in a decrease of the extraction time because less intensional information is extracted. In particular, support determines how many frequent subtrees will be extracted, while confidence influences the number of rules that will be extracted from each subtree. The support threshold has a higher impact on the performance because to a linear decrease of the number of extracted subtrees corresponds an exponential decrease of the number of possible rules. A high support threshold means both small amount of subtrees and small subtrees, from which very little time is required to extract rules. On the other hand, the confidence threshold allows to prune rules from each subtree; however, if the support is small, bigger subtrees will be extracted making the number of rules to check greater.
5.2.4 Accuracy

Precision and recall are commonly used to evaluate the accuracy of approaches which return approximate answers. Based on the soundness theorems of Section 4, we can say that TARs are characterized by 100 percent precision for all query classes described there, since the soundness demonstrates that there are no “false positive” TARs.

Recall depends on the support threshold established in the mining process. Indeed, the application of our mining algorithm returns only frequent subtrees and the number of such trees depends on the support threshold. Thus, since the minimum support threshold strongly influences query recall, it is a relevant parameter for tuning the intensional representation of information.

To understand how the support threshold influences the accuracy of the intensional answers we performed experiments by extensionally querying some real data sets and also by extracting intensional answers from them. In our setting, the traditional definition of recall does not make much sense, thus we need to find a measure of recall that conveys the same intuition as the traditional one.

Given a query \( q_E \) over an XML document \( D \), let \( A_E \) denote the extensional answer to \( q_E \) (i.e., a set of XML documents—or trees—\( T \)). Let \( A_I \) be the set of subtrees \( t \) of \( D \) that are consequents of the TARs extracted from \( D \) which have been returned as intensional answer to \( q_I \) (the intensional rewriting of \( q_E \)).

Given a tree \( T \in A_E \), if there is at least one \( t \in A_I \) such that \( t \) is a subtree of \( T \) then we say that \( t \) is a hit for \( A_E \), since it means that \( t \) represents summarized information of \( T \). We denote this by writing \( t \in \text{hits}(A_I, A_E) \). Then, we compute the recall of \( A_I \) as the ratio of hits w.r.t. the whole extensional answer, that is, as \( |\text{hits}(A_I, A_E)|/|A_E| \).

Our experiments are performed on real XML data sets and cover all three classes of queries introduced in our proposal. Figs. 14a, 14b, and 14c show the results obtained for Classes 1, 2, and 3 queries, respectively. The queries we apply depend on the queried document, for example the queries performed on the DocBook data set are: Q1) find all books that contain “Java” as a keyword and return them ordered by year of publication; Q2) count all books that contain “Java” as a keyword; Q3) find the top 3 used keywords.

Notice that the recall values strongly depend on the support threshold. The greater the support the smaller the recall since a very limited number of TARs can be mined. Similarly, the smaller the support the greater the recall since more intensional information is extracted, thus, trees in the extensional answers are better represented by the mined TARs. Notice that, even with a small support threshold, for Class 1 queries, recall values can be smaller than one. This is reasonable, as it is due to the fact that TARs are not able to represent all the properties of the extensional answer. The difference between the two curves in Fig. 14a is that the eBay data set is quite regular: it represents a list of auctions that have very similar structure. A TAR representing such structure is extracted with low support, and it describes a property of each subtree in the extensional answer. When the support grows, a smaller TAR is extracted that represents properties of fewer subtrees in the extensional answer. On the other hand, the DocBook data set is very irregular in the sense that it describes data with very different structures. Therefore, even with low support thresholds, the extracted TARs are not able to represent a property which is common to all the subtrees in the extensional answer. On the contrary, for Classes 2 and 3 queries, recall values are higher with lower support thresholds because, if we have extracted at least a rule that satisfies query constraints, we are able to compute the exact answer to count requests.

6 COMPARISON WITH OTHER WORK

The problem of association rule mining was initially proposed in [1] and many implementations of the algorithms, downloadable from [12], were developed in the database literature. More recently the problem has been investigated in the XML context [6], [31], [24], [8], [10], [19], [33]. In [31], Wan and Dobbie use XQuery [29] to extract association rules from simple XML documents. They propose a set of functions, written in XQuery, which implement the Apriori algorithm [1]. In [31], Wan and Dobbie show that their approach performs well on simple XML documents but it is very difficult to apply to complex XML documents with an irregular structure. This limitation is overcome in [6], where Braga et al. introduce a proposal to enrich XQuery with data mining and knowledge discovery capabilities, by introducing XMINE RULE, an operator for mining association rules for native XML documents. They formalize the syntax and semantics for the operator and propose some examples of complex association rules.

However, XMINE is based on 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 [6] and [31] 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 does not have a 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 data set in Fig. 1 to explain this consideration. In order to infer the relationship among the features of the bullets in the data set it is necessary to fix the root node of the rules in the \texttt{ballistic\_items} element, the body and the head in \texttt{bullet}. In such way it is possible to learn that “Full Metal Jacket” type of bullets frequently have a “Rimless” type of case. However, once we fix the root of the rule in the \texttt{ballistic\_items} element, we cannot mine item sets stating that, frequently, “robberies” have occurred in “Italy.” Indeed, to mine such property the head of the rule should be fixed in the \texttt{type} element, and the body in the \texttt{country} element, which is not contained in the subtree of the \texttt{ballistic\_items} node.

Our idea is to take a more general approach to the problem of extracting association rules, i.e., to mine all frequent rules, without any a-priori knowledge of the XML data set. A similar idea was presented in [24] where Paik et al. introduced \texttt{HoPS}, an algorithm for extracting association rules from 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. The two trees $X$ and $Y$ have to be disjunct; moreover, both $X$ and $Y$ are embedded subtrees of the XML documents which means that they do not always represent the actual structure of the data. Another limitation of the proposal in [24] is that it does not consider the possibility to mine general association rules within a single XML data set; achieving this feature is one of our goals.

The idea of using association rules as summarized representations of XML documents was also introduced in [4] where the XML summary is based on the extraction of rules both on the structure (schema patterns) and on content (instance patterns) of XML data sets. The limitations of this approach are: 1) the root of the rule is established a-priori and 2) the 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 less precise and reliable.

In our work, 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 [36], [18], [15] and tree-based data representations [37], [35], [2], [25], [26], [17], [16], [23], [17]. In this paper, we focus on tree mining since XML documents are represented with a tree-shaped structure.

Table 2 shows a brief overview of the best known 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 unordered XML trees, however, as described in Section 3, the algorithm at the basis of our work can mine also ordered trees.

In [26], Termier et al. show that DRYADEPARENT is currently the fastest tree mining algorithm and CMTTreeMiner is the second with respect to efficiency. However, DRYADEPARENT extracts embedded subtrees which are trees that maintain the ancestor relationship between nodes but do not distinguish, among the (ancestor, descendant) pairs, the (parent, child) ones. In this paper, we are interested in extracting subtrees which maintain the parent-child relationship. Therefore, we propose an algorithm that extends CMTTreeMiner to mine generic tree-based association rules from XML documents.

### 7 Conclusions and Future Work

The main goals we have achieved in this paper are: 1) mine all frequent association rules without imposing any a-priori restriction on the structure and the content of the rules; 2) store mined information in XML format; 3) use the extracted knowledge to gain information about the original data sets. We have developed a C++ prototype that has been used to test the effectiveness of our proposal. We have not discussed the updatability of both the document storing TARs and their index. As an ongoing work, we are studying how to incrementally update mined TARs when the original XML data sets change and how to further optimize our mining algorithm; moreover, for the moment we deal with a (substantial) fragment of XQuery; we would like to find the exact fragment of XQuery which lends itself to translation into intensional queries.

### Acknowledgments

This research has been partially funded by the European Commission: Project FP7-SEC-2007-1 - 218237-Odyssey and Project IDEAS-ERC-227977-SMScom.

### References


Mirjana Mazuran received the MSc degree in computer science engineering from Politecnico di Milano. She spent six months as a research associate in Information Technology at Politecnico di Milano, working on a project for the extraction of relevant information from XML documents. Since January 2009 she is working toward the PhD degree in computer science at Politecnico di Milano. In particular, she is currently working on mining and querying tree-based patterns from XML documents and on relax and temporal data, the application of data mining techniques to support advanced database functionalities.

Elisa Quintarelli received the master’s degree in computer science from the University of Verona, Italy. On January 2002, she completed the PhD degree in computer and automation engineering at Politecnico di Milano and is now working as an assistant professor at the Dipartimento di Elettronica e Informazione, Politecnico di Milano. Her main research interests concern the study of efficient and flexible techniques for specifying and querying semistructured and semantically annotated databases, and the application of data mining techniques to provide intensional query answering. More recently, her research has been concentrated on context aware data management.

Letizia Tanca received the PhD degree in computer science in 1988. She worked for four years as a software engineer. Currently, she is working as a full professor at Politecnico di Milano. She has held the chair of the degree and master courses in computer science and engineering at the Leonardo campus, from 2000 to 2006. She has taught and teaches mainly courses on Databases and Information System Technologies. She is the author of more than 100 papers on databases and database theory, published in international journals and conferences, and co-author of the book “Logic Programming and Databases.” Recently, she has edited the book “Semantic Web Information Management.” Her research interests range over deductive and active databases, graph-based languages for databases, semantic-web information management, and semistructured information. Her most recent research interests concern context-aware knowledge management. She is a member of the board of the Informatics Europe association.