A Succinct Answering Prototype for XML Data

K.Chiranjeevi[1], Kumar Vasantha[2], Dr.C.Mohan Rao[3].
[1]. M.Tech II Semester, Computer Science & Engineering, Avanthi Institute of Engineering & Technology,AP,India.
[2]. Head of the Department, Computer Science & Engineering, Avanthi Institute of Engineering &Technology,AP,India.
[3]. Professor&Principal, Avanthi Institute of Engineering &Technology,AP,India.

Abstract: Extracting query from the xml data is very time taking task. It is harder task for more amounts of xml documents. So introduced tree based extraction prototype. It first finds association rules and then finds sub trees among the sub-trees. Then analyze the query then find the answer from the tree based association rules. It provides intentional answers from the extracted gist. Gist means the overall idea of the data. From the gist it answers the query and it takes less time to process. It maintain sub trees and their sequences for xml documents. This prototype works efficiently and accurate answers to the queries.

I. INTRODUCTION

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable and is defined the XML 1.0 Specification produced by the W3C and several other related specifications all gratis open standards. The design goals of XML emphasize generality and usability over the Internet. The textual data format with strong support via Unicode for the languages. Although the design of XML focuses on documents is widely used for the representation of arbitrary data structures and many application programming interfaces (APIs) have been developed to aid software developers with processing XML data and several schema systems exist to aid in the definition of XML-based languages.

An **XML schema** is a description of a type of XML document and is expressed in terms of constraints on the structure and content of documents of that type above and beyond the basic syntactical constraints imposed by XML itself, constraints are generally expressed using some combination of grammatical rules governing the order of elements and Boolean predicates that the content must satisfy the data types governing the content of elements and attributes and also more specialized rules such as uniqueness and referential integrity constraints. There are languages developed specifically to express XML and which is native to the XML specification is a schema language that is of relatively limited capability but that also has other uses in XML aside from the expression of schemas. There are two more expressive XML schema languages in widespread use are XML Schema (with a capital *S*) and RELAX NG. The mechanism for associating an XML document with a schema varies according to the schema language and may be achieved via markup within the XML document itself or via some external means.

There are two main approaches to XML document access: keyword-based search and query-answering. Paradigm for information discovery especially over HTML documents in the World Wide Web. The key advantage of keyword search querying is its simplicity – users do not have to learn a complex query language and can issue queries without any prior knowledge about the structure of the underlying data and the keyword search query interface is very flexible and queries may not always be precise and can potentially return a large number of query results and especially in large document collections. It is an important requirement for keyword search is to rank the query results so that the most relevant results appear first.

Despite the success of HTML-based keyword search engines, certain limitations of the HTML data model make such systems ineffective in many domains. These limitations stem from the fact that HTML is a presentation language and hence cannot capture much semantics. XML data model addresses this limitation by allowing for *extensible element tags* it can be arbitrarily nested to capture additional semantics. As an illustration, consider the repository of conference and workshop proceedings. Each conference/workshop has the full-text of all its papers. In addition, information such as titles and references and sections and sub-sections are explicitly captured using nested, application specific XML tags are not possible using HTML.

Given the nested, extensible element tags supported by XML, it is natural to exploit this information for querying. There is an approach is to use sophisticated query languages such as Query to query XML documents. This approach can be very effective in some cases, a downside is that users have to learn a complex query language and understand the schema of underlying XML. An alternative approach, and the one and in this paper we consider to retain the simple keyword search query interface, but exploit XML's tagged and nested structure *during query processing*. Keyword searching over XML introduces many new challenges. The result of the keyword search query is not always the entire document and it can be a deeply nested XML element. It will be good to return the XML element corresponding to the sub-section rather than returning the entire workshop proceedings (as would be done in a standard HTML search). XML keyword search results can be arbitrarily nested elements, and returning the "deepest" node containing the keywords usually gives more context information.

Frequent, dramatic outcomes of this situation are either the information overload problem and where too much data are included in the answer because the set of keywords specified for the search captures too many meanings the information deprivation problem and where either the use of inappropriate keywords the wrong formulation of the query and prevent the user from receiving the correct answer. Consequence when accessing for the first time a large data set and gaining some general information about its main structural and semantic characteristics helps investigation on more specific details. In this paper addresses the need of getting the gist of the document before querying it and both in terms of content and discovering recurrent patterns inside XML documents provides high-quality knowledge about the document content: frequent patterns are in fact intentional information about the data contained in the document itself and they specify the document in terms of a set of properties rather than by means of data. It was opposed to the detailed and precise information conveyed by the data this information is partial and often approximate because of synthetic and concerns both the document structure and its content.

II.RELATED WORK

XQuery is currently still under development by the W3C (XQuery 1.0), and is also known as W3C XML Query. The purpose of XQuery is extracting data from entire XML documents, collections of XML documents, or only document fragments. XQuery is derived from an XML query language called Quilt and which is in turn borrowed features from several other languages and including XPath 1.0, XQL, XML-QL, SQL, and OQL. XQuery 1.0 is the superset of XPath 2.0 both in syntax and semantics. XQuery is a functional expression language that can be used to query or process XML data or any data that can be represented within the same model as XML. Being purely an expression language, XQuery programs are easier to understand and maintain than XSLT, because they do not include the complexities or management of templates (rule-based system) (Funderburg, et al. 2002). This is especially true for highly structured data, and for longer programs. XQuery will still be able to effectively process semi-structured data. The query language is small and powerful. Moreover, XQuery is a full-fledged programming language. It provides if/then statements, loops, variables, quantified expressions and a set with the most important functions. Applications are made simpler by performing a single XQuery request over these views and receiving satisfactory results in one step also it has both an easy, human readable form and an XML representation (IBM Journal 2002).

XQuery can be used to query XML data which has no schema at all or XML Schema or by a Document Type Definition (DTD). The data model used by XQuery has some differences from the classical relational model. Unlikely to relational model, it has no hierarchy treats order as insignificant and does not support identity and it is a functional language and instead of executing commands as procedural languages do every query is an expression to be evaluated expressions can be combined quite flexibly with other expressions to create new expressions. When we compare XPath and XQuery, we see that XPath only designed for select a node out of an existing XML document or database. XPath can't create new XML, it can't select only part of an XML node, and it can be hard to read and understand. XPath also can't define variables or namespace bindings and it has a very simple type system, essentially just data types such as string and Boolean node set. If we need to work with date values and calculate the maximum of a set of numbers or sort a list of strings and then XPath is not suitable for this method. XQuery takes a different approach from XSLT 1.0. They both produce same results but XQuery is more functional. XQuery is very good at expressing joins and sorts and can operate sequences of values and nodes in arbitrary order and ordered documents and XQuery takes a procedural approach to query processing, making it easy to write user defined functions, including recursive ones, but more difficult to perform pattern matching. Support for XML Schema 1.0 is built into XQuery and XQuery was designed with optimization in mind.

Funderburg points that number of special features can be added to the database XQuery processor which makes it different than other query languages. A default view can be generated across the entire database. If this is supported, seamless queries against meta-data and data will be possible. For example, one can ask for all the tables that have a column named salary and have a value larger than 10000. XML query languages naturally query across meta-data (tags) and data (node values). Exposing any XML view affords this ability. As more data are placed in the view, the queries can become more powerful and abstract. For example, a view could also expose type, ownership, and data cataloging information as well as data values.

A. XML Mining Techniques

One of the simple methods to mine XML documents is probably to transform the data from XML to relations. However, the drawbacks of this method are:

1) The transformation itself is usually complex and time-consuming;

2) It may lose some important information for generating rules of interests, for example, some explicit hierarchical relationship between XML elements may become inexplicit when transformed into relations. Before knowledge discovery in XML documents occurs, it is necessary to querying XML tags and content to prepare the XML material for mining. A structured query language based query can extract data from XML documents.

a)Tree Mining over XML

The method proposed in (Zaki, et.al 2003) finds frequent structures within XML documents in order to classify them, i.e., a set of pre-classified XML documents (training dataset) is used to develop a model to classify XML documents (test dataset) that still do not belong to a class. The model is created using the underlying structure of the pre-classified documents.

Zaki models an XML document as an ordered, labeled, rooted tree. There is no distinction between attributes and elements of an XML document; both are mapped to the label set. More precisely an XML document is denoted as T = (V, B), where V is the set of labeled nodes, and B the set of branches. The label of each node is taken from a set of items L = {1, 2, 3, . . . ,m}; different nodes can have the same label. Each branch, b = (x, y), is an ordered pair of nodes where x is the parent of y.

Consider a node x in a tree T with root r, then any node y on the unique path from r to x is called an ancestor of x, and is denoted as y ·l x, where l is the length of the path from y to x. If y ·1 x that is y is an immediate ancestor of x), then y is called the parent of x, and x the child of y. A tree S = (Vs,Bs) is an embedded sub tree of T = (V,B), denoted as S ¹ T, if and only if Vs µ V , b = (x, y) 2 Bs, and x is an ancestor of y in T. Note that in the traditional definition of an induced sub tree, for each branch b = (x, y) 2 Bs and x must be a parent of y in T.

b)Mining association rules using Query

Wan and Dobbie presented a new native XML data mining approach in (Wan and Dobbie 2003).The authors show that extracting association rules from XML documents without any preprocessing or post-processing using XQuery is possible. They propose the XQuery implementation of the well-known Apriori. XML document structure proposed by the authors to mine association rules over XML reflects simply the relational model of the Association Rule Mining problem. The set of transactions is identified by the tag <transactions> and each transaction in the transactions set is identified by the tag <transaction>. The set of items in each transaction is identified by the tag <items> and an item is identified by the tag <item>. The Apriori algorithm is implemented in a classical way: first an XQuery expression is used to create the set of frequent items then another XQuery expression is used to obtain the association rules from the frequent item sets.

c)Scientio XML Miner

XML Miner classifies XML fragments within one or more XML documents. The training set and test set are selected by using XPath expressions. The classifier model is created using Fuzzy Logic, i.e., an extension to conventional logic that allows representing the truth of an assertion by any real number between 1 and 0. Llogic allows only the value 1 that states the truthful of an assertion and 0 that states is falsehood and using Fuzzy Rules XML Miner generates a rule set that explains and predicts selected values in a test dataset. The resulting rule set is expressed in Meta rule, a dialect of XML. As an example the Fuzzy Logic uses the numeric values of the characteristic of the Iris species to define Fuzzy concepts of small, medium and large for each characteristic of the flower. Then the model is applied to the rest of the data to predict the species.

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 by implementing graph- or tree-based algorithms. Our approach introduce a proposal for mining and storing Tree-Based Association Rules (TARs) as a means to represent intentional knowledge in native XML TAR represents intentional knowledge in the form SB) SH and SB is the body tree and SH the head tree of the rule and SB is a sub tree of SH. The rule SB) SH states that, if the tree SB appears in an XML document D and is likely that the "wider" tree SH also appears in D.

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

1. It allows to obtain and store implicit knowledge of faces a data set for the first time and 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 and there is a significant portion of XML documents which have structure because its only implicitly 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. Tree based association rules represent a data guide that helps users to be more effective in query formulation; c) it supports query optimization design and 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 also because frequent patterns allow to discover hidden integrity constraints can be used for semantic optimization; d) for privacy reasons and 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. Approximate answers particularly useful not only when quick answers are needed but also when the original documents are unavailable. In fact it 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 content structure of an XML document and 2) and to use them for intentional query-answering allowing the user to query the extracted TARs rather than the original document.

B. 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 is relation and where X and Y are two arbitrary sets of data items, such that X $\cap$Y =ф ;. The quality of an association rule is measured by means of support and confidence, Minimum Support corresponds to the frequency of the set X ^{U} Y in the data set and while confidence corresponds to the conditional probability of finding Y , having found X and is given by supp(X ^{U} Y)/supp(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 Info-set conventions and we represent an XML document as a tree^2(N,E, r; l, c_i), where N is the set of nodes, r € N is the root of the tree and edges E is the set of edges, l :N$\rightarrow$L is the label function which returns the tag of nodes (with L the domain of all tags) and c:N$\rightarrow$C u {$\perp$} is the content function which returns the content of nodes (with C the domain of all contents) and let consider the element-only Info-set content model [28], where XML non-terminal tags include only other elements and attributes while the text is confined to terminal elements. We are interested in finding relationships among sub trees of XML documents.

III.PROPOSED WORK

TAR mining is a process composed of two steps: 1) mining frequent sub trees with a support above a user defined threshold from the XML document 2) computing interesting rules with a confidence above a user defined threshold, from the frequent sub trees. As will be discussed in more detail and the problem of finding frequent sub trees has been widely treated in the literature [1]. Algorithm 1 presents our extension to a generic frequent sub tree mining algorithm in order to compute interesting TARs. Algorithm 1 inputs are the XML document D the threshold for the support of the frequent sub trees min supp, and the threshold for the confidence of the rules, min conf. Algorithm 1 finds frequent sub trees and then hands each of them over to a function that computes all the possible rules depending on the number of frequent sub-trees and their cardinality amount of rules generated by a naive Compute-Rules function may be very high. Given a sub tree with n nodes, we could generate 2n - 2 rules. This explosion occurs in the relational context based on similar features, it is possible to state the following property allows us to propose the optimized version of Compute-Rules shown in Function 2.

Algorithm 1. Get-Interesting-Rules (D, minsupp, minconf)

```
1: // frequent subtrees
2: FS = FindFrequentSubtrees (D, minsupp)
3: ruleSet =φ ;
4: for all s 2 FS do
5:         // rules computed from s
6:         tempSet = Compute-Rules(s; minconf)
7:        // all rules
8:        ruleSet =ruleSet [ tempSet
9: end for
10: return ruleSet
```

Function 2. Compute-Rules ðs; minconfÞ

```
1: ruleSet =φ; blackList =φ ;
2: for all cs, subtrees of s do
3:         if C is not a subtree of any element in blackList then
4:                 conf = supp(s) / supp(cs)
5:                 if conf ≥minconf then
6:                         newRule =(c_s, s, conf, supp(s)
7:                         ruleSet =ruleSet u{newRule}
8:                 else
9:                 blackList =blackList  u c_s
10:                end if
11:      end if
12: end for
```

```
13: return ruleSet
```

In Function 2, TARs are mined exploiting Remark 1 by generating first the rules with the highest number of nodes of body tree. Consider two rules Tr1 and Tr2 whose body trees contain one and three nodes, suppose both rules have confidence below the fixed threshold. If the algorithm considers rule Tr2 first, all rules whose bodies are induced sub trees of Tr2 will be discarded when Tr2 is eliminated. So is more convenient to first generate rule Tr2 and in general to start the mining process from the rules with a larger body. We can use this solution we can lower the complexity of the algorithm though not enough to make it perform better than exponentially. Notice that the process of deriving TARs from XML documents is only done periodically. Intentional knowledge represents frequent information to update it and is desirable to perform such process after a big amount of updates have been made on the original document. So in the case of stable documents 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 and they are stored in XML format. The 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 the support and confidence of the rule that follows the list of elements and one for each node in the rule head and We exploit the fact that the body of the rule is a sub-tree of the head and use a Boolean attribute in each node to indicate if it also belongs to the body and each blank node is described by an element <blank>. Finally, the rules in the XML file are sorted.

One of the (obvious) reasons for using TARs instead of the original document is that processing iTARs for query answering is faster than processing. To take full advantage of this and we introduce indexes on TARs to further speed up the access to mined trees—and in general of intentional query-answering. The literature survey the problem of making XML query-answering faster by means of path-based indexes has been. In general path indexes are proposed to quickly answer queries that follow some frequent path template and are built by indexing only those paths having highly frequent queries. It start from a different perspective we want to provide a quick and often approximate, answer also to casual queries. Given a set R of rules and the index associates with every path p present in at least one rule of R, the references to rules that contain p in SH and an index is an XML document containing a set of trees T_1 to T_n such that each node n of each tree Ti contains a set of references to the rules containing in S_H the path from the root node of T_i to n. A TAR-index contains references both to iTARs and sTARs and is constructed by Algorithm 3.

Algorithm 3. Create-Index (D)

```
1: for all $D_i$ € D do
2:     for all $d_j$ € $D_i$ with j € (2, 3, . . . n) do
3:         references$_i$(root($d_1$)) = references(root($d_1$))
           [ references(root($d_j$))
4:         sumChildren ($d_1$, $d_j$)
5:     end for
6: end for
7: return D
```

Function 4. sumChildren (T1,T2)

```
1: for all x €children(root(T2)) do
2:     if Ǝ c €children(root(T1)) | c = x then
3:     references(root(c)) = references(root(c))
       [ references(root(x))
4:     c = sumChildren(c,x)
5:     else
6:     addChild(root(T1),x)
7:     end if
8: end for
9: return T1
```

Before applying the algorithm, two sets A and C are constructed containing consequent trees of all the TARs which are 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 = (P1, . . . , Pn) and D = (D1, . . ,Dm) are obtained that are partitions of A and C, respectively, where each Pi and Di contains trees having same root.

The Algorithm 3 is applied to merge the trees in each set using the same rationale behind the Data Guide construction procedure, in particular, for each set, the first tree is merged together with the other and this 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).

iTARs provide an approximate intentional 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 and a user query over the original data set can be automatically transformed into a query over the extracted. The answer will be intentional and 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 intentional answers are in some cases more useful than the extensional ones . For example, if a user asks for the incidents registered in the data, the extensional answer is the list of all incidents (possibly megabytes) to be inspected manually while an intentional answer might be that “80 percent of incidents were robberies.”

Class 1: σ/π-queries. Used to impose a simple, or complex restriction on the value of an attribute or the content of a leaf node possibly results are ordered. 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 and ordered by the date the incident was reported.”

Class 2: count-queries. Used to count the number of elements having a specific content and query creates a set containing the elements which satisfy the conditions and then returns the number of elements in such set used to select the best k answers satisfying a counting and grouping condition and this 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. Let us take an example that Retrieve the k most used types of bullets and in all classes of queries conditions can be imposed on the descendants of the element that is returned and not on its ancestors. The 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 and it is possible to obtain the intentional answer in two steps: 1) rewrite q_E into q_I ; 2) apply qI on the intentional knowledge. 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 intentional query 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.

. V_W= (vw_j, vw_j is a variable of the paths in the WHERE clause of q_E, in the same orderi. . CONN = ($conn_k$, $conn_k$) is a connective in the WHERE clause of q_E, in the same order i.

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

Algorithm 5. Class1-Query (v_F ,V_W,CONN,v_{OB})

```
1:// the intensional query is empty
2: IQ=ф
3: if VW ≠ф ; then
4:  instance rules for paths with a constraint
5:        IQ =IQ .get_iTARs(v_F; VW;CONN; false)
6: else
7:  The path without constraint
8:        IQ = IQ . get_sTARs(v_F )
```

```
9: end if
10:  order the results
11: IQ= IQ .“for $r in $Rules/Ruleorder by $r/v_F/v_OB
return $r”
12: return IQ
```

Function 6. get_iTARs for variables and connectives of count

```
1: IQ=ϕ
2: for all v_j € variables do
3:         if count = true then
4: sql count queries match only in the antecedent
5:                 Q =Q .“let RefI_j:=referencesA(for, v_j)”
6:        else
7:  match both in antecedent and consequent
8:         Q =Q . “let RefI_j:=references(for, vj)”
9:        end if
10: end for
11: Q = Q. “let Rules :=”
12: for all vj 2 variables, j€ (1 . . . n) do
13: Q = Q. “ruleset($RefI_j) connectivej”
14: end for
15: return Q
```

Function 7. get_sTARs (variable)

```
1: Q =“let $Ref_S:=references(variable,“”)
        let $Rules := ruleset($RefS)”
2: return Q
```

According to above algorithms we called that frame work as Tree Ruler prototype is a tool that integrates the functionalities are proposed in our approach and also given an XML document it enables users to extract intentional knowledge and compose traditional queries as well as queries over the intentional knowledge receiving both extensional and intentional answers and users formulate XQueries over the original data and queries are automatically translated and executed on the intentional knowledge and the answer is given in terms of the set of TARs which reflect the search criteria.

Tree Ruler interface offers three tabs. . Get the gist allows intentional information extraction from an XML document given the support and confidence and the files where the extracted TARs and their index are to be stored. . Get the idea allows showing the intentional 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 intentional knowledge and the original XML document and users have to write an extensional query when the query belongs to the classes we have analyzed it is translated and applied to the intentional knowledge. Once it is executed, the TARs that reflect the search criteria are shown.

A) Comparative Analysis of previous and proposed approaches*:*

There are so much usage is increasing day by for answering a query, the answering is predictable but not accurate. The answer content is so large, so its leads to more occupation of the memory. Considering the above reasons its taking more amount of time to search. The content is in paragraph format there is increasing time complexity to retrieve the query answer.

So we introduced answering the query is intentional and accurate by maintaining the content in xml format. And also by finding the association rules of the searching query keyword. We introduced TARs (Tree based Association Rules) for finding the answering the query. The information is stored in the xml format, so that it is very to search and it reduce the processing time and retrieval time. We can the data find as per the query is intentional answers based on the mining of the gist (summary of the content).

Our proposed system process as follows: 1. first input the content (xml documents) 2. Get the gist of the xml documents and the extracted content is stored in TAR and index the sub trees present in the Tree. 3. According to the query the system have to give intentional answers, the system have to compare the content in the extracted content from the TARs. 4. If the query belongs to the classes analyzed from the TARs.

IV.CONCLUSION

We introduced tree based query answering prototype for xml documents. It reduces the processing of the xml documents. All process designed the prototype in the form of xml format only. The extracted sub trees are stored in xml format only. These are so called as tree based association rules. It will find 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. This is particularly useful not only when quick answers are needed but also when the original documents are unavailable. In fact, once extracted, tree based association rules can be stored in a (smaller) document and be accessed independently of the data set they were extracted from.

REFERENCES

[1] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," Proc. 20th Int'l Conf. Very Large Data Bases, pp. 478-499, 1994.

[2] T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, and S. Arikawa, "Efficient Substructure Discovery from Large Semi- Structured Data," Proc. SIAM Int'l Conf. Data Mining, 2002.

[3] T. Asai, H. Arimura, T. Uno, and S. Nakano, "Discovering Frequent Substructures in Large Unordered Trees," Technical Report DOI-TR 216, Dept. of Informatics, Kyushu Univ., http:// www.i.kyushu-u.ac.jp/doitr/trcs216.pdf, 2003. 1406 IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 24, NO. 8, AUGUST 2012 TABLE 2 Tree-Mining Algorithms Overview

[4] E. Baralis, P. Garza, E. Quintarelli, and L. Tanca, "Answering XML Queries by Means of Data Summaries," ACM Trans. Information Systems, vol. 25, no. 3, p. 10, 2007.

[5] D. Barbosa, L. Mignet, and P. Veltri, "Studying the XML Web: Gathering Statistics from an XML Sample," World Wide Web, vol. 8, no. 4, pp. 413-438, 2005.

[6] D. Braga, A. Campi, S. Ceri, M. Klemettinen, and P. Lanzi, "Discovering Interesting Information in XML Data with Association Rules," Proc. ACM Symp. Applied Computing, pp. 450-454, 2003.

[7] Y. Chi, Y. Yang, Y. Xia, and R.R. Muntz, "CMTreeMiner: Mining both Closed and Maximal Frequent Subtrees," Proc. Eighth Pacific- Asia Conf. Knowledge Discovery and Data Mining, pp. 63-73, 2004.

[8] C. Combi, B. Oliboni, and R. Rossato, "Querying XML Documents by Using Association Rules," Proc. 16th Int'l Conf. Database and Expert Systems Applications, pp. 1020-1024, 2005.

[9] A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke, "Privacy Preserving Mining of Association Rules," Proc. Eighth ACM Int'l Conf. Knowledge Discovery and Data Mining, pp. 217-228, 2002.

[10] L. Feng, T.S. Dillon, H. Weigand, and E. Chang, "An XMLEnabled Association Rule Framework," Proc. 14th Int'l Conf. Database and Expert Systems Applications, pp. 88-97, 2003.

[11] S. Gasparini and E. Quintarelli, "Intensional Query Answering to XQuery Expressions," Proc. 16th Int'l Conf. Database and Expert Systems Applications, pp. 544-553, 2005.

[12] B. Goethals and M.J. Zaki, "Advances in Frequent Itemset Mining Implementations: Report on FIMI 03," SIGKDD Explorations Newsletter, vol. 6, no. 1, pp. 109-117, 2004.

[13] R. Goldman and J. Widom, "DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases," Proc. 23rd Int'l Conf. Very Large Data Bases, pp. 436-445, 1997.

[14] R. Goldman and J. Widom, "Approximate DataGuides," Proc. Workshop Query Processing for Semistructured Data and Non-Standard Data Formats, pp. 436-445, 1999.

[15] A. Inokuchi, T. Washio, and H. Motoda, "Complete Mining of Frequent Patterns from Graphs: Mining Graph Data," Machine Learning, vol. 50, no. 3, pp. 321-354, 2003.

[16] A. Jime´nez, F. Berzal, and J.C. Cubero, "Mining Induced and Embedded Subtrees in Ordered, Unordered, and Partially- Ordered Trees," Proc. 17th Int'l Symp. Methodologies for Intelligent Systems, pp. 111-120, 2008.

[17] D. Katsaros, A. Nanopoulos, and Y. Manolopoulos, "Fast Mining of Frequent Tree Structures by Hashing and Indexing," Information and Software Technology, vol. 47, no. 2, pp. 129-140, 2005.

[18] M. Kuramochi and G. Karypis, "An Efficient Algorithm for Discovering Frequent Subgraphs," IEEE Trans. Knowledge and Data Eng., vol. 16, no. 9, pp. 1038-1051, Sept. 2004.

[19] H.C. Liu and J. Zeleznikow, "Relational Computation for Mining Association Rules from XML Data," Proc. 14th ACM Conf. Information and Knowledge Management, pp. 253-254, 2005.

[20] G. Marchionini, "Exploratory Search: From Finding to Understanding," Comm. ACM, vol. 49, no. 4, pp. 41-46, 2006.

[21] M. Mazuran, E. Quintarelli, and L. Tanca, "Mining Tree-Based Association Rules from XML Documents," technical report, Politecnico di Milano, http://home.dei.polimi.it/quintare/ Papers/MQT09-RR.pdf, 2009.

[22] M. Mazuran, E. Quintarelli, and L. Tanca, "Mining Tree-Based Frequent Patterns from XML," Proc. Eighth Int'l Conf. Flexible Query Answering Systems, pp. 287-299, 2009.

[23] S. Nijssen and J.N. Kok, "Efficient Discovery of Frequent Unordered Trees," Proc. First Int'l Workshop Mining Graphs, Trees and Sequences, 2003.

[24] J. Paik, H.Y. Youn, and U.M. Kim, "A New Method for Mining Association Rules from a Collection of XML Documents," Proc. Int'l Conf. Computational Science and Its Applications, pp. 936-945, 2005.
[25] A. Termier, M. Rousset, and M. Sebag, "Dryade: A New Approach for Discovering Closed Frequent Trees in Heterogeneous Tree Databases," Proc. IEEE Fourth Int'l Conf. Data Mining, pp. 543-546, 2004.
[26] A. Termier, M. Rousset, M. Sebag, K. Ohara, T. Washio, and H. Motoda, "DryadeParent, an Efficient and Robust Closed Attribute Tree Mining Algorithm," IEEE Trans. Knowledge and Data Eng., vol. 20, no. 3, pp. 300-320, Mar. 2008.
[27] World Wide Web Consortium, XML Schema, http:// www.w3C.org/TR/xmlschema-1/, 2001.

AUTHORS DETAILS:

K.Chiranjeevi received the B.Tech degree in Computer Science and Engineering from SRKR Engineering College, Bhimavaram, A.P in 2008. He is pursuing his M.Tech in Computer Science and Engineering in Avanthi Institute of Engineering & Technology,Vizag,A.P.His area of interests include Data Warehousing and Data Mining, and Secure Database Applications.

Kumar Vasantha, M.Tech (CSE)
He received the B.Tech degree in Computer Science and Information Technology from JNT University, Kukatpalli, Hyderabad and received the M.Tech degree in Software Engineering from JNT University, Kakinada. Presently he is working as Head of the Department in Computer Science and Engineering in Avanthi Institute of Engineering and Technology, Vizag, A.P.His area of interests include Data Warehousing and Data Mining, RDBMS and Web Technologies.He has Published more than 10 papers in various national and international journals.

Dr.C.Mohan Rao.M.Tech (CST), Ph.D.
He received the M.Tech degree in Computer Science and Technology from Andhra University College of Engineering,Vizag and awarded PhD by Andhra University in 2000.He has 18 years of teaching and research experience and guided number of M.Tech students for their projects.Presently he is working as Principal in Avanthi Institute of Engineering and Technology, Vizag,A.P.His research interests include Data Warehousing and DataMining,Cryptography and Network Security and Artificial Intelligence..He has published 23 papers in various national and international journals.He is guiding 2 research scholars for Ph.D.He received the Best Teacher Award from JNTU,Kakinada in 2009.