XML Algebras for Data Mining

M. Zhang J.T. Yao

Department of Computer Science
University of Regina
Regina, Saskatchewan, Canada S4S 0A2
E-mail: [zhang2mi, jtyao]@cs.uregina.ca

ABSTRACT

The XML is a new standard for data representation and exchange on the Internet. There are studies on XML query languages as well as XML algebras in literature. However, attention has not been paid to research on XML algebras for data mining due to partially the fact that there is no widely accepted definition of XML mining tasks. This paper tries to examine the XML mining tasks and provide guidelines to design XML algebras for data mining. Some summarization and comparison have been done to existing XML algebras. We argue that by adding additional operators for mining tasks, XML algebras may work well for data mining with XML documents.

1. INTRODUCTION

Ever since the emergence of the Internet, the amount of information on the Web has been growing exponentially. The World Wide Web has become a public knowledge base. However, the ever-increasing size of the web is beyond the control of any personnel or organizations. Thus, how to find desired information in such a large web space poses a big challenge.

The present dominant data publishing format on the internet is HTML (a markup language that specifies the rendering of the web documents). Recently, XML (eXtensible Markup Language) is gaining popularity as a new standard for data representation and exchange on the internet. Although both HTML and XML documents can be viewed as a hierarchical structure, the former could not express the internal relationship between its elements.

Web mining based on HTML documents normally consider text mining, or traditional information retrieval techniques. Various languages to query XML documents are proposed by researchers all around the world. Some research has been done on algebras to discover previous unknown knowledge from relational databases. However, attention has not been paid to research on XML algebras for data mining. An XML algebra is a kind of operational query language. A query expressed in XML algebras describes the procedures to obtain the answer. A basic step to study XML algebras for data mining is to understand its mining tasks. Unfortunately, there is no widely accepted definition of XML mining tasks. This paper tries to examine the XML mining tasks and provide guidelines to design XML algebras for data mining.

The paper is organized as follows: Section 2 gives background information and motivation of this research. Section 3 compares three recently proposed XML algebras based on the attributes formally defined in this section. Section 4 discuss the guidelines to define and represent knowledge on XML as well as the principles for extending the existing XML algebras for mining tasks. Section 5 is the summary.

2. MOTIVATIONS AND BACKGROUND INFORMATION

HTML documents have hierarchical structure with elements being nested and enclosed by pre-defined tags. The tags define the display format of the content embedded in them. In other words, HTML documents have no knowledge of the semantic internal relationships between elements. As a result, the existing search engines only employ keyword matching strategy in retrieving data. HTML documents are treated as plain text documents, tags are simply ignored in the content searching process.

An XML document consists of nested elements enclosed by user-defined tags, which indicate the meaning of the content contained. Figure 1 shows an example of an XML document named pub.xml, which contains some publication information.

```
<?xml version="1.0" ?>
<publication>
   <journal title="DBMS">
       <editor>Jack</editor>
       <article>
           <title>index construction</title>
           <author>Smith</author>
       </article>
       <article>
           <title>query evaluation</title>
       </article>
   </journal>
   <journal title="data mining"> </journal>
</publication>
```

Figure 1. An example of an XML document: pub.xml

As illustrated, an XML document has a hierarchical structure. In most cases, we can view this structure as a tree. In Figure 1, the root element *publication* contains two *journal* elements, each with an attribute *title*. The first *journal* element contains three elements, one *editor* and two *article*s, which also have their own sub-elements.

Since the user-defined tags of XML documents reflect the inherent relationships between elements, we may use traditional database techniques to query XML documents. If the structural information of XML documents could be fully exploited, we may obtain much more accurate results. The explicit internal relationships between elements makes it possible to query XML.

It is natural to think about using query languages similar to SQL (Structured Query Language) in relational databases to query XML. However, the traditional SQL language is unable to capture the internal structures of XML data. In a relational database, data is represented in relations, where each relation consists of a relation schema and relation instances. The data in relational database is flat with no containment relationship between elements(tuples). Based on the relational model, SQL queries only employ value-based predicates to make selection without considering the internal structure. In comparison, the XML data model is more complex. It can not be viewed as a table but a directed, edge-labelled tree or directed, edge-labelled graph. Nodes in the tree/graph represent elements or attributes, while the edges connecting nodes represent parent-child relationship between elements or element-attribute relationship. Each edge is labelled tag name of the element or name of the attribute to which this edge points. Containment is the main relationship between elements. Moreover, unlike the relational model, XML does not have a pre-defined strict schema, it allows missing or repetition of sub-elements. For example, in Figure 1, the first *journal* element contains three sub-elements but the second *journal* has none. Therefore, SQL for relational databases can not be applied on XML data directly.

There are some approaches in literature to solve the query problem over XML data. We classify them into two categories:

Transformation of data map XML documents into instances of relational databases in order to use a traditional query language,[19, 20]

Development of query language develop new query languages[2–4, 22] to extract the information of internal structure of XML, specifically, the containment relationships between elements.

The first approach usually produces too many relations and loses important information on relationship. For example, the explicit hierarchical relationship between XML elements may be lost. We adopt the second approach in this paper.

Although XML query languages in literature differ in detailed grammars, they share a common feature, that is: querying structure as well as values of elements. In other words, a query makes selection based on the elements' positions in the

data source in addition to their values. For example, Figure 2 is a query expressed in Xquery[22] over the document *pub.xml* in Figure 1 where "//" indicates ancestor-descendant relationship, and "/" indicates parent-child relationship. This query retrieves from *pub.xml* titles of all the articles written by Smith.

```
FOR  $b IN document("http://.../pub.xml")/publication//article
    $a IN $b/title
WHERE $b/author="Smith"
RETURN
    <article>$a</article>
```

Figure 2. A query in XQuery to pub.xml

Query languages are designed to express human information needs. A user's query only describes the desired answer without specifying how the answer is to be computed. In order to obtain the answer, the system must translate the user's query into an execution plan and then carry out the plan to get the results. In the relational database, when a query is given by users, the system translates the query into the form of relational algebra which describes a step-by-step procedure for computing the answer, based on the order in which operators are applied in the query. Naturally, we need some expressions similar to relational algebra to describe queries for XML in an operational manner. Since these expressions are the counterpart of relational algebra in XML manipulation, it could be called XML algebras. As some research has been done on manipulating query and data mining on relational database with an unified language, it might be possible to do so on XML data. The existing XML algebras might be a good starting point to design XML mining algebras.

3. DEFINITION AND COMPARISON OF XML ALGEBRAS

XML algebras play the same role over XML documents as relational algebra in relational database, that is, as the intermediate representation to user's queries. Analogous to relational algebra, an XML algebra can be defined as:

DEFINITION 3.1. *An XML algebra X is a quadruple $\{M, B, P, R\}$, where M is the XML data model; B is the basic unit of information; P is a set of operators that operate on the instances of the data model; Each operator accepts one or two collections of Bs as input and produces a collection of Bs as output; R is a set of equivalence rules (or called optimization rules) of operators used for optimization.*

As mentioned above, the XML data model could be either a tree or graph. A forest could be transformed to one tree by simply adding a root node as the parent to all trees. Basic unit of information is the individual member of collections that are feed to the operators. As we know, in relational algebra, each operator accepts relations as input and produces a relation as its output. A relation is composed of tuples which is the basic unit of information in relational algebra. In XML algebras, however, basic unit of information could be regarded as the correct counterpart for a relational tuple. Equivalence rules reveal the associative or commutative properties of some operators. Thus we could reorder the operators of a query to reduce the size of intermediate results without affecting the final results.

Since XML algebras are designed to generate efficient step-by-step procedures for users' queries in a certain query language, there are at least two criteria that can be used to evaluate an XML algebra: expressiveness and efficiency. Expressiveness is the ability to express all the queries in certain query languages such as Xquery.[22] Efficiency is the ability to produce efficient evaluation plan by applying the optimization rules.

There are many XML algebras that have been proposed in literature. We will focus on three recently proposed algebras: Niagara Algebra,[6] TAX(Tree Algebra for XML)[11] and XAL[5] in this paper. They will be compared from five aspects, namely, data model, basic unit of information, operators, expressiveness, and optimization rules.

TAX is an algebra for XML data developed for TIMBER XML database system at the University of Michigan.[9] It manages collections of data trees directly, that is, a collection of data trees is feeded to the operators directly. TAX uses the notion of *pattern tree*, which identifies the subset of nodes of interest in any tree in a collection of trees. Most operators in TAX use a *pattern tree* to select needed information. Niagara is an XML algebra designed by University of Wisconsin for Niagara Internet query system.[14] Instead of feeding the whole trees to the operators, Niagara uses a bag of vertices which contains the vertex being operated upon and all the vertices already visited along the path to that vertex. XAL is simpler than the above two. The operators of XAL receive collections of vertices as input. There are

Relational	TAX	Niagara	XAL
selection	selection	select	selection
projection	projection	follow	projection
product	product	product	product
join	join	join	join

Table 1. Algebra operators

three categories of operators: *extraction* operators that retrieve the information from XML document, *meta*-operators that control the evaluation of expression, and *construction* operators that build new XML documents from the extracted data. XAL also provides a heuristic optimization algorithm.

3.1. Data model

In XAL, each XML document is represented as a rooted directed graph with a partial order relation defined on its edges. Niagara also model the XML document as rooted directed graph with elements and attributes corresponding to vertices in the graph. TAX treats the XML documents as forest of labelled rooted trees. Each node of the trees has a virtual attribute called *pedigree* which carries the history of "where it came from" as trees are manipulated by operators.

3.2. Basic unit of information

Analogous to relational algebra operators who operate on collections of tuples, an operator of an XML algebra performs manipulation on collections of entities. Basic unit of information refers to the individual member of these collections, or counterpart in XML for a relational tuple.

In XAL, the basic unit is the vertex which represents either an element or attribute. An operator receives a set of vertices as input and produces a set of vertices as output.

Niagara thinks of a bag of vertices as the basic unit. Thus the operators operate on a collection of bags of vertices, where each bag contains the vertex being operated upon as well as all the vertices already visited along the path to that vertex. Galanis, *et al.*[6] pointed out the drawbacks of treating vertex as basic unit. XML document allows repetition of sub-elements, consequently, an element may have several sub-elements with same tag names but different content. Suppose we apply an operator that filters elements based on the content value of their sub-element, and an element qualifies if the content value of any of its sub-elements satisfies the qualification. The output of this operator consists of elements that may have sub-elements that do not satisfy the qualification.

TAX takes a labelled rooted tree as the basic unit by introducing the notion of pattern tree and witness tree. A pattern tree is a pair of $P = (T, E)$, where T is a node-labelled and edge-labelled tree, E is a formula with value-based predicates applicable to tree nodes. Each node in T has a distinct integer as its label and each edge is either labelled pc (parent-child) or ad (ancestor-descendant). A witness tree is an instance in the data trees that matches the pattern tree. In other words, a witness tree is an embedding of a pattern tree.

3.3. Operators

Each algebra defines a set of operators. Some of them have the same name with their counterparts in relational algebra. Table 1 summarizes the major operators of these three XML algebras, comparable operators are in the same row.

These XML algebras have similar operators. Like the *selection* operator in relational algebra, *selection* or *select* operator in XML algebras chooses qualified entities from the input set of entities. Entity refers to the basic unit of information. The *projection* or *follow* operators have operations differing greatly from their counterpart in relational algebra. In relational algebra, the *projection* operator output all the tuples from the input with certain fields being removed. However, in *Niagara* and *XAL*, the *projection* or *follow* operators produce a set of entities reachable by following the paths specified by the parameter from the input set of entities. In *TAX*, the *projection* finds entities that match the input pattern and output these entities with some attributes being removed. In summary, the *projection* operators work more like the combination of *selection* and *projection* in relational algebra. *Join* operators receive two sets of entities and produces a set of compound entities based on the join condition. A *Product* operator is a join operation with join condition always true.

3.4. Expressiveness

As XML algebras serve as intermediate representation of user's queries, it must be powerful enough to express all possible queries in certain query language. *TAX* supports most queries in many popular XML query languages, such as XQuery,[22] XML-QL[4] *etc*. *Niagara* supports queries expressed in Quilt.[3] *XAL* supports queries in XQuery.[22] In terms of expressiveness, TAX is the most powerful one.

3.5. Optimization rules

Queries expressed in XML algebras describe the step-by-step operations to compute the answer. Performance of the operations largely rely on the order of operators in the algebra expression. Optimization rules mainly refer to the equivalence rules to reorder the operators in the algebra expression. The goal of optimization is to reduce the size of intermediate results.

Niagara explicitly provides the equivalence rules. For example, *follow* operators are interchangeable, two *selections* on the same expression could be collapsed into a single selection using the conjunction of the respective predicates, etc. *TAX* does not give rules explicitly, but it points out that its set operators such as *union*, *intersection* are associative and commutative but *product* operator is neither associative nor commutative. *XAL* also provides a set of equivalence rules. More importantly, it gives a heuristic algorithm that uses the equivalence rules to transform a query tree into an optimized tree.

3.6. Summary

XAL is easy to understand and implement because it uses the vertex as the basic unit of information. TAX and Niagara are more efficient. In terms of expressiveness, TAX is the most powerful. However, it is unwise to say that one is better than the other as each has its advantages. The importance here is that these algebras provide a very good starting point to develop XML algebras for mining tasks.

4. ALGEBRAS FOR XML MINING

Data mining is to find previously unknown and potentially useful knowledge from a large amount of data.[17] Finding association rules, data classification, and clustering are some tasks of data mining. Researchers have been conducting intensive studies in supporting individual mining tasks. However, they fail to support multi-step mining tasks.[10] We could not perform complex mining task which is the combination of several simple tasks of different types. In this case, the output of one task need to be served as the input to another task. Decades of successful experiences in database systems enlighten many researchers to employ methods similar to query manipulation, i.e. designing standard data mining query languages. The ideal scenario is that complex data mining tasks could be expressed in a standard data mining query language. The data mining system can therefore translate the given tasks into an internal representation form which is actually a step-by-step procedure to get the mining result. Furthermore, this step-by-step procedure, or execution plan, could be optimized by reordering the sub-tasks. To achieve this goal, the mined knowledge or pattern and the raw data must be represented in the same form.[8] Imielinske and Mannila[8]introduced the concept of inductive databases where data and generalized patterns are kept.

Knowledge discovery is an interactive process as traditional querying, i.e. users can query data and patterns with an unified language. Many data mining query languages have been proposed to provide an integration of data mining environment with database management systems. Han *et al.*[7] was probably one of pioneers in designing the data mining query languages for relational database and suggested design guidelines. To evaluate data mining queries, several algebras for data mining were proposed.[10, 18] However, they were designed specifically for mining relational databases. Little research has been done in the field of mining XML with an unified language.

One of the simplest 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) the transformation 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.

The other way is to mine knowledge on XML directly. However, due to the vague definition of data mining, there is no widely accepted interpretation of "knowledge" on XML. For example, what is an association rule on XML data?

What is the individual object in data clustering? Recently, Nayak,*et al.*[16] gave taxonomy of XML mining and classified XML mining into structure mining and content mining, upon which different tasks of mining were informally defined. For example, in XML structure mining, clustering is to identify DTD (Document Type Definition) similarities among XML documents. DTD defines the legal building blocks of an XML document. Association rule discovery is to describe relationship between co-occurred tags. In content mining, clustering and association rule discovery are applied to the content.

Generally, the individual object in mining tasks is an XML document. In other words, an XML document is the counterpart of tuple in relational database mining. However, this taxonomy has some limitations:

1. In XML documents, the path formed by concatenating tags from the root to the current element describes the meaning of the enclosed content. The separation of structure mining and content mining often results in incorrect rules.

2. The structure mining mainly deals with DTD, which itself is not an XML document. Thus it is difficult to represent knowledge obtained from structure mining in the same form as that from content mining.

The prerequisite of designing XML mining algebras is that all generalized knowledge must be expressed in XML, i.e. input and output of operators must have the same form. Extension of existing XML algebras by adding some mining operators might work well as a mining algebra. We first need to examine possible mining operators. We may take an easy way to design an operator for each mining task. Each operator is associated with an algorithm for the mining task. However, this method leaves little room for optimization. A further development is to decompose mining tasks into several sub-tasks. These sub-tasks are shared among different mining tasks. We could design an operator for each sub-task. With such consideration, we should find the minimal set of sub-tasks such that all the mining tasks could be fulfilled by the execution of a sequence of sub-tasks. We can then define an operator for each sub-task. All operators should have the following properties:

1. The operation of each operator should be clearly defined and easily implemented.

2. The input and output of each operator should be in the same form.

However, due to the heterogeneity of knowledge, it is very difficult to find such a set of sub-tasks or define the operators for the sub-tasks. Knowledge are at different abstraction levels. Some knowledge is extracted from the primitive data, others are abstracted from generated knowledge. Since all types of knowledge ultimately comes from the raw data, they have certain relationships between them. The relationship is either horizontal or vertical. Horizontal relationship means the compared knowledge are at the same abstraction level. Vertical relationship means the compared knowledge are at different abstraction levels. The lowest level knowledge is extracted directly from primitive data. The higher level knowledge is generated from the lower level knowledge. All types of knowledge forms a hierarchy. For example, the frequent pattern within an XML document is extracted from the primitive data. When these patterns are produced, the inter-document clustering is performed based on the frequent patterns they contain. Each type of knowledge corresponds to one mining task. If the mining tasks on XML could be defined in this way, any type of mining task could be decomposed to a sequence of operations starting from the primitive data. As a result, most operators of the existing XML algebras could be used in mining. We only need to add some operators for mining. These operators may include: an operator that counts the number of given patterns, an operator that extracts all patterns of the given size, an operator that extracts the patterns containing given sub-patterns, an operator that computes the similarity/distance between patterns, *etc*. Patterns could either be linear or non-linear. Since all the patterns are represented in XML, linear pattern is regarded as a special case of non-linear pattern.

5. TOWARDS XML MINING SUBTASKS AND OPERATORS

We will discuss some XML mining tasks in detail based on previous works in this section. With a clear understanding of these tasks. A set of subtasks and operators can be identified. Therefore XML mining algebras could be proposed based on the knowledge. One of the important XML mining tasks is to mine frequent tree patterns in a collection of XML documents. The patterns are modelled as either ordered labelled trees or unordered labelled trees. An ordered tree refers to a tree where the left-to-right order exists among the sibling nodes. An unordered tree is a tree where the order of sibling nodes are ignored. Formally, the problem is defined as,

DEFINITION 5.1. *Given a collection of data trees D and a user-defined minimum support s, find frequently occurred tree patterns in D with frequency greater than or equal to s.*

This problem is closely related to the *tree inclusion problem*: given a pattern tree P and a target tree T, find the instances of P in T. Kilpelainen[12] studied the tree inclusion problem under five variations of *inclusion* in either an ordered or unordered target tree. The five variations are as follows:

Tree inclusion A pattern tree P is an included tree of target tree T, if there is a label-preserving mapping f from the nodes of P to nodes of T such that for all nodes $q_i, q_j \in P, q_i$ is the parent of $q_j \Longleftrightarrow f(q_i)$ is an ancestor of $f(q_j)$;

Path inclusion A pattern tree P is a path-included tree of target tree T, if there is a label-preserving mapping f from the nodes of P to nodes of T such that for all nodes $q_i, q_j \in P, q_i$ is the parent of $q_j \Longleftrightarrow f(q_i)$ is the parent of $f(q_j)$;

Region inclusion A pattern tree P is a region-included tree of target tree T, if (1) P is a path-included tree of T,(2) for every pair of matching nodes $f(q_i), f(q_j) \in T$, $f(q_i)$ is a left sibling of $f(q_j)$, all nodes of T that are right siblings of $f(q_i)$ and left siblings of $f(q_j)$ are matching nodes;

Child inclusion A tree P is a child-included tree of target tree T,if (1) P is a path-included tree of T, (2)whenever a node $q \in P$ has k children, $k > 0$, $f(q)$ has also k children;

Subtree inclusion : A tree P is a subtree of target tree T, if P is isomorphic to a subtree of T;

Kilpelainen[12] also gave the complexity of these inclusion problems as shown in Table 2 where m is the size of the pattern tree and n is the size of the target tree.

	tree-inclusion	path-inclusion	region-inclusion	child-inclusion	subtree-inclusion
unordered	NP-complete	$O(m^{1.5}n)$	$O(m^2n)$	$O(m^{1.5}n)$	$O(n)$
ordered	$O(mn)$	$O(mn)$	$O(mn)$	$O(n\sqrt{m}polylog(m))$	$O(n)$

Table 2. Complexity of tree inclusion problems

The above mentioned variations of tree inclusion are all based on label-preserving mapping, that is, the label of nodes in pattern P are the same to the label of their matching nodes in target tree T. Amer-Yahia[1] proposed the concept of *node generalization* of tree patterns. *Node generalization* permits the label of a node to be generalized to a super-type. Suppose there exist a knowledge hierarchy in which "publication" is a super-type of "paper". The label "paper" could be generalized to "publication" so as to produce more matching instances. It is useful in classifications when no frequent pattern can be extracted from training set of the same class using label-preserving mapping.

The solution of *tree inclusion* problem provides a foundation for frequent tree pattern mining. Termier *et al.*[21] introduced an algorithm called *treefinder* to find frequent trees from a collection of trees under three variations of tree inclusion: *subtree inclusion*, *inclusion by tree embedding*(equivalent to the *tree-inclusion* in[12]), and *inclusion by tree subsumption*. A data tree d is said to include a tree pattern p by *subsumption* if there is a mapping f from the nodes of p into the nodes of d such that for all nodes $q_i, q_j \in p, q_i$ is the parent of $q_j \Longrightarrow f(q_i)$ is an ancestor of $f(q_j)$. *Treefinder* is not guaranteed to find all frequent trees. The useful feature of this algorithm is that it is a two-step algorithm, which raises the possibility that this task could be decomposed into two sub-tasks. Zaki's[23] *treeminer* algorithm tried to discover all frequent tree *embedded* (equivalent to the *tree-inclusion* in[12]) in a collection of ordered trees. This algorithm could be applied to unordered trees with slight modification. *Treeminer* is based on the property of frequent patterns: a super-pattern is less frequent or as frequent as a sub-pattern. There are two steps of this algorithm: 1) enumerate candidate sub-trees of size k and 2) count the frequency of these sub-trees. The sub-tree of size $k+1$ is generated from the sub-trees of size k that have frequency larger than the threshold. Obviously, we can design an operator for sub-tree enumeration and an operator for frequency counting.

Frequent pattern is a kind of knowledge derived directly from raw data. It is used to classify XML documents. Zaki and Aggarwal[24] proposed the concept of *structural rule* which is denoted as $T \Longrightarrow c_i$, where T is a frequent tree pattern and c_i is one of the classes. A structural rule $T \Longrightarrow c_i$ implies that the presence of pattern T in an XML document is related to its likelihood of belonging to class c_i. Usually a classification task can be divided into two phases: training phase

and testing phase. In the training phase, structural rules are mined from the classified training data. In the testing phase, the input XML document is classified according to its matching structural rules.

Classification task consists of the following sub-tasks: 1) the frequent tree patterns are mined from training documents, it could be fulfilled by the operators just discussed above; 2) predictive structural rules are generated based on the frequent pattern and class labels of the training documents; 3) all the matching patterns (antecedents of structural rules) are retrieved from the test document, which is the *tree inclusion* problem we discussed in previous section; 4) the test document is classified by combining the statistics from each matching rules. Since an XML document may match more than one structural rule, we need a formula to put these rules together to predict its class. It could be performed by the operator that receives a user-defined function as parameter. This function defines the strategy on how to use the structural rules.

The problem of clustering has been extensively studied by many researchers. Most research focus on clustering text documents where no internal structure is considered. Two important factors of clustering are *distance metric* and *clustering algorithm* . In XML clustering, the clustering algorithms for text document could still be used, but the distance metric must be defined on the XML data modelled as trees. Generally, there are two kind of methods to define the distance metric for XML documents[13]:

1. Set Based Metrics: each XML tree is decomposed into a set of objects (edge, path or prefix of path) and two such sets are compared to compute the distance between two XML trees.

2. Tree Edit Distance Based Metrics: edit distance is the minimum cost of transforming tree A to tree B using a pre-defined set of edit operations, where each operation being associated with a cost.

Usually the edit distance metric performs better than set based metrics in capturing the structure of XML trees. Unfortunately, the general tree edit distance problem for unordered trees are NP-hard.[25] Zhang and Shasha[25] gave an $O(n^3 \log n)$ algorithm for computing tree edit distance between ordered trees, using operations of single-node insertion and deletion. Nierman and Jagadish[15] expanded the set of operations by adding sub-tree insertion and deletion and associated different cost to operations. No matter what operations are used, the clustering task is composed of two sub-tasks: 1) compute the distance between XML documents; 2) apply the clustering algorithm. The resulting clusters could be a partition or a covering of the input data, depending on the algorithm. Therefore, we need at least three operators, one for distance computation, the other two for partition-clustering and covering -clustering.

6. SUMMARY

The ultimate goal of XML algebras is to provide operational procedure to obtain answers of user's queries. XAL is simple and could be implemented easily. Niagara extends the definition of basic unit from one vertex to bag of vertices. TAX is an algebra managing collections of tree directly. These XML algebras are good starting point to design algebras for XML mining. Extension of the existing algebras by adding some operators for mining tasks might work well. The intuitive way to design one operator for each type of mining task is inefficient in execution and leaves little room for optimization. A better solution is to design one operator for each sub-task whose operation is much simpler. How to decompose the mining tasks into sub-tasks is still an open problem, and it is largely dependent on the nature of the mining tasks. We propose the guidelines to define the mining tasks: mining tasks are defined at different abstract levels. Any type of knowledge could be generated from knowledge at lower levels. Mining operators may include those performing the basic operations for mining tasks, such as counting the number of given pattern, computing the similarity/distance between patterns, generating patterns within the given size, etc.

REFERENCES

1. S. Amer-Yahia, S. Cho, and D. Srivastava. Tree pattern relaxation. *International Conference on Extending Database Technology (EDBT)*, 2002.
2. S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J.L. Wiener. The Lorel query language for semistructured data. International Journal on Digital Libraries, 1(1):68-88, April 1997.
3. D. Chamberlin, J. Robie, and D. Florescu. Quilt: An XML query language for heterogeneous data sources. In international Workshop on the Web and Database (WebDB'2000), May 2000.

4. A. Deutsch, M. Fernandez, D. Florescu, A. Levy and D. Suciu. A Query Language for XML. In proceedings of 8th International World Wide Web Conference, pp. 77-91, May 1999.
5. F. Frasincar, Geert-Jan Houben, and C. Pau. XAL: an Algebra for XML Query Optimization. In proceedings of 13th Australasian Database Conference (ADC2002)
6. L. Galanis, et al., Following the paths of XML Data: An algebraic framework for XML query evaluation. Technical report, University of Wisconsin, 2001. available at http://www.cs.wisc.edu/niagara/papers/algebra.pdf
7. J. Han, Y. Fu, K. Koperski, W. Wang, and O. Zaane, " DMQL: A Data Mining Query Language for Relational Databases", 1996 SIGMOD'96 Workshop. on Research Issues on Data Mining and Knowledge Discovery (DMKD'96), pp. 27-33, Montreal, Canada, June 1996.
8. T. Imielinski and H. Mannila. A Database Perspective on Knowledge Discovery. Communications of the ACM Vol. 39, No. 11 pp.58-64,1996
9. H.V. Jagadish, et.al. TIMBER: A native XML database. The VLDB journal, 11(4):274-291, 2002
10. T. Johnson, L.V.S. Lashmanan and Raymond T.Ng. The 3W Model and Algebra for Unified Data Mining. Proceedings of the 26th VLDB Conference, Cairo, Egypt, 2000
11. H.V. Jagadish, L.V.S. Lakshmanan, D. Srivastava and K.Thompson. TAX: A Tree Algebra for XML. In Proc. DBPL Conf. pp. 149-164. Rome, Italy, 2001.
12. P. Kilpelainen. Tree Matching Problems with Applications to Structured Text Databases. PhD thesis, University of Helsinki, 1992.
13. D. Lee. Query Relaxation for XML Model. PhD thesis. University of California, Los Angeles, 2002.
14. J. Naughton, D. DeWitt, D. Maier, et al. The Niagara Internet Query System. IEEE Data Eng. Bulletin, 24(2).
15. A. Nierman and H.V. Jagadish. Evaluating Structural Similarity in XML Documents. *Proceedings of the Fifth International Workshop on the Web and Databases* (WebDB 2002)
16. R. Nayak, R. Witt and A. Tonev. Data Mining and XML documents. Proceedings of the 2002 International Conference on Internet Computing, June 24-27, Nevada, USA, 2002
17. R. Ramakrishnan, J. Gehrke. Database Management Systems. 2000
18. E. Schallehn, K. Sattler, and G. Saake. Extensible and similarity-based grouping for data integration Poster paper.In 8th Int. Conf. on Data Engineering (ICDE), San Jose, CA, 2002
19. J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In Proc. VLDB Conf. pp. 302-314, Edinburgh, Scotland, 1999
20. A. Schmidt, M. Kersten, M. Windhouwer, F. Waas. Efficient Relational Storage and Retrieval of XML Documents. In proc. of Intl. Workshop on the Web and Databases, pp. 47-52, 2000
21. A. Termier, M-C. Rousset, M. Sebag. TreeFinder: a First Step towards XML Data Mining. *IEEE Int'l Conf. on Data Mining*, 2002.
22. W3C Recommendation. XQuery 1.0: An XML Query Language. In http://www.w3.org/TR/xquery.
23. M.J. Zaki. Efficiently Mining Frequent Trees in a Forest. *SIGKDD*, 2002.
24. M.J. Zaki, C.C.Aggarwal. XRules: An Effective Structural Classifier for XML Data. *SIGKDD*, 2003.
25. K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal of Computing*,18(6):1245-1262, 1989