Discovering Frequently Changing Structures from Historical Structural Deltas of Unordered XML

Qiankun Zhao[†] Sourav S Bhowmick[†] Mukesh Mohania[‡] Yahiko Kambayashi[§]
[†]CAIS, Nanyang Technological University, Singapore. {*pg04327224, assourav*}*@ntu.edu.sg*
[‡]IBM India Research Lab, India. *mkmukesh@in.ibm.com*
[§]Department of Social Informatics, Kyoto University, Japan.

ABSTRACT

Recently, a large amount of work has been done in XML data mining. However, we observed that most of the existing works focus on the snapshot XML data, while XML data is dynamic in real applications. To the best of our knowledge, none of the existing works has addressed the issue of mining the history of changes to XML documents. Such mining results can be useful in many applications such as XML change detection, XML indexing, association rule mining, and classification etc. In this paper, we propose a novel approach to discover the *frequently changing structures* from the sequence of historical *structural deltas* of unordered XML. To make the structure discovering process efficient, an expressive and compact data model, **H**istorical-**D**ocument **O**bject **M**odel (**H-DOM**), is proposed. Using this model, two basic algorithms, which can discover all the *frequently changing structures* with only two scans of the XML sequence, are presented. Experimental results show that our algorithms, together with the optimization techniques, are efficient and scalable.

Categories and Subject Descriptors: H.2.8 [Database Management]: Database Applications – *Data Mining*.

General Terms: Algorithm, Design, Experimentation.

Keywords: XML, data mining.

1. INTRODUCTION

Recently, there has been increasing research efforts to mine XML data. Existing work on mining XML data includes frequent substructure mining [6, 7, 9, 10], classification [11] and association rule mining [1]. Among these, the frequent substructure mining is the most well researched topic. The basic idea is to extract substructures (subtrees or subgraphs), which occur frequently among a set of XML documents or within an individual XML document. Frequent substructures have been used for efficient querying and classification of XML documents [11].

1.1 Motivation

CIKM'04, November 8–13, 2004, Washington, DC, USA.

Existing research on frequent substructure mining, such as AGM [6], FSG [7], TreeMiner [10], gSpan [9], focus on discovering frequently occurred substructures from structural data collections. For example, suppose there is a collection of XML documents that describe the detail information about university professors. Figures 1(a) and (d) are the tree representations of two XML documents. By applying existing state-of-the-art structure mining techniques, frequent substructures among them can be discovered. For example, by applying the gSpan [9] mining approach, the structures shown in Figures 1 (b) and (c) will be returned as frequent substructures mining results.

However, all the existing approaches of XML mining focus only on snapshot XML data, while in real life XML data is dynamic. That is, XML data may change at any time in different ways. For example, consider document 2 in Figure 1 (d), the *publication* and *activity* of a professor may change over time. Figures 1 (e), (f), (g) are the tree representations of three versions of document 2. The black circles represent for the newly inserted nodes (elements/attributes). The gray circles denote for the deleted nodes. The bold circles are nodes whose contents have been updated. Insertions and deletions of nodes are considered as *structural changes* while content changes refer to updates. To save space, only structural changes are presented. We use C_i, J_i, and P_i to represent the *Conference, Journal,* and *Paper* element whose attribute *id* is i. The dynamic nature of XML leads to two challenging problems. First, is the maintenance of frequent substructures. As the data source changes, new frequent structures may be added and some old ones may not be frequent any more. Second, is the discovery of novel knowledge hidden behind the historical changes to XML data, some of which is difficult or impossible to be discovered from snapshot data. In this paper, we focus on discovering novel hidden knowledge from the historical changes to XML data. Consider the four different versions of XML documents in Figure 1, following novel knowledge can be discovered. Notice that, this list is by no means exhaustive.

- *Frequently changing structure/content*: The structure rooted at *XML* and the content of *Activity* changed more frequently, while the structure and content rooted at *Bio* never changed in the history.
- *Web delta association*: Whenever the structure rooted at *Publication* changes, structure *Activity* also changes.
- *Change patterns*: More and more *paper*s are inserted under the root of *XML* while nodes under *CFP* are frequently deleted and inserted.

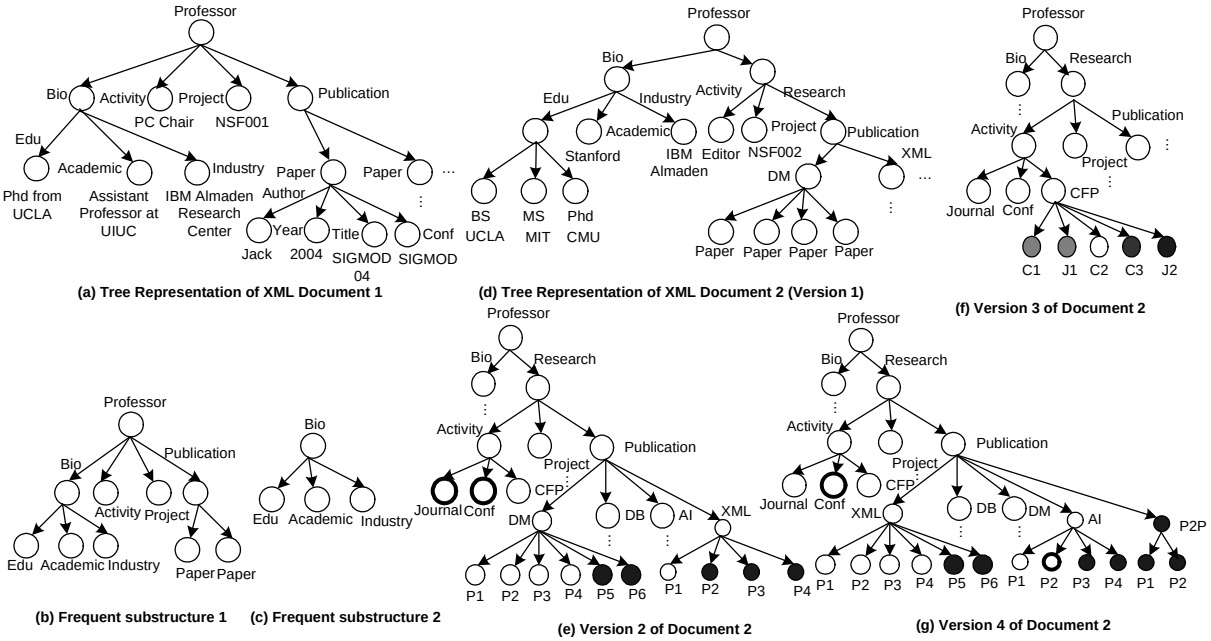


Figure 1: An Example

1.2 Applications

Such knowledge can be useful in many applications such as association rule mining for changes of XML, XML change detection, and web usage mining etc. We use the following applications as representative of the potential wide range of applications.

1.2.1 Association Rule Mining:

Given a set of the Frequently Changing Structures (FCSs), the sequence of structural deltas can be represented as transactions of FCSs. Consequently, association rules among the FCSs can be discovered by treating each delta as a transaction. Considering the structural data shown in Figure 1, suppose the set of structures rooted at nodes *DM, XML, P2P, AI* and *CFP* are FCSs, then the sequence of structural deltas can be transformed into three transactions, {*DM, XML*}, {*CFP*}, and {*XML, AI, P2P*}. By applying the existing association rule techniques [5], the association among structural changes can be discovered. Such association rules can be very useful for monitoring and predicting the underlying change trends of web sites in e-commerce, monitoring the change pattern of users' navigation behavior etc. This issue has been addressed in our previous work [2].

1.2.2 Web Usage Mining:

Recently, a lot of work has been done in web usage mining. However, most of the existing works focus on snapshot web usage data, while usage data is dynamic in real life. Knowledge hidden behind historical changes of web usage data, which reflects how web access patterns (WAP) change, is critical to adaptive web, web site maintenance, business intelligence etc. The web usage data can be considered as a set of trees, which have the similar structures as XML documents. By partitioning web usage data according to the user-defined calendar pattern, we can obtain a sequence of changes from the historical web access patterns. From the changes, useful knowledge such as how certain web access patterns changed, which parts changes more frequently and which parts do not, can be extracted. Some preliminary results of mining the changes to historical web access patterns have been shown in [12].

1.2.3 XML Change Detection:

One of the major limitations of existing XML change detection systems [3, 8] is that they are not scalable for very large XML documents. With the set of web delta association rules and frozen structures extracted from historical changes, the scalability of XML change detection system can be improved. Suppose, we have a association rule which states that when part A of the XML document changes most of the time part B will not change. Then we can skip part B of the document for change detection when part A changes. Similarly, the frozen structures are not expected to change or seldom change. Hence most of time we can eliminate the corresponding parts of the XML document for change detection. With such knowledge, rather than loading the entire XML documents, only parts of the documents that are likely to change are loaded and compared. We believe that this will improve the efficiency and scalability of change detection process, especially for *very large* XML documents.

1.3 Overview

From the above list of knowledge, we may observe that the core foundation of such knowledge is the frequently changing structure/content, from which knowledge are discovered. In this paper, we focus on discovering the Frequently Changing Structure (FCS) since there are many applications where data are described in a more structural way, e.g. chemical compounds, biological data, computer network, and web browsing history [9]. Also, in archive-based applications such as the SIGMOD record and DBLP XML documents, where content update is rare, the structural information is also critical. In this paper, we deal with unordered XML documents since the unordered model of XML is more suit-

able for most database applications [8]. Hereafter, whenever we say XML, we mean unordered XML.

However, the existing state-of-the-art XML mining techniques [9] fail to extract the frequently changing structure. Even if we apply such techniques repeatedly to a sequence of snapshots of XML data, they cannot discover such knowledge accurately and efficiently. Suppose there are n versions of XML collections denoted as X_1, X_2, $\cdots$, X_n. We take the gSpan [9] algorithm as an example. For each version, gSpan is applied and the sets of frequent structure mining results are denoted as M_1, M_2, $\cdots$, M_n. By postprocessing the sequence of mining results, we may find the sets of structures I and J, where $I = M_1 \cap M_2 \cap \cdots M_n$ is the set of structures that are frequent over all the time points from 1 to n; $J = M_p - M_q$ $(1 \leq q < p \leq n)$ is the set of structure that is frequent at time point p but not frequent at time point q. However, such structures may not reflect their change patterns and frequencies accurately. For example, structures in I may have changed or may not have changed, which may depend on the changes of other structures in the data collections. Also, it is possible that some of them may have been deleted from one position/document and inserted into another position/document. Similarly, structures in J may have changed or may not have changed. It may be the result of other changes such as changes to the total number of transactions and the minimal support threshold etc. Moreover, such mining and postprocessing processes can be very expensive. This example shows that even by applying existing XML mining techniques repeatedly, the above list of knowledge cannot be discovered accurately and efficiently.

Given a sequence of XML documents, which are different versions of the same XML document, FCS mining is to discover all the substructures that change *frequently* and *significantly* in the sequence of XML structural deltas.

We propose two algorithms to extract the frequently changing structure from the XML structural deltas. There are three major phases: *H-DOM construction* phase, *FCS extraction* phase, and the *visualization* phase. In the first phase, given a sequence of historical XML documents, the structural deltas are calculated and stored in the H-DOM (Historical Document Object Model). The second phase is to use data mining techniques to extract the frequently changing structures by traversing the H-DOM tree. Lastly, in the visualization phase, the frequently changing structures are presented within the tree representation so that it can be easily interpreted.

Our experiment results show that our algorithms can successfully extract all the frequently changing structures efficiently. Also, the H-DOM tree is very compact, its size is around 50% of the original size of the XML sequence. Moreover, it has been observed that the space optimization techniques can make the H-DOM tree more compact by around 20% and consequently make our algorithms more scalable, which can handle 450Mb XML sequence.

The major contributions of this paper can be summarized as follows.

1. We propose a novel approach, to the best of our knowledge, to discover hidden knowledge from the sequence of historical structural deltas of XML documents.

2. Two algorithms for mining frequently changing structure, with three optimization techniques, have been proposed and implemented.

3. We present a list of applications where the frequently changing structure mining results can be useful.

4. We conduct extensive experiments by using different datasets and varying the parameters to show the performances our algorithms and optimization techniques.

The rest of the paper is organized as follows. In Section 2, we present some related work to our research. Section 3 presents the background of FCS mining. The H-DOM model and the FCS mining algorithms, together with different optimization techniques, are described in Section 4. The detailed experimental results are reported in Section 5. Section 6 concludes this paper.

2. RELATED WORK

Since XML data is semi-structured and widely used, data mining of semi-structured data has attracted many research efforts recently [6, 7, 9, 10]. Most existing work focus on discovering the frequent substructures from a collection of semi-structured data such as XML documents. AGM [6] is an Apriori-based algorithm for mining frequent substructures. But the results of AGM is restricted to only the *induced* substructures. FSG [7] is also an Apriori-based algorithm for mining all *connected* frequent subgraphs. Experiments results in [7] show that FSG is considerably faster than AGM. However, both AGM and FSG do not scale to very large database. gSpan [9] is an algorithm for extracting frequent subgraphs without candidate generation. It employs the depth-first search strategy over the graph database. Like AGM, gSpan needs elaborate computations to deal with structures with non-canonical forms. More recently, TreeMiner [10] is proposed to discover the frequent *embedded* substructure, which is a generalization of induced substructure. But the TreeMiner does not scale to very large XML documents either. The discovered frequent substructures are further explored to improve other applications such as XML querying and XML classification [11] etc.

Different from the above techniques, which focus on designing ad-hoc algorithms to extract structures that *occur frequently* in the snapshot data collections, FCS mining is to extract structures that *change frequently* from the sequence of historical XML versions.

Considering the dynamic nature of XML data, many efforts have been directed into the research of change detection for XML data. Different techniques have been proposed [3, 8, 4]. XML TreeDiff [4] computes the difference between two XML documents using hash values and simple tree comparison algorithm. XyDiff [3] is proposed to detect changes of ordered XML documents. Besides *insertion, deletion,* and *updating*, XyDiff also support *move* operation. X-Diff [8] is designed to detect changes of unordered XML documents. In our FCS mining, we extend the XML change detection technique in [8] to discover hidden knowledge from the history of changes to unordered XML data with data mining techniques.

3. BACKGROUND

In this section, we present the background knowledge on *types of structural changes for XML data* and *dynamic metrics*. We also give the problem statement of *frequently changing structure mining*.

3.1 Types of XML Structural Changes

The structure of XML document can be modelled as a tree structure according to the Document Object Model (DOM) specification. In this section, we present different types of structural changes of XML documents in their tree representations. Traditionally, all changes of XML documents can be represented by five types of edit operations [8]. The first three are basic operations and the last two are composite operations that can be represented as a list of basic operations.

- *Insert*(x(*name*, *value*), y): insert a node x, with node name *name* and node value *value*, as a leaf child node of node y.
- *Delete*(x): delete a leaf node x.
- *Update*(x, *new_value*): change the value of a leaf node x to *new_value*. Note that only the value can be updated, but not its name.
- *Insert*(T_x,y): insert a subtree T_x, which is rooted at x, to node y.
- *Delete*(T_x): delete a subtree T_x, which is rooted at node x.

Based on the edit operations, an *edit script* is defined as a sequence of edit operations that transform an XML document from one version to another [8]. However, not all the edit operations can change the structure of the XML documents. For example, the *Update* operation will not change the structure of a document. Corresponding to the structural changes, we define the *structural edit script* as a sequence of edit operations that convert one structure to another. It is similar to the *edit script* except that all *Update* operations are excluded in the *structural edit script*. For example, the structural edit script of the structure in Figures 1 (e) and 1 (f) from version 2 to version 3 is $\langle$ Delete(C_1), Delete(J_1), Insert($C_3(v_1)$, *CFP*), Insert($J_2(v_2)$, *CFP*)$\rangle$. To make it easier to locate the edit operation, an *affiliated node* is defined for each edit operation. For all the insertion operations, their *affiliated nodes* are nodes y; for all the deletion and updating operations, their *affiliated nodes* are x.

3.2 Definitions

We model the structures of XML documents as unordered, labeled, rooted tree structures. We denote the structure of an XML document as $S = (N, E, r)$, where N is the set of labeled nodes, E is the set of edges, $r \in N$ is the root. We do not distinguish between elements and attributes, both of them are mapped to the set of labeled nodes. Each edge, $e = (x, y)$ is an ordered pair of nodes, where x is the parent of y. The *size* of the structure S, denoted by $|S|$, is the number of nodes in N.

DEFINITION *1.* [**Substructure**] A structure $S' = (N', E', r')$ is a *substructure* of $S = (N, E, r)$, denoted as $S' \preceq S$, provided *i*) $N' \subseteq N$, and *ii*) $e = (x, y) \in E'$, if and only if x is the parent of y in E.

If $S' \preceq S$, we also say that S *contains* S', and S is the *superstructure* of S'. For example, according to this definition, the structure in Figure 1 (b) is a *substructure* of the structure in Figure 1(a) but not a *substructure* of the structure in Figure 1(d).

DEFINITION *2.* [**Structural Delta**] Let S_i and S_{i+1} be the tree representations of two XML documents X_i and X_{i+1}. The structural delta from X_i to X_{i+1} is represented as $\triangle_i$, where $\triangle_i$ is a structural edit script $\langle o_1, o_2, \cdots, o_m\rangle$ that transforms S_i into S_{i+1}, denoted as $S_i \overset{o_1}{\rightarrow}\overset{o_2}{\rightarrow} \cdots \overset{o_m}{\rightarrow} S_{i+1}$.

The size of the structural delta, denoted as $|\triangle_i|$, is defined as the number of basic edit operations in the structural edit script. Consider the previous example, the structural delta from version 2 to version 3 is $\triangle_2$=$\langle$Delete(C_1), Delete(J_1), Insert($C_3(v_1)$, *CFP*), Insert($J_2(v_2)$, *CFP*)$\rangle$ and the value of $|\triangle_2|$ is 4 since there are 4 basic edit operations shown as colored circles in Figure 1 (f).

DEFINITION *3.* [**Structural Delta of Substructures**] Let $\langle\triangle_1, \triangle_2, \cdots, \triangle_i\rangle$ be a sequence of structural delta for an XML document X, whose tree representation is S. Suppose $s = (N_s, E_s, r_s)$ is a substructure of S. The sequence of structural delta for s is denoted as $\langle\triangle_{s_1}, \triangle_{s_2}, \cdots, \triangle_{s_i}\rangle$, where $\triangle_{s_i} \subseteq \triangle_i$ and for all operation $o_j \in \triangle_{s_i}$, their affiliated nodes should be in N_s.

Reconsider the examples in Figure 1. For the substructure rooted at node *Activity*, the corresponding of structural delta is $\langle\ \triangle_1, \triangle_2, \triangle_3\ \rangle$ where $\triangle_1$ and $\triangle_3$ are empty, $\triangle_2$ is the same as in the structural delta example.

DEFINITION *4.* [**Consolidate Structure**] Given two structures s_i and s_j, where $r_i = r_j$. The *consolidate structure* of them is denoted as $s_i \uplus s_j$, where *i*)$N_{s_i \uplus s_j} = N_{s_i} \cup N_{s_j}$, *ii*) $e = (x, y) \in E_{s_i \uplus s_j}$, if and only if x is the parent of y in E_{s_i} or E_{s_j}.

Consider the structures in Figure 1. For the substructures rooted at node *Bio* in Figure 1 (a) and (d), the consolidate structure is the structure rooted at node *Bio* in Figure 1 (d).

From the example in Figure 1, we observed that different substructures of the XML document might change in different ways at different frequencies. To evaluate the historical behavior of different substructures, we propose a set of dynamic metrics.

DEFINITION *5.* [**Structure Dynamic**] Let $\langle S_i, S_{i+1}\rangle$ be the tree representations of XML documents $\langle X_i, X_{i+1}\rangle$. Suppose $s \preceq S_i$. The *structure dynamic* of s from document X_i to document X_{i+1}, denoted by $N_i(s)$, is defined as: $N_i(s) = \frac{|\triangle_{s_i}|}{|s_i \uplus s_{i+1}|}$.

Here $N_i(s)$ is the structural dynamic of s from version i to $i+1$. By using the consolidation structure, the total number of unique nodes in the two versions can be obtained as $|s_i \uplus s_{i+1}|$. It includes not only nodes that are in version $i+1$ but also nodes that have been deleted in version i. $N_i(s)$ is the percentage of nodes that have changed from X_i to X_{i+1} in s against the number of nodes in its consolidation structure. For example, consider the two structures shown in Figures 1 (d) and 1 (e). We calculate the structure dynamic value for the substructure rooted at node *DM* from version 1 to version 2. Based on the definition, $|\triangle_{DM_1}| = 2$, $|DM_1 \uplus DM_2| = 6$. Consequently, $N_1(DM) = 0.33$ (2/6). It also can be observed that $N_i(s) \in [0, 1]$. If s is inserted or deleted, then the corresponding value of structure dynamic is 1 since $\triangle_{s_i} = s_i \uplus s_{i+1} = s$. If s did not change from version i to version $i+1$, then the value of structure dynamic

is 0 since $|\triangle_{s_i}|$ is 0. It can be implied that larger the value of structure dynamic is, more significantly the substructure changed.

DEFINITION 6. [**Version Dynamic**] Let $\langle S_1, S_2, \cdots, S_n \rangle$ be the tree representations of XML documents $\langle X_1, X_2, \cdots X_n \rangle$. Suppose $s \preceq S_j$. The *version dynamic* of s, denoted as $V(s)$, is defined as:

$$V(s) = \frac{\sum_{i=1}^{n} v_i}{n-1} \text{ where } v_i = \begin{cases} 1, & \text{if } |\triangle_{s_i}| \neq 0; \\ 0, & \text{if } |\triangle_{s_i}| = 0; \end{cases}$$

Consider the 4 different versions of the XML document in Figure 1. We calculate the version dynamic value for the substructure rooted at node *XML*. The n value here is 4. For the first delta, $|\triangle_{XML_1}|$ is not 0, so $v_1 = 1$. Similarly, $v_2 = 0$, $v_3 = 1$. Then, $\sum_{i=1}^{3} v_i = 2$. Consequently, the version dynamic of this substructure is 0.67 (2/3). It can be observed that $V(s) \in [0, 1]$. If s changed in every version in the history, then the corresponding value of $\sum_{i=1}^{n} v_i$ is $n-1$, so the version dynamic value is 1. If s did not change in the history at all, then the value of $\sum_{i=1}^{n} v_i$ is 0 and version dynamic value is 0. Also, it implies that larger the value of version dynamic is, more frequently the substructure changed in the history.

A major difference between $N_i(S)$ and $V(S)$ is that $V(S)$ measures the overall changes over the history while $N_i(S)$ measures the local changes between two consecutive versions. Thus, for a substructure there is one value of version dynamic and a sequence of values for structure dynamic. To measure the overall change behavior of a substructure in terms of both significance and frequency, we proposed another dynamic metric named *degree of dynamic*, denoted as *DoD*. *DoD* is the extension of structure dynamic by incorporating the version dynamic metric. It represents the overall significance of the structural changes in the history.

DEFINITION 7. [**Degree of Dynamic**] Let $\langle S_1, S_2, \cdots, S_n \rangle$ be the tree representations of XML documents $\langle X_1, X_2, \cdots, X_n \rangle$. Suppose $s \preceq S_j$, $N_i(s)$ and $V(s)$ are the values of structure dynamic and version dynamic of s. The degree of dynamic, *DoD*, for s is defined as:

$$DoD(s, \alpha) = \frac{\sum_{i=1}^{n} d_i}{(n-1) * V} \text{ where } d_i = \begin{cases} 1, if\ N_i \geq \alpha \\ 0, if\ N_i < \alpha \end{cases}$$

where α is the pre-defined threshold for *structure dynamic*.

The metric *DoD* is defined based on the threshold of structure dynamic. It represents the fraction of versions, where the structure dynamic values for the substructure are no less than the predefined threshold α, against the total number of version the substructure has changed over the history. Consider the examples shown in Figure 1. We can calculate the *DoD* value for the substructure rooted at node *XML*. From the previous examples, we know that the structure dynamic values of this substructure are 0.75, 0, and 0.33. The version dynamic value is 0.67. Suppose the threshold for structure dynamic is set to 0.30, then the value of *DoD* is 1 (2/2). If the threshold for structure dynamic is set to 0.35, then the corresponding *DoD* value will be 0.5 (1/2). It is obvious that, $\forall\ \alpha, DoD(s, \alpha) \in [0, 1]$. Extended from the structure dynamic, the value of *DoD* also implies the overall significance of the substructure, larger the value is, more significant the changes are.

3.3 Problem Statement

The problem of frequently changing structure mining is to discover those structures that changed significantly and frequently in the history. Based on the above set of metrics, the frequently changing structure is defined as follows.

DEFINITION 8. [**Frequently Changing Structure**] Let $\langle S_1, S_2, \cdots, S_n \rangle$ be the tree representations of XML documents $\langle X_1, X_2, \cdots, X_n \rangle$. The thresholds for structure dynamic, version dynamic, and degree of dynamic are α, β, γ respectively. A structure $s \preceq S_j$ is a **frequently changing structure (FCS)** in this sequence iff: $V(s) \geq \beta$ and $DoD(s, \alpha) \geq \gamma$.

The FCS is defined based on the predefined thresholds of the dynamic metrics. The significance of changes are defined by structure dynamic α and degree of dynamic γ, while the frequency of changes are defined by version dynamic β.

EXAMPLE 1. Consider the versions of XML shown in Figure 1, an example of the frequently changing structure will be the structure rooted at node *XML* as shown in Figure 1(d). This structure may indicate that the corresponding professor is very active in the research area of *XML*.

4. ALGORITHMS

In this section, we present the algorithms for discovering the frequently changing structures. First, a data model for representing and storing the history of structural delta is proposed. Next, we present an overview of the algorithms followed by two algorithms for FCS mining. We identify some limitations of the two algorithms and present a set of optimization techniques.

4.1 H-DOM Model

The structure of an XML document can be represented and stored as a tree such as the DOM tree proposed by W3C. However, in our frequently changing structure mining problem, given a sequence of history XML documents, it is not efficient to store them in a sequence of DOM trees. We present an H-DOM model to represent the history of changes to XML data. The H-DOM is an extension of the DOM model with some historical properties so that it can compress the history of changes to XML into a single H-DOM tree. It is a general model to store and representing historical changes to XML data including both the structure and content such as: *insert*, *delete*, and *update*. Formally, we define an H-DOM tree as follows:

DEFINITION 9. [**H-DOM**] An H-DOM tree is a 4-tuple $H = (N, A, v, r)$, where N is a set of object identifiers; A is a set of labelled, directed arcs (p, l, c) where $p, c \in N$ and l is a string; v is a function that maps each node $n \in N$ to a set of values (C_n, C_v), C_n is an integer and C_v is a binary string; r is a distinguished node in N called the root.

We now elaborate on the parameters C_n and C_v. The two parameters are introduced to record the historical changes for each substructure. C_n is an integer that records the number of versions that a substructure has changed significantly enough (the structure dynamic is no less the corresponding threshold). C_v is a binary string that represents the historical changes of a substructure. The length of the string is equal to the number of deltas in the XML sequence. The ith

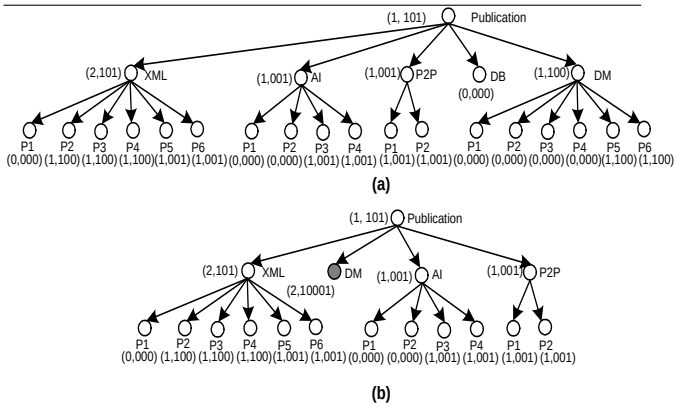


Figure 2: Part of the H-DOM Tree

digit of the string denotes the change status of the structure from X_i to X_{i+1}, where the value of 1 means this structure changed, the value of 0 means it did not change. The types of changes are not specified in C_v since we focus on the frequency and significance of the changes. In the H-DOM tree, the C_v value for each structure is finally updated by using the formula: $C_v(s) = C_v(s_1) \vee C_v(s_2) \vee \cdots \vee C_v(s_j)$, where $s_1, s_2, \cdots, s_j$ are the substructures of s.

Figure 2 (a) is part of the H-DOM for the structure sequence in Figure 1. Suppose the threshold for structure dynamic is 0.30, the C_n value of node *XML* is 2, which means that this structure has changed twice in the history with a structure dynamic value no less than 0.30. The C_v value 100 of node p_3 means that this node has changed from X_1 to X_2. The C_v value of the internal nodes and root node are calculated according to the above formula. With C_v and C_n, values of the dynamic metrics can be calculated as follows.

- $N_i(s) = \frac{\sum C_v(s_j)[i]}{|s_{t_i} \uplus s_{t_i+1}|}$, where s_j is the list of descendant nodes of s, $C_v(s_j)[i]$ is the ith digit of $C_v(s_j)$.
- $V(s) = \frac{\sum_{i=1}^{n-1} C_v[i]}{n-1}$, where $C_v[i]$ is the ith digit of $C_v(s)$; n is the total number of XML documents.
- $DoD(s) = \frac{C_n}{\sum_{i=1}^{n-1} C_v[i]}$, where $C_v[i]$ is the ith digit of $C_v(s)$; n is the total number of XML documents.

The H-DOM model is inspired by the FP-Tree in association rule mining [5]. It is designed to preserve and compress the historical structural information of XML versions. H-DOM compresses the historical structural data by representing the identical nodes only once in the H-DOM tree, while the related historical information is preserved using a binary string and an integer. Compared to the FP-Tree, the compactness of H-DOM should be higher since the same nodes may appear more than once with *h-links* in the FP-tree. Moreover, the FCSs can be extracted without any candidate generation process by traversing the H-DOM exactly once, while in FP-Tree there is an conditional FP-Tree generation process. Another feature of the H-DOM tree is that it expresses the temporal features of the XML structures.

4.2 FCS Algorithm

There are three phases in our FCS mining. The *H-DOM construction* phase, the *FCS extraction* phase, and the *visualization* phase. Given a sequence of historical XML documents, an H-DOM tree will be constructed in the first phase. Rather than store each structural delta into XML files as in the original X-Diff algorithm, we integrate the change detection and mapping processes. The second phase is to extract FCS from the H-DOM tree. Finally, the mining results are displayed to users in the visualization phase. In this section, we focus on H-DOM construction and the FCS extraction since the visualization phase is quite straightforward.

4.2.1 H-DOM Construction:

Algorithm 1 in Figure 3 describes the process of H-DOM construction. Given a sequence of historical XML documents, the H-DOM tree is initialized as the structure of the first version. After that, the algorithm iterates over all the other versions by extracting the structural deltas and mapping them into the H-DOM tree. The SX-Diff function is a modification of the X-Diff [8] algorithm that generates only the structural change, given two different versions of the document. The structural delta is mapped into the H-DOM tree according to mapping rules as described in Algorithm 2. This process iterates until no more XML document is left in the sequence. Finally, the H-DOM tree is returned as the output of this phase.

Algorithm 2 in Figure 3 describes the mapping function. Given the H-DOM tree and the structural changes, this function is to map the deltas into the H-DOM tree and return the updated H-DOM tree. The idea is to update the corresponding values of the nodes in the H-DOM tree, for all the nodes in the structural delta. The values of the nodes are updated according to following rules:

i) If the node does not exist in the H-DOM tree, then the node is inserted. The value of C_v is set to $000\cdots1$ where the ith digit of the string is set to 1 and i is the version number of the structural delta. In addition, the N_i value is calculated. If $N_i \geq \alpha$, then C_n is set to 1 and the C_n values of its parent nodes are incremented by 1 until N_i is less than α. Otherwise, C_n is set to 0 and the process terminates.

ii) For nodes that exist in the H-DOM, the value of C_v is updated by inserting a 1 at the ith digit of C_v where i is the version number of the structural delta. The value of C_n is also updated based on N_i and α. Similarly, If $N_i \geq \alpha$, then C_n is incremented by 1 and the C_n values of its parent nodes are updated based on the same rule until N_i is less than α. Otherwise, C_n does not change and the process terminates.

4.2.2 FCS Extraction:

In this phase, given the H-DOM tree, the values of the required parameters (structure dynamic, version dynamic, and degree of dynamic) for each node are calculated and compared against the predefined thresholds. Since for a FCS, both its version dynamic and degree of dynamic should be no less than the thresholds, we first calculate only one of the parameters and determine whether it is necessary to calculate the other parameter. Because if any of the two parameters does not satisfy the definition, the substructure cannot be a FCS. In our algorithm, the version dynamic for a node is checked against the corresponding threshold first. If it is no less than the threshold, then we check its degree of dynamic. Considering the traversal strategy of the H-DOM tree, two approaches are analyzed: the bottom-up (level by level) approach and the top-down (breath first) approach. Before we come to the details of the traversal strategies, we present two lemmas that will be used to make the extraction

phase more efficient.

LEMMA *1.* Let $n_i, n_j \in N$ be any two nodes, the substructures rooted at n_i and n_j are denoted as S_{n_i} and S_{n_j} respectively. If n_i is the ancestor of n_j, then $V(S_{n_i}) \geq V(S_{n_j})$.

PROOF. The proof is intuitive. Based on the previous definition, once a node changes, superstructures that include this node are considered as changed. It indicates that the number of versions a superstructure has changed should be no less than its substructures. Consequently, it can be concluded that the version dynamic of a superstructure should be no less than the version dynamic of its substructures, while the total number of versions is the same. □

LEMMA *2.* Let S_1 and S_2 be any two structures, S_2 is a substructure of S_1. Given the threshold for DoD as γ, the necessary condition for structure S_1 to be a FCS is that $C_n(S_1) \geq \gamma \times V(S_2) \times (n-1)$.

PROOF. From Lemma 1, we can infer that $V(S_1) \geq V(S_2)$. The necessary condition for structure S_1 to be a FCS is that its degree of dynamic is no less than the threshold γ, which is $\gamma \leq \frac{C_n(S_1)}{V(S_1)\times(n-1)}$. Then, $C_n(S_1) \geq \gamma \times V(S_1) \times (n-1)$, while $V(S_1) \geq V(S_2)$, it can be inferred that $C_n(S_1) \geq \gamma \times V(S_2) \times (n-1)$. □

Based on the above lemmas, we observed that it is not necessary to traverse the entire H-DOM tree. We can skip checking some structures that cannot be FCSs. Lemma 1 can be used in the top-down traversal strategy. When we reach a node where its version dynamic is less than the threshold, it is not necessary to further traverse down this substructure since the version dynamic of its substructures will definitely be less than the threshold and they cannot be FCSs. Lemma 2 can be used in the bottom-up traversal strategy. In this case, for any node, rather than calculate its version dynamic value, the C_n value of the node is checked against the value of $\gamma \times V(S_i)$, where S_i is any of its substructures. If $C_n < \gamma \times V(S_i)$, then it is not necessary to calculate the version dynamic and degree of dynamic for this structure since it cannot be a FCS. Based on the lemmas, the top-down FCS extraction algorithm and the bottom-up FCS extraction algorithm are presented in Algorithm 3 and Algorithm 4 in Figure 3.

4.3 Algorithm Analysis

In this section, we analyze the time complexity and space complexity of the FCS basic algorithms. Since the visualization is trivial, we focus on analysis of the first and second phases.

4.3.1 Time Complexity:

In phase 1, the H-DOM tree is constructed based on the sequence of historical XML documents. In this phase, each XML document is parsed once and only consecutive versions are compared. Let $\langle\ |T_1|, |T_2|, \cdots, |T_n|\ \rangle$ and $\langle\ |t_1|, |t_2|, \cdots, |t_{n-1}|\ \rangle$ denote the number of nodes in the sequence of XML documents and the structure deltas respectively. The complexity of SX-Diff is $O(|T_i| \times |T_{i+1}|) \times max\{\deg(T_i), \deg(T_{i+1})\} \times \log_2(\max\{\deg(T_i), \deg(T_{i+1})\}))$ according to [8]. The complexity of the mapping process is $O(|t_i|)$. The SX-Diff and mapping process iterate $k-2$ times in this phase, while the cost of the initialization is $O(|T_1|)$. Since $|t_i| \leq |T_i|$, the dominant of this iteration is the SX-Diff. The overall complexity of phase 1 is $O((k-2)\times \max\{\ |T_i| \times |T_{i+1}|)\times \max\{\deg(T_i), \deg(T_{i+1})\} \times \log_2(\max\{\deg(T_i), \deg(T_{i+1})\})\})$, where $i \in [2, k-1]$. In phase 2, the H-DOM is traversed and the parameters for all the potential FCSs are calculated and compared against the predefined thresholds. No matter which traversal strategy we choose, the upper bound of this phase is $O(|T|)$, which is an entire traversal of the H-DOM tree, where $|T|$ is the total number of nodes in the H-DOM tree. In practice, the actual cost of this phase is substantially cheaper than this, since we use Lemma 1 and Lemma 2 to reduce the traversal space. From the above analysis, it can be inferred that the bottleneck of the FCS mining is the structural change detection process, which is the most expensive process.

4.3.2 Space Analysis:

In the FCS basic algorithms, the history of XML structural deltas is stored in the H-DOM tree and it is processed in memory. The space cost of this algorithm is the size of the H-DOM tree. Based on the algorithm, we observed that the size of the H-DOM tree depends on the overlaps between the consecutive versions. For the same number of XML documents with the same value of average number of nodes, the more significantly they change, larger the size of the H-DOM is. Since only the structural data is stored and each unique node is store only once, the size of the H-DOM should be no larger than the total size of the sequence of XML documents. However, as the sizes of the XML documents increase or the changes become more significant, or the number of XML documents increases, the size of H-DOM will increase accordingly. However, the upper bound of the space requirement is $O(|S_1 \uplus S_2 \uplus \cdots, \uplus S_n|)$, where $\langle S_1, S_2, \cdots, S_n\rangle$ are the tree representations of the XML documents sequence $\langle X_1, X_2, \cdots, X_n\rangle$.

4.4 Optimization Techniques

Based on the analysis of the FCS basic algorithms, in this section we propose three optimization techniques. The *compression techniques*, the *build and merge strategy*, and the *DTD-based pruning technique*. The objective of these techniques is to make the algorithm more scalable by reducing the size of the H-DOM tree.

4.4.1 Compression Technique:

In the H-DOM model, suppose there are n versions of XML in the sequence. Then, for each node a length n binary string is used to represent the history of changes. However, we observed that the size of the string can be very large, while only 2 out of n digits are useful since each node itself in the H-DOM could change at most twice, *insertion* and *deletion*. Consequently, rather than using the binary string, we use two integers to represent the changes. Consider the H-DOM tree in Figure 2 as an example. For node p_2, suppose it is deleted in the i+1th version, then the C_v value of this node will be $1000\cdots01$ in the basic approach. Now we only store two integers 1 and i to represent the changes. Using the basic algorithm the space requirement is i bites, but using this strategy it only requires 8 bites (for two integers). It is obvious that when $i > 8$, the later strategy is more efficient in terms of space. Usually, to get useful knowledge from the changes, the number of versions is greater than 8.

4.4.2 Building and Merging Strategy:

```
Input:
  A sequence of XML versions: <X 1, X2, i > Xn>
  Threshold of structure dynamic:   alpha
Output:
  H-DOM tree: H
1: H is initialized as S(X 1)
2:   for all  k less than  n do
3:     Delta = SX-Diff(X k, X(k-1))
4:     H= Mapping (H, Delta) //algorithm 2
5:     k++
6:   endfor
7: Return(H)
```

(a) Algorithm 1: H-DOM construction

```
Input:
  H-DOM tree: H
  Threshold of structure dynamic:   alpha
  Structural delta: Delta
Output:
  Updated H-DOM tree: H
1:   for all  ni in Delta
2:     if ni is not null
3:       for all  ni in H do
4:         update C n(ni)
5:         if Ni(ni) is no less than  alpha
6:           update C v(ni)
7:           n i = ni.parent(H)
8:         endif
9:       endfor
10:    endif
11: endfor
12:Return(H)
```

(b) Algorithm 2: Mapping

```
Input:
  H-DOM tree: H
  Threshold of version dynamic and DoD:   beta, gamma
Output:
  A set of FCS root nodes: F
1:  for all  ni is not null do (Top-down bread first approach)
2:   if V(ni) is no less than  beta then
3:     if DoD(n i) is no less than  gamma
4:       F is updated by incorporating n i
5:     endif
6:   else
7:     prune all descendants of n i
8:   endif
9:  endfor
10:Return(F)
```

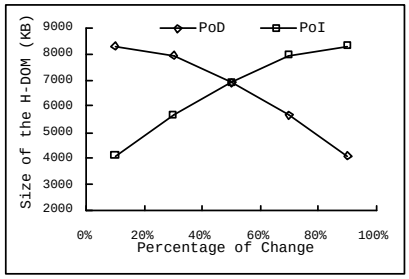
(c) Algorithm 3: FCS extraction (Top-down)

```
Input:
  H-DOM tree: H
  Threshold of version dynamic and DoD:   beta, gamma
Output:
  A set of FCS root nodes: F
1:  for all  ni that is not null in H
2:    if Cn is less than  gamma  times V(s) then
3:      ni = n i.next
4:    else
5:      if V(ni) is no less than  beta
6:        if DoD(n i) is no less than  gamma
7:          F is updated by incorporating n i
8:        endif
9:      endif
10:   endif
11: endfor
12:Return(H)
```

(d) Algorithm 4: FCS extraction (Bottom-up)

```
Input:
  H-DOM tree: H
  Threshold of structure dynamic: alpha
  Structural delta: Delta
Output:
  Updated H-DOM tree: H
1:  for all  ni in Delta
2:    if ni is deleted and V(n i) is less than  alpha then
3:      update(n i, H)
4:      prune all descendants of n i
5:    else
6:      update(n i, H)
7:    endif
8:  endfor
9:Return(H)
```

(e) Algorithm 5: Building and merging



(f) Experimental results: Size of H-DOM tree

Figure 3: Algorithms and Experimental Results

(a) Statistics of Datasets

Symbol	Description
NoN	Number of Nodes
NoV	Number of Versions
PoC	% of Changes

(b) Parameters

Symbol	Description
α	Threshold of $N_i(s)$
β	Threshold of $V(s)$
γ	Threshold of $DoD(s, \alpha)$

(c) Description of Datasets

dataset	NoN	NoV	PoC	α	β	γ
1	10644	20	10%	0.2	0.2	0.4
2	21464	20	10%	0.2	0.2	0.4
3	43196	20	10%	0.2	0.2	0.4
4	87642	20	10%	0.2	0.2	0.4

(d) Description of Datasets

Source data	NoN	NoV	PoC	α	β	γ
$SIGMOD_1$	1124	20	10%	-	0.2	0.4
$DBLP_1$	1143	20	10%	0.2	-	0.4
$Synthetic_1$	1264	20	10%	0.2	0.4	-

(e) Description of Datasets

Source data	NoN	NoV	PoC	α	β	γ
$SIGMOD_2$	-	30	10%	0.2	0.2	0.4
$DBLP_2$	5743	-	10%	0.2	0.2	0.4
$Synthetic_2$	1264	20	-	0.2	0.2	0.4

Table 1: Symbols, Descriptions, and Datasets

Based on the basic algorithms, we observed that for any structure that has been deleted their C_n and C_v values would not change since no change could happen to them again. Thus, whether this structure is a FCS or not can be determined by then. Differ from the FCS basic algorithms, we propose not to keep all substructures in the H-DOM tree. If the structures are not a FCS when they are deleted, only the root nodes are stored in the H-DOM, with the summarized historical information. By using this strategy, the size of the H-DOM tree will be reduced. Consider the H-DOM tree in Figure 2. Suppose in the coming version the substructures rooted at *DM* and *DB* are deleted. Based on the thresholds substructure *DM* is a FCS while substructure *DB* is not. Then, rather than store the entire substructure of *DM* and *DB*, only the root node of *DM* is stored with the summarization of historical information. Similarly, the substructure *DB* is merged into its parent node as shown in Figure 2 (b).The building and merging algorithm is shown in Algorithm 5 in Figure 3.

4.4.3 DTD-based Pruning Technique:

We observed from the history of the structural changes that some of the nodes never change in the history. Thus, it is not necessary to store such information since it cannot be used for FCS mining. If we can prune such nodes during the H-DOM construction phase, then the H-DOM tree will be more compact and the efforts of checking such nodes can be avoided. With the help of DTD and schema, elements and attributes in the XML documents can be categorized into two classes. Elements and attributes that can be inserted or deleted individually are classified to class 1, while elements and attributes that cannot be inserted or deleted individually are in class 2. With the DTD, we know that elements defined with more than one occurrences can be inserted or deleted while elements that are defined as default and required for exactly one occurrence cannot be deleted or inserted individually. Our DTD-based pruning strategy maps only nodes belong to class 1, while nodes in class 2 are merged into their parent nodes to save space. For example, suppose elements *BS, MS* and *PhD* are defined as required

subelements with exactly one occurrence for element *Edu* in Figure 1 (d). Then, rather than store the entire substructure rooted at node *Edu*, it can be represented as a single node *Edu* in the H-DOM tree.

5. PERFORMANCE STUDY

5.1 Experimental Setup and Dataset

We ran experiments on a PC with Intel Pentium 4, 1.7GHz CPU, 256 RAM, 40G hard disk, and Microsoft Windows 2000. We have implemented two basic algorithms, the bottom-up based algorithm FCS-BASIC-B and the top-down based algorithm FCS-BASIC-T. We also implemented optimization-based algorithms by combining the optimization techniques we proposed. Since their performances are quite similar, only two of them are presented in this section due to space constraint. FCS-A is implemented by integrating the three optimization techniques. FCS-C is implemented with the compression technique and the building and merging technique for XML data sequences without DTDs. Note that the optimization-based algorithms are implemented using the bottom-up traversal strategy since the metrics can be calculated more efficiently in this way.

We use synthetic XML delta sequences generated from three XML documents, which are real and synthetic XML documents. The two real XML documents we use are DBLP and SIGMOD XML downloaded from UW XML repository [1], while the synthetic XML is generated by IBM XML Generator [2]. From such XML documents, sequences of XML versions are generated by using our synthetic XML delta generator. We do experiments by using datasets of different characteristics and varying the parameters of each algorithm. For each algorithm, different XML datasets are used to show how the datasets affect the performance. Experiments with the same dataset and all possible variations of the parameters have also been done to show how the parameters can affect the performance. The symbols for characteristics of the datasets and parameters of the algorithms are shown in Tables 1(a) and 1(b) with their descriptions.

5.2 Variation of Algorithm Parameters

We evaluate the performance of the four algorithms, FCS-BASIC-T, FCS-BASIC-B, FCS-A, and FCS-C, by varying the thresholds of the three major parameters, Structure dynamic, version dynamic, and degree of dynamic. Table 1 (d) shows the characteristics of the datasets and some parameters used in our experiments. Hereafter, we use the symbol "-" to denote the parameters or characteristics of the dataset that will be varied in the experiments. Figure 4 (a) shows the performance of the algorithms by varying the threshold α. We use the $SIGMOD_1$ XML dataset. Figure 4 (b) shows how the algorithms perform when the threshold β changes. We use the $DBLP_1$ XML dataset. Figure 4 (c) describes how the changes of threshold γ may affect their performances. We use the $Synthetic_1$ XML dataset. From the above figures, following observations can be made.

None of the algorithms is sensitive to the changes of α, β or γ. However, the overall observation is that as any of the thresholds increases, the execution time decreases. This is due to the fact that when the threshold increases, the pruning techniques are more efficient and the search space of FCS is reduced. The execution time does not change significantly with the changes of thresholds because that the major cost of the algorithms is the cost of SX-Diff, which is independent to the thresholds. As shown in Figure 4 (d), the SX-Diff cost is more than 50% of the total cost. From Figure 4 (d), we also observed that as the total number of nodes increases the percentage of SX-Diff cost also increases.

The FCS-BASIC-T algorithm is more stable than others as α changes, but it is more sensitive to the changes of β. This is due to the fact that FCS-BASIC-T uses the heuristic in Lemma 1 to prune the H-DOM tree, and other algorithms use the heuristic in Lemma 2, where Lemma 1 is solely based on β and Lemma 2 is based on α.

For the same dataset, with the same parameters, we observed that the differences of execution time for the four algorithms are in constant order. It means that although different traversal strategies, optimization techniques are used, the time cost does not change significantly.

5.3 Characteristics of Datasets

We evaluate the performance of the four algorithms by varying the characteristics of the datasets. Table 1 (e) shows the values of the parameters and some of the characteristics of the datasets used in our experiments. Figure 4 (e) shows the performance of the algorithms using $SIGMOD_2$ by varying NoN, the average number of nodes in each XML document, from 8,000 to 40,000 (the corresponding size of each XML document is from 3M to 15M), with 30 versions in the sequence. Figure 4 (f) presents the performance of the algorithms using $DBLP_2$ by varying NoV, the number of version in the sequence, from 30 versions to 150 versions . With the average size of each XML document is 2.3M (5743 nodes). Figure 4 (g) evaluates the performance of the algorithms by varying PoC. The $Synthetic_2$ dataset is used. From the above figures, several observations can be made.

As the average number of nodes NoN in the XML document increases, the time cost increases. Changes are more significant compared to the changes in Figures 4 (f) and 4 (g). It is because when the average number of nodes increases, the SX-Diff cost increases, the pruning and extraction phase become more expensive too.

As the total number of versions NoV in the XML sequence increases, the execution time of the algorithms increases too. It is obvious that when the total number of versions increases, the number of comparison increases accordingly. Consequently, the cost for detection the structural changes increases. Compared with the changes of NoN, the changes of NoV do not affect the performance significantly.

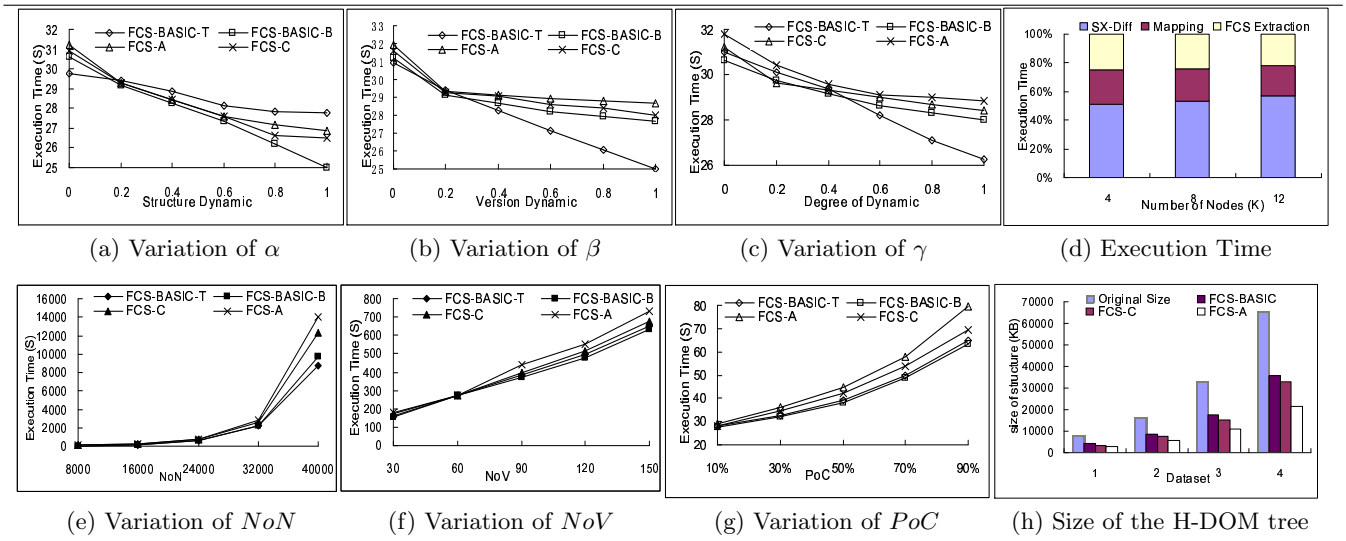
As the percentage of changes PoC in the XML sequence increases, the execution time also increases. Since our FCS mining is actually dealing with the deltas rather than the original sequence, as the PoC increases, the size of the delta increases. Consequently, the SX-Diff, the pruning and extraction phases become more expensive.

5.4 Compression Efficiency Experiments

We evaluate the space efficiency of the algorithms by comparing the compactness of the H-DOM tree. Table 1 (c) shows the characteristics of the datasets and parameters used in the experiments. Figure 4 (h) show the size of the H-DOM trees for different datasets and different algorithms.

From Figure 4 (h), we can observe that compared to the

[1] http://www.cs.washington.edu/research/xmldatasets
[2] http://www.alphaworks.ibm.com/tech/xmlgenerator



(a) Variation of α (b) Variation of β (c) Variation of γ (d) Execution Time

(e) Variation of NoN (f) Variation of NoV (g) Variation of PoC (h) Size of the H-DOM tree

Figure 4: Experiment Results

original dataset, the H-DOM trees are very compact. The compression rate of the H-DOM tree is almost 50% without any optimization techniques. With the optimization techniques, the FCS-C, and FCS-A are more compact than the FCS-BASIC. Especially, the H-DOM tree built using the FCS-A is the most compact one. The compression rate of the FCS-A is around 30% according to our experiments. This fact also explains why the time cost of the FCS-C, and FCS-A are relatively more expensive as shown in the results shown in Figures 4 (a) to 4 (g).

We evaluate the compression rate of the H-DOM tree using datasets of different characteristics. Among all the characteristics of the dataset, we observed that only the percentages of deletion and insertion affect the compression rate significantly. Figure 3 (f) shows how the size of the H-DOM tree changes as the percentages of deletion and insertion change. The H-DOM tree is built by the FCS-A algorithm. We use dataset 1 in Table 1 (c). The percentage of changes is set to 60%. From Figure 3 (f), we observed that as the percentage of deletion increases and the percentage of insertion decreases the size of the H-DOM decreases. This is because when more and more nodes are deleted the pruning and merging function is more efficient.

5.5 Summary

From the above observations, we can conclude that our proposed algorithm FCS-BASIC-T and FCS-BASIC-B are efficient and scalable while three optimization techniques, *compression technique, building and merging strategy,* and *DTD-based pruning strategy*, have improved the space efficiency substantially. Based on the experiment results, if users want to find out FCS with higher version dynamic, the FCS-BASIC-T is recommended. Otherwise, the FCS-BASIC-B is the best choice, since the three optimization techniques work in a bottom-up manner. The FCS-C algorithm can be applied to any datasets, while the FCS-A can only be used for datasets with DTDs.

6. CONCLUSIONS

In this paper, we propose an approach to discover useful and hidden knowledge from the history of XML structural changes. Specifically, we focus on extracting the FCSs. We propose an H-DOM model to represent and store the XML structural data, in which the history of structural data is preserved and compressed. Based on the H-DOM model, we present two basic algorithms, FCS-BASIC-T and FCS-BASIC-B, to discover the FCSs. By analyzing the performance of the basic algorithms, some optimization techniques are also discussed. Extensive experimental evaluations using dataset generated from both synthetic and real XML documents have been conducted for all the algorithms we proposed. Recommendations of when to choose which algorithm are also discussed based on the experiment results. Finally, we show the usefulness of the frequently changing structures by presenting a list of potential applications.

7. REFERENCES

[1] D. Braga, A. Campi, S. Ceri, M. Klemettinen, and P. L. Lanzi. A tool for extracting XML association rules. *In Proc. ICTAI*, 57–65, 2002.

[2] L. Chen, S. S. Bhowmick and C. Chia. Mining Association Rules from Structural Deltas of Historical XML Documents. In *In Proc. PAKDD*, 452–457, 2004.

[3] G. Cobena, S. Abiteboul, and A. Marian. Detecting changes in XML documents. *In Proc. ICDE*, 41–52, 2002.

[4] Curbera and D. A. Epstein. Fast difference and update of XML documents. *In Proc. XTech'99*, 1999.

[5] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *In Proc. SIGMOD*, 1–12, 2000.

[6] A. Inokuchi, T. Washio, and H. Motoda. An apriori based algorithm for mining frequent substructures from graph data. *In Proc. PKDD*, 13–23, 2000.

[7] M. Kuramochi and G. Karypis. Frequent subgraph discovery. *In Proc. ICDM*, 313–320, 2001.

[8] Y. Wang, D. J. DeWitt, and J.-Y. Cai. X-diff: An effective change detection algorithm for XML documents. *In Proc. ICDE*, 519–530, 2003.

[9] X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. *In Proc. ICDM*, 721–724, 2002.

[10] M. J. Zaki. Efficiently mining frequent trees in a forest. *In Proc. SIGKDD*, 71–80, 2002.

[11] M. J. Zaki and C. C. Aggarwal. XRules: An effective structural classifier for XML data. *In Proc. SIGKDD*, 316–325, 2003.

[12] Q. Zhao and S. S. Bhowmick. Mining changes to historical web access patterns. *In Pro. PKDD*, 2004.