XML Structural Delta Mining: Issues and Challenges

Qiankun Zhao [a] Ling Chen [a] Sourav S. Bhowmick [a]
Sanjay Madria [b]

[a] *School of Computer Engineering, Nanyang Technological University, Singapore*
[b] *Department of Computer Science, University of Missouri-Rolla, USA*

Abstract

Recently, there is an increasing research efforts in XML data mining. These research efforts largely assumed that XML documents are static. However, in reality, the documents are rarely static. In this paper, we propose a novel research problem called *XML structural delta mining*. The objective of XML structural delta mining is to discover knowledge by analyzing structural evolution pattern (also called *structural delta*) of history of XML documents. Unlike existing approaches, XML structural delta mining focuses on the dynamic and temporal features of XML data. Furthermore, the data source for this novel mining technique is a sequence of historical versions of an XML document rather than a set of snapshot XML documents. Such mining technique can be useful in many applications such as change detection for very large XML documents, efficient XML indexing, and XML search engine etc. Our aim in this paper is not to provide a specific solution to a particular mining problem. Rather, we present the *vision* of the mining framework and present the issues and challenges for three types of XML structural delta mining: *identifying various interesting structures, discovering association rules from structural deltas,* and *structural change pattern-based classification.*

Key words: Versions of XML documents, structural delta, dynamic metrics, XML structural delta mining, research issues, applications.

1 Introduction

Recently, XML is widely used as the *de facto* standard for data exchanging in the internet. As more and more data is stored and represented in

Email address: assourav@ntu.edu.sg (Sourav S. Bhowmick).

XML format, there have been increasing research efforts in mining XML data. Existing works on mining XML data include frequent substructure mining [18, 21, 40, 9, 38, 16, 11], classification [41, 12, 17] and association rule mining [4], etc. Among these, the frequent substructure mining is one of the most well-researched topics. The basic idea is to extract substructures, subtrees or subgraphs, which occur together frequently among a set of XML documents. The set of discovered frequent substructures can be useful in different XML-based applications such as efficient querying and XML integration [39]. Moreover, they can be useful for data analysis in different domains such as bioinformatics, chemistry, and network analysis [12, 17], since data in these domains can be easily modelled as XML documents.

1.1 Motivation

A key feature of XML data is its dynamic property. XML data may change at any time in any way. New data are inserted into the XML document; obsolete data are deleted or updated. That is, the dynamic nature of XML data results in two types of changes: *structural deltas* and *content deltas*. *Structural deltas* typically modify the structure of the XML data whereas *content deltas* modify the textual content. In this paper, we focus our attention to structural deltas only. *Hereafter, unless otherwise specified, changes to the XML data refer to structural deltas.*

The dynamic nature of the structure of XML data leads to the following challenging problems in the context of data mining.

- **Maintenance of XML data mining results:** Due to the changes to XML data, knowledge extracted from obsolete data may not be valid any longer. Let us elaborate further. Suppose there is a collection of XML documents that record the detail information about products and services for a e-commerce web site. By applying existing frequent structure mining techniques [18, 21, 37, 38], frequent substructures among the XML documents can be discovered. However, with the evolution of the XML documents, some parts of the structure may be deleted while new structures may be inserted. Consequently, the set of frequent substructures extracted at time t_1 may change at time t_2 where $t_2 > t_1$. That is, some of the previously discovered frequent substructures may not be frequent any more. Similarly, some of the substructures, which were infrequent previously, may become frequent. To address this problem, some incremental data mining techniques [13, 28] can be applied to maintain the mining results.
- **Discovering novel knowledge:** Historical collection of XML data contains rich temporal information. Consequently, *novel knowledge may be hidden behind the history of changes to XML data* that cannot be discovered by data mining techniques designed for static data. For example, we may find

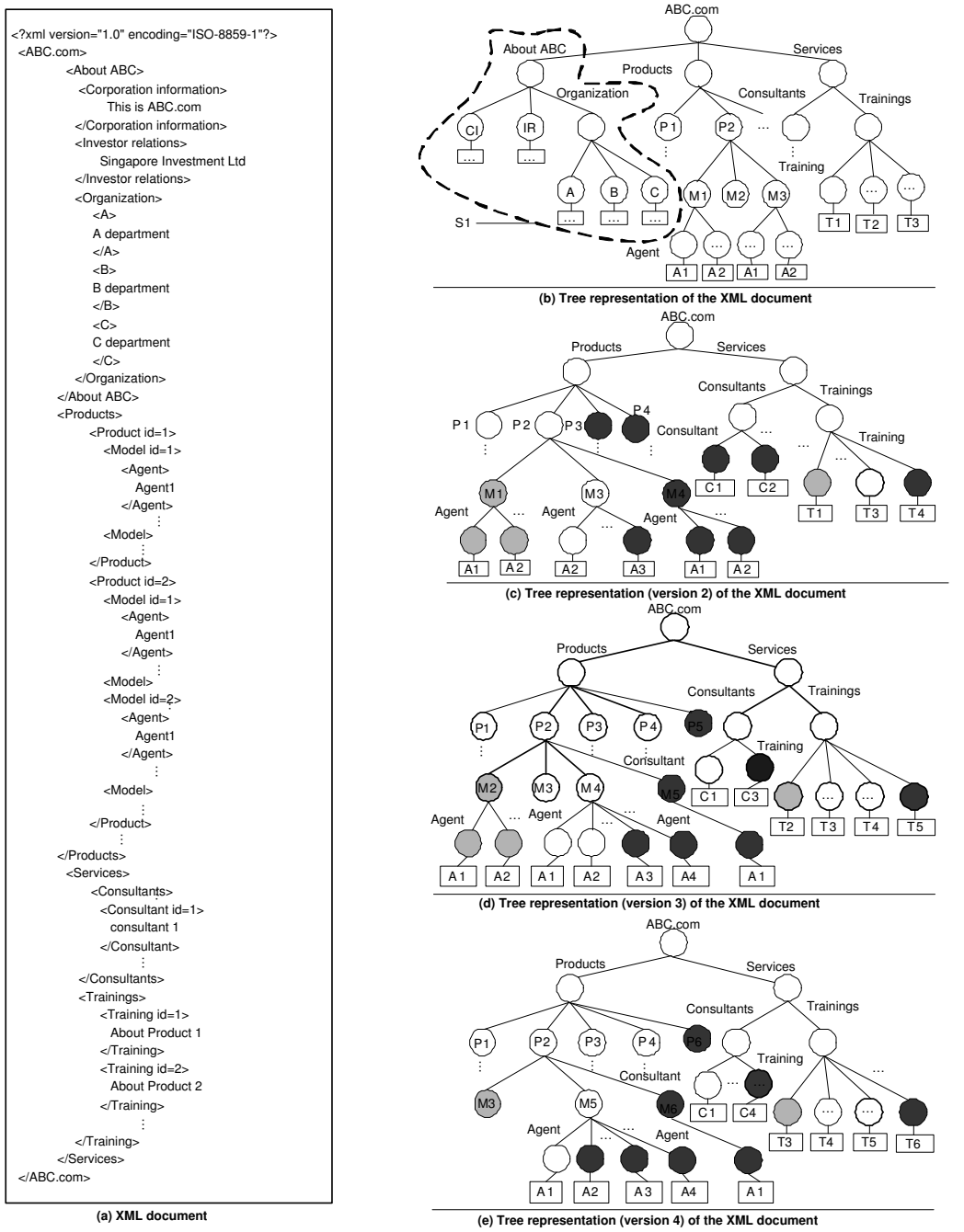


Fig. 1. Versions of XML documents.

that some particular parts of the XML documents evolve more frequently than the other parts. As we shall see later, while knowledge obtained from snapshot data is interesting, knowledge discovered by analyzing evolution pattern of XML data may be also critical in many applications such as change detection for XML data, XML indexing, semantic meaning extraction etc.

In this paper, we focus on the second problem in the context of structural

evolution of XML data. That is, we focus on discovering novel knowledge by analyzing the patterns of the structural deltas of historical XML documents. We call such knowledge discovery process as *XML structural delta mining*. We now illustrate with an example the types of knowledge that can be discovered. Fig. 1(a) is an XML document that records the information such as *products*, and *services* of an e-commerce Web site *ABC.com*. Fig. 1(b) is the tree representation of the XML document. Fig. 1(c), (d) and (e) are the tree representations of another three historical versions of the same XML document in Fig. 1(a). For brevity, we use P_i, C_i, M_i, and T_i to represent the *Product, Consultant, Model* and *Training* element, respectively whose attribute *id* is i. In these figures, each node represents an element or attribute and each edge represents the parent and child relationship among the elements and attributes. The gray circles in this figure denote elements/attributes that are deleted; the black ones denote the elements/attributes that are inserted; the circles with bold lines represent the elements/attributes whose values have been updated. From the versions of XML documents in Fig. 1, the following novel knowledge can be extracted by employing XML structural delta mining techniques.

- *Frequently changing structures*: Some parts of the structures change more frequently and significantly (such as the subtrees rooted at node *products* and *services*) compared with other parts (such as the subtree rooted at node *About ABC*). Such structures represent the relatively more dynamic parts of the XML document.
- *Frozen structures*: Some parts of the structures never or seldom change in the history (such as the substructure s_1 shown in Fig. 1(b) with dotted line). Frozen structures represent the most stable parts of the XML document.
- *Association rules*: Some structures are associated in terms of their change patterns. For instance, whenever the subtree rooted at node P_2 changes, the subtree rooted at node *Training* also changes. Such an association rule implies the concurrence of changes to different parts of the XML document.
- *Change patterns*: Consider the sequence of versions in Fig. 1. It is possible to detect certain trend-based patterns from the historical versions. For instance, one can observe that more and more nodes are inserted in some substructures such as *Consultant*, while nodes are frequently inserted in and deleted from other substructures such as *Training*. We may also discover many other types of change patterns such as trends, seasonal patterns, and periodical change patterns, etc.

In this paper, we present the issues and challenges related to the discovery of above types of knowledge. Note that by no means we claim that the above list is exhaustive. We use them as representatives for various types of knowledge behind the sequence of structural changes to XML documents. Such novel knowledge can be useful in different applications such as change detection for very large XML documents, dynamic XML indexing, and semantic meaning extraction etc. The details about different applications will be discussed in

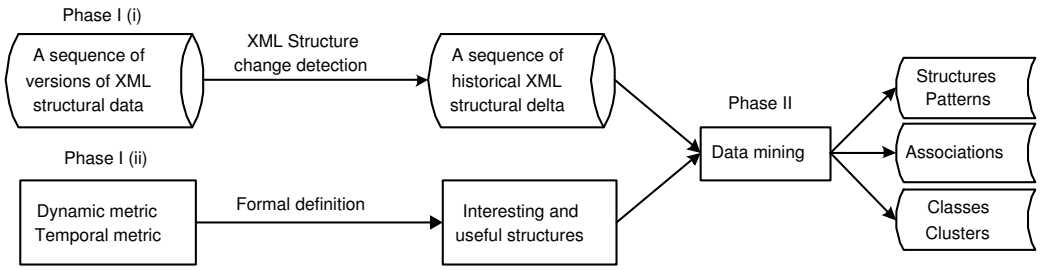


Fig. 2. Architecture of XML structural delta mining.

Section 8.

To the best of our knowledge, the state-of-the-art XML structure mining techniques [37, 38] fail to extract such knowledge. This is due to the following differences between XML structural delta mining and existing works on XML structure mining. First of all, XML structural delta mining focuses on the dynamic and temporal features of XML data. Secondly, the data source for XML structural delta mining is a sequence of historical versions of an XML document rather than a set of snapshot XML documents collected at a specific time point. The major contributions of this paper can be summarized as follows.

- We propose to discover novel knowledge from changes to historical versions of XML structure data (XML structural delta). The challenges and potential applications for XML structural delta mining are also discussed.
- Three representative types of novel XML structural delta mining are discussed. They are *identifying interesting substructures, mining XML structural delta association rules,* and *classifying XML data based on structural change patterns*, respectively.

We caution that our framework for XML structural delta mining is still in its formative stages. *Our aim in this paper is not to provide a specific solution to a particular problem.* Rather, we are presenting the *vision* of our framework, with the hopes that others in the community will explore further on specific problems in the arena of XML structural delta mining. We strongly believe that mining historical versions of XML documents is an exciting research area that has not yet been well explored. This paper identifies some of the major issues that needs to be addressed in this context.

1.2 The Framework

The general framework of XML structural delta mining is depicted in Fig. 2. Given a sequence of historical versions of an XML document, the objective of XML structural delta mining is to extract useful knowledge such as interesting

structures, association rules, and classification/clusters from the history of XML structural deltas. There are two major phases in the XML structural delta mining process.

In the first phase, there are two parallel sub-phases as shown in Fig. 2. In phase I(i), we need a structural change detection system to obtain the sequence of structural deltas from versions of XML documents. Recently, a number of techniques for detecting the changes to XML data have been proposed such as XyDiff [7], X-Diff [35], and TreeDiff [10].These techniques detect both content and structural changes. Consequently, a structural change detection system can be built based on these existing systems. The results of this sub-phase is a sequence of structural delta. In phase I(ii), we define some *dynamic metrics* to measure the interestingness of different substructures based on their *dynamic* and *temporal* properties. Based on these metrics, different types of interesting structures are defined and discovered in the subsequent phase. In the second phase, given a sequence of historical structural deltas and user-defined constraints with respect to a set of dynamic metrics, data mining techniques are applied to extract various types of knowledge as highlighted earlier. New algorithms or techniques need to be designed to mine such types of knowledge.

1.3 Paper Organization

The organization of this paper is as follows. In Section 2, we present the related works. In Section 3, we formally introduce the XML structural delta mining problem and present some concepts that shall be used subsequently to discuss various types of knowledge discovery problem. Sections 4–6 present various types of knowledge that we can discover using XML structural delta mining technique. Specifically, the issue of discovering different types of *interesting substructures* is elaborated in Section 4. In Section 5, *XML structural delta association rule mining* is discussed. The *change pattern-based classification* problem is presented in Section 6. Next, key research issues for XML structural delta mining are discussed in Section 7. Then, in Section 8, we briefly present some real-life applications of the novel knowledge discussed in the preceding sections. Finally, the last section concludes this paper.

2 Related Work

Our proposed XML structural delta mining is largely influenced by several recent technologies by three major research communities. First, in the recent times, the database and XML communities have looked at XML change detection problem. Second, the data mining and AI communities have been increasingly active in addressing various issues related to XML data mining.

We compare our approach with these technologies and highlight the novelty of our approach.

2.1 XML Change Detection

Recently, a number of techniques for detecting the changes to XML data have been proposed. XMLTreeDiff [10] and XyDiff [7] are main-memory algorithms for detecting the changes in *ordered* XML documents. In an *ordered* XML, both the parent-child relationship and the left-to-right order among siblings are important. In [10], Curba and Epstein proposed the TreeDiff algorithm to compute the differences between two XML documents using DOMHash values. TreeDiff is a tool developed by IBM to detect changes to ordered XML documents. Using the hash values, TreeDiff can reduce the size of the trees by filtering the identical subtrees. However, the results generated by TreeDiff may not be the optimal. XyDiff [7] is another algorithm for detecting changes to ordered XML documents. The algorithm computes a signature (i.e., hash value) and a weight (i.e., subtree size) for every node in both documents in a bottom-up fashion. Based on the signature and weight, subtrees with the largest weight are always compared first. If the signatures are equal, the two nodes are matched. The algorithm starts from finding a match between the heaviest nodes and heavier subtrees have higher priority to be chosen for comparison. Once a match is found, it will propagate to ancestors to get more matches. Insertions/Deletions and Moves will be computed after all exact matches are found. However, XyDiff cannot guarantee any form of optimal or near-optimal result because of the greedy rules used in the algorithm. Wang et al. proposed X-Diff [35] for computing the changes to *unordered* XML documents. In *unordered* XML, the parent-child relationship is significant, while the left-to-right order among siblings is not important. In X-Diff algorithm, for each pair of nodes from the input documents the distance between their respective subtrees is obtained by finding the minimum cost mapping for matching children (by reduction to the minimum cost maximum flow problem). X-Diff generates more accurate results compared with XyDiff. The main strength of X-Diff algorithm is that it reduces the mapping space significantly and achieves polynomial time complexity. However, the change detection response time is slower than XyDiff and it cannot handle very large XML documents.

All these algorithms suffer from scalability problem as they fail to detect changes to large XML documents due to lack of memory. Consequently, a number of approaches [23, 24, 22] have been proposed to address the scalability problem of XML change detection by using relational databases. In this approach, first, XML documents are stored in RDBMS. Then, the changes are detected by using a set of SQL queries. Experimental results show that this approach has better scalability, running time, and comparable result quality compared to X-Diff.

Similarly, there are some works about detecting and monitoring the changes of data schemas in the data warehouse scenario [2, 3, 29]. In [2], a set of change operations, similar to those proposed in the XML change detection process, was proposed to describe the changes in the data schemas. In [3], the authors presented a concept and an implementation of a multi-version data warehouse that is capable of handling changes in the structure of its schema. An approach to handling changes in data warehouse structure and content based on a multi-version data warehouse is proposed in [29]. Each data warehouse version describes a schema and data at certain period of time or a given business scenario, created for simulation purposes. In order to appropriately analyze multi-version data, an extension to a traditional SQL language is required.

Compared to the above works, the proposed XML structural delta mining is different in the following way. Give a sequence of historical versions of an XML document, XML change detection systems generates a sequence of changes between each pair of consecutive XML documents. These systems are not designed beyond change detection. However, the detected changes can be too large for the users to understand. XML structural delta mining process the detected changes further to extract hidden knowledge so that the results can be understood and utilized by users easily.

2.2 XML Data Mining

With the ever-increasing amount of available XML data, the data mining community has been motivated to discover knowledge from collections of XML documents. For example, there have been increasing research efforts in mining frequent patterns [4, 31] or sequential patterns [25] from XML repositories, classifying [41] and clustering [27] XML documents. We review some of the works in the following two related areas, *mining association rules or frequent patterns from XML data* and *classifying XML data*.

As one of the most important problems of data mining, association rule mining has been applied to discover some underlying associations from XML repositories. *XMINE* [4] is a tool developed to extract XML association rules from XML documents. Based on XPath and inspired by the syntax of XQuery, *XMINE* allows to express complex mining tasks. However, *XMINE* aims to discover association rules from the contents of XML documents whereas we consider the structures of XML documents.

Since XML documents are typically viewed as semi-structured data, they do not have rigid structure. Major work on XML structure mining focuses on discovering frequent substructures from a collection of XML documents [1, 31, 34, 40]. Wang and Liu [34] developed an Apriori-like algorithm to mine frequent substructures based on the "downward closure" property. They first found the

frequent *1-tree-expressions* that are frequent individual *label path*s. Discovered frequent *1-tree-expression*s are joined to generate candidate *2-tree-expression*s. The process is executed iteratively till no candidate *k-tree-expression*s is generated. Asai et al. [1] developed another algorithm, FREQT, to discover all frequent tree patterns from large semi-structured data. They modeled the semi-structured data as *labeled ordered tree* and discover frequent trees level by level. At each level, only the rightmost branch is extended to discover frequent trees of the next level. Thus, efficiency can be obtained without generating duplicate candidate frequent trees. TreeMinerH and TreeMinerV [40] are two algorithms for mining frequent trees in a forest. As the name of the algorithm indicates, TreeMinerH is an Apriori-like algorithm based on a horizontal database format. In order to efficiently generate candidate trees and count their frequency, a smart *string encoding* is proposed to represent the trees. In contrast, TreeMinerV uses vertical *scope-list* to represent a tree. Frequent trees are searched in depth-first way and the frequency of generated candidate trees are counted by joining *scope-lists*. TreeFinder [31] is an algorithm to find frequent trees that are *approximately* rather than *exactly* embedded in a collection of tree-structured data modelling XML documents. Each labelled tree is described in *relaxed relational description* which maintains ancestor-descendant relationship of nodes. Input trees are clustered if their atoms of *relaxed relational description* occur together frequently enough. Then maximal common trees are found in each cluster by using algorithm of *least general generalization*. Recently, there is another line of work that employs the pattern-growth strategy to discover frequent subtrees [33, 36].

Classification of XML documents has also been addressed by some recent research works [32, 41]. In [41], Zaki proposed an algorithm to construct *structural rules* in order to classify XML documents. The basic idea is to relate the presence of a particular kind of structural pattern in an XML document to its likelihood of belonging to a particular class. In addition to the usual text-based term frequency vectors, Theobald et al. [32] explored other features such as XML twigs and tag paths on which an XML classifier operates. Moreover, they also leveraged the ontological background for the construction of more expressive feature spaces.

The critical difference between our XML structural delta mining and existing works on XML data mining is that we address the dynamic nature of XML data. Existing works on XML data mining extract knowledge from the snapshot version of XML documents, whereas we extract knowledge from a sequence of historical *structural deltas* of an XML document.

Symbol	Description
X	An XML document.
T	Tree representation of an XML document.
t	A subtree in XML tree T.
e_i	A basic structural edit operation.
$\triangle_i^{i+1}(t)$	Structural delta of subtree t from i-th to $(i+1)$-th version.
$\|\triangle_i^{i+1}(t)\|$	Size of structural delta $\triangle_i^{i+1}(t)$.
Ψ_t	Structural delta sequence of subtree t.
G_t	Structural edit script sequence of a subtree t.
$V(t)$	Version dynamic of subtree t.
$V(S)$	Version dynamic of a set of subtrees S.
$N_i(t)$	Structure dynamic of subtree t.
α	Threshold for structure dynamic.
$DoD(t,\alpha)$	Degree of dynamic of subtree t.
$DoD(S,\alpha)$	Degree of dynamic of a set of subtrees S.
β	Threshold for version dynamic.
γ	Threshold for DoD.
ν	Threshold for periodic dynamic structure.
$\rho_{in}(t)$	Increasing strength of subtree t.
$\rho_{de}(t)$	Decreasing strength of subtree t.
$\tau_{in}(t)$	Increasing tolerance of subtree t.
$\tau_{de}(t)$	Decreasing tolerance of subtree t.
σ	Threshold for strength.
κ	Threshold for tolerance.
$U_i(t)$	Surprise of a subtree t.
δ	Threshold for surprise.
ϕ	Threshold for confidence.
θ	Threshold for interest.

Table 1
Summary of symbols.

3 Preliminaries

In this section, we introduce some concepts that are used to define various types of knowledge discovered by XML structural delta mining. First, we give an overview of different types of change operations that result in XML structural deltas. Then, we present various types of *dynamic metrics*. Finally, we formally define the problem of XML structural delta mining. In the subsequent sections, we shall use these concepts to discuss various types of novel knowledge. Table 1 provides a summary of symbols used in the subsequent sections.

The structure of an XML document can be modelled as a rooted tree or a graph. In this paper, we use the rooted tree structure as example. Formally, an XML tree structure is defined as following.

Definition 1 (XML Tree) *The structure of an XML document can be modelled as a rooted tree T that has 3-tuple T = (V, E, V_0), where V is a set of nodes $\{V_0, V_1, \cdots, V_n\}$; E is a set of edges $\{e_{0i}, e_{ij}, \cdots\}$, where e_{ij} = (V_i, V_j) connects node V_i to node V_j; $V_0 \in V$ is a distinguished node named the root of the tree structure.* □

In the tree structure representation of an XML document, each node represents an element/attribute and each edge represents the parent and child relationship between the two nodes. The *size* of the tree T, denoted by $|T|$, is the number of nodes in V. After representing the structure of an XML document as a tree, a substructure can be represented correspondingly as a subtree, which is defined as follows.

Definition 2 (XML Subtree) *A rooted tree structure t = (V′, E′, V_0') is a* ***subtree*** *of an XML tree T = (V, E, V_0), denoted as $t \preceq T$, provided i) $V' \subseteq V$; ii) e_{ij} = (V_i, V_j) $\in E'$ if and only if $e_{ij} \in E$; iii) $V_0' \in V$.* □

Then, change operations on XML documents can be defined based on the tree structures. In [35], all changes to XML documents can be described by five types of edit operations, including three basic operations and two composite operations that can be decomposed into a list of basic operations. As we are only concerned of structural changes to XML document, the *update* operation, which only results in the changes to contents, is not considered in our research. Hence, we only consider the following two operations as *basic* edit operations.

- *Insert*(x(*name*, *value*), y): insert a node x, with node name *name* and node value *value* (possibly empty), as a child node of node y.
- *Delete*(x): delete a node x.

Note that using the above basic operations, the following *composite* operations can be defined.

- *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.

Observe that we have not considered the *move* operation to be a basic or composite edit operation. A move can be represented by a sequence of delete and insert operations. The structural change is identical although they may have different semantics. However, the structural significance of the semantics

is application-specific. That is, for certain applications move operation may not carry any additional semantics compared to insert and delete operations. On the other hand, for other applications the semantics may be significant. Hence, for this paper, we assume move to be equivalent to an insert and a delete operations. Our edit operation definition can easily be extended if a specific application requires the move operation to be considered as a basic edit operation.

3.2 XML Structural Delta

Based on the edit operations, an *edit script* is defined as a sequence of edit operations that transforms an XML document to another one [35]. Corresponding to the structural changes, we define the *structural edit script* as a sequence of *basic* edit operations that converts the structure of one version to the structure of another version. Note that it differs from the definition of edit script in two ways. First, a structural edit script does not include any update operation. Second, unlike edit script, it is composed of *basic* edit operations (insertion and deletion of a node). Note that an edit script may contain composite edit operations. For example, the structural edit script for the subtree rooted at node *Products* from the version in Fig. 1(b) to the version in Fig. 1(c) is $\langle$ Insert((P_3, *val*), Products), Insert((P_4, *val*), Products), Insert((M_4, *val*), P_3),Insert((A_1, *val*), M_4), Insert((A_2, *val*) M_4), Insert((A_3, *val*), *Agent*), Insert((C_1, *val*), Consultant), Insert((C_2, *val*), Consultant), Insert((T_4, *val*), Training), Delete(M_1), Delete(A_1), Delete(A_2), Delete(T_1) $\rangle$.

Given two versions of an XML document, formally, the structural delta between them is defined as follows.

Definition 3 (Structural Delta) *Let T_i and T_{i+1} be the tree representations of two versions of an XML document, denoted as X_i and X_{i+1}. Let $t_i \preceq T_i$. The corresponding structure of t_i in T_{i+1} is t_{i+1}, denoted as $t_{i+1} \preceq T_{i+1}$. The **structural delta** for the subtree t_i from T_i to T_{i+1}, denoted as $\triangle_i^{i+1}(t)$, is defined as a structural edit script $\langle e_1, e_2, \cdots, e_m \rangle$ that transform the structure of t_i into t_{i+1}. That is, $\triangle_i^{i+1}(t) = \langle e_1, e_2, \cdots, e_m \rangle$ where e_k is a basic edit operation $\forall\ 0 < k \leq m$. The **size** of the structural delta $\triangle_i^{i+1}(t)$, denoted as $|\triangle_i^{i+1}(t)|$, is m. That is, $|\triangle_i^{i+1}(t)| = m$. Furthermore, the structural delta from X_i to X_{i+1} is denoted as $\triangle_i^{i+1}$.* □

Reconsider the structural edit script for the subtree rooted at node *Products* from the version in Fig. 1(b) to the version in Fig. 1(c) as discussed above. Here, the size of its structural delta is 13 as there are 13 basic edit operations between the two versions of XML documents.

In the above definition, the XML structural delta is defined for two consecu-

tive versions of an XML document. To represent the sequence of changes to more than two versions of an XML document, we define the notion of *XML structural delta sequence*.

Definition 4 (Structural Delta Sequence) *Let $\langle T_1, T_2, \cdots, T_n\rangle$ be the sequence of tree representations of n historical versions of an XML document X. Let $t \preceq T_1$. The* ***structural delta sequence*** *for the subtree t from T_1 to T_n is $\Psi_t = \langle \triangle_1^2(t), \triangle_2^3(t), \cdots, \triangle_{n-1}^n(t)\rangle$, where $\triangle_i^{i+1}(t)$ is the XML structural delta for t from ith version to (i+1)th version. Also, structural delta sequence of X is denoted as $\Psi_X = \langle \triangle_1^2, \triangle_2^3, \cdots, \triangle_{n-1}^n\rangle$.* □

Definition 5 (Structural Edit Script Sequence) *Let $\Psi_t = \langle \triangle_1^2(t), \triangle_2^3(t), \cdots, \triangle_{n-1}^n(t)\rangle$. Then, the* ***structural edit script sequence*** *for a subtree t, denoted as G_t, is defined as $\langle e_1^1, e_2^1, \cdots, e_p^1, e_1^2, e_2^2, \cdots, e_i^{n-1}\rangle$, where e_k^s is the k^{th} structural edit operation in $\triangle_s^{s+1}(k)$, $0 < s < n$ and $0 \leq k \leq |\triangle_s^{s+1}(k)|$.* □

3.3 Dynamic Metrics

Reconsider the example in Fig. 1. We observe that different substructures of an XML document might change in different ways. For example, they may change at different frequencies with different *significance*. To quantify the changes to different substructures, we propose a set of *dynamic metrics*. The first metric is called *structure dynamic*. It measures how significantly a substructure has changed from one version to another. The second metric is called *version dynamic*. It measures how frequently a substructure has changed in a sequence of historical versions. DoD (*Degree of Dynamic*) is a metric to measure the significance of the changes of a substructure in the entire history. We elaborate on them in turn.

Definition 6 (Structure Dynamic) *Let T_i and T_{i+1} be the tree representations of two versions of XML documents. Suppose $t \preceq T_i$. The* ***structure dynamic*** *of t from T_i to T_{i+1}, denoted as $N_i(t)$, is defined as:*

$$N_i(t) = \frac{|\triangle_i^{i+1}(t)|}{|t_i \uplus t_{i+1}|}$$

where $|t_i \uplus t_{i+1}|$ is distinct number of nodes in t_i and t_{i+1}. □

Here $N_i(t)$ is the structural dynamic of t from version i to $i+1$. $N_i(t)$ is computed as the percentage of nodes that have changed from T_i to T_{i+1} in t. For example, consider the first two versions shown in Fig. 1(b) and 1(c). We calculate the structure dynamic value for the subtree rooted at node P_2 from version 1 to version 2. Based on the definition, $|\triangle_1^2(P_2)| = 7$, $|P_{2_1} \uplus P_{2_2}| = 12$. Consequently, $N_1(P_2) = 0.58$ (7/12). It can be observed that $0 \leq N_i(t) \leq 1$. If t is inserted or deleted, then the corresponding value of structure dynamic

is 1 since $|\triangle_i^{i+1}(t)| = |t_i \uplus t_{i+1}|$. If t did not change from version i to version $i+1$, then the value of structure dynamic is 0 since $|\triangle_i^{i+1}(t)|$ is 0. It can be inferred that the larger the value of structure dynamic, the more significantly the subtree changed.

Definition 7 (Version Dynamic) *Let $\langle\ T_1, T_2, \cdots, T_n\ \rangle$ be the sequence of n versions of an XML document. Suppose $t \preceq T_j$ $(1 \leq j \leq n)$. The* ***version dynamic*** *of t, denoted as $V(t)$, is defined as:*

$$V(t) = \frac{\sum_{i=1}^{n-1} v_i}{n-1} \ \ where \ \ v_i = \begin{cases} 1, \ if \ |\triangle_i^{i+1}(t)| \neq 0; \\ 0, \ if \ |\triangle_i^{i+1}(t)| = 0; \end{cases}$$

□

Consider the four versions of the XML document in Fig. 1. We calculate the version dynamic value for the subtree rooted at node P_2. Here $n = 4$. Observe that this subtree changed in all the versions. Consequently, $\sum_{i=1}^{3} v_i = 3$, the version dynamic of this substructure is 1 (3/3). It can be observed that $0 \leq V(t) \leq 1$. If t changed in every version in the history, then the version dynamic value is 1. If t did not change in the history at all, then the version dynamic value is 0. Also, it implies that the larger the value of version dynamic, the more frequently the subtree changed in the history.

Definition 8 (Degree of Dynamic) *Let $\langle T_1, T_2, \cdots, T_n\ \rangle$ be the sequence of tree representations of n historical versions of an XML document. Suppose $t \preceq T_j$ $(1 \leq j \leq n)$. The* ***degree of dynamic****, DoD, for t is defined as:*

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

where α is the user-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 value of the subtree is no less than the predefined threshold α. Consider the example shown in Fig. 1. Now, we calculate the DoD value for the subtree rooted at node P_2. Consider the structure dynamic and version dynamic values in the previous example. Suppose the threshold for structure dynamic is set to 0.30, then the value of DoD is 1 (3/3). If the threshold for structure dynamic is set to 0.9, then the corresponding DoD value will be 0 (0/3). It is obvious that, $\forall\ \alpha, 0 \leq DoD(t, \alpha) \leq 1$. The value of DoD implies the significance of the overall changes to the subtree, the larger the value, the more significant the changes are.

Compared with the classic measures, *trend, seasonality, and residue*, used in the time series community, our proposed dynamic metrics more accurately reflects significance of structural changes. For instance, the classic trend components cannot work well when the time series is non-monotonous or nonlinear, while our *version dynamic* can monitor any subtle changes between any consecutive versions. Furthermore, dynamic metrics are more flexible than the classic time series components. For instance, the original sequences, the sequence of support values of certain substructure, can be easily recovered to the exact values, while it is difficult to do so with the classic time series components. Note that the three classic time series components can be derived from the version dynamics, structural dynamic, and degree of dynamic.

3.4 XML Structural Delta Mining

Based on the dynamic metrics defined above, various types of useful knowledge about the changed subtrees can be identified. The problem of XML structural delta mining is to discover knowledge from historical versions of XML structural delta. Formally,

Definition 9 (XML Structural Delta Mining) *Let $\langle T_1, T_2, \cdots, T_n \rangle$ be the tree representations of a sequence of historical versions of an XML document X where T_i is the tree structure of the i-th version of the XML document. Let $\Psi_X = \langle \triangle_1^2, \triangle_2^3, \cdots, \triangle_{n-1}^n \rangle$ be the structural delta sequence of X. Then **XML structural delta mining** on X can be defined by the following function:*

$$\mathcal{R} = XSDMine(\Psi_X, \Gamma)$$

where Γ is a set of constraints on dynamic metrics and $\mathcal{R}$ is a set of structures extracted from X that satisfies the constraints in Γ. □

As mentioned in Section 1, we focus on the following three types of knowledge, *interesting structures*, *association rules*, and *change pattern based classifications*. We elaborate on them in the subsequent sections in turn.

4 Discovering Interesting Structures

In the preceding section, we have introduced a set of dynamic metrics to measure the degree and significance of structural changes to XML documents. In this section, we use these metrics as a foundation to present different types of interesting substructures that can be discovered from XML structural deltas. The interesting substructures include *frequently changing structure, frozen structure, periodic dynamic structure, increasing dynamic structure, decreasing dynamic structure*, and *outlier structure*. Real-life applications

of these structure will be discussed in Section 8. Some preliminary research results related to the discovery of some of these structures are available in [42, 43, 45, 46].

4.1 Frequently Changing Structure

Given a sequence of historical versions of an XML document, we may observe that different substructures change at different frequencies with different significance. *Frequently changing structure* refers to substructures that change *frequently* and *significantly* enough with respect to the user-defined thresholds. Based on the dynamic metrics proposed in the previous section, it is formally defined as follows.

Definition 10 (Frequently Changing Structure) *Let $\langle T_1, T_2, \cdots, T_n \rangle$ be the tree representations for versions of an XML document X. Let the thresholds of structure dynamic, version dynamic, and degree of dynamic be α, β, γ respectively. A structure $t \preceq T_j$ $(1 \leq j \leq n)$ is a* ***frequently changing structure*** *in this sequence iff: $V(t) \geq \beta$ and $DoD(t, \alpha) \geq \gamma$.* □

Observe that *frequent changing structures* defined in terms of both the frequency of the changes and the significance of the changes. To be a *frequently changing structure*, a substructure must change at certain frequency ($V(t) \geq \beta$) and corresponding changes must be significantly enough ($DoD(t, \alpha) \geq \gamma$). Given a sequence of XML documents, *frequently changing structure* mining is to discover all the *frequently changing structure*s according to the user defined thresholds for *version dynamic, structure dynamic* and *degree of dynamic*. For example, consider the motivating example in Fig. 1 again. Suppose the predefined thresholds for structure dynamic, version dynamic, and degree of dynamic are $0.3, 0.5$, and 0.8, respectively. The set of *frequently changing structure*s are the subtrees rooted at nodes *Services* and *Products*. In real life applications, the thresholds for these metrics may be set according to the corresponding domain knowledge and requirements.

Observe that the definition of *frequently changing structure* is different from existing definitions of frequent substructures [18, 21, 37, 9] in two aspects. Firstly, the *frequently changing structure* is defined based on the dynamic metrics, while the existing frequent substructures are defined based on the frequency of concurrence of the substructures in the static XML documents. Secondly, the *frequently changing structure* mining uses *approximate matching* technique [9], while others use *exact matching* [18, 21, 37]. The approximate matching allows slight variation of the substructures, which means that if certain fraction (defined by α) of two substructures is different, then they are matched. The approximation of the matching can be controlled by tuning the threshold of structure dynamic in *frequently changing structure* mining.

4.2 Frozen Structure

Inverse to the *frequently changing structure*, some substructures seldom or never change over time. To identify such kind of substructures, we introduce another type of interesting structure named *frozen structure*. Intuitively, frozen structures refer to the structures that are relatively stable and seldom change in the history. Similar to the *frequently changing structure*s, *frozen structure* can be defined as structures whose values of *structure dynamic* and *version dynamic* do not exceed certain predefined thresholds. Formally,

Definition 11 (Frozen Structure) *Let $\langle T_1, T_2, \cdots, T_n \rangle$ be the tree representations of n versions of an XML document X. Let the thresholds of structure dynamic, version dynamic, and degree of dynamic be α, β, γ respectively. A structure $t \preceq T_j$ $(1 \leq j \leq n)$ is a* ***frozen structure*** *in this sequence iff: $V(t) < \beta$ and $DoD(t, \alpha) < \gamma$.* □

For instance, the substructure within the dotted line in Fig. 1(b) is a frozen structure for any user-defined non-zero thresholds, since this substructure did not change in the history and both values of its version dynamic and structure dynamic are 0. The smaller the values of *structure dynamic* and *version dynamic*, the less significantly and frequently the substructures change. Similarly, a smaller value of *degree of dynamic* indicates a smaller fraction of significant changes among all the versions it has changed. In the *frequently changing structure* mining, the larger the thresholds for the parameters, the more interesting the mining results are. But in *frozen structure* mining, the smaller the thresholds, the more interesting the mining results are.

4.3 Periodic Dynamic Structure

Another type of interesting structure is *periodic dynamic structure*. It refers to the structures that change periodically. Based on the survey of existing periodic sequential pattern mining research [15], a *periodic pattern* is a sequence of items that appears repeatedly in certain sequence. For example, in the sequence $\langle s_1, s_2, s_3, s_1, s_2, s_4, s_1, s_2, s_5 \rangle$, pattern $\langle s_1, s_2 \rangle$ repeated perfectly for three times. In this case, we say $\langle s_1, s_2 \rangle$ is a periodic pattern in this sequence. Some relaxations or restrictions can be integrated to define different types of periodic patterns.

In the context of XML structural delta mining, we treat each basic edit operation as an item and the edit operations that transform the structure of an XML document from one version to another version as an itemset. Consequently, the structural delta sequence can be modelled as a sequence of itemsets. Then, existing periodic sequential pattern mining [15] can be applied to extract periodic edit operations to a particular substructure from versions of an XML

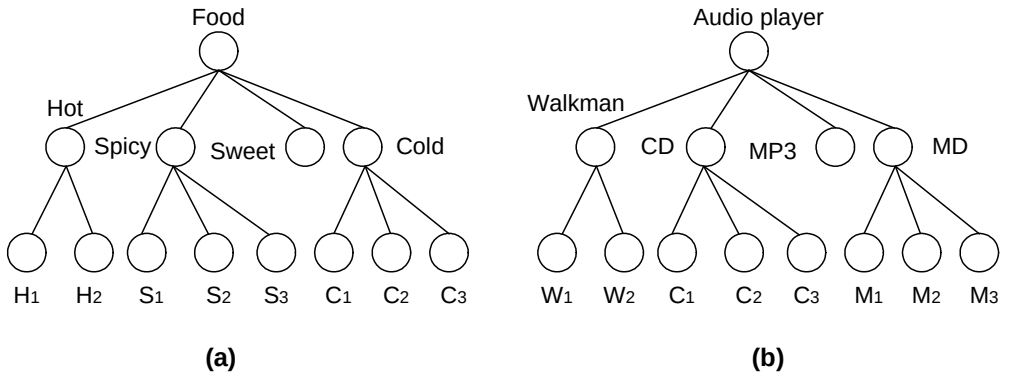


Fig. 3. Example structures.

document. The discovered substructure is referred as *periodic dynamic structure*. We now formally define *periodic dynamic structure*.

We first define the notion of *subsequence* in the context of structural edit script sequence. Recall the definition of structural edit script sequence in Definition 5. A structural edit script sequence $G'_t = \langle e'_1, e'_2, e'_3, \ldots, e'_m \rangle$ is called a *subsequence* of another sequence $G_t = \langle e_1, e_2, e_3, \ldots, e_n \rangle$, denoted as $G'_t \subseteq G_t$, if and only if there exist $1 \leq i_1 \leq i_2 \leq \ldots \leq i_m \leq n$ such that $e'_j = e_{i_j}$ for $1 \leq j \leq m$.

Definition 12 (Periodic Dynamic Structure) *Let $\langle T_1, T_2, \cdots, T_n \rangle$ be the sequence of tree representations of n versions of an XML document X. Let $t \preceq T_j$ where $1 \leq j \leq n$. Let G_t be the structural edit script sequence of t. Given a threshold ν, substructure t is a* ***periodic dynamic structure*** *if there exist any nonempty structural edit script sequence G' such that*

$$\sum_{i=1}^{n} \frac{s_i}{n} \geq \nu \text{ where } s_i = \begin{cases} 1, \text{ if } G' \subseteq G_t; \\ 0, \text{ otherwise;} \end{cases}$$

□

Reconsider the example in Fig. 1. Suppose this *ABC* company releases new products of a particular category every three months. Then, there will be edit operations to the substructure of the particular category that occur repeatedly every quarter. This substructure will be discovered as a periodic dynamic structure. Fig. 3(a) is another example of the periodic dynamic structure. It describes a list of food in a restaurant. As different food are available at different seasons, this structure may change periodically. For instance, the *hot* or *cold* food may be inserted and deleted in every half year. The periodic dynamic structure is proposed to represent substructures where such periodic change patterns reside.

4.4 Increasing and Decreasing Dynamic Structures

Reconsider the example in Fig. 1. It can be observed that more and more *agent* nodes are inserted under product models M_4 and M_5, while other structures do not have such change patterns. These change patterns may indicate that product models M_4 and M_5 are becoming more popular or the company is aggressively promoting these product models. We call such structures as *trend-based dynamic structures*. Basically, there are two types of trend-based dynamic structures: *increasing dynamic structure* and *decreasing dynamic structure*. We now define these structures. We begin by defining the notion of *strength* and *tolerance* which we shall use later for the definitions of these structures.

Definition 13 (Increasing and Decreasing Strength) *Let* $\langle T_1, T_2, \cdots, T_n \rangle$ *be the sequence of* n *versions of an XML document. Suppose* $t \preceq T_j$ $(1 \leq j \leq n)$*. Let*

$$\rho(t) = \frac{\sum_{i=1}^{n-1} v_i}{(n-1) * V(t)}$$

- *Then,* $\rho(t)$ *is called* ***increasing strength****, denoted as* $\rho_{in}(t)$*, if* $v_i = 1$ *when* $\frac{N_{i+1}(t)}{N_i(t)} > 1$*. Otherwise,* $v_i = 0$*.*
- *It is called* ***decreasing strength****, denoted as* $\rho_{de}(t)$*, if* $v_i = 1$ *when* $0 < \frac{N_{i+1}(t)}{N_i(t)} < 1$*. Otherwise,* $v_i = 0$*.* □

Definition 14 (Increasing and Decreasing Tolerance) *Let* $\langle T_1, T_2, \cdots, T_n \rangle$ *be the tree representations for* n *versions of an XML document. Let* $t \preceq T_j$ *be a substructure in the XML document. For* $1 \leq i \leq n-1$*, let*

- $\tau_i(t) = \frac{N_i(t) - N_{i+1}(t)}{N_i(t)}$ *when* $N_i(t) \neq 0$*, otherwise* τ_i *=0; and*
- $\tau'_i(t) = \frac{N_{i+1}(t) - N_i(t)}{N_{i+1}(t)}$ *when* $N_{i+1}(t) \neq 0$*, otherwise* τ'_i *=0.*

Then, ***increasing*** *and* ***decreasing tolerance*** *are defined as* $\tau_{in}(t) = \max\{\tau_1(t), \tau_2(t), \cdots, \tau_{n-1}(t)\}$ *and* $\tau_{de}(t) = \max\{\tau'_1(t), \tau'_2(t), \cdots, \tau'_{n-1}(t)\}$*, respectively.* □

Using the above definitions, we can define *increasing* and *decreasing* dynamic structures as follows.

Definition 15 (Increasing and Decreasing Dynamic Structures) *Let* $\langle T_1, T_2, \cdots, T_n \rangle$ *be the sequence of tree representations for* n *versions of XML documents. Let* $t \preceq T_j$*. Let the thresholds of version dynamic, strength, and tolerance be* β*,* σ *and* κ*, respectively. Then,*

- *A structure* t *is an* ***increasing dynamic structure*** *in this sequence iff:* $V(t) \geq \beta$*,* $\tau_{in}(t) \leq \kappa$*, and* $\rho_{in}(t) \geq \sigma$*.*

- *A structure $t \preceq T_j$ is a* ***decreasing dynamic structure*** *in this sequence iff: $V(t) \geq \beta$, $\tau_{de}(t) \leq \kappa$, and $\rho_{de}(t) \geq \sigma$.* □

The *increasing dynamic structure* is defined based on the predefined thresholds for *version dynamic*, *increasing strength*, and *increasing tolerance*. Here the *increasing strength* is used to reflect how the overall changes of the structure dynamic values from one version to another comply with the increasing pattern. It is the number of times a substructure changed following the increasing change pattern against the total number of times a structure changed. The value of strength is between 0 and 1. A greater value of strength implies that this structure complies better with the increasing pattern. Since it is possible that some of the changes to the structure may not comply with the increasing pattern, *tolerance* is defined to restrict the uncomplying changes. In other words, *tolerance* is used to confine the maximal significance of decreasing change patterns in the history. That is none of the changes in the history of an *increasing dynamic structure* can decrease beyond the *tolerance*. The value of *tolerance* is also between 0 and 1. A greater value of tolerance implies a more relaxed constraint of the decreasing patterns. According to the above definition, to be an *increasing dynamic structure*, the structure should change frequently ($V(t) \geq \beta$). Furthermore, changes to the structure should comply with the increasing pattern in certain number of versions ($\rho_{in} \geq \sigma$). There should be no versions where changes to the structure decreases beyond the tolerance ($\tau_{in}(t) \leq \kappa$). The decreasing dynamic structure can be explained in the similar way.

We now illustrate these two structures with an example. Suppose we have an XML document that describes the *audio player* products, whose tree structure is as shown in Fig. 3(b). There are four types of players *Walkman, CD, MP3,* and *MD*. Assuming that *Walkman* and *CD* are going to be out of date, more and more new models of *MP3* and *MD* are inserted in the file. Consequently, the subtrees rooted at nodes *MP3* and *MD* may be discovered as *increasing dynamic structure*s. Similarly, the *decreasing dynamic structure* is illustrated as follows. Reconsider the structure in Fig. 3(b). Observe that more and more nodes in the subtrees rooted at *Walkman* and *CD* are deleted because these products are becoming out of date. Hence, the two subtrees may be discovered as *decreasing dynamic structures*.

4.5 Outlier Structure

Outlier structures refer to these substructures that have undergone surprising changes with respect to their historical behaviors. Based on the sequences of values of its *structure dynamic* and *version dynamic*, it is possible to construct a model for a substructure that reflects its change patterns if there exists some patterns. Based on the model, we can predict the values of *structure dynamic*

and *version dynamic* for the next version. We propose to build an adaptive model that can be updated based on the sequence of real values over time. To identify these outlier structures a new parameter named *surprise* is introduced to measure the deviation between the real value and the expected value.

Definition 16 (Surprise) *Let $\langle T_1, T_2, \cdots, T_n \rangle$ be the sequence of tree representations of n historical versions of an XML document X. Let $t \preceq T_i$ and M be an adaptive model that describes the changes history of t. The* ***surprise*** *of the changes for structure t from version i to version (i+1) is defined as:*

$$U_i(t) = \frac{|N_{t_i} - \overline{M(N_{t_i})}|}{N_{t_i}} * w_1 + \frac{|V_{t_i} - \overline{M(V_{t_i})}|}{V_{t_i}} * w_2$$

where $\overline{M(N_{t_i})}$ and $\overline{M(V_{t_i})}$ are the expected values of N_{t_i} and V_{t_i} based on M, w_1 and w_2 are two non-negative values that represent the weights of structure dynamic and version dynamic and $w_1 + w_2 = 1$. □

By using the metric of surprise, we can define some interesting substructures that cannot be represented by the substructures defined earlier. Moreover, the weights of structure dynamic and version dynamic in the definition makes it flexible and enable users to modify it with respect to their requirements easily. For example, if the structure dynamic is more important in some scenarios, then the weight of structure dynamic can be increased and vise versa. To identify the structures that have greater values of *surprise*, we introduce the type of structure named *outlier structure*. Formally,

Definition 17 (Outlier Structure) *Let $\langle T_1, T_2, \cdots, T_n \rangle$ be the sequence of tree representations of n versions of an XML document X. Let $t \preceq T_j$ where $1 \leq j \leq n$. Given a new version of the XML document X_{n+1}, t is an* ***outlier structure*** *if $U_n(t) \geq \delta$ where δ is the threshold for surprise.* □

Based on the definition, if the changes to a substructure deviate significantly from the expected values, which is defined by the threshold of surprise, we call this substructure an outlier structure. By varying the threshold of surprise, outlier structures that deviate with different significance can be discovered. Usually, in different applications, the threshold value of surprise can be very different. For example, in some applications, a deviation of 1 percent can be significant, while it may be trivial in other applications. Briefly, the above definition is defined in a flexible way so that users in different areas can efficiently extract the interesting structures that they desired by varying the threshold of surprise.

Considering the examples shown in Fig. 1. If the substructures that were identified earlier as *frequently changing structure, frozen structure, periodic dynamic structure*, or *increasing/decreasing dynamic structure* do not change in the same way as they are predicted to change, then they will be consid-

ered as *outlier structures*. The outlier structures can be any structures that have certain predictable change patterns before they went through some surprising changes. That is, their change patterns (values of structure dynamic and version dynamic) can be predicted with certain confidence. We do not consider random structures that have no predictable change patterns. Consequently, extra attention should be paid to the outlier structures because they may reflect certain mistakes, fraud actions, abnormal changes, or intentional changes.

5 Discovering Association Rules

From the sequence of structural deltas of an XML document, we can discover other types of knowledge such as association rules. For example, consider the example in Fig. 1(b). Whenever the structure of the subtree rooted at the node "Products" was changed, the structure of the subtree rooted at the node "Training" mutated as well. Then, an association rule $Products \rightarrow Training$ (we use the root node to represent a changed subtree) may be extracted with respect to some appropriately specified thresholds. As we shall see in Section 8, knowledge obtained from such rules can be useful in applications such as XML search engine, XML clustering. The reader may refer to [5, 6] where we have reported some preliminary results on mining association rules from structural deltas of historical XML documents.

When discovering association rules between changed subtrees of an XML tree, we observe that whenever a subtree changes, all its ancestor subtrees change as well. For example, in Fig. 1(b), when the subtree rooted at the node "P2" changes, the subtree rooted at the node "Products" changes as well. However, discovering an association rule between the two subtrees does not make any sense since the knowledge in such a rule is too obvious. Hence, rather than mine rules that indicate when some subtrees change, some other subtrees frequently change as well, we can mine rules that indicate when some subtrees change significantly, some other subtrees frequently change significantly as well. That is, we capture not only the *version dynamic* but also the *structure dynamic* of associated subtrees to discover nontrivial knowledge. In this section, we discuss three types of association rules that can be mined from a sequence of structural deltas of an XML document.

5.1 Positive Association Rule Mining

In order to discover association rules between changed subtrees, we need to extend the definitions of *Version Dynamic* (Definition 7) and *Degree of Dynamic* (Definition 8). Currently, the two metrics are defined for a single subtree. For a set of subtrees, they can be extended as follows.

Definition 18 (Version Dynamic of Subtree Set) *Let* $\langle T_1, T_2, \cdots, T_n \rangle$ *be the tree representations of n historical versions of an XML document X. Suppose* $S = \{t_1, \ldots, t_m\}, \forall j \in [1, m], \exists i \in [1, n]$ *s.t.* $t_j \preceq T_i$. *The* ***version dynamic*** *of S, denoted as* $V(S)$, *is defined as:*

$$V(S) = \frac{\sum_{i=1}^{n} v_i}{n-1} \text{ where } v_i = \prod_{j=1}^{m} v_{j_i} \text{ and } v_{j_i} = \begin{cases} 1, \text{ if } |\triangle_j^{j+1}(t)| \neq 0; \\ 0, \text{ if } |\triangle_j^{j+1}(t)| = 0; \end{cases}$$

□

Definition 19 (Degree of Dynamic of Subtree Set) *Let* $\langle T_1, T_2, \cdots, T_n \rangle$ *be the tree representations of n historical versions of an XML document X. Suppose* $S = \{t_1, \ldots, t_m\}, \forall j \in [1, m], \exists i \in [1, n]$ *s.t.* $t_j \preceq T_i$, $N_i(t_j)$ *is the value of structure dynamic of each subtree in S and* $V(S)$ *is the value of version dynamic of S. The* ***degree of dynamic****, DoD, for S is defined as:*

$$DoD(S, \alpha) = \frac{\sum_{i=1}^{n} d_i}{(n-1) * V(S)} \text{ where } d_i = \prod_{j=1}^{m} d_{j_i} \text{ and } d_{j_i} = \begin{cases} 1, if\ N_i(t_j) \geq \alpha \\ 0, if\ N_i(t_j) < \alpha \end{cases}$$

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

Based on the two metrics, we can define a set of subtrees as a frequent pattern if subtrees in the set not only frequently change together but also frequently change significantly when they change together.

Definition 20 (Frequent Pattern) *Given the user-specified minimum structure dynamic* α, *minimum version dynamic* β *and minimum degree of dynamic* γ, *a set of subtrees S=*$\{t_1, t_2, \ldots, t_m\}$ *is a* ***frequent pattern*** *if it satisfies the following two conditions:*

- *Version dynamic of the set is no less than the user-specified threshold, V(S)* $\geq \beta$.
- *Degree of dynamic of the set is no less than the user-specified threshold, DoD(S,* α*)* $\geq \gamma$. □

Then we can derive association rules from frequent patterns. The metric *confidence* is defined similarly as in classical association rule mining. It measures the strength of a discovered association rule. Basically, it reflects the conditional probability that subtrees in the consequent of a rule change significantly when subtrees in the antecedent change significantly.

Definition 21 (Confidence) *Given a frequent pattern S=*$\{t_1, \ldots, t_m\}$, *let X be a subset of subtrees in pattern S, let Y be the non-empty subset of remaining*

subtrees in S. ***Confidence*** *of association rule* $X \Rightarrow Y$ *is:*

$$Confidence(X \Rightarrow Y) = \frac{V(S) * DoD(S, \alpha)}{V(X) * DoD(X, \alpha)}$$

□

If subtrees in Y change significantly enough every time when subtrees in X change significantly, then the *confidence* of the rule $X \Rightarrow Y$ will be one; if subtrees in Y never change significantly enough when subtrees in X change significantly, then the *confidence* of the rule $X \Rightarrow Y$ will be zero.

Definition 22 (Positive Association Rule) *Given the user-specified minimum confidence* ϕ*, let* X*,* Y *be two subtree sets s.t.* $X \cap Y = \emptyset$*.* $X \Rightarrow Y$ *is a* ***positive association rule*** *if it satisfies the following conditions: 1)* $X \cup Y$ *is a frequent pattern 2)* $confidence(X \Rightarrow Y) \geq \phi$*.* □

For example, consider the example in Fig. 1(b) again. Although whenever the subtree rooted at the node "P2" changes, the subtree rooted at the node "Products" changes as well, we may not discover an association rule between them because when the former subtree change significantly, the latter may not change significantly as well.

5.2 Negative Association Rule Mining

A positive association rule indicates that when some subtrees change significantly, some other subtrees change significantly as well with certain confidence. Conversely, a negative association rule indicates that when some subtrees change significantly, some other subtrees rarely change significantly with certain confidence. In order to discover negative association rules between changed subtrees, we need a metric *Interest*, as in classical negative association rule mining, to measure the dependence between two subtree sets. Given two variables a and b, the *Interest* between them can be defined as $Interest(a, b) = \frac{p(a \cup b)}{p(a)p(b)}$, where $p(a)$ is the probability of variable a. If $Interest(a, b)$ is greater than one, variable b is positively dependent on variable a. Otherwise, variable b is negatively dependent on variable a. Similarly, we can define the *Interest* between two set of subtrees as follows.

Definition 23 (Interest) *Given two sets of subtrees* X *and* Y*,* $V(X)$*,* $V(Y)$ *and* $V(X \cup Y)$ *are the version dynamic of subtree sets* X*,* Y *and* $X \cup Y$ *respectively. The* ***interest*** *between the two sets is,*

$$Interest(X, Y) = \frac{V(X) - V(X \cup Y)}{V(X)(1 - V(Y))}$$

□

The numerator is the fraction of the versions when subtrees in X changed whereas subtrees in Y did not change. The denominator is the product of the fraction of versions when subtrees in X changed and the fraction of versions when subtrees in Y did not change. If subtree set Y negatively depends on subtree set X, the value of $Interest(X,\ Y)$ should be greater than some user-specified minimum *Interest* that is greater than 1.

Correspondingly, the confidence of a negative association rule can be defined as follows.

Definition 24 (Confidence) *Given a frequent pattern S=$\{t_1, \ldots, t_m\}$, let X be a subset of subtrees in pattern S, let Y be the non-empty subset of remaining subtrees in S.* ***Confidence*** *of association rule $X \Rightarrow \neg Y$ is:*

$$Confidence(X \Rightarrow \neg Y) = \frac{V(X) * DoD(X, \alpha) - V(S) * DoD(S, \alpha)}{V(X) * DoD(X, \alpha)}$$

□

Thus, the negative association rules that can be mined from a sequence of XML structural deltas can be defined as follows.

Definition 25 (Negative Association Rule) *Given user-specified minimum structure dynamic α, minimum version dynamic β, minimum degree of dynamic γ, minimum Interest θ, and minimum confidence ϕ, let X, Y be two sets of subtrees, s.t. $X \cap Y = \emptyset$. $X \Rightarrow \neg Y$ is a* ***negative association rule*** *if it satisfies the following conditions. 1) $V(X \cup Y) < \beta$; 2) $DoD(X \cup Y) < \gamma$; 3) $Interest(X,\ Y) \geq \theta$; 4) $Confidence(X \Rightarrow \neg Y) \geq \phi$.* □

Hence, the semantic meaning of a negative association rule is that the subtrees in the antecedent and consequent of the rule rarely change together. Even if they change together, they rarely change significantly. The change of subtrees in the consequent negatively depends on the change of subtrees in the antecedent. And when subtrees in antecedent change significantly, subtrees in consequent usually change slightly with certain confidence.

5.3 Specialized Association Rule Mining

Moreover, association rules can also be extracted between the interesting structures discussed in the previous section. For example, we may discover an association rule between a set of *increasing dynamic structures* and a set of *decreasing dynamic structures*, which means when subtrees in the antecedent of the rule change more and more significantly, subtrees in the consequent of the rule change slightly gradually. We call the association rules between the particular interesting structures as *specialized association rule*. We do not

elaborate on the details of the definition of specialized association rule as it is similar to the association rules we defined above.

6 Structure Change Pattern Based Classification

Besides the two types of knowledge discussed above, we can also perform classification based on the XML structural deltas.

As we reviewed in Section 2, XRULE [41] is a classifier that classifies XML documents according to the discriminatory structures extracted from each class of XML documents by some algorithm of frequent subtree mining. However, only the structures embedded in the static XML documents are considered. To classify XML documents with higher precision, the evolutionary patterns of XML structure can be integrated into the feature space on which the classifier operates. For example, suppose there are two XML documents describing the Web site map of an IT company and a consultant company respectively. Both documents contain a subtree of "Products" and a subtree of "Training". According to XRULE, the two documents can be grouped in the same class as they share some common substructures. However, for the XML file describing the information of an IT company, the structure of the subtree "Training" may mutate frequently once the structure of the subtree "Products" changes since training courses are provided for new products. For the XML file describing the information of a consultant company, the subtree of "Products" may contain some financial solutions while the subtree of "Training" may contain some training course for careers. Then, when the structure of the subtree "Products" changes, the structure of the subtree "Training" may not change simultaneously as they do not have any hidden association. Hence, by considering the change patterns of the substructures in XML documents, the two documents can be distinguished as they describe two company sites of different types.

Basically, the idea of classification on XML structural deltas is to relate the presence of particular change patterns of substructures to the likelihood of belonging to a particular class. The classifier based on change patterns of substructures can be a general model that classify all types of evolutional data that are structured or semi-structured. Such techniques can be used not only for XML documents but also for any types of structured data that is dynamic, such as web log data, chemical data, and biological data etc.

7 Research Issues

In the preceding section, we have introduced various types of knowledge that can be discovered using XML structural delta mining technique. In this sec-

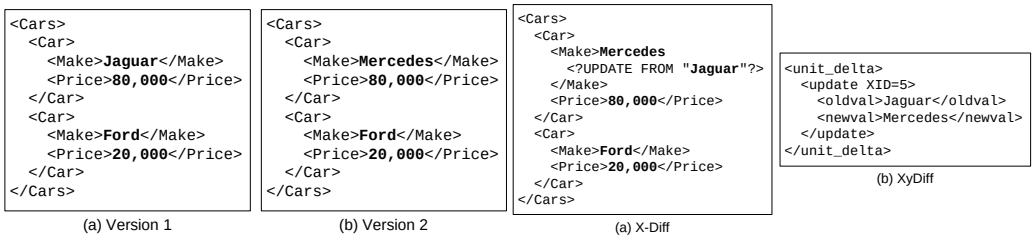
tion, we present the key research issues that needs to be addressed to realize these various mining efforts.

7.1 Duration of Real Data Collection

In order to make the extracted knowledge convincible and accurate, the data collection is expected to go on for a significant time period. However, finding appropriate time duration for data collection is a challenging task as it depends on several parameters such as characteristics of source and application, frequency of changes, number of available versions, type of knowledge to be discovered, etc. It is an open problem to estimate the appropriate duration for different applications. Such estimation will reduce the resource consumption of the data mining process and maximize the quality of results generated by different XML structural delta mining techniques.

7.2 Determining Schedules for Structural Delta Generation

XML data mining research has largely assumed that XML documents are static. However, in reality the documents are rarely static. XML structural delta mining aims to extract knowledge by analyzing the structural evolution pattern of history of XML documents. One key issue is to generate a sequence of structural deltas that can be fed to the mining engine for pattern extraction. To improve the accuracy of the mining process, ideally we should be able to harness the complete set of structural deltas during a particular time period. As these documents often reside in autonomous and remote sources, it is not realistic to assume that structural deltas will be automatically propagated to the mining engine. Hence, finding appropriate schedules for change detection is important so that set of structural deltas used for the mining task is complete and contain sufficient data to guarantee reliable and accurate mining results. Defining schedules for structural delta detection is a challenging task, because the rate of change of the XML document may vary drastically from document to document. Note that the naive solution of polling the sources periodically is not an efficient process for two reasons. First, we may miss some of the intermediate structural deltas as frequency of change of a particular document/source may not match the polling frequency. Second, this approach may unnecessarily overload the mining process by attempting to detect changes when the document have not changed. A more efficient way of solving this problem is to predict the rate of change of documents by analyzing the change history. Recently, Ipeirotis et al. have proposed such schedule prediction technique for web databases by using survival analysis technique [19]. However, this is an open research problem in the context of XML structural delta mining.



(a) Sample documents. (b) Edit scripts.

Fig. 4. Semantic-conscious change detection.

7.3 Efficient and Scalable Structural Change Detection

XML structural delta mining is expected to extract novel knowledge that existing data mining techniques for structured data fail to discover. Experimental results in [46, 42] have verified that XML structural delta mining can indeed produce novel knowledge and the size of XML structural delta sequence is substantially smaller than the original XML data sequence. However, we also observed that the XML change detection process is one of the most expensive tasks in the whole mining process. This is further aggravated by parsing the XML documents repeatedly. Although there are many XML parsers available, it has been acknowledged that the parsing process is the most expensive part of XML data management [30]. When the number of versions is large and the size of the XML documents is also very large, the cost of parsing the XML documents will consume a lot of resources in the XML structural delta mining system. Currently, many change detection systems have been designed for detecting changes to XML documents [7, 35]. However, most of them are not efficient enough and have certain limitations. For instance, the X-Diff [35] algorithm cannot handle *very large* XML documents. Consequently, two key issues need to be addressed. First, we need to rethink and optimize existing change detection systems for our XML structural delta mining. Second, there is a pressing need to improve the efficiency and scalability of the parsing mechanism.

7.4 Scalable and Efficient Mining Algorithms

One of the most important concerns in designing a data mining algorithm is the scalability and efficiency. Similarly, for our XML structural delta mining, we need to design efficient and scalable algorithms that can handle *very long* sequence of historical XML documents that are *very large*. In the context of XML structural delta mining, the algorithm should be scalable with respect to not only the size of the XML delta sequence, but also to various values of different dynamic metrics such as *degree of dynamic, structure dynamic*, and *version dynamic* in discovering the various types of interesting structures,

association rules, and classifications. To address the scalability, a relational database based approach may be more competitive than a memory based one. However, in terms of efficiency, a memory based approach is more competitive than the relational database based approach. To sum up, there should be a trade-off between the scalability and the efficiency. In addition, another issue should be considered. Since the dataset is accumulated over time, the characteristics and knowledge hidden behind the data collection may vary accordingly. This feature makes it desirable to design an incremental data mining algorithm that can use the previously discovered results and the current changes to update the extracted knowledge.

7.5 Semantic-conscious Algorithms

It is important that algorithms designed for XML structural delta mining are *semantic-conscious*. Otherwise, these algorithms may produce results that may not be accurate or reliable. Let us elaborate with an example. It is evident that detecting structural deltas is a fundamental activity in the XML structural delta mining process. Consider the old and new versions of an XML document in Figure 4(a). The results returned by the change detection techniques such as X-Diff and XyDiff are depicted in Figures 4(b). Observe that both the algorithms detect that the `Make` of the first car is updated from *Jaguar* to *Mercedes*. However, in reality the car whose `Make` is *Jaguar* definitely cannot be updated to *Mercedes*. Hence, the results generated by X-Diff and XyDiff are semantically incorrect. The correct types of changes that should be detected here are *deletion* of the first `car` element in Figure 4(a)(i) and *insertion* of a new car (the first `car` element in Figure 4(a)(ii)). Observe that as the change is detected as an update, it will be not part of the structural delta. As a result, semantically incorrect deltas will be propagated to the XML structural delta mining engine which may eventually produce unreliable output! Consequently, it is important to design change detection as well as data mining algorithms that exploit the semantic constraints of the data. By incorporating such semantic knowledge in a general-purpose data mining algorithm,i t is possible to increase the performance of the algorithm.

7.6 Unified Mining Framework

Since data from many fields, such as Web log data and biological data, can be represented in XML format and there are different types of knowledge that can be extracted, it is desirable to design a unified mining framework that is suitable for different applications in different fields and can discover various types of knowledge. For example, different users may be interested in different types of knowledge. The unified mining framework should be able to provide different types of knowledge to different users with regards to their

requirements. That is, for the same data, the mining framework should contain different data mining techniques so that it can provide personalized knowledge for all types of possible users. Similarly, for applications in different areas, the background knowledge will be different. The unified mining framework should be able to incorporate the corresponding background knowledge into the mining process.

8 Applications

As mentioned before, there are many potential applications for the XML structural delta mining. In this section, we present some of the typical applications for the three types of knowledge we discussed earlier. Note that by no means we claim this list to be exhaustive.

8.1 Applications of Interesting Structures

Efficient Change Detection for Very Large XML Documents: One of the major limitations of existing XML change detection systems [7, 35] is that they are not scalable for very large XML documents. The state-of-the-art XML change detection algorithms [35, 7] attempt to compare the entire XML trees to detect changes. For instance, X-Diff [35] requires entire *Xtree* to be memory resident. An *Xtree* is typically much larger than its XML document. Thus, the scheme is not scalable for large XML documents. With the above mentioned interesting structures, the scalability and efficiency of XML change detection system can be improved. For instance, if one can discover substructures that change frequently (*frequently changing structure*s) and those that do not (*frozen structures*), then he/she can use such knowledge to detect changes for different parts of the documents at different frequencies based on their change patterns. For example, we can detect changes to *frequently changing structure*s more frequently than detect changes to structures that change infrequently. Moreover, we may ignore frozen substructures during change detection, as most likely they undergo no changes. For example, the substructure rooted at node *About ABC* in Fig. 1 is a frozen structure, so this portion of the document can be ignored during the change detection process. We believe that this will improve the efficiency of change detection process, especially for *very large* XML documents.

Efficient XML indexing: As we know that one of the key issue of XML indexing is to identify the ancestor and descendant relationship quickly. To this end, different numbering schemes have been proposed [26, 20]. Li and Moon proposed a numbering scheme in XISS (XML Indexing and Storage System) [26]. The XISS numbering scheme uses an *extended preorder* and a *size*. The *extended preorder* allows additional nodes to be inserted without reordering

and the *size* determines the possible number of descendants. More recently, XR-Tree [20] was proposed to index XML data for efficient structural joins. Compared with the XR-tree [20], XISS numbering scheme is more flexible and can deal with dynamic updates of XML data more efficiently. Since extra space is reserved in the *extended preorder* to accommodate future insertions, global reordering is not necessary until all the reserved space is consumed. However, Li and Moon did not highlight on how much extra space should be allocated. Allocating too small reversed space will lead to the ineffectiveness in maintaining the numbering scheme, whereas allocating too much extra space will lead to too large numbers being assigned to nodes in a large XML document. Moreover, in the XISS approach, the gaps are equally allocated, while in practice different parts of the document change with different significance. Based on our mining results, the numbering scheme can be improved by allocating the gaps in a more intelligent manner. For example, for the parts of structure that change frequently and significantly, larger gaps are allocated while for frozen structures, smaller gaps can be reserved. By using this strategy, the numbering scheme should be more efficient in terms of both index maintenance and space allocation.

Dynamic-Conscious XML Caching: Existing XML query pattern-based caching strategies focus on extracting the set of frequently issued *Query Pattern Trees* (QPTs) based on the number of occurrences of the QPTs in the history. Each occurrence of the same QPT is considered equally important for the caching strategy. However, the same QPT may occur at different timepoints in the history of XML queries. This temporal feature can be used to improve the caching strategy. Let us elaborate on it further.

Given an XML data repository, a collection of XML queries are issued by different users over a period of time. These queries can be represented as a collection of QPTs. Each QPT consists of a set of *rooted query paths* (RQPs). A RQP in a QPT is a path starting from the root. The *support* value of a RQP in a particular week represents the number of occurrences of a RQP against the total number of QPTs issued in the specified week. Note that the support values can change in different ways for different RQPs in the history.

Based on the above observation, we can use two dynamic metrics called *frequency* and *degree of dynamic* (DoD), to summarize the support values of the RQPs and the changes to the RQPs. *Frequency* is used to measure the average support values and the standard deviations among the historical support values. *Degree of dynamic* (DoD) is used to measure the *aggregated* changes in the historical support values. Based on these two metrics, an algorithm can be designed to discover two groups of interesting RQPs from the historical QPTs, called the *frequent conserved query paths (FCQP)* and the *infrequent conserved query paths (ICQP)*. An RQP P is called *conserved query path* if and only if the standard deviation of the history of support values of P is no

greater than the *deviation threshold* and the degree of dynamic is no greater than the *DoD threshold*. A conserved query path is *frequent* (FCQP) if and only if the mean value of the sequence of support values in the history is no less than the *minimum mean threshold*. Similarly, a conserved query path is *infrequent* (ICQP) if and only if the mean value is no greater than the *maximum mean threshold*.

Observe that in the frequent and infrequent conserved query paths, the supports of RQPs do not change significantly (conserved) and their support values can be potentially large or small. We can use this concept to devise a more efficient dynamic-conscious caching strategy by ranking the set of extracted conserved query paths using a rank metric. The idea is to cache the results for the FCQPs with the largest rank scores by replacing the cache results of the ICQPs with the smallest rank scores. In [44], we have proposed such caching strategy. Our preliminary experimental results show that the dynamic-conscious caching strategy outperforms the existing XML query pattern tree-based caching strategies.

Semantic meaning extraction: Based on the frequency, significance, and type of the changes, some semantic meaning can be extracted from the interesting structures with related domain knowledge. For example, suppose we find out that the substructure rooted at node *Training* is a frequently changing structure with more and more subtrees inserted with similar labels as shown in Fig. 1. Then, it can be inferred that the service *Training* is becoming more and more popular or profitable. Certainly, semantic meaning can also be extracted from other types of structures. The basic idea is to incorporate some meta data (such as labels of the edges and nodes, types of changes etc) into the interesting structures to get the semantic implications. For instance, consider the structure in Fig. 3(a). Assume that similar changes (insertion and deletion) occurred periodically to this structure. Incorporated with relevant domain knowledge, labels of the nodes, and types of changes, we may infer that certain food are popular in that place during certain period of time. The semantic meaning extracted from the structural delta mining results can be widely used in e-commerce such as monitoring and predicting the competitors' strategies.

8.2 Applications of XML Structural Delta Association

Structure-based Document Clustering: Clustering XML documents based on the structures embedded in documents is proposed in [34]. However, it may not be accurate enough to cluster according to structures existing in the snapshot data only. In some cases, the evolutionary patterns of the structures can distinguish documents with higher precision. For example, consider the example in Section 6 again. If the associations between the changes to substructures

are taken into account, the documents can be distinguished as they describe different types of company. Hence, association rules mined from XML structural deltas can be used to improve the accuracy of clustering.

Semantic XML Search Engine: When substructures frequently change together, it is very likely that the objects represented by the substructures have underlying semantic correlation, such as the "Products" and "Training" in our motivation example. This kind of knowledge can be used by semantic XML search engine [8], which returns semantically related document fragments that satisfy users' queries. When determining whether elements are semantically related or not, the authors heuristically claim that if two nodes share the same label, they represent two different entities. Thus, content under the two nodes are unrelated. For example, the courses under the "Training" node in Fig. 1(b) may have the same label (they may be different in their attribute values). According to [8], all these courses are unrelated. However, some courses may be semantically related if they are provided for a same product. When one of the courses changes, the other courses very likely change as well. Hence, association rules mined from XML structural deltas can be used here to identify semantically related elements. The knowledge discovered from the evolution of the objects is more convincible than the heuristics.

8.3 Applications of Change Pattern Based Classification

XML Structural Summary Maintenance: XML structural summary is an index structure that stores the mapping from index nodes to data nodes [14]. An XML structure summary can improve the path evaluation performance by pruning the search space significantly. Given a collection of XML documents with different structures, we can group them according to their structure characteristics and create structural summary for each class. However, consider the dynamic nature of XML documents, the constructed structural summary should be updated with the changes to structures of XML documents. If only the structures embedded in snapshots of XML documents are considered in classification, the update process can be quite expensive as documents in the same class may experience different change patterns. However, if the change patterns of XML structures are taken into account in classification, the process of updating structural summary can be expected to be more efficient since documents in the same class very likely change similarly.

9 Conclusions

XML data mining research has largely assumed that XML documents are static. However, in reality the documents are rarely static. In this paper, we propose a novel research problem called *XML structural delta mining*. The

objective of XML structural delta mining is to discover knowledge by analyzing structural evolution pattern of history of XML documents. Rather than discussing solutions to a specific mining problem, we present the *vision* of our XML structural delta mining framework, with the hopes that others in the community will explore further on specific problems in this arena. Currently, we are studying some of these problems. We believe that XML structural delta mining is an important new application area for data mining, combining commercial interest with intriguing research questions.

References

[1] T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, and S. Arikawa. Efficient substructure discovery from large semi-structured data. In *Proceedings of the 2nd SIAM International Conference on Data Mining*, pages 158–174, 2002.

[2] M. Blaschka, C. Sapia, and G. Hofling. On schema evolution in multidimensional databases. In *Proceedings of the First International Conference on Data Warehousing and Knowledge Discovery*, pages 153–164, London, UK, 1999. Springer-Verlag.

[3] B. Bobel, J. Eder, C. Koncilia, T. Morzy, and R. Wrembel. Creation and management of versions in multiversion data warehouse. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 717–723, New York, NY, USA, 2004. ACM Press.

[4] D. Braga, A. Campi, S. Ceri, M. Klemettinen, and P. L. Lanzi. A tool for extracting XML association rules. In *Proceedings of IEEE International Conference on Tools with Artificial Intelligence ICTAI*, pages 57–65, 2002.

[5] L. Chen, S. S. Bhowmick, and L.-T. Chia. Mining association rules from structural deltas of historical XML documents. In *Proceedings of the 8th Pacific Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, Sydney, Australia, 2004.

[6] L. Chen, S. S. Bhowmick, and L.-T. Chia. Mining maximal frequently changing subtree pattern from XML documents. In *Proceedings of the 6th International Conference on Data Warehousing and Knowledge Discovery (DaWaK)*, Zaragoza, Spain, 2004.

[7] G. Cobena, S. Abiteboul, and A. Marian. Detecting changes in XML documents. In *Proceedings of ICDE International Conference on Data Engineering*, 2002.

[8] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. Xsearch: A semantic search engine for XML. In *Proceedings of the 2003 International Conference of Very Large DataBase*, 2003.

[9] D. J. Cook and L. B. Holder. Graph-based data mining. *IEEE Intelligent Systems*, 15(2):32–41, 2000.

[10] Curbera and D. A. Epstein. Fast difference and update of XML documents. In *Proceedings of XTech*, 1999.

[11] L. Dehaspe, H. Toivonen, and R. D. King. Finding frequent substructures in chemical compounds. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 30–36, 1998.

[12] M. Deshpande, M. Kuramochi, and G. Karypis. Frequent sub-structure-based approaches for classifying chemical compounds. In *Proceedings of IEEE ICDM International Conference on Data Mining*, 2003.

[13] V. Ganti, J. Gehrke, and R. Ramakrishnan. DEMON: Mining and monitoring evolving data. *Knowledge and Data Engineering*, 13(1):50–63, 2001.

[14] R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *Proceedings of the 1997 International Conference of Very Large DataBase*, 1997.

[15] J. Han, G. Dong, and Y. Yin. Efficient mining of partial periodic patterns in time series database. In *Proceedings of ICDE International Conference on Data Engineering*, pages 106–115, Sydney, Australia, 1999.

[16] J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraph in the presence of isomorphism. In *Proceedings of IEEE ICDM International Conference on Data Mining*, 2003.

[17] J. Huan, W. Wang, A. Washington, J. Prins, R. Shah, and A. Tropshas. Accurate classification of protein structural families using coherent subgraph analysis. In *Proceedings of PSB*, 2004.

[18] A. Inokuchi, T. Washio, and H. Motoda. An apriori based algorithm for mining frequent substructures from graph data. In *Proceedings of European Conference on Principles and Practice of Knowledge Discovery in Databases PKDD*, pages 13–23, 2000.

[19] P. Ipeirotis, A. Ntoulas, J. Choo, and L. Gravano. Modeling and managing content changes in text databases. In *Proceedings of ICDE International Conference on Data Engineering*, 2005.

[20] H. Jiang and H. Lu. XR-Tree: Indexing XML data for efficient structural joins. In *Proceedings of ICDE International Conference on Data Engineering*, 2003.

[21] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Proceedings of IEEE International Conference on Data Mining ICDM*, pages 313–320, 2001.

[22] E. Leonardi and S. S. Bhowmick. Detecting changes on unordered XML documents using relational databases: A schema-conscious approach. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management (ACM CIKM 2005)*, 2005.

[23] E. Leonardi, S. S. Bhowmick, T. Dharma, and S. Madria. Detecting content changes on ordered XML documents using relational databases. In *Proceedings of the 15th International Conference on Database and Expert Systems Applications (DEXA 2004)*, 2004.

[24] E. Leonardi, S. S. Bhowmick, and S. Madria. Detecting changes on large

unordered XML documents using relational databases. In *Proceedings of the 9th International Conference on Database Systems for Advanced Applications (DASFAA 2005)*, 2005.

[25] H. P. Leung, F. L. Chung, and S. C. Chan. A new sequential mining approach to XML document similarity computation. In *Proceedings of the 7th Pacific-Asia Conference on Knowledge Discovery and Data Mining, Seoul, Korea*, 2003.

[26] Q. Li and B. Moon. Indexing and querying XML data for regular path expressions. In *The VLDB Journal*, pages 361–370, 2001.

[27] W. Lian, D. W. Cheung, N. Mamoulis, and S. M. Yiu. An efficient and scalable algorithm for clustering XML documents by structure. In *IEEE Transactions on Knolwedge and Data Engineering, vol.16, no.1*, 2004.

[28] B. Liu, W. Hsu, and Y. Ma. Discovering the set of fundamental rule changes. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 335–340. ACM Press, 2001.

[29] T. Morzy and R. Wrembel. On querying versions of multiversion data warehouse. In *Proceedings of the 7th ACM international workshop on Data warehousing and OLAP*, pages 92–101, New York, NY, USA, 2004. ACM Press.

[30] M. Nicola and J. John. XML parsing: A threat to database performance. In *Proceedings of ACM International Conference on Information and Knowledge Management*, pages 175–178, 2003.

[31] A. Termier, M.-C. Rousset, and M. Sebag. Treefinder: a first step towards XML data mining. In *Proceedings of International Conference on Data Mining ICDM*, 2002.

[32] M. Theobald, R. Schenkel, and G. Weikum. Exploiting structure, annotation, and ontological knowledge for automatic classification of xml data. In *Proceedings of the 6th Internation Workshop on the Web and Databases (WebDB)*, 2003.

[33] C. Wang, M. Hong, J. Pei, H. Zhou, W. Wang, and B. Shi. Efficient pattern-growth methods for frequent tree pattern mining. In *Proceedings of the 8th Pacific-Asia Knowledge Discovery and Data Mining*, 2004.

[34] K. Wang and H. Liu. Discovering structural association of semistructured data. In *IEEE Transaction of Knowledge and Data Engineering, vol.12*, pages 353–371, 2000.

[35] Y. Wang, D. J. DeWitt, and J.-Y. Cai. X-Diff: An effective change detection algorithm for XML documents. In *Proceedings of ICDE International Conference on Data Engineering*, 2003.

[36] Y. Xiao, J. F. Yao, Z. Li, and M. H. Dunham. Efficient data mining for maximal frequent subtree. In *Proceedings of the 3rd International Conference on Data Mining*, page 379, 2003.

[37] X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In *Proceedings of IEEE International Conference on Data Mining ICDM*, pages 721–724, 2002.

[38] X. Yan and J. Han. Closegraph: Mining closed frequent graph patterns. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003.

[39] L. H. Yang, M. L. Lee, and W. Hsu. Efficient mining of XML query patterns for caching. In *Proceedings of Very Large Database VLDB*, pages 69–80, 2003.

[40] M. J. Zaki. Efficiently mining frequent trees in a forest. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 71–80, 2002.

[41] M. J. Zaki and C. C. Aggarwal. XRules: An effective structural classifier for XML data. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003.

[42] Q. Zhao and S. S. Bhowmick. Mining history of changes to web access patterns. In *Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, Pisa, Italy, 2004.

[43] Q. Zhao and S. S. Bhowmick. FASST mining: Discovering frequently changing semantic structure from versions of unordered xml documents. In *Proceedings of 10th International Conference on Database Systems for Advanced Applications*, 2005.

[44] Q. Zhao, S. S. Bhowmick, and G. Le. Mining conserved XML query paths for dynamic-conscious caching. In *Proceedings of CIKM*, 2005.

[45] Q. Zhao, S. S. Bhowmick, and S. Madria. Discovering pattern-based dynamic structures from versions of unordered XML documents. In *Proceedings of the 6th International Conference on Data Warehousing and Knowledge Discovery (DaWaK)*, Zaragoza, Spain, 2004.

[46] Q. Zhao, S. S. Bhowmick, M. Mohania, and Y. Kambayashi. FCS mining: Discovering frequently changing structures from historical XML structural deltas. In *Proceedings of ACM CIKM International Conference on Information and Knowledge Management*, 2004.