Change Detection of XML Documents Using Signatures

Latifur Khan, Lei Wang and Yan Rao
Department of Computer Science
University of Texas at Dallas
Richardson, TX 75083-0688
Email: [lkhan, leiwang, yanrao]@utdallas.edu

Abstract

XML is an emerging standard for the representation and exchange of Internet data. The characteristics of XML, tree-structured (i.e. a collection of nodes) and self-descriptive, facilitate the detection of changes in an XML document in minute detail and at a finer grain than obtainable at the document level. There is at present no really effective method for detecting these changes. We wish to propose such a method here. Rather than inspecting all nodes between two versions of XML documents, we propose an effective algorithm, called top-down, which will detect changes in XML documents by exploring a subset of nodes in the tree. We would like to be certain that if a leaf node changes the algorithm will detect these changes, not only by inspecting the node itself, but also its parent node, grand parent node, and so on. For this, a signature for each node will be constructed which is basically an abstraction of the information stored in a node. There are several ways we can construct such a signature. We will choose exclusive-or (XOR) to construct node signatures which will prevent a user from getting irrelevant information/change and make certain that the user does not miss relevant information. Note that for the web, along with being able to access huge quantities of information, the relevancy of information/change is more important than missing of relevant information/change. For this, in this paper we propose an automatic change detection algorithm which will identify changes between two versions of an XML document based on these signatures. Our proposed algorithm will traverse the least number of nodes necessary to detect these changes. We demonstrate that our algorithm outperforms the traditional algorithm which exhaustively searches the entire space. We will also demonstrate analytically and empirically that the miss of relevant change is within tolerable range.

1. Introduction

The availability of data on the web is constantly expanding. Most of this data is semi-structured, consisting of text (HTML) and multimedia (sound, video, image). Overall, this data constitutes the largest body of information ever accessible to any individual. A major evolution is now occurring that will dramatically simplify the task of developing applications based on this data. This is the coming Extended Markup Language (XML). XML is becoming the new standard for semi-structured data exchange over the Internet. Because of XML, it is becoming possible to deliver higher level services which are difficult or impossible to support through current HTML technology.

The characteristics of XML, tree-structured (i.e. a collection of nodes) and self-descriptive, facilitate some form of automatic semantic integration. Furthermore, this self-descriptive characteristic further facilitates the detection of changes in an XML document in minute detail and at a finer grain than obtainable at the document level. The detection of changes in an XML document can be achieved at the element level. Let us assume that an XML document consists of a set of catalogs for various categories. Suppose a user sends a request that he be notified when there is a sale in a particular catalog under category electronic. In addressing this request, XML technology provides a major opportunity for changing the face of the web in a fundamental way. Web users are not only interested in the current values of documents but may also be interested in their future modifications. Regarding these, we need to address the following two questions. First, we need to be able to detect the relevant changes in XML documents. Second, it is necessary to notify the user effectively regarding the most recent changes. In this paper we address the first question. The second question will be addressed as a part of future work.

In approaching the first question, rather than inspecting all nodes between two versions of XML documents, we propose an effective algorithm, called *top-down*, which will detect changes in XML documents by exploring a subset of nodes in the tree. In other words, the top-down algorithm prunes the search space starting by comparing values in the root nodes of the two

versions. Next, immediate children nodes of the root nodes will be compared. In this procedure, we would like to make sure that if a leaf node changes the algorithm can detect the change not by inspecting the node itself but also its parent node, grand parent node, and so on. For this purpose, signature of nodes will be constructed. The signature is basically an abstraction of the information stored in each node. The signature of an interior node can be constructed using either OR or Exclusive-or (XOR) of all descendant's signatures. In IR for document filtering, the signature is used based on OR which may allow the retrieval of irrelevant documents to query adversely affecting precision [16]. Furthermore, the number of 1 in signature is assumed to be very small as compared to the length of the signature [19]. On the other hand, using XOR it is not possible to get any irrelevant information/change; however, some relevant information/change may be missed adversely affecting recall. Note that in the web along with the availability of huge quantities of information, the relevancy of information/change (precision) is more important as compared to missing of relevant information/change (recall). Therefore, we consider change detection of XML documents using signatures based on XOR.

To reduce the number of nodes which need to be compared between two versions, it will only be necessary to compare node signatures in the old version with corresponding node signatures in the new version. The top-down algorithm follows depth first search exploration. One important question concerns how we can determine which node in the old version corresponds to a node in the new version. For this, we propose a method to determine an identifier for each node from its nested position in XML documents.

The main contributions of this work are as follows: First, we propose an automatic change detection algorithm between two versions of XML documents based on node signatures. Second, we use XOR to construct node signatures. Due to the employment of XOR a user will not get any irrelevant change; however, the user may miss some change which is relevant. We demonstrate analytically and empirically, however, that this miss is very low. Finally, compared to the traditional algorithm, our proposed algorithm traverses fewer nodes in the tree to detect changes between two versions. We also demonstrate that our algorithm outperforms the traditional algorithm, which must exhaustively search the entire space.

The remainder of this paper is organized as follows. Section 2 covers related work. Section 3 covers XML basics. Section 4 present our algorithm for detecting changes in XML documents. Section 5 discusses merits and demerits of the usage of XOR for signature. Section 6 describes in detail the performance of our approach over an exhaustive search and reports a simulation result regarding on the usage of XOR. Section 7 presents our conclusions, and a comment about future work.

2. Related Work

Recent work in change detection has focused on computing differences between flat files. The GNU diff utility and AT&Ts HtmlDiff [2, 4, 5] are examples of this category. These examples use the LCS (Longest Common Subsequence) algorithm [9] to compare two plain text files or two HTML pages. This LCS works well for flat files; however, it fails to take into account the hierarchical structure information possessed by an XML document [3]. Furthermore, names in an XML document indicate context which must also be considered.

Since XML documents can be represented as trees, tree to tree correction technique can be utilized to detect changes in these documents [7,10, 12, 17, 18]. For this purpose, an XML document can be treated either as an ordered tree or an unordered tree. In our case we treat the XML document as an ordered tree. We propose an effective change detection algorithm by discarding irrelevant sub-trees based on the node signatures. Thus, we explore only a part of the tree and the other parts are pruned.

Related work in information filtering uses signature-based technique which is mainly based OR rather than XOR. Signature methods have been used extensively for text retrieval [16], image database [20], multimedia database [21, 22] and other conventional database systems[23]. A signature is an abstraction of the information stored in a record or file. By examining signature, we can estimate whether the record contains the desired information.

3. XML Primer

An element of XML is simply a type declaration or a set of elements [1]. Document type definitions (DTDs) define the structure of an XML document (e.g., what elements, attributes etc. are permitted in the document). XML documents contain data, and must contain exactly one root

element which will contain all the other elements. It might happen that an element contains a set of sub-elements in addition to the data. For example, a sample XML document might be:

```
<?xml version="1.0"?>
<CATALOG>
 <CD>
  <TITLE>Bridge of Spies</TITLE>
  <ARTIST>T`Pau</ARTIST>
  <COUNTRY>UK</COUNTRY>
  <COMPANY>Siren</COMPANY>
  <PRICE>7.90</PRICE>
  <YEAR>1987</YEAR>
 </CD>
  …
```

Figure 1. A Portion of XML Document

In order to design an efficient algorithm to detect changes to XML documents, we need to understand the hierarchical structure in an XML. Based on the Document Object Model specification [14], an XML document can be represented as a tree.

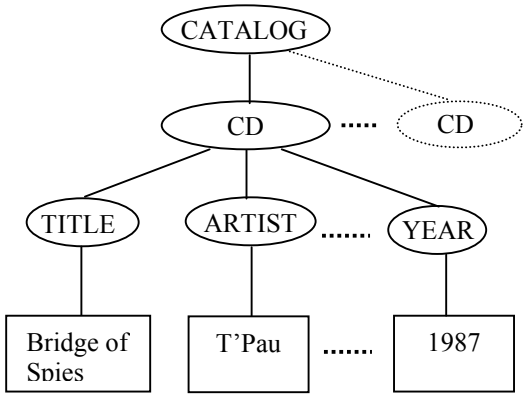

Figure 2. A Partial DOM Tree of the Document

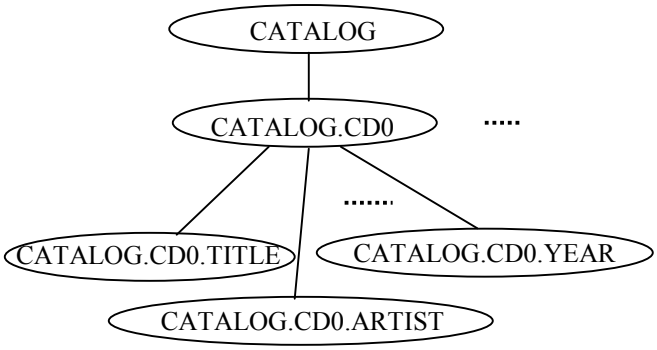

Figure 3. Identifier for Each Element in Figure 2

In Figure 1, three kinds of nodes are observed in the DOM tree, element, text, and attribute nodes. A portion of XML document is shown in Figure 2. Element nodes are represented by ovals, such as CATALOG and CD. They are non-leaf nodes with one label, name. Text nodes are represented by solid rectangles. Text nodes are leaf nodes with one label, value. Attribute nodes are also leaf nodes, but they have two labels name and value. According to the DOM specification, element nodes and text nodes are ordered, while attribute nodes are unordered.

4. Our Approach

We treat an XML document as an ordered tree, in which left to right order among siblings is important. We assume that only text nodes can be changed. Other types of nodes, such as element and attribute nodes, cannot be changed. In other words, in this paper we consider only changes in element values. Our goal is at first to detect whether or not the two versions are identical. If not, we match each element value in the old version with its corresponding value in the new version in order to detect value changes. For example, in Figure 1 and 2, we have a set of CDs. Each CD is described by a set of elements and their values. Let us assume the price of the first CD has been dropped (from 7.9 to 6.9). Therefore, We need to detect this change at a finer grain (i.e., at that particular CDs price level).

4.1 Naive Approach

One possible naive approach is to match each node in the old version with its corresponding node in the new version. Then the entire search space will be explored to detect changes, incurring high execution time. For example, if we apply the naive approach in Figure 1 to detect changes (i.e., first CDs price is dropped), we would have to traverse all the nodes in the tree. However, we would like to devise an algorithm that can identify a sub-tree for further exploration while tossing out other sub-trees if they do not show changes. Thus, the exploration of the search space can be reduced.

4.2 Top-Down

We propose an effective algorithm for the detection of changes in XML documents by exploring only a subset of nodes in the tree. This top-down algorithm starts from the root nodes of the two versions by comparing values. Then the immediate children nodes of the root nodes will be compared. We would like to make sure that if a leaf node changes the algorithm can detect changes, not by inspecting the node itself, but also its parent node, grand parent node, and so on. For this, a signature for each node will be constructed. Therefore, we will not match every node in the first

tree (old version) to every node in the second tree (new version) because each node in an XML has its own context. Exploring context will help us identify corresponding nodes between two XML versions. To match each node in the older version with its corresponding node in the new version we need to use context to define an identifier (Id) for each element by using the root node to this specific node.

4.3 Identifier (Id) Generation

The Id of an element is used to capture the nested position of the element in the XML document. For the formation of Id for an element, first the parent element, and then the grand parent of this element are identified. Finally, the process is stopped all the way back to the root element [15]. The Names of these elements will be concatenated separated by dot (.), where the first Name is the root element's Name, and the last Name is the element Name itself. We can formally define Id of an element as the concatenation of the Id of the parent element and its element Name. One important observation is that this Id generation mechanism depends entirely on an element's Name, but not the content or data values (text). However, it might happen that some elements occur multiple times. If we follow the above strategy for Id generation, we will end up with the same Id for more than one element.

To solve this problem, we need to address two questions. First, we need to identify which elements may occur more than once in a given XML document. Second, for each of these reoccurring elements we need to generate a unique Id. With regard to the first question, we will rely on the DTD. From the DTD we can identify which elements may occur more than once based on *, +. In this way we can identify which elements require special treatment in the generation of Id. With regard to the second question, the Id of each of these elements cannot be generated merely through the addition of the element Name itself at the end of the sequence. Instead, some sequence numbers (started from 0) will be padded with the Name itself so that the Id for each element will be unique. For example, in the XML document (see Figure 1) a sub-element CD occurs multiple times (confirmed by DTD) under element CATALOG. Hence, element CD along with sequence number (started with 0) will be added to its immediate parent's Id. Note that the parent's Id in this case is CATALOG. Therefore, the first and second CD's Id will be CATALOG.CD0, and CATALOG.CD1 respectively. In Figure 2, we show partially a set of Ids for a set of elements.

4.4 Signature

To reduce the number of node comparisons between two versions, we will compare the node signature of a node in the old version with its corresponding node signature in the new version. Note that the signature is basically an abstraction of the information stored in a node [16]. Furthermore, the signature of interior node is simply Exclusive Or (XOR) of all its children nodes' signatures. Rather than using a concatenation of signatures we use XOR to reduce the size of the signature (see Section 5 for details). Formally, the definition of signature is as follows:

Suppose x is a node in a DOM tree T, Signature (x) =Signature (x1) XOR Signature (x2) … XOR Signature(xn) where x1, x2, x3, , and xn are the descendents of x. If x is an element associated with a text node (its value is v), Signature(x) = Hash (v). For example, signature of the element, Title of the first CD in Figure 1 is Signature (Title)=Hash (Bridge of Spies). Note that this title element has a text node whose value is Bridge of Spies. Furthermore, signature of the element, CD (the first CD one in Figure 1) is Signature (Title) XOR Signature (Artist) XOR Signature (Country) XOR Signature (Company) XOR Signature (Price) XOR Signature (Year).

4.5 Change Detection

For the change detection, we will first identify whether there is any change between two versions. If the answer is yes, next we will determine where this change occurs. With regard to the first problem, the signatures of two root nodes of two versions are compared. If these signatures are the same there is no change between documents. Recall that the signature of the root node is XORed together with all of its descendents. If there is a change in any descendent node, the signature of this node will be changed as well as that of its immediate ancestor, and thus that of the root node. If signatures are different we need to address the problem of comparing nodes. For this, first children nodes signature of the root node in the old version will be compared with its corresponding nodes signature in the new version. If these are the same, sub-trees rooted by these nodes from the two versions will be discarded. In other words, no further comparison will take place for these two nodes in the two versions and their descendents. When the leaf nodes of the two versions are reached, and signatures are different, we will have arrived at the nodes where changes appear. For example, let us consider the XML document in Figure 1. The price of the first CD has been

changed. The signatures of the root nodes (Id of this node CATALOG) will be compared between the two versions. Since the price of the first CD has been changed, the signatures will be different. Next, the signatures of the first CD (Id CATALOG.CD0) for two versions will be compared. Since the signatures are obviously different, the signatures of CATALOG.CD0.TITLE,CATALOG.CD0.ARTIST and CATALOG.CD0.COUNTRY, for the two versions will be compared respectively. Since no other node is changed, signatures of nodes CATALOG.CD1, and CATALOG.CD2 of two versions will be the same. Thus, no other sub-tree will be traversed for further change detection.

5. Merits and Demerits of the Usage of XOR

We can get the signature of a parent node by superimposing signatures of all its children using XOR. If some bits change in a parent's signature, we can say that there must be some changes in the children's signatures. But conversely, if some children's signatures change, the parent's signature may remain the same. In this case, we will not be able to detect changes. We call this *miss drop*. This may happen because of the employment of XOR. For example, if a node has two children with signatures 1101, and both of these signatures are changed to 0100 at the same time (last bit changes in both cases), we will not be able to detect the underlying change. This is because 1101 XOR 1101 = 0100 XOR 0100.

In IR for document filtering, the node signature is used based on OR [16]. OR may allow to retrieve irrelevant documents to query (known as *false drop*) that affects precision. Furthermore, the following assumption is made: the number of 1 in the signature can be assumed to be very small as compared to length of the signature [19]. On the other hand, using XOR we will not get any irrelevant change (i.e., false drop probability = 0). While we may miss some change that affects recall; precision will not be affected.

During search or change detection, we want a user to get little irrelevant information/change (precision) while at the same time relevant information will not be overlooked (recall). Note that in the web environment precision is more important than recall. This is because the user wants to get only relevant information/change from the vast quantities of information that are being disseminated. For this, in order to discard irrelevant information, the user may sacrifice some relevant information/change. Furthermore, for web recall cannot be calculated directly [24]. Therefore, we would like to consider change detection of XML documents in a web environment using XOR. XOR guarantees that precision will not be degraded, something which may happen using OR. Furthermore, it does not make any assumption about the least number of 1 in the signature.

Now, we will discuss when our approach fails to detect changes. Let us define that the length of signature be M and we have N signatures to superimpose together. In other words, the parent node has N children nodes. Then we have a bit pattern with N rows and M columns as Figure 2 and the value of each bit is either one or zero. It is obvious the parent's signature has nothing to do with the order of XOR operations. The only factor that can affect the final result is the number of 1 bit in each column. If this number in one column is even, we have 0 at the corresponding bit in the parent's signature. Otherwise, if this number is odd, we have 1. So, if we have one bit changed in a column, no matter what content changes (i.e., from 1 to 0 or 0 to 1), the parents signature in this column will be changed. If we have two bits changed in a column, the total number of 1 in the column will be either even or odd as before, the parent's signature in the respective bit will not be changed. Therefore, if the number of changed bits in every column is even (i.e., all columns), the value of each bit in parent's signature will not be changed. If this happens, we cannot detect the changes in children's signature from their parent's signature, which contributes miss drop. Otherwise, if the number of changed bits in any column is odd, at least this column in parent signature will be changed. So, we can detect the change.

5.1 Calculation of Miss Drop Probability

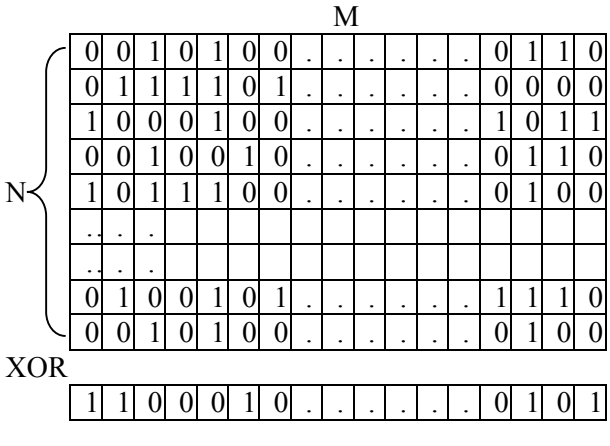

Figure 4. N Signatures with Length M Merge

We assume a particular node with N descendant nodes. In addition, signature length of a

node is M. Therefore, a particular node signature will be simply XOR of all descendant node signatures. In Figure 4 we show a two-dimensional table where one dimension represents M and the other dimension represents N. Therefore, we have total MN bits, any bit can be changed from MN. (i.e., uniformly distributed)

Symbol	Definition
P	Miss drop probability (Similar to $P_i(M)$, but takes N into account.)
M	Signature length
N	Total number of signatures
p	Percentage of changed bits
i	Total number of changed bits
B_j	Total number of changed bits in the j^{th} column
$P_i(M)$	Probability that B_j is even (include zero) for column j when the total number of changed bit is i with M columns (ignoring N); all column changed bits are even; where j varies from 1 to M.
Q_i	Probability of having total number of changed bit is i
S_i	Probability that all bits changed in at least one column when the total number of changed bits is i
$S_L(k)$	Probability that all bits changed in exactly *k* columns when $LN \le i < (L+1)N$
L	At most all bits changed in L columns; L>=k
q	At least all bits changed in *q* columns

Table 1. Symbol Definitions

M, N, and *p* can affect the miss drop probability, P. However, in some case N will not affect P. When i < N, then $p = \frac{i}{MN} \Rightarrow p < \frac{1}{M}$. Intuitively, when the total number of changed bits is less than N, it is impossible to change all bits in any column. In this case, N does not have any impact on P; only M and *p* can affect P. P will be calculated using the following way:

First, let's define $P_i(M)$ is the miss drop probability when total changed bit number is i with M columns. Obviously, we can have expressions as below.

$P_0(M) = 1$ (1)

$P_2(M) = \frac{1}{M}$ (2)

$P_i(M) = 0$ when i is odd (3)

When i is even and i> 2, $P_i(M) =$

$$\frac{1}{M} \times P_{i-2}(M) + [\frac{M-1}{M} \times \frac{2}{M} \times \frac{1}{M}] \times P_{i-4}(M)$$

$$= \frac{1}{M} \times P_{i-2}(M) + \frac{2 \times (M-1)}{M^3} \times P_{i-4}(M) \quad (4)$$

Since all changed bits are uniformly distributed across the columns, i changed bits will be uniformly distributed in M columns. For Equation 1 when i=0, B_j=0 for all j, $P_0(M) = 1$ that means probability of 0 bit changed (i.e., even) in any column is 1. For Equation 2, when only two bits are changed, the column of the first bit does not matter; however the column of the second bit is important. The second bit may appear in the same column as the first bit or not. Since changed bits are distributed uniformly, the probability that 2 bits are in the same column (i.e., second bit will appear in the same column as the first bit), $P_2(M) = \frac{1}{M}$; the probability that both are not in the same column is $= 1 - P_2(M) = \frac{M-1}{M}$.

For Equation 3, if i is odd, no matter how these i bits are distributed, at least for one j, B_j is odd. Hence, number of changed bits in at least one column is odd that allows us to detect changes Recall that node changes will be unnoticed if all column changes are even including zero. Therefore, for all odd i, $P_i(M) = 0$.

For Equation 4, when i>2 we have two cases. Let us assume changed bits are picked one by one. In case 1 the first two bits are in the same column with probability $\frac{1}{M}$; and in case 2 the first two bits are in different columns with probability $\frac{M-1}{M}$. In case 1, we would like to make B_j even for all columns; the first two bits will appear into the same column; rest of i-2 bits will satisfy the same requirement with probability $P_{i-2}(M)$ In case 2, at least two more bits from the rest of i-2 bits will be distributed in these two columns to make B_j even for all columns. Besides these 4 bits, rest i-4 bits will make B_j even for all j with probability $P_{i-4}(M)$.

Now, we can generalize the following way:

$P_0(M) = 1$

$P_2(M) = \frac{1}{M}$

$$P_4(M) = \frac{1}{M} \times P_2(M) + \frac{2 \times (M-1)}{M^3} \times P_0(M)$$

$$= \frac{3M-2}{M^3} = c_{41}M^{-2} + c_{42}M^{-3}$$

$$P_6(M) = \frac{1}{M} \times P_4(M) + \frac{2\times(M-1)}{M^3} \times P_2(M)$$

$$= \frac{5M-4}{M^4} = c_{61}M^{-3} + c_{62}M^{-4}$$

$$P_8(M) = \frac{1}{M} \times P_6(M) + \frac{2\times(M-1)}{M^3} \times P_4(M)$$

$$= \frac{11M^2 - 14M + 4}{M^6}$$

$$= c_{81}M^{-4} + c_{82}M^{-5} + c_{83}M^{-6}$$

……

$$P_i(M) = \frac{1}{M} \times P_{i-2}(M) + \frac{2\times(M-1)}{M^3} \times P_{i-4}(M)$$

$$= c_{i1}M^{-\frac{i}{2}} + c_{i2}M^{-\frac{i}{2}-1} + c_{i3}M^{-\frac{i}{2}-2} \quad (5)$$

Finally, miss drop probability will be sum of all these $P_i(M)$. Recall that miss drop probability arises when number of changed bits in each column is even. Thus, we miss drop probability for any M and N:

$$P = \sum_{i=1}^{MN} P_i(M) \times Q_i \quad (6)$$

If i > N, it would be possible that every bit in one column will be changed. Thus, N will affect on P. However, when *p* is low, even if i > N, we can still use Equation 5. This is because changed bits are uniformly distributed across columns with low *p*, S_i (i.e., the probability of having at least one column in which every bit will be changed) may be very low. In other words, N can affect P only when every bit in at least one column changed. Therefore, N has very little effect to P when *p* is small. When M and N are fixed along with large *p*, then i and S_i will be increased. Furthermore, N will play more impact on P.

Now, we would like to define $S_L(k)$ probability of every bit in exact *k* columns will be changed when $LN \le i < (L+1)N$. We have two boundary values here.

- When i < N, L = 0, then $S_i = 0$, $S_L(0) = 1$, $S_L(k) = 0$ $(0 < k \le M)$;
- If (N-1)*M < i ≤ MN, $S_i = \sum_{k=q}^{L} S_L(k) = 1$, $S_L(k) = 0$ $(0 \le k < q)$. This is because i > (N - 1) * M means that at least in one column every bit will be changed. Hence, *q* is equal to i – (N - 1) * M in this case. Otherwise *q* is equal to zero.

We have $\binom{MN}{i}$ total different bit patterns, and we assume probability for each pattern is the same due to uniform distribution. Based on probability theory, for all other i between N and (N-1)*M, we have

$$S_L(L) = \frac{\binom{M}{L} \times \binom{MN - LN}{i - LN}}{\binom{MN}{i}} \quad (7)$$

$$S_L(L\text{-}1) = \frac{\binom{M}{L-1} \times \binom{MN - (L-1)N}{i - (L-1)N}}{\binom{MN}{i}} - S_L(L) \times \binom{L}{L-1}$$

$$S_L(L\text{-}2) = \frac{\binom{M}{L-2} \times \binom{MN - (L-2)N}{i - (L-2)N}}{\binom{MN}{i}} - S_L(L-1) \times \binom{L-1}{L-2} - S_L(L) \times \binom{L}{L-2}$$

…

$$S_L(k) = \frac{\binom{M}{k} \times \binom{MN - kN}{i - kN}}{\binom{MN}{i}} - \sum_{l=k+1}^{L} S_L(l) \times \binom{l}{k} \quad (8)$$

Using Equation 7 and 8, we get all S_L recursively. For example, if L = 1, we have $S_L(1)$ using Equation 7:

$$S_L(1) = \frac{\binom{M}{1} \times \binom{(M-1)\times N}{i - N}}{\binom{MN}{i}} = \frac{M \times \prod_{l=0}^{N-1}(i-l)}{\prod_{l=0}^{N-1}(MN-l)}$$

And then, we get $S_L(0)$ using Equation 8.

$$S_L(0) = \frac{\binom{M}{0} \times \binom{MN}{i} - \binom{M}{1} \times \binom{MN - N}{i - N}}{\binom{MN}{i}}$$

$$= 1 - \frac{\binom{M}{1} \times \binom{(M-1)\times N}{i - N}}{\binom{MN}{i}} \quad (9)$$

$$S_i = \sum_{k=1}^{L} S_L(k) = 1 - S_L(0)$$

$$= \sum_{k=1}^{L} \left[\frac{\binom{M}{k} \times \binom{MN - kN}{i - kN}}{\binom{MN}{i}} - \sum_{l=k+1}^{L} S_L(l) \times \binom{l}{k} \right] \quad (10)$$

When N is odd, miss drop probability P is as below,

$$P = S_L(0) \times P_i(M) =$$

$$\left\{1 - \sum_{k=1}^{L} \left[\frac{\binom{M}{k} \times \binom{MN-kN}{i-kN}}{\binom{MN}{i}} - \sum_{l=k+1}^{L} S_L(l) \times \binom{l}{k} \right] \right\} \times P_i(M) \quad (11)$$

When N is even:

$$P = S_L(q) \times P_{i-qN}(M-q) + S_L(q+1) \times P_{i-(q+1)N}(M-q-1) + \ldots + S_L(L) \times P_{i-LN}(M-L)$$

$$= \sum_{l=q}^{L} S_L(l) \times P_{i-lN}(M-l) \quad (12)$$

6. Results

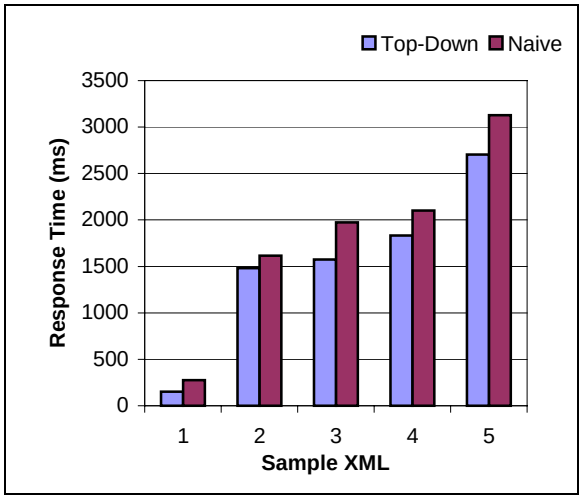

Figure 5. Performance Result of Different Algorithms for 10% Changes in Different XML Documents

In the first set of experiments, we have randomly changed 10% of the nodes of these documents. We have implemented naive and top-down algorithms and measured the response time to detect changes with each algorithm. Here we have pre-computed signatures of nodes. In Figure 5, the X-axis represents XML documents. Note that the first, second, third, fourth, and fifth documents consist of 533, 3153, 3757, 4141, and 5984 nodes for their DOM tree representation respectively. The Y-axis represents the response time, which needed to detect changes. For each XML document, the first and second bars represent top-down, and naive algorithms respectively. Furthermore, the data demonstrates that the response time of top-down outperforms naive.

In the second set of experiments, we need to calculate miss drop probabilities based on M, N and *p*. As we are not sure that how many signatures will be changed and how many bits in a signature will be changed, we would like to do a simulation study. Thus, we can only vary *p*, M and N. M varies from 4 to 16 increased by 4. N varies from 5 to 65 increased by 5 and *p* varies from 10% to 100% increased by 10%. In simulation miss drop probability has been calculated for every fixed M, N and *p*.

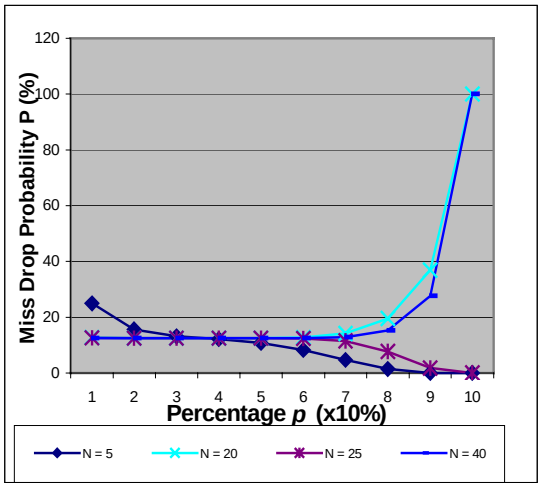

Figure 6. Miss Drop Probability, P for Different *p* (Fixed M)

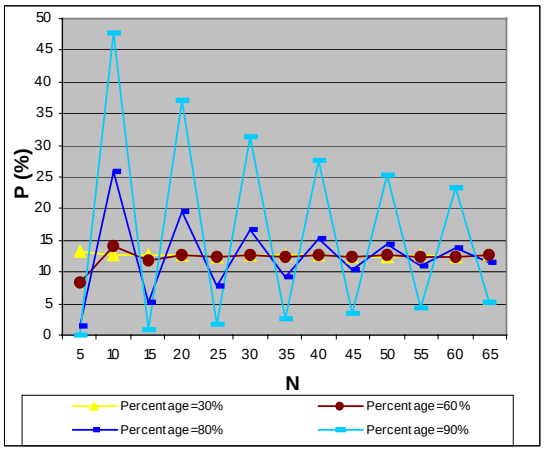

Figure 7. Miss Drop Probability, P for Different N (Fixed M)

For simulation, first we have determined the number of changed bits i (assume even). Then we uniformly have distributed i changed bits across M columns and recorded how many changes happened in each column. If a certain column has already had N changes, then distribute it in some other column randomly that has less than N number of changes. Recall that a column has only N number of bits, and maximum possible bits changed

in a column is N. After distribution, we have counted total number of changed bits in every column. If this number is even for every column, then miss drop has occurred. We have repeated the above steps ten million times and observed how many miss drops we have.

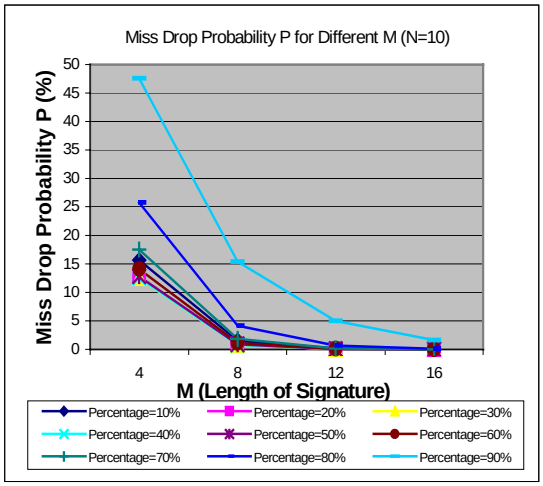

Figure 8. Miss Drop Probability P for Different M (Fixed N)

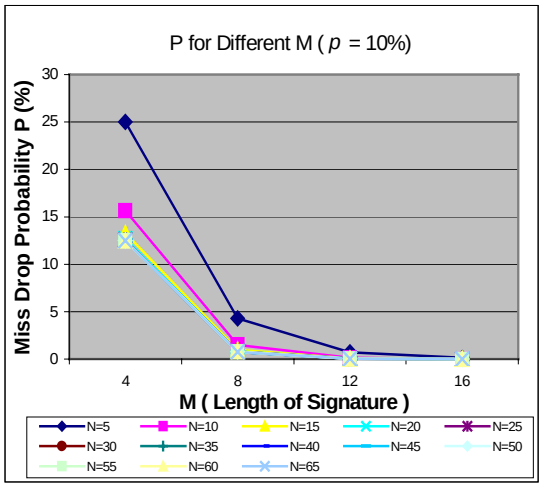

Figure 9. Miss Drop Probability P for Different M (Fixed *p*)

In Figure 6, 7, 8 and 9, the X-axis represents *p*, N, M and M respectively. The Y-axis represents miss drop probability, P. In Figure 6 we demonstrate for different N how miss drop probability varies along with *p* when M is fixed (=4). We have observed P increases with the increased value of *p* when N is even (e.g., N=10, 20…). However, P decreases with the increased value of *p* when N is odd (e.g., N=5, 15). This is because N can affect P only when S_i is not equal to zero. If S_i is large, N will take more effect on P. From Equation 11 and 12, we know that the increase of S_i will also increase P when N is even; if N is odd, the increase of S_i will cause a decrement of P. From Equation 9 and 10, we know that S_i will increase with the increased value of *p*. So, P will increase with the increased value of *p* when N is even and decrease when N is odd. In Figure 7, we have observed that P decreases with the increased value of N when N is even. On the other hand, when N is odd P increases with the increased value of N. This is because that S_i decreases along with the increased value of N when M and *p* are fixed.

In Figure 8 and 9 we show how miss drop probability varies with M for fixed N and different *p*, and for fixed *p* and different N respectively. The data demonstrates that P decreases with increased value of M for any N and *p*. From Equation 9 and 10, S_i will decrease with the increased value of M. That implies that P should increase with the increased value of M when N is odd. But in fact, according to Equation 5, 11 and 12, the other item will decrease sharply with the increased value of M. This will overcome the N's effect. So, no matter N is even or odd, P will decrease sharply with the increased value of M. In our simulation, when M = 16, P will be less than 0.01% for most N and *p*. In our implementation, Java provides 32 bit signatures which suggest that P will be insignificant. In other words, in web setting a tolerably small number of changes may be unnoticed.

7. Conclusions and Future Work

We have proposed a method of automatic change detection between two versions of XML documents based on node signatures. We have chosen exclusive-or (XOR) to construct node signatures which does not allow a user to get irrelevant information/change and allows to miss some relevant information. Note that for the web along with huge information, relevancy of information/change is more important as compared to missing of relevant information/change. For this, in this paper we propose an automatic change detection algorithm which will identify changes between two versions of an XML document based on these signatures. We have demonstrated that our novel approach outperforms a traditional approach, which searches the entire space exhaustively. Our approach traverses fewer nodes in trees to detect changes between two versions of XML documents. We also demonstrate analytically and empirically that miss of relevant change in the web environment is in tolerable range. We would like to extend this work in the following directions. First, we would like to store an old version of XML document into a

database and then evaluate the impact of the database on change detection. Next, we would like to modify the algorithm to detect not only the change of element values, but also the change of the elements and even the order of the elements. Finally, we would like to address how we can continually notify users of ongoing changes.

References

[1] "Extensible Markup Language (XML)", *World Wide Web Consortium*, http://www.w3.org/XML/.

[2] E. Berk, "HtmlDiff: A Differencing Tool for HTML Documents", *Student Project*, Princeton University, http://www.htmldiff.com.

[3] S. Chawathe, A. Rajaraman, H. Garcia-Molina and J. Widom, "Change Detection in Hierarchically Structured Information", *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Montreal, June 1996.

[4] F. Douglis, T. Ball, "Tracking and Viewing Changes on the Web", *1996 USENIX Annual Technical Conference*, 1996.

[5] F. Douglis, T. Ball, Y. F. Chen, E. Koutsofios, "The AT&T Internet Difference Engine: Tracking and Viewing Changes on the Web", *World Wide Web*, 1(1): 27-44, January 1998.

[6] D. S. Hirschberg, "Algorithm for the Longest Common Subsequence Problem", *Journal of the ACM*, 24(4):664-675, October 1977.

[7] C. M. Hoffmann, M. J. O'Donnell, "Pattern Matching in Trees", *Journal of the ACM*, 29: 68-95, 1982.

[8] H. Maruyama, K. Tamura and R. Uramoto, "Digest values for DOM (DOMHash) proposal", *IBM Tokyo Research Laboratory*, http://www.trl.ibm.co.jp/projects/xml/domhash.htm, 1998.

[9] D. McArthur, "LSR Gene Expression RFI Response", *Rosetta Inpharmatics*, http://cgi.omg.org/cgi-bin/doc?lifesci/99-08-11, August 1999.

[10] E. W. Myers, "An O(ND) Difference Algorithm and Its Variations", *Algorithmica*, 1(2): 251-266, 1986.

[11] S. M. Selkow, "The Tree-to-Tree Editing Problem", *Information Processing Letters*, 6(6): 184-186, 1977.

[12] K. C. Tai, "The Tree-to-Tree Correction Problem", *Journal of the ACM*, 26: 485-495, 1979.

[13] R. E. Tarjan, "Data Structures and Network Algorithms", *CBMS-NSF Regional Conference Series in Applied Mathematics*, 1983.

[14] L. Wood, et. al., "Document Object Model (DOM) Level 1 Specification (Second Edition)", *World Wide Web Consortium*, http://www.w3.org/TR/2000/WD-DOM-Level-1-20000929/, 2000.

[15] L. Khan and Y. Rao, "A Performance Evaluation of Storing XML Data in Relational DBMS," in Proc. of *ACM 3rd International Workshop on WEB Information and Data Management (WIDM)*, Georgia, Nov, 2001.

[16] C. Faloutsos and S. Christodoulakis, Signature Files: An Access Method for Documents and its Analytical Performance Evaluation, *ACM Trans. on Office Information Systems (TOOIS)*, 2, 4, pp. 267-288, Oct. 1984.

[17] Y. Wang, D. J. DeWitt, and Jin-Yi Cai, X-Diff: A Fast Change Detection Algorithm for XML Documents, Submitted for Publication.

[18] B. Nguyen, S. Abiteboul, G. Cobena, and M. Preda, Monitoring XML data on the Web, In *Proc. of the ACM SIGMOD*, Santa Barbara, 2001.

[19] H. Kitagawa, Y. Fukushima, Y. Ishikawa and N. Ohbo, Estimation of False Drops in Set-valued Object Retrieval with Signature Files, in *Proc. of 4th Intl. Conf. on Foundations of Data Organization and Algorithms*, Chicago, Illinois, October 1993, Splinger-Verlag, pp. 146-163.

[20] S.-Y. Lee, M.-C. Yang, and J.-W. Chen, Signature file as a spatial filter for iconic image database, Journal of Visual Languages and Computing, vol. 3, no. 4, pp. 373-397, 1992.

[21] F.Rabitti and P. Zezula, A dynamic signature technique for multimedia databases, Proceedings of the 13th International Conference on Research and Development in Information Retrieval, September 1990, pp 193-210.

[22] P. Tiberio and P. Zezula, Selecting signature files for specific applications, Information Processing and Management, vol. 29, no. 4, pp. 487-498, 1993

[23] W.W. Chang and H.J. Schek, A signature access methods for the starburst database system, Proceedings of the Fifteenth International Conference on Very Large Data Base, Amsterdam, The Netherlands, August 1989, pp. 145-153.

[24] R. Baeza-Yates, B. Ribeiro-Neto, Modern Information Retrieval, ISBN 0-201-39829.