Change Detection in XML Documents for Fixed Structures using Exclusive-Or (XOR)

Latifur Khan, Qing Chen, and Lei Wang
Department of Computer Science
University of Texas at Dallas
Richardson, TX 75083-0688
Email: [lkhan, qingch, leiwang]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. Furthermore, for a fixed schema (structure) changes may frequently happen in the content or data values of XML documents in the Web. We wish to propose a method that effectively detects this content or data value changes. 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 using XOR. 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, image and video). Overall, this data constitutes the largest body of information 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 evolving Extended Markup Language (XML), which is becoming the new standard for semi-structured data exchange over the Internet. The characteristics of XML, tree-structured (i.e. a collection of nodes) and self-descriptive, facilitate interpretation of data. This means that a program receiving an XML document can interpret this document in multiple ways. Based upon content, the document can be filtered and restructured to suit the needs of a given application.

This self-descriptive characteristic 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. The fixed schema or structure may be a very common scenario in the Web along with content changes. Hence, the focus of this paper is to detect content changes in XML documents.

One may argue that to detect content changes for fixed structures why do not we just inspect all leaf nodes. This method will be expensive when there are too many leaf nodes. 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 [4, 6, 12]. Signature methods have been used extensively for text retrieval[5, 6, 7, 11], image database[8, 9], multimedia database[3, 10], and other conventional database systems [1, 11]. A signature is an abstraction of the information stored in a record or file. Furthermore, the number of 1 in signature is assumed to be very small as compared to the length of the signature [6]. 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. 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.

The main contributions of this work are as follows: We propose an automatic change detection algorithm between two versions of XML documents based on node signatures. 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 as compared to that obtained from using OR-based signatures, widely used in document filtering.

The remainder of this paper is organized as follows: Section 2 describes content changes in XML documents using XOR signatures. Finally, Section 3 describes the implementation of our system, including details about the performance of various approaches.

2. Change Detection in XML documents for Fixed Structure

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, 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 CD's price level).

![Figure 1. A Partial Tree of an XML Document](image)

2.1 Top-Down Approach

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.

2.2 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 [6]. 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. Formally, the definition of signature is as follows:

Suppose x is a node in a XML 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).

2.3 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 descendendent 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 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 for two versions will be compared. Since the
signatures are obviously different, the signatures TITLE, ARTIST and COUNTRY, for the two versions will be compared respectively. Since no other node is changed, signatures of other nodes of two versions will be the same. Thus, no other sub-tree will be traversed for further change detection.

2.4 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 0100 = 0101 XOR 0100.

In IR for document filtering, the node signature is used based on OR [5, 6, 7]. 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 [6]. 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 using XOR cannot be calculated directly. 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 3 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.

2.5 Calculation of Miss Drop Probability

![Figure 2 N Signatures with Length M Merge](image)

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 2 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).
Table 1 Symbol Definitions

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

$M$, $N$, and $p$ can affect the miss drop probability, $P$. However, in some case $N$ will not affect $P$. When $i < N$, then $p = \frac{i}{MN}$, 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.

\[
P_0(M) = 1 \quad \text{(1)}
\]

\[
P_2(M) = \frac{1}{M} \quad \text{(2)}
\]

\[
P_i(M) = 0 \text{ when } i \text{ is odd} \quad \text{ (3)}
\]

\[
P_i(M) = \frac{1}{M} \quad \text{when } i \text{ is even and } i > 2 \quad \text{ (4)}
\]

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),

\[
P_2(M) = \frac{1}{M} \quad \text{ (5)}
\]

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 $\frac{M^2-1}{M^3}$. 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:

\[
P_i(M) = \frac{1}{M} \quad \text{ (6)}
\]

\[
P_2(M) = \frac{1}{M} \quad \text{ (7)}
\]

\[
P_i(M) = \frac{1}{M} \quad \text{ when } i \text{ is even and } i > 2 \quad \text{ (8)}
\]

\[
P_i(M) = \frac{M^2 - 1}{M^3} \quad \text{ when } i \text{ is odd and } i > 2 \quad \text{ (9)}
\]

\[
P_{i+2}(M) = \frac{3M^2 - 2}{M^3} \quad \text{ (10)}
\]

\[
P_{i+4}(M) = \frac{c_{41}M^{i+2} + c_{42}M^{i+3}}{M^4} \quad \text{ (11)}
\]
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 $N$:

$$P = \sum_{i=1}^{\min(M,N)} P_i(M) \cdot Q_i$$  \hspace{1cm} (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 $LN = i < (L + 1)N$. We have two boundary values here.

**When $i < N$, $L = 0$, then $S_i = 0$, $S_i(0) = 1$, $S_i(k) = 0 (0 < k = M)$;**

**If $(N-1)*M < i = MN$, $S_i = \frac{i}{k^l \cdot q^2} \cdot L_i = 1$, $S_i(k) = 0 (0 = k < q)$. 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 $\frac{MN}{3} \cdot i^3$ 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

$$S_L(L-1) = \frac{1}{L-1} \frac{S_L(L-2) + S_i}{M}$$  \hspace{1cm} (7)

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

$$S_i(1) = \frac{1}{1} \frac{S_i}{i}$$

...
\[
S_L(0) = \prod_{i=0}^{N-1} S_L(k) = S_L(0) \prod_{k=0}^{N-1} S_L(k)
\]

When \( N \) is odd, miss drop probability \( P \) is as below,
\[
P = S_L(0) \cdot P_i(M) = \prod_{k=0}^{N-1} S_L(k) \cdot P_i(M)
\]

When \( N \) is even,
\[
P = \prod_{q=N}^{L} S_L(q) \cdot P_i(q) \cdot S_L(q) \prod_{k=q}^{L} S_L(k) \cdot P_i(k)
\]

Finally, miss drop probability \( P \) is:
1) When \( i \) is odd, \( P = 0 \);
2) When \( i \) is even, we have several sub cases that demand different ways to calculate \( P \) (as shown in Table 2).

| \( \text{Table 2 Calculation of } P \text{ when } \text{i is even} \) |
|---------------|-----------------|-----------------|
| \( N \) is even | \( i < N \)       | \( N = i = (N-1)^*M \) | \( i > (N-1)^*M \) |
| \( N \) is odd  | Use Equation (5) | Use Equation (12) |                      |

Actually, \( P \) is not the final miss drop probability. It is just the miss drop probability when the number of changed bits is \( i \). So, strictly say, it should be written as \( P(i) \). Similar to the equation (3), final miss drop probability will be:
\[
\text{Miss Drop Probability} = \prod_{i=1}^{N} P(i) \cdot Q_i
\]

4. Implementation

We implemented a prototype in Java 1.4 and we used the IBM parser XML4J to create the Document Object Model (DOM) tree from the XML document. All the experiments presented herein were conducted on a Sun SPARCstation running Solaris 2.5.
We have randomly changed 10% of the nodes of these documents. We have implemented naïve (i.e., inspect all nodes) 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 3, 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 naïve algorithms respectively. Furthermore, the data demonstrates that the response time of top-down outperforms naïve.

For these five documents with a 10% change we constructed 64 bit signatures using OR and XOR, and measured miss drop probability (see Table 3). In Table 3, miss drop probability for OR varies from 87% to 98% whereas XOR contributes less than 0.5% for each XML document. Higher miss drop probability happens due to the nature of OR logic. For example, suppose a node has three children with signatures 0001, 0000, and 0000 and the signature of the node is 0001 OR 0000 OR 00000=0001 in the OR case, and 0001 XOR 0000 XOR 0000=0001 in the XOR case. Now, if the second children signature has been changed from 0000 to 0001, the signature of the node will be 0001 OR 0001 OR 00000= 0001 in the OR case, and 0001 XOR 0001 XOR 0000=0000 in the XOR case. Thus, XOR will detect the change while OR will fail to detect the change. It is obvious that in the OR case when a children node has 1 at a particular bit position and then later some other children nodes change from 0 to 1/1 to 0 at that position, this change will not be detected. This happens when at least one children node's signature sets to 1 at a particular bit position. The signature of the node will be 1 at that position regardless of the content of other children. Therefore, this empirical result demonstrates the superiority of XOR over OR. We can conclude that traditional Or-based signature, widely used in document filtering, has not worked well here. We have observed that our proposed XOR-based signature detects changes in XML documents with a reasonably good accuracy. In future, we would like to detect structural changes in XML documents using signatures based on XOR.

Table 3 Miss Drop Probability of the Usage of Or and XOR Change in Different XML Documents

<table>
<thead>
<tr>
<th>Method/Document</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR</td>
<td>0.87</td>
<td>0.95</td>
<td>0.94</td>
<td>0.94</td>
<td>0.98</td>
</tr>
<tr>
<td>XOR</td>
<td>0.001</td>
<td>0.002</td>
<td>0.004</td>
<td>0.002</td>
<td>0.002</td>
</tr>
</tbody>
</table>

References