An Efficient Algorithm to Compute Differences between Structured Documents

Kyong-Ho Lee, *Member*, *IEEE Computer Society*, Yoon-Chul Choy, *Member*, *IEEE Computer Society*, and Sung-Bae Cho, *Member*, *IEEE Computer Society*

Abstract—SGML/XML are having a profound impact on data modeling and processing. This paper presents an efficient algorithm to compute differences between old and new versions of an SGML/XML document. The difference between the two versions can be considered to be an edit script that transforms one document tree into another. The proposed algorithm is based on a hybridization of bottom-up and top-down methods: The matching relationships between nodes in the two versions are produced in a bottom-up manner and then the top-down breadth-first search computes an edit script. Faster matching is achieved because the algorithm does not need to investigate the possible existence of matchings for all nodes. Furthermore, it can detect structurally meaningful changes such as the movement and copy of a subtree as well as simple changes to the node itself like insertion, deletion, and update.

Index Terms—Change detection, difference computation, edit script, edit operation, structured document, SGML, XML.

1 Introduction

Since SGML (Standard Generalized Markup Language) [1] and XML (eXtensible Markup Language) [2] are good tools for embedding logical structure information into documents, they are widely accepted as a standard format for representing documents in various fields such as CALS (Commerce At Light Speed), EC (Electronic Commerce), Digital Library, and the Web. Thus, a lot of attempts have been made to process structured documents based on SGML/XML.[1]

Computing differences between structured documents is important for various fields such as version management [3], [4], [5], [6], [7], data warehousing [8], and active database [9]. Recently, the availability of large repositories of XML documents is increasing on the Web. Accordingly, automated change detection of XML documents that are visited often by users will be able to provide a more convenient Web environment. For example, it can automatically inform the user, who invests in a stock or bids on an auction item, whether the price of the stock has been changed or a new bid has been posted for the item.

SGML/XML documents include logical structures with hierarchy. For example, SGML/XML representations of technical journal articles have hierarchical structures with logical components such as a title, authors, an abstract, keywords, sections, paragraphs, etc. Therefore, logical structures as well as textual data should be considered for efficient change detection in SGML/XML documents.

Most previous works for computing differences have dealt with flat-file [10], [11], [12], [13], relational data [14], or hierarchical data [15], [16], [17], [18], [19], [20], [21]. However, because the works based on flat-file or relational data do not take hierarchical information into consideration, they do not handle structured documents effectively. Most previous works targeting hierarchical data extract only simple changes such as the insertion, deletion, and update of nodes. They represent the movement or copy of a subtree as a set of deletions and insertions or a set of insertions, respectively. However, these are not structurally meaningful especially in SGML/XML documents. Therefore, structurally meaningful changes such as the movement and copy of a subtree should be supported. Previous works that detect meaningful changes have a shortcoming of being too slow.

In this paper, we present an efficient algorithm for detecting meaningful changes in structured documents more quickly than previous works. To this end, the method is designed according to the domain characteristics of SGML/XML documents. Specifically, the proposed algorithm takes the hybrid of top-down and bottom-up methods to compute differences between old and new versions of a structured document. Matchings[2] between nodes in two versions are created by applying a path-matching algorithm in a bottom-up manner. Top-down, breadth-first search computes an edit script,[3] which is a sequence of edit operations needed to transform an old version of a document to its new version. Furthermore, for more efficient representation of structured documents, we propose a document model, which reflects their domain

1. This paper uses the three terms "structured document," "SGML/XML document," and "structured document based on SGML/XML" with the same meaning.

2. The notion of a correspondence between nodes that are identical or similar is formalized as a *matching* between nodes as was in the work of Chawathe et al. [20], whereas the Zhang and Shasha method [15] defines it as a *mapping*.

3. An edit script means a sequence of operations that transforms an old version of a document into its new version. As with the Zhang and Shasha method and the work of Chawathe et al., we define these operations as edit operations.

• *The authors are with the Department of Computer Science, Yonsei University, 134, Shinchon-dong, Sudaemoon-ku, Seoul, 120-749, South Korea. E-mail: {khlee, ycchoy, sbcho}@cs.yonsei.ac.kr.*

Manuscript received 15 May 2000; revised 23 July 2002; accepted 8 May 2003.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 112104.

```
1: <!--     Element Name    Content Model                -->
2: <!ELEMENT    MEMO        (TO+, FROM, BODY, CLOSE?)    >
3: <!ELEMENT    TO          (#PCDATA)                    >
4: <!ELEMENT    FROM        (#PCDATA)                    >
5: <!ELEMENT    BODY        (PARAGRAPH)*                 >
6: <!ELEMENT    PARAGRAPH   (#PCDATA | QUOTATION)*       >
7: <!ELEMENT    QUOTATION   (#PCDATA)                    >
8: <!ELEMENT    CLOSE       (#PCDATA)                    >
```

Fig. 1. An example of an XML DTD for a memorandum.

characteristics, based on a rooted ordered tree that has a root node and order among nodes.

Compared with the previous works, [15], [16], [17], which detect simple changes such as the insertion, deletion, and update of a node, the proposed method makes it possible to detect the movement and copy of a subtree because it supports not only one-to-one but also one-to-many matchings. In addition, since the algorithm does not investigate possible existence of matchings for all nodes, it produces matchings faster than the work of Chawathe et al. and the Chawathe and Molina method [21], which detect structurally meaningful changes such as the movement or copy of a subtree. The algorithm proposed to produce matchings runs in time $O(l_1 l_2 + (m_1/l_1 + m_2/l_2) l_2)$, where m_1 (and m_2) and l_1 (and l_2) denote the numbers of interior and leaf nodes of an old version (and a new version), respectively. With the increasing number of XML documents and the growing interest in their sophisticated processing, the application potential of the proposed method is very high.

The organization of this paper is as follows: In Section 2, a document model that represents structured documents is proposed. Edit operations are defined and a cost model for computing an edit script is introduced. In Section 3, a brief discussion about related work and the properties and constraint of our work are given. In Section 4, the proposed algorithm is presented in two steps: producing matchings and computing an edit script. In Section 5, the performance of the algorithm is analyzed, and the conclusions are summarized in Section 6.

```
<MEMO>
  <TO>Comrade Napoleon</TO>
  <FROM>Snowball</FROM>
  <BODY>
    <PARAGRAPH>
          In Animal Farm, George Orwell says:
          <QUOTATION>These were large sheets ... </QUOTATION>
          Do you think SGML would be helpful?
    </PARAGRAPH>
  </BODY>
  <CLOSE>Comrade Snowball</CLOSE>
</MEMO>
```

Fig. 2. A document instance that conforms to the DTD in Fig. 1.

2 Background

In this section, a document model is proposed for efficient representation of structured documents. We define edit operations that describe meaningful changes between structured documents. Differences from the operations of conventional methods are also highlighted. Finally, a cost model is introduced to determine an edit script.

2.1 Document Model

An SGML/XML document is generally composed of three parts: a declaration, a DTD (Document Type Definition), and a document instance [22], [23], [24]. The primary goal of the proposed method is to detect changes between document instances that conform to the same DTD. This is because editing an SGML/XML document normally means modifying the document instance under the logical schema of the corresponding DTD.

As shown in Fig. 1, a DTD is usually made up of element declarations that describe a logical structure of a document, [23], [25]. The declaration of line 2 identifies the element named MEMO. Its content model follows the element name. The content model defines what an element may contain. MEMO must contain TO, FROM, and BODY and may contain CLOSE. The commas between element names indicate that they must occur in succession. The plus mark after TO indicates that it may be repeated more than once but must occur at least once. The question mark after CLOSE means that it is optional. Both FROM and BODY with no punctuation must occur exactly once. Additionally, a keyword #PCDATA indicatestextual data (lines 3, 4, 6-8).

We define a document model based on a rooted ordered tree (hereafter, when we use the term "tree," it means an "ordered tree"). Specifically, each node that makes up a tree has a label and a value. Elements and textual data comprising an SGML/XML document become nodes in the corresponding tree according to the proposed document model. Nodes that correspond to an element and textual data are called an "element node" and a "text node," respectively. An element node takes the name of an element as its label. Textual data is represented as the value of a text node.

Fig. 2 shows an example of a structured document that conforms to the DTD of Fig. 1. Its tree representation is illustrated in Fig. 3. The node with a label "TEXT" is a text node (this paper uses "TEXT" as the label of a text node for convenience). Text nodes and element nodes become leaf nodes and interior nodes, respectively.[4] In this model,

4. In this paper, text and leaf nodes have the same meaning and also element and interior nodes are the same.

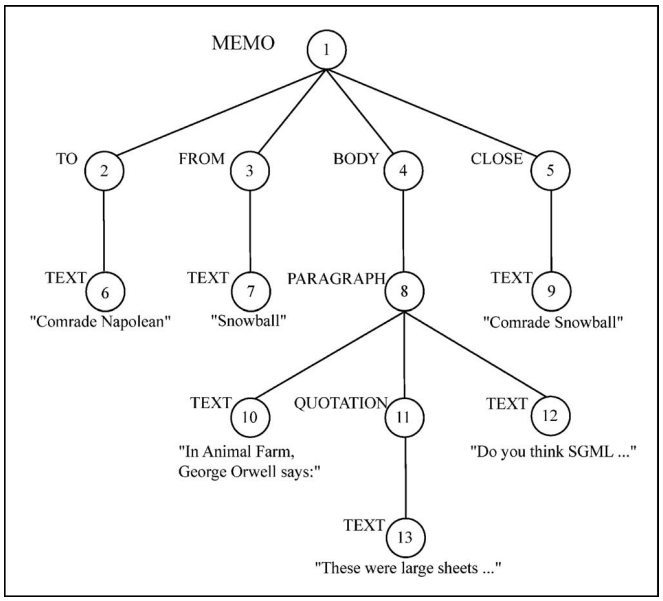

Fig. 3. The representation of the document in Fig. 2 using the document model.

interior nodes are defined as nodes that are not leaf nodes and their degrees are greater than 0.

2.2 Edit Operation

As with the previous works [15], [16], [17], [18], [19], [20], [21], this paper represents differences between structured documents as an edit script, that is, a sequence of edit operations that transforms an old version, T_1, of a document into its new version, T_2.

As shown in Table 1, in cases of the insert and delete operations, the proposed ones can describe the insertion and deletion of all nodes as with the Zhang and Shasha method and the Shasha and Zhang method [16]. On the other hand, the work of Chawathe et al. does not allow the insertion and deletion of an interior node, thus requiring the movement or deletion of all descendants before deleting an interior node.

An update operation is allowed for textual data only (value of a text node). This is due to the domain characteristics of SGML/XML editing. When we edit SGML/XML documents, generally we do not change the name of an element but insert or delete an element, move or copy a subtree, or update textual data. Therefore, like the work of Chawathe et al., the proposed update operation expresses the change of textual data. The update of the Zhang and Shasha method and the Shasha and Zhang method means the change of a label. On the other hand, the move and copy operation have the same meaning as the move operation in the work of Chawathe et al. and the copy operation of the Chawathe and Molina method.

2.3 Cost Model

There could exist a number of edit scripts for expressing differences between two versions, T_1 and T_2. This paper presents a model for determining an optimal[5] edit script. Let $c_i(x)$, $c_d(x)$, and $c_u(x)$ denote, respectively, the costs of inserting, deleting, and updating a node x, and let $c_m(x)$ and $c_c(x)$ denote the costs of moving and copying a subtree x,[6] respectively.

In this paper, for easier computation, we define $c_i(x)$, $c_d(x)$, $c_m(x)$, and $c_c(x)$ as unit cost; that is, $c_i(x) = c_d(x) = c_m(x) = c_c(x) = 1$ for all x. For a text node x, $c_u(x)$ is determined by a function, $compare(v_1, v_2)$, which computes the difference between the old value v_1 and the new one v_2 of x. The *compare* function is based on the longest common subsequence (LCS) [12], [15], as follows:

$$compare(v_1, v_2) = -2 \times lcsr(v_1, v_2) + 2, \tag{1}$$

where $lcsr(v_1, v_2)$denotes the rate of an LCS between two texts v_1 and v_2, and is defined by[7]

$$lcsr(v_1, v_2) = \frac{2 \times \mid lcs(v_1, v_2)}{|v_1| + |v_2|}. \tag{2}$$

The range of $c_u(x)$ is between 0 and 2. An example of determining edit operations based on the proposed cost model is as follows: Given text nodes x and y in the two versions, whose values and relative positions are different from each other, the difference between the two versions can be represented as one of two cases: 1) moving node x and then updating its value with the value of y and 2) deleting node x and inserting node y. The two cases express the differences in the move/update and delete/insert pairs, respectively.

To determine an optimal edit script, we consider the value of the *compare* function. If the value of the function between the nodes is less than 1, the cost of the move and update operations (<2) is less than the cost of the insert and delete operations (=2). Thus, it is more cost effective to choose the move and update operations. Otherwise, the insert and delete operations will be chosen.

3 Related Work

As mentioned before, previous works can be categorized into three classes by their targets that are flat-file, relational data, and hierarchical data. In this section, a brief discussion about conventional methods that deal with hierarchical data, especially ordered trees, is presented along with their constraints. Additionally, the properties and constraints of our approach are introduced.

The Zhang and Shasha method and the Shasha and Zhang method target a labeled ordered tree and represent differences between two trees as simple operations such as the insertion, deletion, and update of a node. They have several constraints. First, they allow only one-to-one matchings[8] between nodes. As a result, the copy of a subtree is not supported. However, as illustrated in the DTD of Fig. 1, because repeated subtrees are common in XML documents, subtree operations such as copy and movement should be supported.

5. This paper does not prove that the edit script found is minimal. This means that it may be nonminimal.

6. This means the subtree that has a node x as its root node.

7. $|lcs(v_1, v_2)|$, $|v_1|$, and $|v_2|$ denote the length of an LCS of v_1 and v_2, the length of v_1, and the length of v_2, respectively.

8. For any matchings $[i_1, j_1]$ and $[i_2, j_2]$, $i_1 = i_2$ if and only if $j_1 = j_2$.

TABLE 1
Edit Operations

Operation	Description
Insert	The insertion of a new node n into T_1 is represented by an *insert* operation, denoted by $INS((n, l, v), p, c, k)$. A node n with a label l and a value v is inserted as the kth child of a node p of T_1 and has c, that is, a partial set of child nodes of p, as its children. As a result of performing an *insert* operation, n becomes the kth child of p and includes c as its children.
Delete	A *delete* operation, $DEL(n)$, is an inverse of an *insert* operation. Each child node of the deleted node again becomes the child of the deleted node and child ordering is preserved. Because of the characteristic of SGML/XML, a root node, that is, a document element, cannot be deleted.
Update	An *update* operation, $UPD(n, v_1, v_2)$, updates the value of a node n from v_1 to v_2. In a structured document, the update of an element does not mean the update of its label but the update of the textual data. The proposed *update* operation is only for the value of a text node and the update of an interior node is not supported.
Move	A *move* operation, $MOV(n, p, k)$, moves a subtree n that has n as its root node to the kth child of a node p.
Copy	A *copy* operation, $CPY(n, p, k)$, copies a subtree n that has n as its root node to the kth child of a node p.

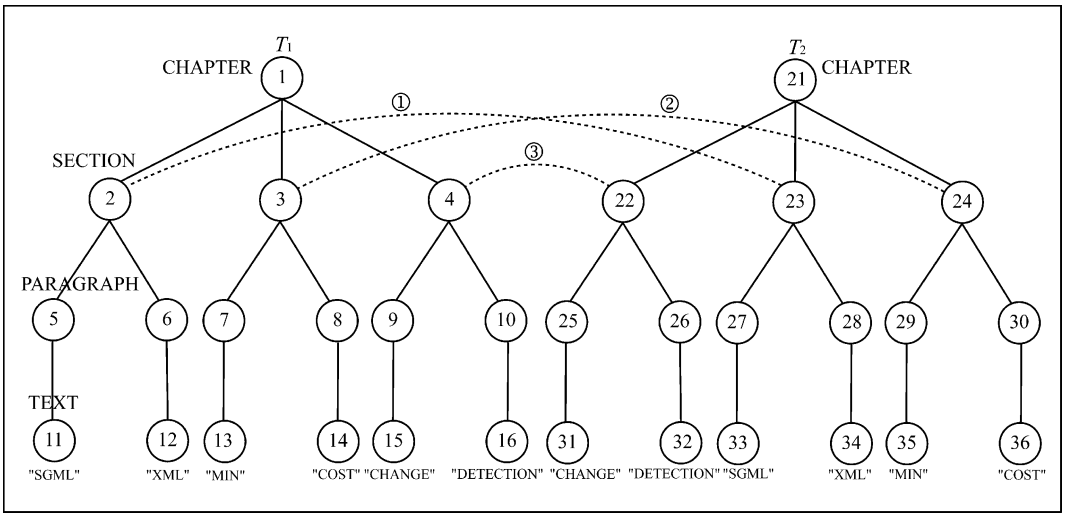

Fig. 4. An example of matching relationships violating the sibling order (the dotted lines represent matching relationships between nodes).

Second, because they have the constraint that the sibling order[9] between nodes that give rise to matchings should be preserved, they cannot detect the movement of a subtree. For example, in Fig. 4 where subtree 4 has been moved to the first child of node 1, they do not allow matchings ①, ②, and ③ that violate the sibling order but are meaningful and intuitive. Therefore, they cannot detect the movement of the subtree 4.

Third, the ancestor order[10] among nodes that form matching relationships must be preserved. For example, assume that T_1 has been transformed into T_2 by applying two move operations sequentially as shown in Fig. 5. Likewise, matchings ① and ② in Fig. 6 are not allowed because they violate the ancestor order.

The matchings that are accepted by the Zhang and Shasha method are as shown in Fig. 7. The resultant edit script is a sequence of three deletions (deletion of nodes 4, 6, and 7) and three insertions (insertion of nodes 12, 14, and 16).

However, if the method allowed the matchings, ① and ②, of Fig. 6, it could detect more cost-effective and structurally meaningful changes, movements of subtree 4 and subtree 2 as in Fig. 5.

Zhang [17] proposes an algorithm with a time complexity $O(|T_1||T_2|)$ with additional constraints to the Zhang and Shasha method that runs in time

$$O(|T_1||T_2| \min(\text{depth } (T_1), \text{leaves } (T_1)) \min(\text{depth } (T_2), \text{leaves } (T_2))).$$

9. For any matchings $[i_1, j_1]$ and $[i_2, j_2]$, i_1 is to the left of i_2 if and only if j_1 is to the left of j_2.

10. For any matchings $[i_1, j_1]$ and $[i_2, j_2]$, i_1 is an ancestor of i_2 if and only if j_1 is an ancestor of j_2.

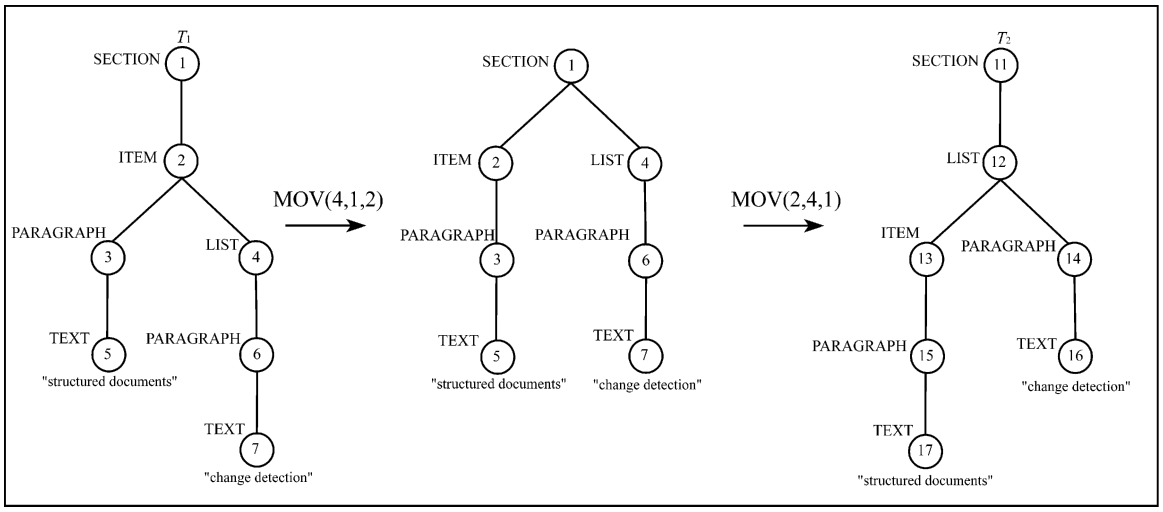

Fig. 5. An example of two movements.

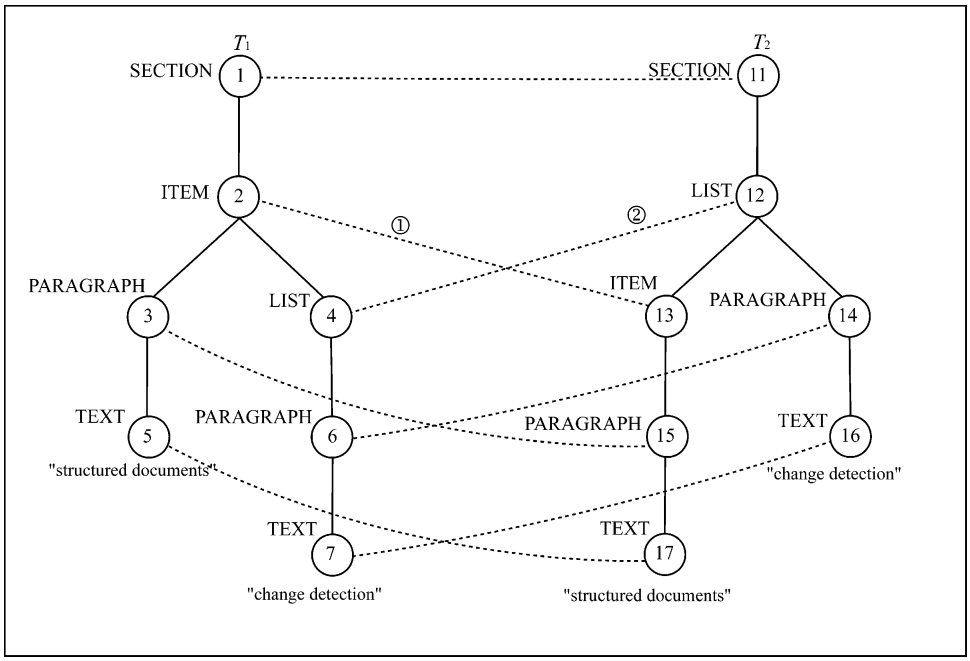

Fig. 6. An example of matching relationships, ① and ②, that violate the ancestor order.

Chang et al. [18] and Wang et al. [19] have implemented a system that computes differences in SGML documents based on the Zhang and Shasha method.

Chawathe et al. propose an algorithm for change detection in structured data. The algorithm consists of two stages: finding matchings and computing an edit script. It can detect the movement of a subtree as well as the insertion, deletion, and update of nodes. However, it assumes an *acyclic labels condition* for labels. For this reason, it cannot process a document that contains cyclic elements (e.g., elements ITEM and LIST contain LIST and ITEM, respectively, as shown in Fig. 6). Actually, the article and book DTDs of ISO 12083 [26] allow the parent and child ITEM and LIST elements to be inverted. Another constraint is that any two-leaf nodes do not have the same or similar values. Because the work of Chawathe et al. targets structured data with limited conditions, it has faster processing speed compared to the Zhang and Shasha method. However, an SGML/XML document has not only quite a few cyclic elements but also textual data that are the same or almost identical. Therefore, the method proposed by Chawathe et al. cannot extract an optimal edit script and its possible area of application is limited.

On the other hand, Chawathe and Molina have recently presented a heuristic algorithm for meaningful change detection that supports the movement and copy of a subtree and has no constraints of the work of Chawathe et al. The method deals with the problem for an unordered tree, of which the NP-hardness has been proven by the work of Zhang et al. [27]. Although this is a current state of the art method, it is slow and its data model is missing ordering information about elements defined in an SGML/XML DTD.

Our work aims at developing an algorithm that computes meaningful changes in SGML/XML documents faster than previous works. To this end, the proposed method is based on the domain characteristics of SGML/XML documents.

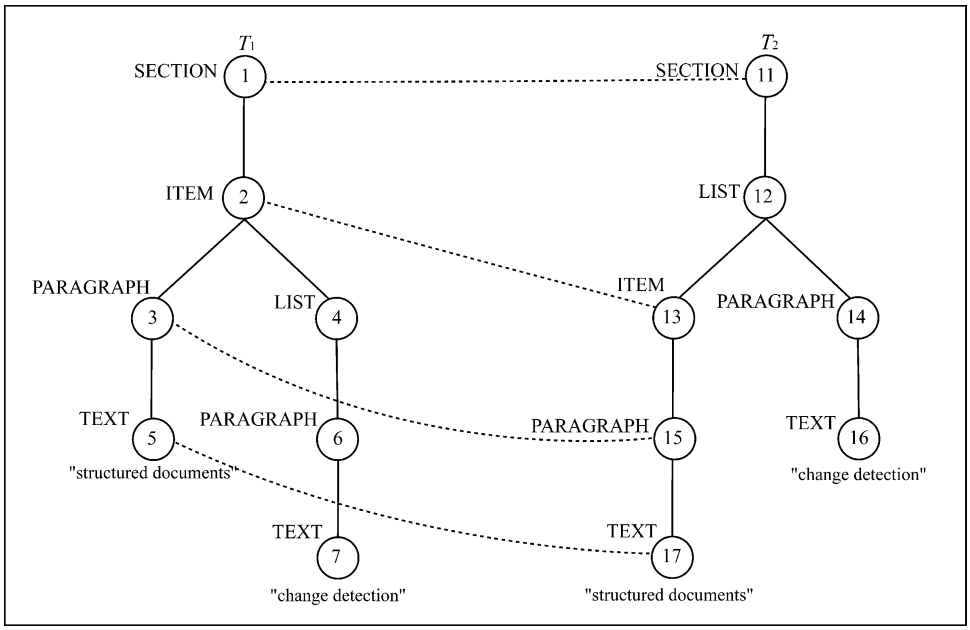

Fig. 7. An example of matching relationships preserving the ancestor order.

First, we require that the labels of parent nodes of text nodes, which match each other, be the same. This is because the minimum independent logical unit in an SGML/XML document is an element. Second, it is inefficient to investigate possible existence of matchings for all interior nodes that have the same label. The nodes whose leaf-node descendents matcheach other are compared.

Definition 1: Path. *For a node x, $Path\ (x)$ is a sequence of nodes from the parent node of x to the root node if x is not the root.*

Definition 2: The Corresponding Paths (CP). *$Two\ paths$, $Path\ (x)$ and $Path\ (y)$, are called corresponding when there is a matching relationship $[x, y]$. The correspondence of $Path\ (x)$ and $Path\ (y)$ is denoted by $CP\ (x, y)$. For example, $Path\ (5)$ and $Path\ (7)$ correspond to $Path\ (17)$ and $Path\ (16)$, respectively, in Fig. 6.*

Particularly, we constrain the ancestor order not in the whole trees but in the corresponding paths to be preserved. This is because the movements of subtrees are not allowed within the corresponding path under the definition of the move operation. For example, we allow the matchings ① and ② in Fig. 6, which violate the ancestor order in the whole trees but preserve the ancestor order in the corresponding paths. Specifically, because the text nodes 5 and 17 match each other, matchings [3, 15], ①, and [1, 11] are created from the corresponding paths, {3, 2, 1} and {15, 13, 12, 11}. Additionally, matchings [6, 14] and ② are produced from *CP* (7, 16).

In Fig. 8, where a new version has been created by actually deleting node 4 and inserting node 12, we don't allow matching ⑤ because it violates the ancestor order in CP (7, 17). In the case of matching ⑤, where the two nodes have the same label and share common descendents, if a method for producing matchings does not consider the proposed ancestor-order preservation of the corresponding paths, it may not compute an optimal edit script or should use an additional search module that explores matchings to find a better solution with an exploration that may be extremely time-consuming. Previous works that support the movement of a subtree does not consider this constraint. Section 4.1 describes the proposed method for producing matchings based on these properties and constraints in detail.

4 The Proposed Algorithm

The proposed algorithm is divided into two stages: producing matchings and computing an edit script. First, we find matching relationships between old and new versions of a document by applying the path-matching algorithm in a bottom-up manner. Then, an edit script is computed by doing a top-down breadth-first search on the two versions.

4.1 Producing Matchings

Previous works [14], [15], [16], [17], [21] generate matchings with consideration only of the labels of nodes. In the case of structured documents, elements might exist with the same label but with different content. For example, an SGML/XML document usually has quite a few *sections* that have different content. Therefore, the proposed algorithm takes both textual and structural information into account. For instance, for element nodes that have the same label, if there is no structural similarity between their children, a matching between them is not allowed.

The proposed method produces matching relationships-between text nodes, and on the basis of these relationships, determines matchings between element nodes in a bottom-up manner. A detailed description about the method for producing matchings is as follows:

Matching Criterion 1. Let $label\ (x)$, $value\ (x)$, and $parent\ (x)$ denote the label, value, and parent node of a

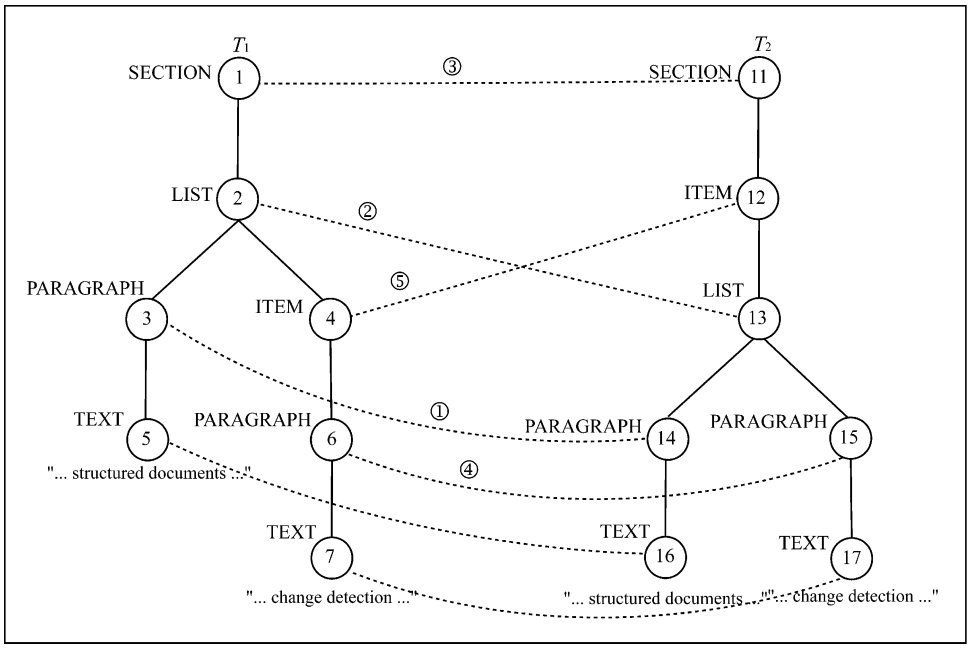

Fig. 8. An example of violating the ancestor order in the corresponding paths.

node x, respectively. For each text node $y_j \in T_2, 1 \leq j \leq n$, there might exist zero, one, or more than one text node $x \in T_1$, where $label\ (parent\ (y_j)) = label(parent\ (x))$ and $compare\ (value\ (y_j),\ value\ (x)) < t$ for some threshold t. We choose the nodes $x_{i1} \ldots x_{ik}, 1 \leq i_1 < \ldots < i_k \leq m$, whose values of the compare function correspond to the minimum, and produce matching relationships, $[x_{i1}, y_j] \ldots [x_{ik}, y_j]$.

First, the method produces matchings between text nodes in the two versions using the Matching Criterion 1. As a result, text nodes in the old version have the possibility of one-to-one, one-to-many, many-to-one, or no matching relationships with text nodes of the new version. In the case of many-to-one matchings, it can be considered that, except for one node, the rest of the nodes have already been deleted. Because it is not necessary to produce matchings for deleted nodes, many-to-one matchings are changed to one-to-one according to the Matching Criterion 2. Therefore, text nodes of the old version have one-to-one, one-to-many, or no matchings. As a result, text nodes of the new version have either one-to-one or no matchings with text nodes of the old version.

Definition 3: The LCS of the Corresponding Paths. *Given a sequence $S = l_1 \ldots l_n$, a sequence S' is a subsequence of S if it can be obtained by deleting zero or more elements from S. That is, $S' = l_{i1} \ldots l_{im}$, where $1 \leq i_1 < \ldots < i_m \leq n$. Given the corresponding paths, $CP\ (x, y)$, its LCS is defined as a sequence $S'' = (x_1, y_1) \ldots (x_k, y_k)$ such that 1) $\{x_1 \ldots x_k\}$ is a subsequence of $Path\ (x)$, 2) $\{y_1 \ldots y_k\}$ is a subsequence of $Path\ (y)$, 3) the labels of nodes x_i and y_i, $1 \leq i \leq k$, are the same, and there is no sequence that satisfies 1), 2), and 3) and is longer than S''.*

Definition 4: The Corresponding Paths with Highest Similarity. *For a node y that has one-to-many matchings with k nodes $x_1 \ldots x_k$, the LCS (refer to Definition 3) of each $CP\ (y, x_i)$, $1 \leq i \leq k$, can be computed. The corresponding path with the highest similarity is defined as the CP with the longest LCS. If there exists more than one CP, we can choose the CP whose LCS has the largest number of pairs of nodes with the same sibling order. If more than one CP has the same priority, an arbitrary one might be selected.*

Matching Criterion 2. For any text node $y \in T_2$, one-to-many matchings are changed to one-to-one. Given that there are one-to-many matchings between a node y and nodes $x_1 \ldots x_k \in T_1$, the matching between y and x_i, $1 \leq i \leq k$, whose $CP\ (y, x_i)$ has the highest similarity (refer to Definition 4), is selected.

Second, matchings between interior nodes are produced. The proposed path-matching algorithm produces matchings between interior nodes from the corresponding paths of leaf nodes that match each other. Fig. 9 shows the overall procedure ofthe path-matching algorithm. Specifically, the algorithm constraints the ancestor order within the corresponding paths to be preserved (lines 14-20).

Definition 5: The Similarity of a Matching. *The similarity of a matching $[x, y]$ implies how similar two subtrees with root nodes, $x \in T_1$ and $y \in T_2$, are and is computed by:*

$$\frac{|common(x, y)|}{|descendant(x)|}, \tag{3}$$

where $common\ (x, y)$ denotes a set $\{z \mid [w, z] \in M$ and $w \in descendant\ (x)$, $z \in descendant\ (y)$, for $label\ (x) = label\ (y)\}$. Here, $descendant\ (x)$ is defined as a set of all nodes in subtree x. Let $|S|$ denote the number of nodes that a set S contains.

Matching Criterion 3. For any interior node $y \in T_2$, one-to-many matchings are changed into one-to-one. Consider matchings $[y, x_1] \ldots [y, x_k]$, where $x_1 \ldots x_k$ are interior nodes in T_1. To determine the most appropriate matching $[y, x_i]$, $1 \leq i \leq k$, a matching with the highest similarity is chosen (refer to Definition 5). If there exists more than one

```
1:  Input:  OldPath[m]  - a path of the old version,
2:                        m (∈ positive integer) is the number of nodes
3:          NewPath[n]  - a path of the new version,
4:                        n (∈ positive integer) is the number of nodes
5:          M   - matching
6:  Output: M   - updated matching
7:  Function definition:
8:          char* label(x) ::= returns a string that is the label of a node x.
9:          Boolean IsEqual(s, t) ::= if s equals t, returns TRUE, else returns FALSE.
10: Method:
11:         int j = 0; // index initialization of NewPath[n]
12:         for(int i = 0; i < m; i++) {
13:           for(int k = j; k < n; k++) {
14:                 // A condition for preserving the ancestor order in the corresponding paths.
15:                 if((NewPath[k] is already matched) &&
16:                       (a target node[11] of NewPath[k] exists in OldPath)) {
17:                       // it is not necessary to proceed further
18:                       // due to the characteristics of the path
19:                       return M;
20:                 }
21:                 if(IsEqual(label(OldPath[i]), label(NewPath[k]))) {
22:                       // a new matching [OldPath[i], NewPath[j]] is created
23:                       [OldPath[i], NewPath[j]] → M;
24:                       j++;
25:                       break;
26:                 }
27:           }
28:         }
```

Fig. 9. The path-matching algorithm. 11. The node that is in matching with the current node.

matching with the highest rate of similarity, a matching whose nodes have the same sibling order is selected. If more than one matching is available, the similarity of paths is considered (refer to Matching Criterion 2).

The path-matching algorithm might produce many-to-one matchings between interior nodes as shown in Fig. 10. Likewise, many-to-one matchings are changed into one-to-one by applying the Matching Criterion 3. This paper introducesin particular a new concept, the similarity of a matching, in order to extract structurally meaningful changes.

In Fig. 10, both nodes 22 and 23 of T_2 have one-to-many matchings. According to the Matching Criterion 3, the similarities of matchings are first computed and the one with the highest similarity is chosen. For node 22, the similarities of two matchings, ② and ③, are the same with the value, 0.5(=2/4). Hence, by considering the sibling order, the matching ② is chosen (both 2 and 22 are the first sibling). Similarly, matching ④ is chosen for the node 23.

4.2 Computing an Edit Script

To compute an edit script is to extract a sequence of edit operations by doing top-down breadth-first scans on old and new versions based on their matchings. Detailed specification on the algorithm for computing an edit script is as shown in Fig. 11. The algorithm takes as input old and new versions and their matchings; it returns a sequence of edit operations (lines 1-5). Nodes that have no matching in the old and new versionsare subjectsfor deletion (lines 11-13) and insertion (lines 26-29), respectively.

First, for a node[12] that has a one-to-one matching, if it is a text node and the value of the *compare* functionis not 0, the update operation is extracted (lines 17-18). If it is an interior node, matchings of its children are considered to detect their movement (lines 15-16). If the node has one and only one child node, the possibility of interparent movement[13] is investigated. If the number of child nodes is more than one, the method investigates the possibility of intraparent movement[14] and then determines interparent move operations.

12. In this section, the method considers the node of the old version thathas some matching, except when explicitly stated.

13. The movement corresponding to a matching $[x, y]$ such that $[parent\ (x), parent\ (y)] \notin M$.

14. The movement that changes the relative ordering of siblings.

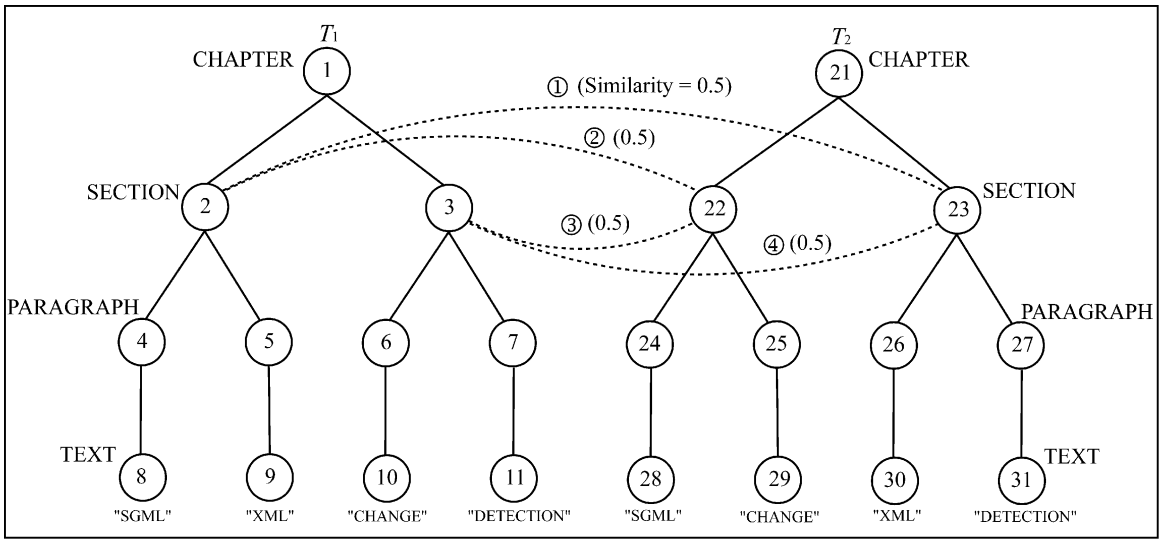

Fig. 10. An example of many-to-one matchings between interior nodes.

```
1:  Input: T1  - an old version
2:         T2  - a new version
3:         M   - matchings
4:  Output: S = S1 ··· Sm, where Si ∈ {NIL, INS, DEL, UPD, MOV, CPY},
5:                                 1 ≤ i ≤ m  - a sequence of edit operations
6:  Function definition:
7:      float compare (x, y) ::= returns a value whose range is between 0 and 2.
8:      MOV extractMOV (x, y) ::= extracts MOVs from matching relationships
9:                                  between children of x and y.
10: Method:
11:     while (top-down breadth-first search of an old version) {
12:         if (a node x ∈ T1 does not have a matching)
13:             DEL (x) → S;
14:         else if (a node x has a one-to-one matching with y ∈ T2) {
15:             if (x is an interior node)
16:                 extractMOV(x, y) → S;
17:             else if ((a text node) && (compare (x, y) > 0))
18:                 UPD (x, y) → S;
19:         }
20:         else if (a node x has one-to-many matchings with y1 ... yk) {
21:             determines the matching [x, yi], 1 ≤ i ≤ k, that corresponds to NIL;
22:             computes CPY and INS operations from the remaining matchings;
23:             extractMOV(x, yi) → S;
24:         }
25:     }
26:     while (top-down breadth-first search of a new version) {
27:         if (a node y ∈ T2 that does not have a matching)
28:             INS (y) → S;
29:     }
```

Fig. 11. The algorithm for computing an edit script.

To determine the intraparent move operation, we apply the LCS-based method that is suggested by Chawathe et al. to extract the minimum number of move operations. For instance, two cases can be considered from matchings ①, ②, and ③ of Fig. 4. One is moving nodes 2 and 3 to the right of node 4 and the other is moving node 4 to the left of node 1. Therefore, we can select the second one that is more cost-effective. The procedure for extracting the intraparent move operation is a process of aligning a set of misaligned nodes to preserve the sibling order.

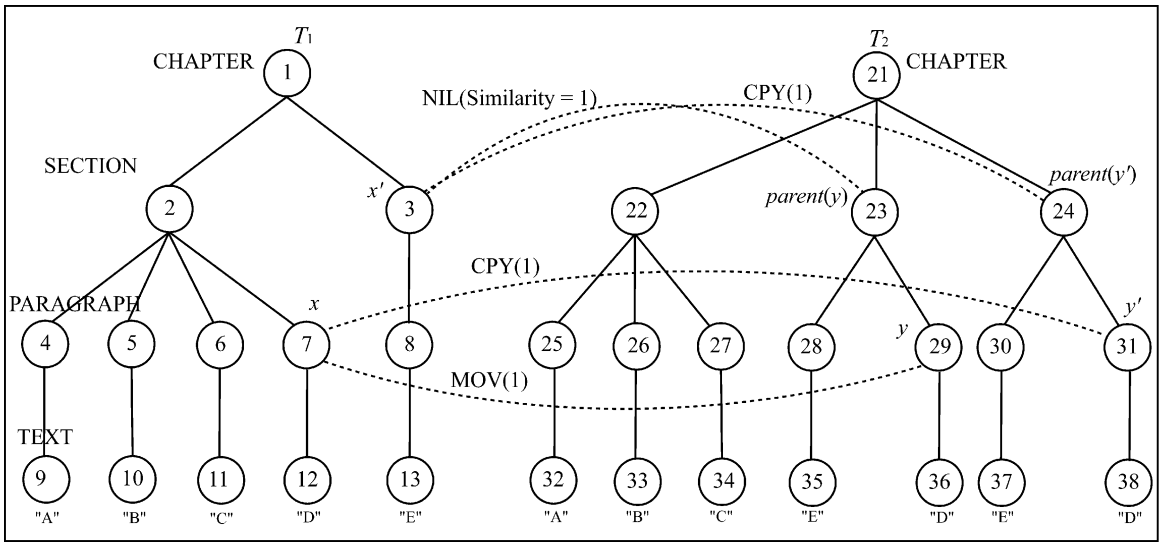

Fig. 12. Edit operations with nonoptimal cost.

NIL[15] **Criterion**. If there has already been a matching that was found to be a subject for the move operation, it is also considered to be a subject for NIL. For other cases, for a node $x \in T_1$, a matching $[x, y_i]$, $1 \leq i \leq k$, which has the highest similarity, is chosen out of the matchings, $[x, y_1] \ldots [x, y_k]$. If there exists more than one matching satisfying the above condition, the one where the matching between their parent nodes, $[parent\ (x), parent\ (y_i)]$, already marked with NIL, is selected. Unless there exists a matching $[parent\ (x), parent\ (y_i)]$ marked with NIL, an arbitrary matching $[x, y_i]$ can be selected and the move operation should be computed from this one.

A method for computing an interparent move operation is as follows: For two arbitrary matchings, $[x, y]$ and $[parent\ (x), z]$, if node z is not the parent node of node y, this is the case where node x has been moved to the position of y. In particular, if the moved node is a text node and compare(x, y) is less than 1, this is the case where the text node in question has been moved and then updated. The following Interparent Movement Criterion describes a formal specification on extracting the interparent move operation.

Interparent Movement Criterion. For a matching $[x, y]$ that corresponds to NIL (refer to the NIL Criterion), let $c_1 \ldots c_m$ be the children of node x. If c_i, $1 \leq i \leq m$, does not have any matching with children of node y, this is the case where c_i has been moved. If c_i has more than one matching, the one with the highest similarity is chosen as a target for the move operation. If more than one matching with the same rate of similarity exists, a matching is chosen nondeterministically among them.

If the similarity of matching $[c_i, z]$, which has been chosen by the Interparent Movement Criterion, is 1, every descendant of c_i and z has a one-to-one matching that corresponds to NIL. If the rate of similarity is less than 1, this is the case where some descendants of c_i have already been moved. For example, in Fig. 6, the similarity of matching [2, 13] is 0.4 (=2/5). Hence, we can figure out that a part of descendants (subtree 4) has been moved already. In this case, for accurate extraction of edit operations, the order in which the movement has been applied should be considered. On the other hand, if the similarity is larger than 1, this implies that new nodes have been inserted or copied after the movement.

Fig. 12 shows the case where subtree 7 has been moved to the second child of node 3 and then subtree 3 has been copied to the third child of node 1. However, from this case, the algorithm extracts the copy operation and the copy and move operations for subtree 3 and subtree 7, respectively, with the result of nonoptimal edit script.

A method for solving this problem is as follows: Assume that the move and copy operations are applied to two matchings, $[x, y]$ and $[x, y']$, respectively, and $parent\ (y)$ and $parent\ (y')$ are matched to node x'. If x' and $parent\ (x)$ are not the same node and the copy operation has been applied to $[x', parent\ (y')]$ either directly or indirectly,[16] this is when, after moving node x to the child of node x', node x' or the ancestor node of x' has been copied. This corresponds to an optimal edit script as shown in Fig. 13.

Second, in the case of a node that has one-to-many matching relationships (lines 20-24), a matching that corresponds to *NIL* is found by applying the *NIL* Criterion, and then the copy and insert operations are extracted from the remaining matchings. Specifically, if the rate of similarity of a matching that has not been marked with *NIL* is 1, and then the copy operation is assigned. Otherwise, the insert operation is assigned. Additionally, the move operations are extracted from matchings of its children in the same manner as given above for the case of a node that has a one-to-one matching.

For example in Fig. 14, the proposed method computes a sequence of edit operations: *INS*(22), *INS*(24), *MOV*(3), *INS*(27), *INS*(30), *MOV*(7), *INS*(33), *CPY*(9), and *CPY*(11). The detailed explanation about the computing process is as follows:

15. Matchings that do not correspond to any edit operations are marked with *NIL*.

16. Due to one of its ancestors being copied.

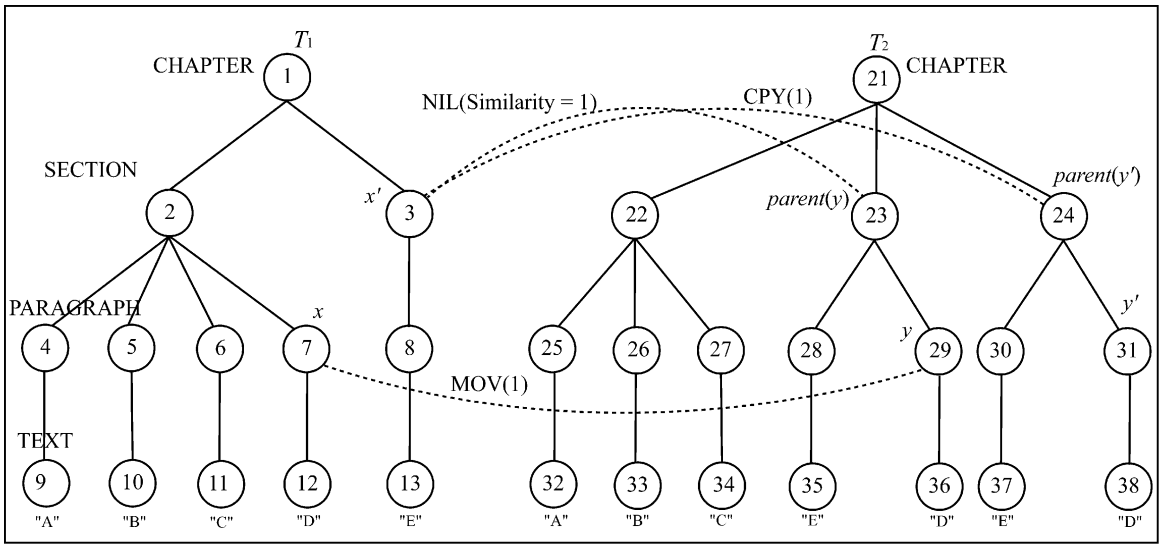

Fig. 13. Edit operations with optimal cost.

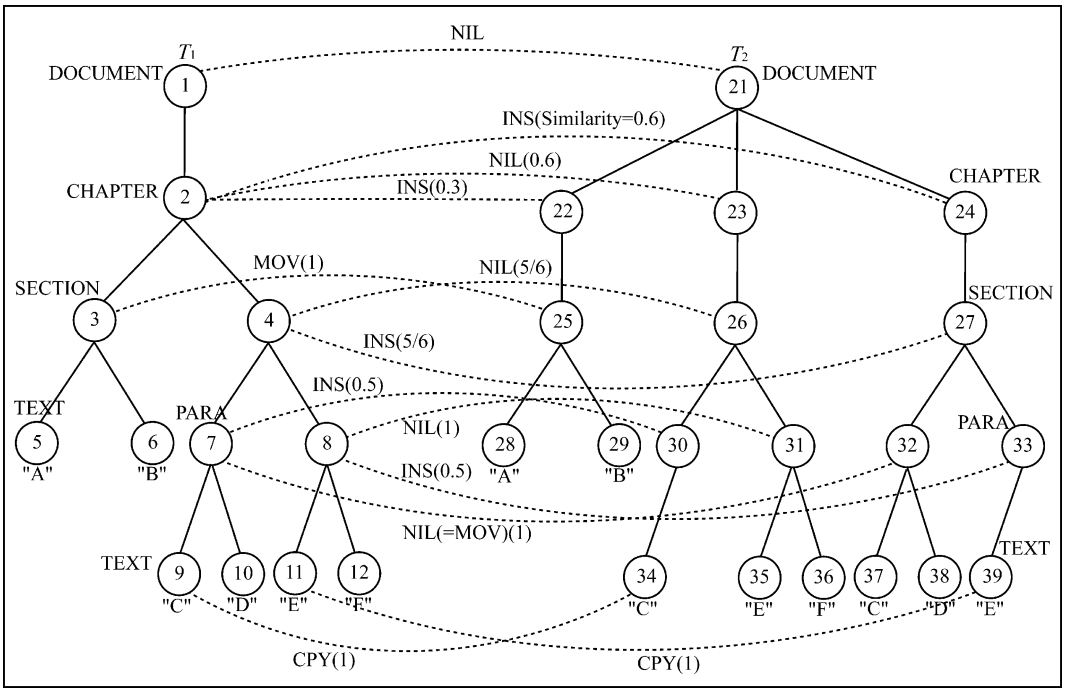

Fig. 14. An example for illustrating the extraction process of an edit script.

1. Since a matching between two root nodes, 1 and 21, is one-to-one, it corresponds to *NIL*.
2. Node 2 has one-to-many matchings. First, by applying the *NIL* Criterion, matching [2, 23] is chosen for *NIL* and two insert operations, *INS*(22) and *INS*(24), are extracted from the remaining matchings. Then, from matchings of child nodes, the existence of the move operation is investigated. Node 3 does not match with the child node of node 23. Hence, the move operation, *MOV*(3), is determined from matching [3, 25].
3. For matching [3, 25], the move operation has been already assigned.
4. Matching [4, 26] is marked with *NIL* according to the *NIL* Criterion. For matching [4, 27], the insert operation, *INS*(27), is determined because its rate of similarity is not 1. In other words, this is the case where node 27 is inserted as the child of node 24. On the other hand, two nodes, 5 and 6, form *NIL* relationships with nodes 28 and 29, respectively.
5. Node 7 matches with two nodes, 30 and 32. By considering the rate of similarity, matching [7, 32] is drawn as a target for *NIL* and matching [7, 30] is chosen for the insert operation, *INS*(30). Especially, since the parent nodes of nodes, 7 and 32, are not in *NIL* relationship, the move operation, *MOV*(7), is extracted from matching [7, 32].
6. Matching [8, 31] corresponds to *NIL* and the insert operation, *INS*(33), is extracted from matching [8, 33].
7. According to the *NIL* Criterion, two matchings, [9, 37] and [11, 35], correspond to *NIL* and two copy

TABLE 2
A Comparison of the Method with Related Works in Time Complexity

Step \ Algorithm	The Zhang and Shasha method	The method of Chawathe et al.	The proposed
Producing matchings	$O(n^2)$ [17]	$O(l^2c + ml)$	$O(l^2c + (m_1/l_1 + m_2/l_2)l_2))$
Computing an edit script	$O(n^2 \log^2 n)$	$O(nd)$	$O(nd)$

n: the total number of nodes, l: the total number of leaf nodes
l_1: the total number of leaf nodes of T_1, l_2: the total number of leaf nodes of T_2
m: the total number of internal nodes, m_1: the total number of internal nodes of T_1
m_2: the total number of internal nodes of T_2, d: the total number of misaligned nodes
c: the average cost of $c_M(x, y)$ for a pair of leaf nodes x and y

17. Chawathe et al. described that matchings can be produced in time $O(n^2)$ by postprocessing the output of the Zhang and Shasha method.

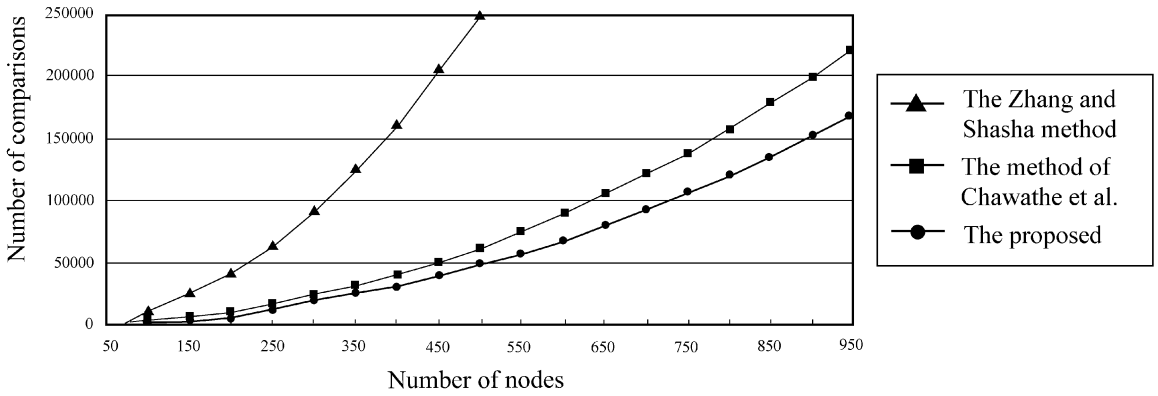

Fig. 15. Running times of producing matchings.

operations, *CPY*(9) and *CPY*(11), are extracted from matchings, [9, 34] and [11, 39], where the similarity is 1.

5 Performance Analysis

This paper presents an efficient algorithm for meaningful change detection between SGML/XML documents based on their domain characteristics. In this section, the performance of the proposed algorithm is quantitatively analyzed and compared with previous works that target ordered trees or structured documents in terms of the processing speed. Additionally, by experimenting with large number of documents, we have investigated the accuracy of the proposed method. The trade offs and implications of our approach are also qualitatively discussed.

Table 2 shows the result of comparing the processing speed of the proposed algorithm with conventional methods. Particularly, Fig. 15 shows how the running times of the Zhang and Shasha method, the work of Chawathe et al., and the proposed algorithm vary with the number of nodes in terms of producing matchings.

Theorem 1. *The average running time of the path-matching algorithm for producing matchings between interior nodes in old and new versions is proportional to $O((m_1/l_1 + m_2/l_2)l_2)$, where m_1 (and m_2) and l_1 (and l_2) denote the numbers of interior nodes and leaf nodes of T_1 (and T_2), respectively.*

Proof. Because the proposed algorithm does not allow many-to-one matchings (it produces one-to-one or one-to-many), the maximum number of matchings for leaf nodes is l_2. Thus, it is possible to assume that the average numbers of nodes comprising paths of leaf nodes in T_1 and T_2 are m_1/l_1 and m_2/l_2, respectively. As a result, it takes the time that is proportional to $(m_1/l_1 + m_2/l_2)$, to create matchings from the corresponding paths. Therefore, the time complexity of the path-matching algorithm for producing matchings between interior nodes is $O((m_1/l_1 + m_2/l_2)l_2)$. □

Theorem 2. *The time complexity for computing an edit script is $O(nd)$, where n is the total number of nodes in T_1 and T_2 and d is the number of misaligned nodes that are targets for the intraparent move operation.*

Proof. Other than aligning nodes to determine the intraparent movement, the breadth-first search of the proposed algorithm runs in time $O(n)$. First, assume that $|x|$ is the number of the children of a node $x \in T_1$. Also, assume that $d(x, y)$ is the number of the misaligned children, which are the targets for the move operation, of nodes x and $y \in T_2$. Then, the time required for aligning the children of x and y is bounded by $O((|x| + |y|)d(x, y))$. Therefore, the total time complexity is $O(nd)$. □

The proposed method produces matchings between leaf nodes in the two versions and, on the basis of these matchings, determines matchings between interior nodes, like the work of Chawathe et al. However, due to the fact

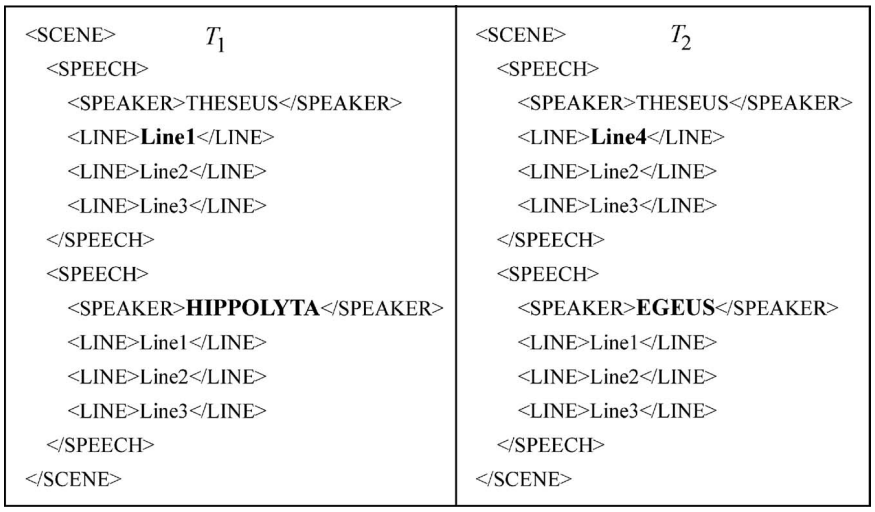

Fig. 16. Two sample documents.

that the path-matching algorithm is able to find matchings between interior nodes in time $O((m_1/l_1 + m_2/l_2)l_2)$, it is much faster than the algorithm of Chawathe et al. that runs in time $O(m_l)(= O((m_1 + m_2)(l_1 + l_2)))$. Theorems 1 and 2 prove the time complexity of the proposed algorithm.

To evaluate the accuracy of the proposed method, we have experimented with two collections of patent documents and the plays of Shakespeare. The patent documents have been published by the Korean Intellectual Property Office[18] in July 1998, which consists of a total of 200 documents with an average size of 18KB. The average numbers of element nodes and text nodes are 156 and 116, respectively, with the ratio of 1.3:1. The size of the plays of Shakespeare[19] used in our experiments ranges 139KB to 289KB. Concerning ground-truth data, the new versions of documents have been synthetically prepared by human experts who had much experience in SGML/XML documents. This is because of the difficulty of collecting real documents with two or more versions. Also, we are not aware of any formal method for getting representative samples of the way people would modify XML documents.

Specifically, to produce matchings between text nodes, we have used 1 as a value of threshold t of the Matching Criterion 1 for convenience sake. Depending on data-centric or document-centric documents, an appropriate value of the threshold may be different. Experimental results show that the proposed method has detected structurally meaningful changes. All of the movement and copy of subtrees have been also computed correctly.

There was one case where the proposed method did not generate minimum-cost edit scripts. For examples, Figs. 16 and 17 are two sample documents and their tree representations, respectively. From these two documents, the proposed method computed as a sequence of edit operations {UPD(13, "Line1", "Line4"), DEL(8), DEL(16), INS(8'), INS(16')}, where the minimum-cost script would be either {UPD(13, "Line1", "Line4"), DEL(16), INS(16')} or {UPD(13, "Line1", "Line4"), UPD(16, "HIPPOLYTA", "EGEUS")}. This is because our path-matching algorithm considers only interior nodes from the corresponding paths of leaf nodes that match each other. The result is not minimal but still meaningful because the two documents have different speakers.

Comparing with previous works from the perspective of producing matchings, we constrain the ancestor order in the corresponding paths to be preserved. If it were not considered, the method could not compute an optimal edit script or should use an additional search module that might be extremely time-consuming. As mentioned before, this is because the movement of a subtree is not allowed within the same path under the definition of the move operation. However, the previous works that supports the movement of a subtree do not take this into account.

We introduce the concept of the similarity of a matching in order to indicate structural similarity between interior nodes. Meanwhile, Chawathe et al. allow interior nodes to be matched with each other if and only if more than 50 percent of their leaf node descendents have already been matched. Hence, the matching criteria in their work might not support an edit script with optimal cost. It is somewhat difficult to evaluate which one among related works better reflects the degree of similarity between documents. This is because their goals are different from method to method. For example, while the Zhang and Shasha method computes a minimal edit script between labeled ordered trees, it does not support our structurally meaningful changes in XML documents.

On the other hand, this paper does not take attributes of an SGML/XML document into account because they are unordered and do not correspond to logical constituents. However, the problem can be solved through the extension of the document model. Specifically, an attribute list of an element, that is, a set of pairs of an attribute name and its value, might be represented as the value of the corresponding interior node. At the phase of computing an edit script, the attribute lists of the corresponding nodes can be considered.

18. http://www.kipo.go.kr/.
19. http://www.ibiblio.org/bosak/xml/eg/shaks200.zip.

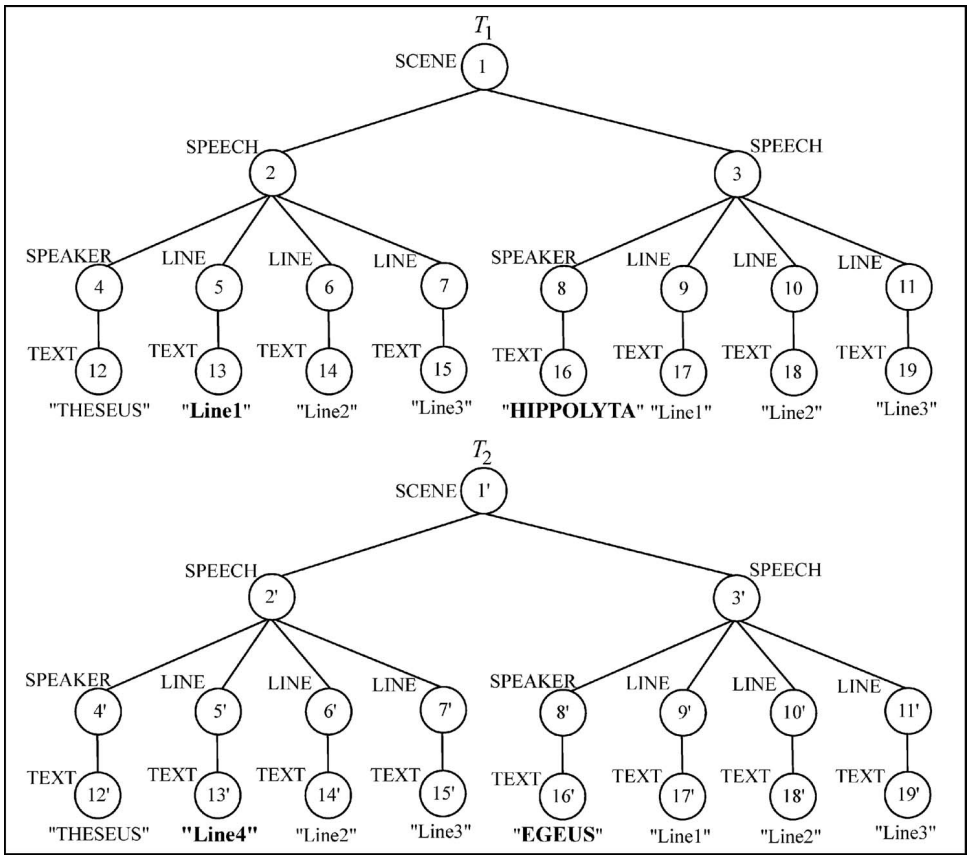

Fig. 17. The tree representations of two documents in Fig. 16.

Because this paper is based on the domain characteristics of SGML/XML documents, there might exist application domains where this method is not appropriate and the other algorithms are better. However, recently, the availability of large repositories of XML documents has been increasing rapidly and there is a growing interest in sophisticated processing tools for structured documents. Therefore, as mentioned in the introduction, the application potential of the proposed method is very high.

6 Conclusions

In this paper, we have presented an efficient algorithm for computing differences between old and new versions of an SGML/XML document faster than previous works. To this end, the proposed method is based on the domain characteristics of structured documents.

The proposed algorithm is based on a hybridization of bottom-up and top-down methods: The matching relationships between nodes in the two versions are produced in a bottom-up manner and then the top-down breadth-first search computes an edit script. Because the proposed path-matching algorithm does not need to investigate possible existence of matchings for all nodes, faster matching is achieved. We have also proposed the document model that facilitates the efficient representation of structured documents.

Additionally, we have introduced the concept of the similarity ofa matching and supported not only one-to-one but also one-to-many matchings in order to extract structurally meaningful changes. Therefore, the proposed algorithm can detect meaningful changes such as the movement and copy of a subtree as well as simple changes to the node itself like insertion, deletion, and update.

The larger the resultant edit script is, the more expensive it is to apply this script later. The "undo" that recovers an old version from the newest version of a document requires computing inverse operations of the resultant edit operations. The more concise the edit script is, the less time it takes to compute the inverse operations. In this perspective, the proposed edit operations are efficient because the move and copy operations can represent a set of deletions and insertions and a set of insertions, respectively.

As mentioned before, this paper does not prove that the edit script found is minimal. As a future work, we will investigate whether the proposed method can detect changes with minimum cost. On the other hand, for a more thorough evaluation of the proposed method, our future works include experiments with a larger set of documents includingboth data-centric and document-centric documents as well as comparisons with recently published algorithms [28], [29].

Acknowledgments

S.-B. Cho was supported by Korean Research Foundation grant KRF-2002-005-H20002.

REFERENCES

[1] ISO/IEC 8879, *Information Processing—Text and Office Systems —Standard Generalized Markup Language (SGML),* Int'l Organization for Standardization, 1986.

[2] W3C Recommendation REC-xml-200001006, *Extensible Markup Language (XML) 1.0,* second ed., World Wide Web Consortium, 2000, http://www.w3c.org/TR/REC-xml.

[3] A. Haake, "CoVer: A Contextual Version Server for Hypertext Applications," *Proc. Fourth ACM Conf. Hypertext,* pp. 43-52, Nov. 1992.

[4] K. Osterbye, "Structural and Cognitive Problems in Providing Version Control for Hypertext," *Proc. Fourth ACM Conf. Hypertext,* pp. 33-42, Nov. 1992.

[5] H. Moeller, "Versioning Structured Technical Document," *Proc. Workshop Versioning in Hypertext Systems,* Sept. 1994.

[6] P.B. Berra, S.J. Yoo, Y.K. Lee, and K. Yoon, "Version Management in Structured Document Retrieval Systems," *Proc. Eighth Int'l Conf. Software Eng. and Knowledge Eng.,* pp. 537-544, June 1996.

[7] M.A. Noronha, L.G. Golendziner, and C.S. dos Santos, "Extending a Structured Document Model with Version Control," *Proc. Int'l Database Eng. and Applications Symp.,* pp. 234-243, July 1998.

[8] W. Labio and H.G. Molina, "Efficient Snapshot Differential Algorithms for Data Warehousing," *Proc. 20th Conf. Very Large Databases,* pp. 63-74, Sept. 1996.

[9] J. Widom and S. Ceri, *Active Database Systems: Triggers and Rules for Advanced Database Processing.* Morgan Kaufmann, 1995.

[10] R. Wagner and M. Fischer, "The String-to-String Correction Problem," *J. Assoc. of Computing Machinery,* vol. 21, no. 1, pp. 168-173, Jan. 1974.

[11] R. Wagner, "On the Complexity of the Extended String-to-String Correction Problem," *Proc. Seventh ACM Symp. Theory of Computation,* 1975.

[12] E. Myers, "An O(ND) Difference Algorithm and Its Variations," *Algorithmica,* vol. 1, no. 2, pp. 251-266, 1986.

[13] S. Wu, U. Manber, G. Myers, and W. Miller, "An O(NP) Sequence Comparison Algorithm," *Information Processing Letters,* vol. 35, pp. 317-323, Sept. 1990.

[14] S. Ghandeharizadeh, R. Hull, D. Jacobs, J. Castillo, M.E. Molano, S.H. Lu, J. Luo, C. Tsang, and G. Zhou, "On Implementing a Language for Specifying Active Database Execution Models," *Proc. 19th Int'l Conf. Very Large Data Bases,* pp. 441-454, Aug. 1993.

[15] K. Zhang and D. Shasha, "Simple Fast Algorithms for the Editing Distance between Trees and Related Problems," *SIAM J. Computing,* vol. 18, no. 6, pp. 1245-1262, 1989.

[16] D. Shasha and K. Zhang, "Fast Algorithms for the Unit Cost Editing Distance between Trees," *J. Algorithms,* vol. 11, pp. 581-621, 1990.

[17] K. Zhang, "Algorithms for the Constrained Editing Distance between Ordered Labeled Trees and Related Problems," *Pattern Recognition,* vol. 28, no. 3, pp. 463-474, Mar. 1995.

[18] G.J.S. Chang, G. Patel, L. Relihan, and J.T.L. Wang, "A Graphical Environment for Change Detection in Structured Documents," *Proc. 21st Ann. Int'l Computer Software and Applications Conf. (COMPSAC),* pp. 536-541, Aug. 1997.

[19] J.T.L. Wang, D. Shasha, G.J.S. Chang, L. Relihan, K. Zhang, and G. Patel, "Structural Matching and Discovery in Document Databases," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 560-563, May 1997.

[20] S. Chawathe, A. Rajaraman, H.G. Molina, and J. Wisdom, "Change Detection in Hierarchically Structured Information," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 493-504, June 1996.

[21] S. Chawathe and H.G. Molina, "Meaningful Change Detection in Structured Data," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 26-37, May 1997.

[22] C.F. Goldfarb, *The SGML Handbook.* Oxford: Clarendon Press, 1990.

[23] M. Bryan, *SGML: An Author's Guide to the Standard Generalized Markup Language.* Addison-Wesley, 1988.

[24] H. Brown, "Standards for Structured Documents," *The Computer J.,* vol. 32, no. 6, pp. 505-514, 1989.

[25] C.F. Goldfarb and P. Prescod, *The XML Handbook.* Prentice Hall, 1998.

[26] ISO 12083, *Information and Documentation—Electronic Manuscript Preparation and Markup,* Int'l Organization for Standardization, 1994.

[27] K. Zhang, J.T.L. Wang, and D. Shasha, "On the Editing Distance between Undirected Acyclic Graphs," *Int'l J. Foundations of Computer Science,* vol. 7, no. 1, pp. 43-58, 1996.

[28] G. Cobéna, S. Abiteboul, and A. Marian, "Detecting Changes in XML Documents," *Proc. 18th Int'l Conf. Data Eng. (ICDE'02),* pp. 41-52, Feb. 2002.

[29] Y. Wang, D.J. Dewitt, and J.-Y. Cai, "X-Diff: An Effective Change Detection Algorithm for XML Documents," *Proc. 19th Int'l Conf. Data Eng. (ICDE '03),* Mar. 2003.

Kyong-Ho Lee received the BS, MS, and PhD degrees in computer science from Yonsei University, Seoul, Korea, in 1995, 1997, and 2001, respectively. Previously, he worked as a guest researcher of IT Laboratories at the NIST (National Institute of Standards and Technology), Maryland. Currently, he is an assistant professor in the Department of Computer Science at Yonsei University. His research interests include multimedia document engineering, knowledge and data engineering, pattern matching, and XML. He is a member of the Korea Information Science Society, the Korea Information Processing Society, the Korea Multimedia Society, the ACM, and the IEEE Computer Society.

Yoon-Chul Choy received the BS degree from Seoul National University, Korea, in 1973, and the MS and PhD degrees from the University of California, Berkeley, in 1976 and 1979, respectively. He is a professor of computer science at Yonsei University in Korea. After working as a research staff member for Lockheed Corp. and Rockwell International, he joined Yonsei University in 1984. Currently, he leads the Multimedia and Graphics Laboratory at Yonsei University. He is the president of the Korea Multimedia Society and has served as the chairman of Korea eBook Standard Committee. He was also a visiting professor at the University of Massachusetts and the Keio University (SFC) in Japan. His main areas of interests are multimedia document processing, 3D user interface in cyberspace, virtual reality, and eLearning. He is a member of the IEEE Computer Society.

Sung-Bae Cho received the BS degree in computer science from Yonsei University, Seoul, Korea, in 1988 and the MS and PhD degrees in computer science from KAIST (Korea Advanced Institute of Science and Technology), Taejeon, Korea, in 1990 and 1993, respectively. He worked as a member of the research staff at the Center for Artificial Intelligence Research at KAIST from 1991 to 1993. He was an invited researcher of Human Information Processing Research Laboratories at ATR (Advanced Telecommunications Research) Institute, Kyoto, Japan, from 1993 to 1995. In 1998, he was invited as a visiting scholar at University of New South Wales, Canberra, Australia. Since 1995, he has been a professor in the Department of Computer Science, Yonsei University. His research interests include neural networks, pattern recognition, intelligent man-machine interfaces, evolutionary computation, and artificial life. Dr. Cho was awarded outstanding paper prizes from the IEEE Korea Section in 1989 and 1992, and another one from the Korea Information Science Society in 1990. He was also the recipient of the Richard E. Merwin prize from the IEEE Computer Society in 1993. He was listed in *Who's Who in Pattern Recognition* from the International Association for Pattern Recognition in 1994, and received the best paper awards at the International Conference on Soft Computing in 1996 and 1998. He is a member of the Korea Information Science Society, INNS, the IEEE Computer Society, and the IEEE Systems, Man, and Cybernetics Society.