HybridDiff: An Algorithm for A New Tree Editing Distance Problem

I-Chen Wu*, Bing-Hung Lin*, Loon-Been Chen**, Jui-Yuan Su*, and Po-Chun Hsu*
*Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan
**Department of Computer Science and Information Engineering, Tunghai University, Taichung, Taiwan
icwu@csie.nctu.edu.tw, bhlin@csie.nctu.edu.tw

***Abstract*— Change detection between documents plays an important role in many applications. Zhang and Shasha defined an editing distance problem, called theChange detection between documents plays an important role in many applications. Zhang and Shasha defined an editing distance problem, called the *general editing distance (GED) problem* in this paper, between two ordered labeled trees T_1 and T_2, and devised a new algorithm to solve the problem in time $O(|T_1||T_2|H(T_1)H(T_2))$. $H(T)$ denotes $\min(D(T), L(T))$, where $D(T)$ is the longest depth of tree T and $L(T)$ is the number of leaves of tree T. Zhang also defined an editing distance problem, called the *constrained editing distance (CED) problem* in this paper, and devised a new algorithm to solve it in time $O(|T_1||T_2|)$.**

This paper proposes a new editing distance problem, called the *hybrid editing distance (HED) problem*, a hybrid problem of both GED and CED problems. Some tree nodes, called *C-nodes*, follow the restrictions of the CED problem, while all the other tree nodes, called *G-nodes*, follow the restrictions of the GED problem. In a tree T, a G-subtree T_u is defined to be a maximal connected component in T whose root is a C-node u and whose other nodes are G-nodes. Thus, this paper presents a new algorithm to solve it in time $O(|T_1||T_2|H_1^{max}H_2^{max})$, where H_1^{max} is the maximum $H(T_u)$ for all G-subtrees T_u in T_1, and H_2^{max} is the maximum $H(T_u)$ for all G-subtrees T_u in T_2. In the case of all C-nodes, that is $H_1^{max} = H_2^{max} = 1$, the time complexity is equal to that of Zhang's algorithm. In the case of all G-nodes, that is $H_1^{max} = H(T_1)$ and $H_2^{max} = H(T_2)$, the time complexity is equal to that of Zhang and Shasha's. Finally, from our observation on HTML files, this problem can be applied to the editing distances of the document trees of HTML files. In HTML files, inline elements are close to C-nodes, while block-level elements are close to G-nodes.

Keywords: Algorithm, Change detection, Tree editing distance, General editing distance problem, Constrained editing distance problem, Hybrid editing distance problem

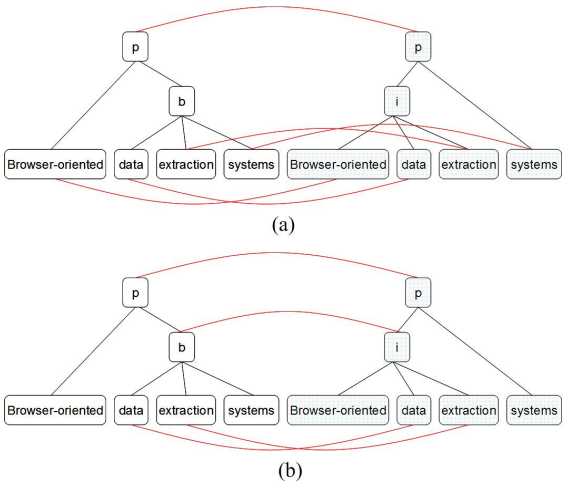

Fig. 1. (a) general edit mapping (b) constrained edit mapping

I. INTRODUCTION

The problems of tree editing distance are to measure the differences between trees. Applications include web page comparisons, pattern recognition, biology computing, image analysis, database, compiler optimization, and natural language processing, etc. [2], [6], [11]

Zhang and Shasha [10] defined an editing distance problem, called the *general editing distance (GED) problem* in this paper, between two ordered labeled trees. The GED problem is to derive the minimum editing changes in both trees. On the other hand, it is to derive the maximum number of matched nodes (or called mapped nodes) which still maintain *ancestor-descendant* and *sibling-ordering relationship*s, defined in Section II in more detail. For example, one edits an sentence from "Browser-oriented **data extraction systems**" to "*Browser-oriented data extraction* systems" by simply modifying font styles in different parts of the sentence. In HTML, the original is "`<p>Browser-oriented <b>data extraction systems<b></p>`", while the targeted is "`<p><i>Browser-oriented data extraction</i> systems</p>`". Figure 1 (a) shows the maximum number of mappings between

both sentences in HTML trees for the GED problem. Thus, clearly, the minimum editing distances are to remove <b> from the original and to add <i> only. From our observation with HTML files, the problem is well suited to inline elements [5] such as <b> and <i>. Zhang and Shasha [10] also devised an algorithm to solve the GED problem with two trees T_1 and T_2 in time $O(|T_1||T_2|H(T_1)H(T_2))$. $H(T)$ denotes $\min(D(T), L(T))$, where $D(T)$ is the longest depth of tree T and $L(T)$ is the number of leaves of tree T.

Zhang [8] also defined a different editing distance problem, called the *constrained editing distance (CED) problem*. In the CED problem, all *matched nodes* (or called *mapped nodes*) need additionally maintain the *least-common-ancestor (LCA) relationship*, defined in Section II in more detail. For the above example, Figure 1 (b) shows the maximum number of mappings between both sentences for the CED problem. From our observation with HTML files, the problem is well suited to block-level elements [5] such as <p> and <table>. Zhang [8] also devised an algorithm to solve the GED problem with two trees T_1 and T_2 in time $O(|T_1||T_2|)$.

In this paper, we present a new editing distance problem for document trees, where some nodes, called *C-nodes*, follow the restrictions of CED problems and all the others, called *G-nodes*, follow that of GED problems. This problem is called the *hybrid editing distance (HED) problem*, a hybrid problem of both GED and CED problems. For example, the elements of HTML documents are categorized into block-level and inline elements, according to HTML 4.0 specification [5] developed by W3C; and users may want to derive edit distances among HTML files by letting block-level elements be C-nodes and inline elements be G-nodes. In a tree T, a *G-subtree* T_u is defined to be a maximal connected component in T whose root[1] is a C-node u and whose other nodes are G-nodes. For simplicity of discussion, we assume that all document trees are rooted at C-nodes. Note that a pseudo C-node can be added into a document tree as the root without loss of generality. Then, all trees can be broken into a set of disjoint G-subtrees. In this paper, we present a new algorithm to solve the HED problem in time $O(|T_1||T_2|H_1^{max}H_2^{max})$, where H_1^{max} is the maximum $H(T_u)$ for all G-subtrees T_u in T_1, and H_2^{max} is the maximum $H(T_u)$ for all G-subtrees T_u in T_2. In the case of all C-nodes, that is $H_1^{max} = H_2^{max} = 1$, the time complexity is equal to that of Zhang's algorithm. In the case of all G-nodes, that is $H_1^{max} = H(T_1)$ and $H_2^{max} = H(T_2)$, the time complexity is equal to that of Zhang and Shasha's.

The remainder of this paper is organized as follows. Section II reviews the GED and CDE problems. Section III discusses the HED problem. Section IV concludes this paper.

[1]In this paper, the root of a connected component C in a tree T is defined to be the node in C that is the closest to the root of T.

II. General and Constrained Edit Distance Problems

In this section, we review the GED and CED problems. In Subsection II-A, we describe the general edit mapping, the GED problem and Zhang and Shasha's algorithm solving the GED problem. In Subsection II-B, we describe the constrained edit mapping, the CED problem and Zhang's algorithm solving the CED problem.

A. General Editing Distance

A tree T is a *labeled tree* if each node of T is assigned a symbol from a finite alphabet Σ. A *null symbol* is denoted by λ. T is an *ordered tree* if there is a left-to-right order among all siblings in T. Consider two ordered labeled trees T_1 and T_2. Let u be a node with a symbol $text_1(u) \in \Sigma \cup \lambda$ in T_1 where $text_1$ is the label function of T_1, and v be a node with a symbol $text_2(v) \in \Sigma \cup \lambda$ in T_2 where $text_2$ is the label function of and T_2. Following [10] and [8], there are three edit operations defined for ordered labeled trees: insert (denoted by$\lambda \rightarrow v$), delete (denoted by $u \rightarrow \lambda$), and update (denoted by $u \rightarrow v$).

An edit distance mapping M_G is called a *general edit distance mapping* from T_1 to T_2, if the following three conditions hold for any two links $u_1 \rightarrow v_1$ and $u_2 \rightarrow v_2$ in M_G.

- C1 One-to-one condition: $u_1 = u_2$ iff $v_1 = v_2$.
- C2 Ancestor condition: u_1 if an ancestor of u_2 iff v_1 is an ancestor of v_2.
- C3 Sibling condition: u_1 is to the left of u_2 iff v_1 is to the left of v_2.

Let the cost of M_G be denoted by $\gamma(M_G)$. The *general edit distance* is defined to be the minimum cost among all the general editing distance mappings from T_1 to T_2, and denoted by $\delta_G(T_1, T_2)$. The *GED (general edit distance) problem* is to find the general edit distance.

In [10], Zhang and Shasha proposed an algorithm solving this problem in time $O(|T_1||T_2|H(T_1)H(T_2))$ and space $O(|T_1||T_2|)$. Let u be any node in the tree using the left-to-right postorder numbering. $p(u)$ is the parent of u. $anc(u)$ are all ancestor nodes of u including $p(u)$. $lleaf(u)$ is the leftmost leaf node in the subtree rooted at u. Generally, $1..u$ is a forest in the tree and is denoted by $T[1..u]$, while $T[lleaf(u)..u]$ is denoted by $T[u]$, where $lleaf(u)$ is the leftmost node in the subtree rooted at u. An example is shown in Figure 2.

Zhang and Shasha's algorithm [10] used the technique of dynamic programing to solve the problem without the need of computing all subtree-to-subtree distances. The distance between $T_1[u'..u]$ and $T_2[v'..v]$ is denoted by $fdist(u'..u, v'..v)$. Besides, $fdist(1..u, 1..v)$ is also denoted by $fdist(u, v)$. The distance between subtrees $T_1[u]$ and $T_2[v]$ is then denoted by $tdist(u, v)$. Figure 3 shows all possible mappings between $T_1[u]$ and $T_2[v]$. With the recurrence, we can use dynamic programming to solve the GED problem in time $O(|T_1||T_2|H(T_1)H(T_2))$, as proved in [10]. For simplicity of discussion, in the rest of this paper, we omit the recurrences for all edit distances

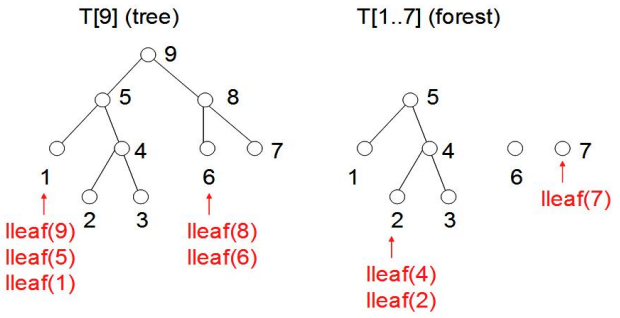

Fig. 2. Tree and Forest

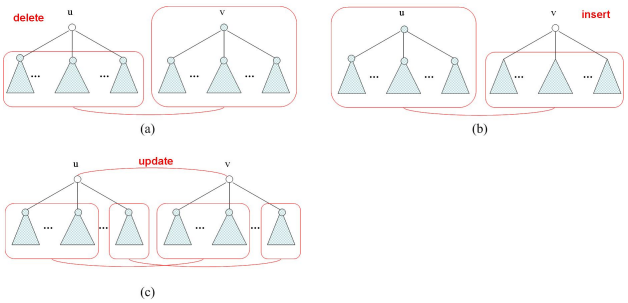

Fig. 3. General edit distance mapping: (a) delete (b) insert (c) update

when empty trees are involved. For example, $fdist(u..u-1, v..v-1) = 0$, $fdist(u_1..u_2, v..v-1) = \Sigma_{s=u_1}^{u_2}\gamma(s \rightarrow \lambda)$ and $fdist(u..u-1, v_1..v_2) = \Sigma_{s=v_1}^{v_2}\gamma(\lambda \rightarrow s)$.

B. Constrained Editing Distance

A general edit distance mapping M_C is also called a *constrained edit distance mapping* from T_1 to T_2, if the following condition holds in M_C.[2]

C4 Least common ancestor (LCA) condition: For any three links $u_1 \rightarrow v_1$, $u_2 \rightarrow v_2$ and $u_3 \rightarrow v_3$ in M_C, $lca(u_1, u_2)$ is a proper ancestor of u_3 iff $lca(v_1, v_2)$ is a proper ancestor of v_3, where $lca(u, v)$ represents the least common ancestor of nodes u and v.

The LCA condition can also be written as: $lca(u_1, u_2) = lca(u_1, u_3)$ iff $lca(v_1, v_2) = lca(v_1, v_3)$. Figure 4-(a) illustrates a mapping satisfying the lca condition, while Figure 4-(b) illustrates a mapping violating the condition.

[2]When we say that a is an ancestor of b, a can be b itself. When we say that a is an proper ancestor of b, a cannot be b itself.

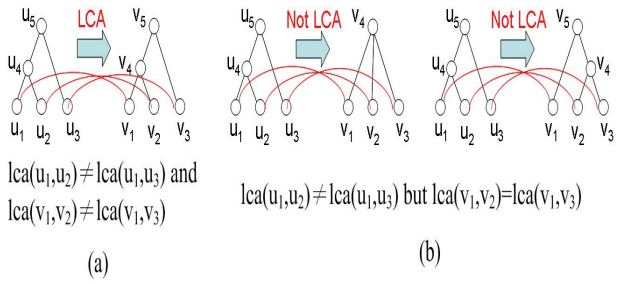

Fig. 4. (a) Constrained edit mapping (b) Not constrained edit mapping

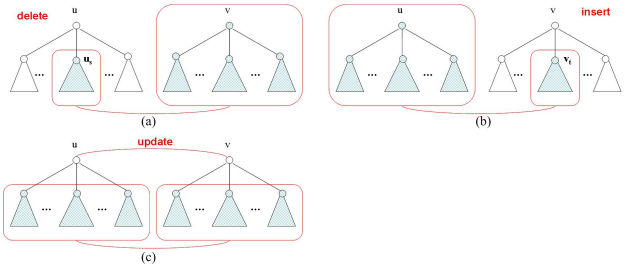

Fig. 5. Constrained edit distance mapping: (a) delete (b) insert (c) update

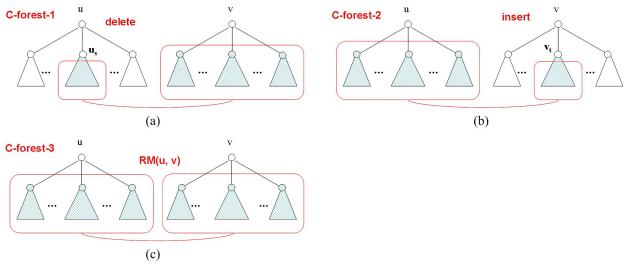

Fig. 6. Constrained edit distance mapping: (a) C-forest-1 (b) C-forest-2 (c) C-forest-3

The *constrained edit distance* is defined to be the minimum cost among all the constrained editing distance mappings from T_1 to T_2, and denoted by $\delta_C(T_1, T_2)$. The *CED (constrained edit distance) problem* is to find the constrained edit distance.

In [8], Zhang proposes an algorithm solving this problem in time $O(|T_1||T_2|)$ and space $O(|T_1||T_2|)$. Let u be any node in T_1 and v be any node in T_2. Also, let the children of u be u_1, u_2,..., u_{n_u} and the children of v be v_1, v_2,..., v_{n_v}. For simplicity, the forest of w, $T_1[lleaf(u)..u-1]$, is denoted by $F_1[u]$. The subtree and forest of v are denoted likewise.

The constrained edit distance between subtrees $T_1[u]$ and $T_2[v]$ is denoted by $D(T_1[u], T_2[v])$, while the constrained edit distance between forests $F_1[u]$ and $F_2[v]$ is denoted by $D(F_1[u], F_2[v])$. Figure 5 and 6 illustrates mappings between $T_1[u]$ and $T_2[v]$. With these mappings, Zhang [8] solve the CED problem in time $O(|T_1||T_2|)$.

III. Hybrid Editing Distance Problem

In this section, we first define a new editing distance problem, called the *hybrid editing distance (HED) problem*, a hybrid problem of both GED and CED problems, in Subsection III-A. In Subsection III-B, we propose a new algorithm for solving the HED problem, whose time complexity is derived in Subsection III-C.

A. Problem Definition

In the HED problem, all the tree nodes are composed of *G-nodes* or *C-nodes*. Conceptually, G-nodes satisfy Conditions C1, C2 and C3, the same as those satisfied in the nodes in the GED problem, while C-nodes satisfy Conditions C1, C2, C3 and C4, the same as those satisfied

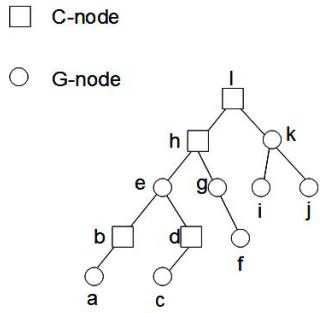

Fig. 7. A hybrid tree

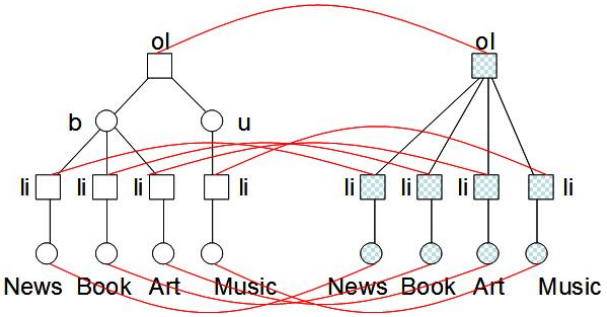

Fig. 8. Hybrid edit distance mapping

in the nodes in the CED problem. Practically, Condition C4 in the nodes in the HED problems are defined in more detail in this subsection.

The *constrained least common ancestor* of two nodes u_1 and u_2 in a tree T, denoted by $CLCA(u_1, u_2)$, is the least common ancestor C-node of u_1 and u_2. The *C-node frontier* of u, denoted by $CF(u)$, is the set of C-node descendants v where all nodes on the path from u (exclude) to v (exclude) are G-nodes. For example, in Figure 7, the following hold, $CLCA(a,c)$=h, $CLCA(a,d) = h$, $CLCA(b,d) = h$, $CLCA(e,f) = h$, $CLCA(e,g) = h$, $CLCA(i,j) = l$, $CLCA(e,i) = l$, $CF(l) = h$, $CF(h) = b, d$, $CF(b) = \{\}$, and $CF(d) = \{\}$. Without loss of generality, a tree is assumed to be rooted at a C-node, by adding a pseudo C-node into a document tree as a root.

A general edit distance mapping M_H is called a *hybrid edit distance mapping* from T_1 to T_2, if the following conditions hold in M_H.

C5 For any three links $u_1 \rightarrow v_1$, $u_2 \rightarrow v_2$ and $u_3 \rightarrow v_3$ in M_H, $CLCA(u_1, u_2)$ is a proper ancestor of u_3, if and only if $CLCA(v_1, v_2)$ is a proper ancestor of v_3.

The *hybrid edit distance* is defined to be the minimum cost among all the hybrid editing distance mappings from T_1 to T_2, and denoted by $\delta_C(T_1, T_2)$. The *HED (Hybrid Edit Distance) problem* is to find the hybrid edit distance.

For example, in Figure 8, the hybrid edit mapping can match items and delete node "b" and "u", by letting inline elements be G-nodes and block-level elements be C-nodes in HTML files. Such an edit distance seems more reasonable for HTML documents, since users only change the outlook of contents.

B. *Algorithm for the HED Problems*

In this subsection, a new algorithm is proposed to solve the HED problem. A subgraph of tree T is called a *G-forest* of C-node u, denoted by $F_H[u]$, if all nodes v in the subgraph are G-nodes and all the nodes on the path from u (exclusive) to v (exclusive) are G-nodes. The *G-subtree* of u, denoted $T_H[u]$, is defined to be $F_H[u] \cup \{u\}$. Let $lleaf_H(u)$ denote the leftmost node $T_H[u] \cup CF(u)$. As in Figure 7, $T_H[l] = i, j, k, l$, $T_H[h] = e, f, g, h$, $T_H[b] = a, b$, $lleaf_H(l) = h$, $lleaf_H(e) = b$, and $lleaf_H(g) = f$. From the above definition, we can easily obtain that a document tree consists of disjoint G-subtrees.

Let u be a C-node in T_1 and v be a C-node in T_2. The hybrid edit distance between subtrees $T_1[u]$ and $T_2[v]$ is denoted by $D_H(T_1[u], T_2[v])$ and the hybrid edit distance between forests $F_1[u]$ and $F_2[v]$ is denoted by $D_H(F_1[u], F_2[v])$. Let G-subtree $T_{1H}[u] = i_1, i_2, ..., i_k, u$ and G-subtree $T_{2H}[v] = j_1, j_2, ..., j_{k'}, v$. The hybrid edit distance between G-subtrees $T_{1H}[u]$ and $T_{2H}[v]$ is denoted $tdist_H(u, v)$. The hybrid edit distance between G-forests $F_{1H}[u]$ and $F_{2H}[v]$ is denoted by $fdist_H(i_1..u, j_1..v)$. Let $CF(u) = d_1, d_2, ..., d_{n'_u}$ and $CF(v) = d'_1, d'_2, ..., d'_{n'_v}$. Then, Lemma 1 shows that the following recurrences hold.

$$D_H(T_1[u], T_2[v]) =$$

$$\min \begin{cases} D_H(T_1[u], \lambda) + \min_{s=1}^{n'_u} \{D_H(T_1[d_s], T_2[v]) - D_H(T_1[d_s], \lambda)\} \\ \quad \text{(illustrated in Fig. 9-(a))} \\ D_H(\lambda, T_2[v]) + \min_{t=1}^{n'_v} \{D_H(T_1[u], T_2[d'_t]) - D_H(\lambda, T_2[d'_t])\} \\ \quad \text{(illustrated in Fig. 9-(b))} \\ D_H(F_1[u], F_2[v]) + \gamma(u \rightarrow v) \\ \quad \text{(illustrated in Fig. 9-(c))} \end{cases}$$

$$D_H(F_1[u], F_2[v]) =$$

$$\min \begin{cases} D_H(F_1[u], \lambda) + \min_{s=1}^{n'_u} \{D_H(F_1[d_s], F_2[v]) - D_H(F_1[d_s], \lambda)\} \\ \quad \text{(illustrated in Fig. 10-(a))} \\ D_H(\lambda, F_2[v]) + \min_{t=1}^{n'_v} \{D_H(F_1[u], F_2[d'_t]) - D_H(\lambda, F_2[d'_t])\} \\ \quad \text{(illustrated in Fig. 10-(b))} \\ fdist_H(i_1..u, j_1..v) \\ \quad \text{(illustrated in Fig. 11)} \end{cases}$$

Lemma 1: The above two recurrences hold.

Proof: The proof is skipped due to page limitation. ■

Lemma 2: Let $T_i[u] = T_{1H}[u] \cup CF(u) = i_1, i_2, ..., i_k, u$ and the $T_j[v] = T_{2H}[v] \cup CF(v) = j_1, j_2, ..., j_{k'}, v$. By letting $i_1 \leq u' \leq u$ and $j_1 \leq v' \leq v$, the following recursions hold.

$$fdist_H(i_1..u', j_1..v') =$$

$$\min \begin{cases} fdist_H(i_1..u'-1, j_1..v') + \gamma(u' \rightarrow \lambda) \\ \quad (u' \in \text{G-node, illustrated in Fig. 11-(a)}) \\ fdist_H(i_1..u'-1, j_1..v') + D_H(T_1[u'], \lambda) \\ \quad (u' \in \text{C-node, illustrated in Fig. 11-(b)}) \\ fdist_H(i_1..u', j_1..v'-1) + \gamma(\lambda \rightarrow v') \\ \quad (v' \in \text{G-node, illustrated in Fig. 11-(c)}) \\ fdist_H(i_1..u', j_1..v'-1) + D_H(\lambda, T_2[v']) \\ \quad (v' \in \text{C-node, illustrated in Fig. 11-(d)}) \\ G - forest - 3 - 3 - cost \\ \quad (u' \in \text{G-node and } v' \in \text{G-node}) \\ fdist_H(i_1..u'-1, j_1..v'-1) + D_H(T_1[u'], T_2[v']) \\ \quad (u' \in \text{C-node and } v' \in \text{C-node, illustrated in Fig. 11-(f)}) \end{cases}$$

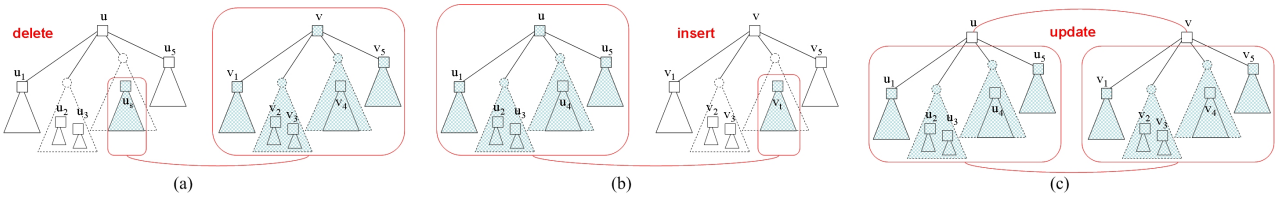

Fig. 9. Hybrid edit distance mapping: (a) delete (b) insert (c) update

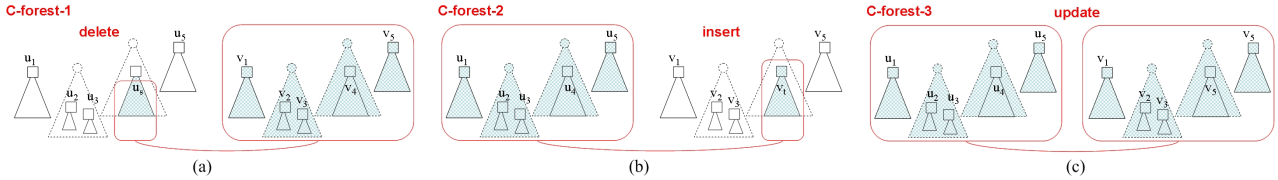

Fig. 10. Hybrid edit distance mapping: (a) C-forest-1 (b) C-forest-2 (c) C-forest-3

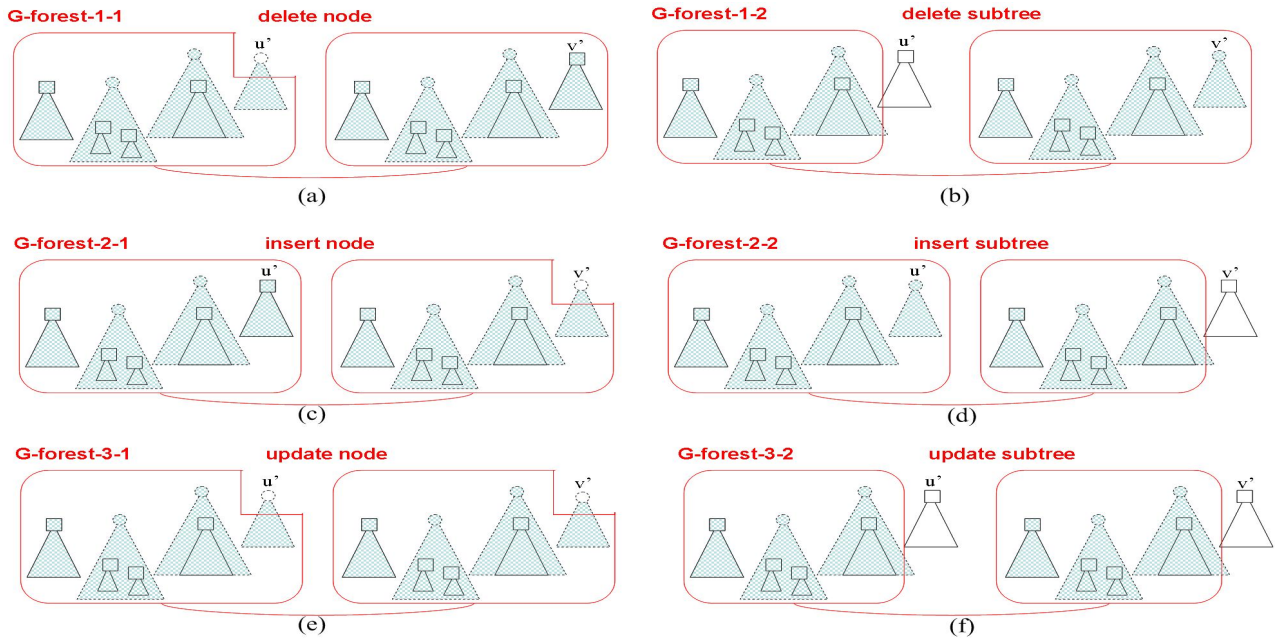

Fig. 11. Hybrid edit distance mapping: (a) G-forest-1-1 (b) G-forest-1-2 (c) G-forest-2-1 (d) G-forest-2-2 (e) G-forest-3-1 (f) G-forest-3-2

There are two conditions for deriving the $G-forest-3-3-cost$:

1) If $lleaf_H(u') = lleaf_H(u)$ and $lleaf_H(v') = lleaf_H(v)$, the $G-forest-3-3-cost$ is

$$fdist_H(i_1..u'-1, j_1..v'-1) + \gamma(u' \to v'),$$

as illustrated in Fig. 11-(e).

2) If $lleaf_H(u') \neq lleaf_H(u)$ or $lleaf_H(v') \neq lleaf_H(v)$, the $G-forest-3-3-cost$ is

$$fdist_H(i_1..u'-1, j_1..v'-1) + tdist_H(u', v'),$$

as illustrated in Fig. 11-(f), but u' and v' are both G-nodes.

Proof: The proof is skipped due to page limitation. ■

Figure 9, 10 and 11 show mappings of above Lemmas. From these recurrences, we can solve the hybrid edit distance problem by a dynamic programming algorithm 1. Like [10], we use $keyroots$ to reduce the time complexity of Lemma 2.

C. Time Complexity

In this subsection, we show the space and time complexity of our HED algorithm. Also, we discuss conditions that our algorithm runs as same time complexity as the Zhang and Shasha's algorithm [10] or the Zhang's algorithm [8].

First, consider the space complexity of our algorithm. In our algorithm, four matrices are used to store results of $D_H(F_{1H}[u], F_{2H}[v])$, $D_H(T_{1H}[u], T_{2H}[v])$ and $fdist_H(i_1..u, j_1..v)$ during the computation (see the HED algorithm 1). The size of each matrix is at most $|T_1||T_2|$. Hence, the space complexity is $O(|T_1||T_2|)$.

Second, investigate the time complexity of our algorithm. Let $C(T)$ be the set of C-nodes and $G(T)$ be the set of G-nodes in T. Let $H(T) = \min(D(T), L(T))$, $D(T)$ be the longest depth of T, and $L(T)$ be the number of leaves of T. In our HED algorithm, the routine $gConnectedComponent$ dominates the time complexity. Because of the property of $Keyroot$ [10], the time complexity of procedure $gConnectedComponent$

Algorithm 1 The HED algorithm

Input: Two subtrees T_1 and T_2
Output: $D_H(T_1[u], T_2[v])$, where $1 \le u \le |T_1|$ and $1 \le v \le |T_2|$
Main loop
for u=1 to $|T_1|$
  initialize D($F_1[u]$, λ);
  initialize D($T_1[u]$, λ);
for v=1 to |T2|
  initialize D(λ, $F_2[u]$);
  initialize D(λ, $T_2[u]$);
for u=1 to |T1|
  for v=1 to |T2|
    if $u \in$C-node and $v \in$C-node
      The G-subtrees are $T_{1H}[u] = \{i_1, i_2, ..., i_k, u\}$ and $T_{2H}[v] = \{j_1, j_2, ..., j_{k'}, v\}$
      Compute $fdist_H(i_1..u, j_1..v)$ ($gConnectedComponent(u, v)$)
      Compute $D_H(F_1[u], F_2[v])$; (Lemma 1)
      Compute $D_H(T_1[u], T_2[v])$; (Lemma 1)
End

Procedure $gConnectedComponent(u, v)$
Input: Two hybrid subtrees $T_{1H}[u]$ and $T_{2H}[v]$
Output: $fdist_H(i_1..u, j_1..v)$
Compute $lleaf_H$ and $Keyroots$ of T_{1H} and T_{2H} separately
for u"=1 to $|Keyroots(T_{1H})|$
  for v"=1 to $|Keyroots(T_{2H})|$
    $s = Keyroots[u"]$;
    $t = Keyroots[v"]$;
    Compute $tdist_H(s, t)$;

Procedure $tdist_H(u, v)$
for u'=i_1 to u
  initialize $fdist_H(i_1..u', \lambda)$;
for v'=j_1 to v
  initialize $fdist_H(\lambda, j_1..v')$;
for u'=i_1 to u
  for v'=j_1 to v
    Compute $fdist_H(i_1..u', j_1..v')$; (Lemma 2)

is $O(|T_{1H}[u]| \times |T_{2H}[v]| \times H(T_{1H}[u]) \times H(T_{2H}[v]))$ where $H(T_{1H}[u]) = \min(D(T_{1H}[u]), L(T_{1H}[u]))$ and $H(T_{2H}[v]) = \min(D(T_{2H}[v]), L(T_{2H}[v]))$. The total time complexity is:

$$\Sigma_{u \in C(T_1)} \Sigma_{v \in C(T_2)} O(|T_{1H}[u]| \times |T_{2H}[v]| \times H(T_{1H}[u]) \times H(T_{2H}[v]))$$
$$\le O(\Sigma_{u \in C(T_1)}(|T_{1H}[u]| \times H(T_{1H}[u])) \times \Sigma_{v \in C(T_2)}(|T_{2H}[v]| \times H(T_{2H}[v])))$$
$$= O(time_1).$$

Let H_1^{max} be the maximum $H(T_{1H}[u])$ among all G-subtrees $T_{1H}[u]$ in T_1 and H_2^{max} be the maximum $H(T_{2H}[v])$ among all G-subtrees $T_{2H}[v]$ in T_2. The total time complexity is:

$$O(time_1)$$
$$\le O(H_1^{max} \times H_2^{max} \times \Sigma_{u \in C(T_1)}(|T_{1H}[u]|) \times \Sigma_{v \in C(T_2)}(|T_{2H}[v]|))$$
$$= O(time_2).$$

Since $\Sigma_{u=1}^{C_1} |T_{1H}[u]|$ is the sum of sizes of all G-subtrees $T_{1H}[u]$ in T_1, the summation is at most $O(|T_1|)$. $\Sigma_{v=1}^{C_2} |T_{2H}[v]|$ is the sum of sizes of all G-subtrees $T_{2H}[v]$ in T_2, and the summation is at most $O(|T_2|)$. The total time complexity is:

$$O(time_2) \le O(H_1^{max} \times H_2^{max} \times |T_1| \times |T_2|).$$

Third, compare the time complexity of ours with that of Zhang and Shasha's algorithm in [10] and [8]. If all nodes are G-nodes, then $|T_1|$=$|G_1|$ and $|T_2|$=$|G_2|$. Since our HED algorithm computes $gConnectedComponent$ only once, the total time complexity is:

$$O(|T_{1H}[u]| \times |T_{2H}[v]| \times H(T_{1H}[u]) \times H(T_{2H}[v]))$$
$$= O(|T_1| \times |T_2| \times \min(D(T_1), L(T_1)) \times \min(D(T_2), L(T_2)).$$

the same as that of Zhang and Shasha's algorithm. However, when these trees contain some more C-nodes, e.g., block-level elements in HTML files, the G-subtrees get smaller and the values H_1^{max} and H_2^{max} also get smaller. When the sizes of G-subtrees become all ones, i.e., all are C-nodes, the time complexity is reduced to $O(|T_1||T_2|)$, the same as that of Zhang's algorithm.

IV. CONCLUSION

The first contribution of this paper is to propose a new editing distance problem, called the *hybrid editing distance (HED) problem*, a hybrid of GED and CED problems. For the document trees in this problem, some nodes (called *C-nodes*) follow the restrictions of CED problems and others (called *G-nodes*) follow those of GED nodes. Based on our observation on HTML files, this problem can be applied to the editing distances of the document trees of HTML files. In HTML files, inline elements are close to C-nodes, while block-level elements are close to G-nodes.

The second contribution of this paper is to present a new algorithm to solve the HED problem in time $O(|T_1||T_2|H_1^{max}H_2^{max})$, where H_1^{max} is the maximum $H(T_u)$ for all G-subtrees T_u in tree T_1, and H_2^{max} is the maximum $H(T_u)$ for all G-subtrees T_u in tree T_2. (G-subtrees are defined in Subsection III-B.) In general, our algorithm runs as fast as Zhang and Shasha's algorithm, when $H_1^{max} = T_1$ and $H_2^{max} = T_2$; and faster than Zhang and Shasha's algorithm when $H_1^{max} < T$ and $H_2^{max} < T_2$. When $H_1^{max} = 1$ and $H_2^{max} = 1$, our algorithm runs as fast as Zhang's algorithm.

REFERENCES

[1] P. Bille, A survey on tree edit distance and related problems, Theoretical Computer Science 337(1-3), 217-239 (2005).
[2] M. Chodorow and J. L. Klavans, Locating syntactic patterns in text corpora, Mnuscript, Lexical systems, IBM Research, T. J. Watson Research Center, Yorktown Heights, New York (1990).
[3] S. Dulucq and H. Touzet, Decomposition algorithms for the tree edit distance problem, Journal of Discrete Algorithms 3, 448-471 (2005).
[4] C. L. Lu, Z. Y. Su, and C. Y. Tang, A new measure of edit distance between labeled trees, Proceedings of the 7th Annual International Conference on Computing and Combinatorics (2001).
[5] L. Quinn, HTML 4.0 Reference: http://www.htmlhelp.com/reference/html40/, WDG (1998).
[6] B. A. Shapiro and K. Zhang, Comparing multiple RNA secondary structures using tree comparisons, Computer Appl. Biosci. 6(4), 309-318 (1990).
[7] R. A. Wagner and M. J. Fischer, The string-to-string correction problem. Journal of the ACM 21(1), 168-173 (1974).
[8] K. Zhang, Algorithms for the constrained editing distance between ordered labeled trees and related problems, Pattern Recognition 28(3), 463-474 (1995).
[9] K. Zhang, A constrained edit distance between unordered labeled trees, Algorithmica 15(3), 205-222 (1996).
[10] K. Zhang and D. Shasha, Simple fast algorithms for the editing distance between trees and related problems, SIAM Journal of Computing 18(6), 1245-1262 (1989).
[11] H. Zhang, D. Shasha and J. T. L. Wang, Approximate tree matching in the presence of variable length don't cares, Journal of the ACM 16, 33-66 (1994).