Faster algorithms for guided tree edit distance

Dekel Tsur*

Abstract

The guided tree edit distance problem is to find a minimum cost series of edit operations that transforms two input forests F and G into isomorphic forests F' and G' such that a third input forest H is included in F' (and G'). The edit operations are relabeling a vertex and deleting a vertex. We show efficient algorithms for this problem that are faster than the previous algorithm for this problem of Peng and Ting [5].
Keywords: Design of algorithms; String algorithms; Edit distance; Ordered trees

1 Introduction

Tree edit distance is a popular metric for the similarity of two forests and arises in XML comparisons, computer vision, compiler optimization, natural language processing, and computational biology. The tree edit distance problem is to transform two rooted, ordered, and labelled forests F and G into isomorphic forests F' and G' by using a minimum cost series of edit operations on F and G. The edit operation are relabeling a vertex and deleting a vertex. Deleting a vertex v means removing v and all edges incident to v. The children of v become children of the parent of v (if it exists) instead of v.

Tai [6] gave the first algorithm for tree edit distance. The time complexity of Tai's algorithm is $O(|F|\,\text{leaves}(F)^2 \cdot |G|\,\text{leaves}(G)^2)$, where $|F|$ denotes the number of vertices in F and $\text{leaves}(F)$ denotes the number of leaves in F. Shasha and Zhang [7] improved this result to $O(|F|\,\text{cdepth}(F) \cdot |G|\,\text{cdepth}(G))$, where $\text{cdepth}(F)$ denotes the minimum between $\text{leaves}(F)$ and the maximum height of a tree in F. In the worst case, their algorithm runs in $O(|F|^2\,|G|^2)$ time. Klein [4] gave an $O(|F| \log |F| \cdot |G|^2)$ time algorithm where $|F| \geq |G|$, and Demaine et al. [3] gave an $O(|F|\,|G|^2\,(1 + \log \frac{|F|}{|G|}))$ time algorithm.

Peng and Ting [5] introduced a generalization of tree edit distance called *guided tree edit distance*. In this problem the input is three forests F, G, and H. The goal is to find a minimum cost series of edit operations that transforms F and G into isomorphic forests F' and G' such that H can be obtained from F' (or G') by a series of vertex deletion operations. Peng and Ting gave an algorithm for this problem with time complexity $O(|F|\,\text{cdepth}(F) \cdot |G|\,\text{cdepth}(G) \cdot |H|\,\text{leaves}(H)^2)$ and space complexity $O(|F|\,|G|\,|H|\,\text{leaves}(H)^2)$.

In this paper we show that the guided edit distance problem can be solved in $O(|F|\,\text{cdepth}(F) \cdot |G|\,\text{cdepth}(G) \cdot |H|\,d(H)^2(\log\log d(H))^3/\log^2 d(H))$ time or in $O(|F|\,|G|^2\,(1+\log \frac{|F|}{|G|}) \cdot |H|\,d(H)^2\,(\log\log d(H))^3/\log^2 d(H))$ time, where $d(H)$ is the maximum between the number of trees in H, and the maximum number of children of a vertex in H. The space complexity of our algorithms is $O(|F|\,|G|\,(d(H)^2 + |H|))$. As $d(H) \leq \text{leaves}(H)$, our first algorithm is always faster than the algorithm of Peng and Ting and uses less memory. Moreover, if we measure the time complexity as a function of $n = |F| + |G| + |H|$ (i.e. the size of the input) the algorithm of Peng and Ting has time complexity $O(n^7)$ while our second algorithm has time complexity $O(n^6(\log\log n)^3/\log^2 n)$.

*Department of Computer Science, Ben-Gurion University of the Negev. Email: dekelts@cs.bgu.ac.il

2 Preliminaries

Let F be a forest with trees $F_1, \ldots, F_t$ (from left to right) and let v be a vertex in F with children $v_1, \ldots, v_d$ (from left to right). We define $F[v, i, j]$ to be the subforest of F that is induced by $v_i, v_{i+1}, \ldots, v_j$ and their descendants (note that this subforest does not include the vertex v). Moreover, we define $F[\phi, i, j]$ to be the subforest of F that contains the trees $F_i, F_{i+1}, \ldots, F_j$. If $i > j$ then $F[\phi, i, j]$ is an empty forest. Let L_F denote the leftmost tree in a forest F, and let R_F denote the rightmost tree. Let l_F and r_F be the roots of L_F and R_F, respectively.

Let $t(F)$ denote the number of trees in a forest F. For a vertex v, let $d(v)$ denote the number of children of v. Define $d(F) = \max(\max_{v \in F} d(v), t(F))$ and $\mathrm{cdepth}(F)$ is the minimum between the number of leaves in F and the maximum height of a tree in F.

The edit distance between two forests F and G is denoted $\delta(F, G)$. The guided edit distance between forests F and G with guiding forest H is denoted $\delta(F, G, H)$ (note that $\delta(F, G, \emptyset) = \delta(F, G)$ where $\emptyset$ denotes an empty forest). We denote by $c_{\mathrm{del}}(v)$ the cost of deleting a vertex v, and by $c_{\mathrm{rel}}(v, w)$ the cost of changing the label of vertex v to be equal to the label of w.

We say that a vertex $v \in F$ is *to the left of* a vertex $w \in F$ if v appears before w in the postorder of F and v is not a descendant of w.

The guided tree edit distance problem can be formulated in terms of matchings. Let F and G be two forests. We say that a set $M \subseteq V(F) \times V(G)$ is an *edit matching* if

1. M is a matching, namely each $v \in F$ appears in at most one pair of M and each $v \in G$ appears in at most one pair.
2. For every $(v, v'), (w, w') \in M$, v is an ancestor of w if and only if v' is an ancestor of w'.
3. For every $(v, v'), (w, w') \in M$, v is to the left of w if and only if v' is to the left of w'.

A *matchings pair* for F, G, H is a pair (M, M') where M is an edit matching between F and G, and M' is a matching between F and H such that

1. All vertices of H are matched in M'.
2. If $v \in F$ is matched in M' then v is also matched in M.

The cost of (M, M') is

$$\mathrm{cost}_{F,G,H}((M, M')) = \sum_{v \in U} c_{\mathrm{del}}(v) + \sum_{(v,v') \in M_0} c_{\mathrm{rel}}(v, v') + \sum_{(v,v',v'') \in A} (c_{\mathrm{rel}}(v, v'') + c_{\mathrm{rel}}(v', v'')),$$

where U is the set of vertices in F and G that are unmatched in M, M_0 is the set of all pairs $(v, v') \in M$ such that v is not matched in M', and A is the set of all triplets (v, v', v'') such that $(v, v') \in M$ and $(v, v'') \in M'$. It is easy to verify that $\delta(F, G, H)$ is equal to the minimum cost of a matchings pair for F, G, H.

Several algorithms for tree edit distance are based on the following recurrence (see [3]): For two forests $F \neq \emptyset$ and $G \neq \emptyset$,

$$\delta(F, G) = \min \left\{ \begin{array}{l} \delta(F - r_F, G) + c_{\mathrm{del}}(r_F), \\ \delta(F, G - r_G) + c_{\mathrm{del}}(r_G), \\ \delta(F - R_F, G - R_G) + \delta(R_F - r_F, R_G - r_G) + c_{\mathrm{rel}}(r_F, r_G) \end{array} \right\} \qquad (1)$$

and

$$\delta(F,G) = \min \left\{ \begin{array}{l} \delta(F - l_F, G) + c_{\mathrm{del}}(l_F), \\ \delta(F, G - l_G) + c_{\mathrm{del}}(l_G), \\ \delta(F - L_F, G - L_G) + \delta(L_F - l_F, L_G - l_G) + c_{\mathrm{rel}}(l_F, l_G) \end{array} \right\}. \tag{2}$$

Moreover, $\delta(F, \emptyset) = \sum_{v \in F} c_{\mathrm{del}}(v)$ and $\delta(\emptyset, G) = \sum_{v \in G} c_{\mathrm{del}}(v)$.

A *strategy algorithm* for tree edit distance is an algorithm that computes $\delta(F, G)$ as follows. Given F and G, the algorithm computes $\delta(F_1, G_1), \ldots, \delta(F_k, G_k)$ for some pairs of subforest of F and G, with $F_k = F$ and $G_k = G$. Each value $\delta(F_i, G_i)$ is computed by applying either rule (1) or rule (2) on F_i and G_i. Each of the $\delta(F', G')$ terms that appear in the right hand side of the selected rule is a value that was computed previously by the algorithm, namely it appears in $\delta(F_1, G_1), \ldots, \delta(F_{i-1}, G_{i-1})$. Note that the algorithm can discard the value $\delta(F_i, G_i)$ at some step $j > i$ if the computation of the values $\delta(F_{j'}, G_{j'})$ for $j' \geq j$ does not depend on $\delta(F_i, G_i)$. Each pair (F_i, G_i) is called a *relevant pair w.r.t.* F, G, S, where S denotes the strategy algorithm. We also say that F_i is a *relevant subforest* of F and G_i is a relevant subforest of G (w.r.t. F, G, S).

The algorithm of Shasha and Zhang [7] is a strategy algorithm that always uses rule (1). The time complexity of the algorithm of Shasha and Zhang is $O(|F| \operatorname{cdepth}(F) \cdot |G| \operatorname{cdepth}(G))$. Demaine et al. [3] gave a strategy algorithm with time complexity $O(|F| \, |G|^2 \, (1 + \log \frac{|F|}{|G|}))$ and showed that this algorithm is asymptotically optimal among all strategy algorithms when the time complexity is measured as a function of $|F|$ and $|G|$.

We use the following Lemma from [3].

Lemma 1. *Let S be a strategy algorithm for tree edit distance and let F and G be two forests. For every pair of vertices $v \in F$ and $w \in G$ we have that $(F[v, 1, d(v)], G[w, 1, d(w)])$ is a relevant pair w.r.t. F, G, S.*

3 The algorithm

Lemma 2. *Let F,G and H be non-empty forests. Then*

$$\delta(F,G,H) = \min \left\{ \begin{array}{l} \delta(F - r_F, G, H) + c_{\mathrm{del}}(r_F), \\ \delta(F, G - r_G, H) + c_{\mathrm{del}}(r_G), \\ \min\limits_i \left\{ \begin{array}{l} \delta(F - R_F, G - R_G, H[\phi, 1, i]) \\ \quad + \delta(R_F - r_F, R_G - r_G, H[\phi, i+1, t(H)]) \end{array} \right\} \\ \qquad + c_{\mathrm{rel}}(r_F, r_G), \\ \delta(F - R_F, G - R_G, H[\phi, 1, t(H) - 1]) \\ \qquad + \delta(R_F - r_F, R_G - r_G, H[r_H, 1, d(r_H)]) \\ \qquad + c_{\mathrm{rel}}(r_F, r_H) + c_{\mathrm{rel}}(r_G, r_H) \end{array} \right\}, \tag{3}$$

where the inner minimum is taken over $i = 0, \ldots, t(H)$. Furthermore,

$$\delta(F,G,H) = \min \left\{ \begin{array}{l} \delta(F - l_F, G, H) + c_{\mathrm{del}}(l_F), \\ \delta(F, G - l_G, H) + c_{\mathrm{del}}(l_G), \\ \min\limits_i \left\{ \begin{array}{l} \delta(F - L_F, G - L_G, H[\phi, i+1, t(H)]) \\ \quad + \delta(L_F - l_F, L_G - l_G, H[\phi, 1, i]) \end{array} \right\} \\ \qquad + c_{\mathrm{rel}}(l_F, l_G), \\ \delta(F - L_F, G - L_G, H[\phi, 2, t(H)]) \\ \qquad + \delta(L_F - l_F, L_G - l_G, H[l_H, 1, d(l_H)]) \\ \qquad + c_{\mathrm{rel}}(l_F, l_H) + c_{\mathrm{rel}}(l_G, l_H) \end{array} \right\}. \tag{4}$$

Proof. We only prove equality (3). Let $\alpha(F,G,H)$ denote the right hand side of (3). Let (M,M') be a minimum cost matchings pair for F,G,H. If r_F is not matched in M then (M,M') is a matchings pair for $F-r_F,G,H$ and $\text{cost}_{F-r_F,G,H}((M,M')) = \text{cost}_{F,G,H}((M,M'))-c_{\text{del}}(r_F) = \delta(F,G,H) - c_{\text{del}}(r_F)$. Therefore, $\delta(F-r_F,G,H) \leq \delta(F,G,H) - c_{\text{del}}(r_F)$, so $\alpha(F,G,H) \leq \delta(F,G,H)$. The same is true when r_G is not matched in M.

Suppose now that both r_F and r_G are matched in M. Since M is an edit matching, r_F must be matched to r_G. Let $M_1 = \{(v,v') \in M : v \in R_F - r_F\}$ and $M_2 = M \setminus (M_1 \cup \{(r_F,r_G)\})$. We have that M_1 is an edit matching between the vertices of $R_F - r_F$ and the vertices of $R_G - r_G$, and M_2 is an edit matching between the vertices of $F - R_F$ and the vertices of $G - R_G$.

We now consider two cases. In the first case suppose that r_F is matched in M'. r_F must be matched to r_H (as M' is an edit matching). Let $M_1' = \{(v,v') \in M' : v \in R_F - r_F\}$ and $M_2' = M' \setminus (M_1' \cup \{(r_F,r_H)\})$. Again, we have that M_1' is an edit matching between the vertices of $R_F - r_F$ and the vertices of $R_H - r_H$, and M_2' is an edit matching between the vertices of $F - R_F$ and the vertices of $H - R_H$. Moreover, (M_1,M_1') is a matchings pair for $R_F - r_F, R_G - r_G, R_H - r_H$ and (M_2',M_2) is a matchings pair for $F - R_F, G - R_G, H - R_H$. Therefore,

$$\text{cost}_{R_F-r_F,R_G-r_G,R_H-r_H}((M_1,M_1')) + \text{cost}_{F-R_F,G-R_G,H-R_H}((M_2,M_2')) =$$
$$\text{cost}_{F,G,H}((M,M')) - c_{\text{rel}}(r_F,r_H) - c_{\text{rel}}(r_G,r_H) = \delta(F,G,H) - c_{\text{rel}}(r_F,r_H) - c_{\text{rel}}(r_G,r_H),$$

so

$$\delta(F-R_F,G-R_G,H-R_H) + \delta(R_F-r_F,R_G-r_G,R_H-r_H) \leq \delta(F,G,H) - c_{\text{rel}}(r_F,r_H) - c_{\text{rel}}(r_G,r_H).$$

It follows that $\alpha(F,G,H) \leq \delta(F,G,H)$ (as $H - R_H = H[\phi,1,t(H)-1]$ and $R_H - r_H = H[r_H,1,d(r_H)]$).

If r_F is not matched in M', let i be the maximum index such that there is a vertex of $F-R_F$ that is matched in M' to a vertex in $H[\phi,i,i]$. Define $M_1' = \{(v,v') \in M' : v \in R_F - r_F\}$ and $M_2' = M' \setminus M_1'$. We have that M_1' is an edit matching between the vertices of $R_F - r_F$ and the vertices of $H[\phi,i+1,t(H)]$, and M_2' is an edit matching between the vertices of $F - R_F$ and the vertices of $H[\phi,1,i]$. It follows that

$$\delta(F-R_F,G-R_G,H[\phi,1,i]) + \delta(R_F-r_F,R_G-r_G,H[\phi,i+1,t(H)]) \leq \delta(F,G,H) - c_{\text{rel}}(r_F,r_G).$$

Therefore, $\alpha(F,G,H) \leq \delta(F,G,H)$.

In all cases above we showed that $\alpha(F,G,H) \leq \delta(F,G,H)$. Showing the opposite direction is similar. ∎

Observation 3. *Let $H_1 = H[x,a,b]$ for some x, a, and b, and let $H_2 = H_1[\phi,i,j]$. Then $H_2 = H[x,a+i-1,a+j-1]$.*

Theorem 4. *Let S be a strategy algorithm for tree edit distance with time complexity $t(F,G)$ and space complexity $s(F,G)$. Then there is an algorithm for guided tree edit distance with time complexity $O(t(F,G) \cdot |H|\, d(H)^2 (\log\log d(H))^3 / \log^2 d(H))$ and space complexity $O(s(F,G) \cdot d(H)^2 + |F|\,|G|\,|H|)$.*

Proof. We define an algorithm $\hat{S}$ for guided tree edit distance. Let F, G, and H be an input to the algorithm. For the rest of this section, let $d(\phi) = t(H)$. Let $(F_1,G_1),\ldots,(F_K,G_K)$ be the relevant pairs w.r.t. F,G,S according to the computation order. Algorithm $\hat{S}$ is as follows.

Initialize tables A_x for every $x \in H \cup \{\phi\}$.
for each vertex $x \in H$ in postorder and for $x = \phi$ **do**
  for $t = 1, \ldots, K$ **do**
    Compute $\delta(F_t, G_t)$ using the same rule as S uses to compute $\delta(F_t, G_t)$.
    if algorithm S uses rule (1) to compute $\delta(F_t, G_t)$ **then**
      Compute $\delta(F_t, G_t, H[x, i, j])$ for all $1 \leq i \leq j \leq d(x)$ using rule (3).
    else
      Compute $\delta(F_t, G_t, H[x, i, j])$ for all $1 \leq i \leq j \leq d(x)$ using rule (4).
    if $(F_t, G_t) = (F[v, 1, d(v)], G[w, 1, d(w)])$ for some v and w **then**
      $A_x[v, w] \leftarrow \delta(F_t, G_t, H[x, 1, d(x)])$.
    for every value $\delta(F_{t'}, G_{t'})$ that S discards after computing $\delta(F_t, G_t)$ **do**
      Discard $\delta(F_{t'}, G_{t'})$ and $\delta(F_{t'}, G_{t'}, H[x, i, j])$ for all $1 \leq i \leq j \leq d(x)$.
  Discard $\delta(F_t, G_t)$ and $\delta(F_t, G_t, H[x, i, j])$ for all t, i, and j.

By Lemma 1 after processing a vertex x, the table A_x contains the value of $\delta(F[v, 1, d(v)], G[w, 1, d(w)], H[x, 1, d(x)])$ for every $v \in F$ and $w \in G$.

Fix some i and j, and consider the terms that appear in the computation of $\delta(F_t, G_t, H[x, i, j])$ using rule (3). By Observation 3, these terms are of the form $\delta(F', G', \emptyset) = \delta(F', G')$ or $\delta(F', G', H[x, i', j'])$ (where F' is a subforest of F_t, G' is a subforest of G_t, and $i \leq i' \leq j' \leq j$), except for the term $\delta(R_{F_t} - r_{F_t}, R_{G_t} - r_{G_t}, H[r_{H[x,i,j]}, 1, d(r_{H[x,i,j]})])$. For each term $\delta(F', G', H[x, i', j'])$ (or $\delta(F', G')$) we have that the term $\delta(F', G')$ appears in the computation of $\delta(F', G')$ using rule (1). Since algorithm S must store the value of $\delta(F', G')$ in memory, it follows that algorithm $\hat{S}$ stores in memory the value of $\delta(F', G', H[x, i', j'])$ (or $\delta(F', G')$). As for $\delta(R_{F_t} - r_{F_t}, R_{G_t} - r_{G_t}, H[r_{H[x,i,j]}, 1, d(r_{H[x,i,j]})])$, this value is stored in the $A_{r_{H[x,i,j]}}$ table (note that $R_{F_t} - r_{F_t} = F[r_{F_t}, 1, d(r_{F_t})]$) .

For each relevant pair (F_t, G_t), algorithm $\hat{S}$ computes the value of $\delta(F_t, G_t, H[x, i, j])$ for every $x \in H \cup \{\phi\}$ and every $1 \leq i \leq j \leq d(x)$. The number of these values is

$$\sum_{x \in H \cup \{\phi\}} \binom{d(x)+1}{2} \leq \sum_{x \in H \cup \{\phi\}} d(x)^2 \leq d(H) \sum_{x \in H \cup \{\phi\}} d(x) = d(H)\,|H|\,.$$

Therefore, the number of δ values computed by algorithm $\hat{S}$ is $O(K \cdot d(H)\,|H|) = O(t(F, G) \cdot d(H)\,|H|)$. Moreover, computing some value $\delta(F_t, G_t, H[x, i, j])$ takes $O(j - i + 1) = O(d(x)) = O(d(H))$ time. Therefore, the time complexity of algorithm $\hat{S}$ is $O(t(F, G) \cdot |H|\, d(H)^2)$.

We now show how to improve the time complexity of algorithm $\hat{S}$. In order to compute $\delta(F_t, G_t, H[x, i, j])$ for all $1 \leq i \leq j \leq d(x)$, we compute $c_{ij} = \min_{k=i,\ldots,j-1} \delta(F_t - R_{F_t}, G_t - R_{G_t}, H[x, i, k]) + \delta(R_{F_t} - r_{F_t}, R_{G_t} - r_{G_t}, H[x, k + 1, j])$ for all $1 \leq i \leq j \leq d(x)$. After the computation of these values, computing each $\delta(F_t, G_t, H[x, i, j])$ takes constant time.

The computation of all c_{ij}-s is done using distance matrix product (see also [2]). Define matrices $A = \{a_{ij}\}_{i,j=1}^{d(x)}$ and $B = \{b_{ij}\}_{i,j=1}^{d(x)}$, where

$$a_{ij} = \begin{cases} \delta(F_t - R_{F_t}, G_t - R_{G_t}, H[x, i, j]) & \text{if } i \leq j \\ \infty & \text{otherwise} \end{cases}$$

and

$$b_{ij} = \begin{cases} \delta(R_{F_t} - r_{F_t}, R_{G_t} - r_{G_t}, H[x, i + 1, j]) & \text{if } i + 1 \leq j \\ \infty & \text{otherwise} \end{cases}.$$

Then, $c_{ij} = \min_k\{a_{ik} + b_{kj}\}$. The matrix $C = \{c_{ij}\}_{i,j=1}^{d(x)}$ is called the *distance matrix product* of A and B, and can be computed in $O(d(x)^3(\log\log d(x))^3/\log^2 d(x))$ time [1]. It follows that the time complexity of algorithm $\hat{S}$ is $O(t(F, G) \cdot |H|\, d(H)^2(\log\log d(H))^3/\log^2 d(H))$.

Finally, we compute the space complexity of algorithm $\hat{S}$. The distance matrix product computation when handling $x \in H \cup \{\phi\}$ takes $o(d(x)^3) = o(d(H)^3) = o(|F|\,|G|\,|H|)$ space. The space used by the A_x tables is $O(|F|\,|G|\,|H|)$. For each value of $\delta(F', G')$ that is stored by S at some step, algorithm $\hat{S}$ stores at most $d(H)^2$ values $\delta(F', G', H[x, i, j])$ for all i and j. Thus, the space complexity of algorithm $\hat{S}$ is $O(s(F, G) \cdot d(H)^2 + |F|\,|G|\,|H|)$. ■

The time complexity of the algorithm of Shasha and Zhang [7] is $O(|F|\,\mathrm{cdepth}(F) \cdot |G|\,\mathrm{cdepth}(G))$. The algorithm of Demaine et al. [3] has time complexity $O(|F|\,|G|^2\,(1 + \log \frac{|F|}{|G|}))$. Both these algorithms have space complexity $O(|F|\,|G|)$.

Corollary 5. *There are algorithms for guided tree edit distance with time complexities* $O(|F|\,\mathrm{cdepth}(F) \cdot |G|\,\mathrm{cdepth}(G) \cdot |H|\,d(H)^2(\log\log d(H))^3/\log^2 d(H))$ *and* $O(|F|\,|G|^2\,(1 + \log \frac{|F|}{|G|}) \cdot |H|\,d(H)^2 (\log\log d(H))^3/\log^2 d(H))$. *The space complexity of these algorithms is* $O(|F|\,|G|\,(d(H)^2 + |H|))$.

References

[1] T. M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. In *Proc. 39th Symposium on the Theory of Computing (STOC)*, pages 590–598, 2007.

[2] W. Chen. New algorithm for ordered tree-to-tree correction problem. *J. of Algorithms*, 40:135–158, 2001.

[3] E. D. Demaine, S. Mozes, B. Rossman, and O. Weimann. An optimal decomposition algorithm for tree edit distance. In *Proc. 34th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 146–157, 2007.

[4] P. N. Klein. Computing the edit-distance between unrooted ordered trees. In *Proc. 6th European Symposium on Algorithms (ESA)*, pages 91–102, 1998.

[5] Z. Peng and H. Ting. Guided forest edit distance: Better structure comparisons by using domain-knowledge. In *Proc. 18th Symposium on Combinatorial Pattern Matching (CPM)*, pages 28–39, 2007.

[6] K. Tai. The tree-to-tree correction problem. *J. Assoc. Comput. Mach.*, 26(3):422–433, 1979.

[7] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. on Computing*, 18(6):1245–1262, 1989.