Space efficient algorithms for ordered tree comparison

Lusheng Wang[1] and Kaizhong Zhang[2]

[1] Department of Computer Science, City University of Hong Kong, Hong Kong
cswangl@cityu.edu.hk
[2] Dept. of Computer Science, University of Western Ontario, London, Ont. N6A 5B7, Canada
kzhang@csd.uwo.ca

Abstract. In this paper we present techniques to significantly improve the space complexity of several ordered tree comparison algorithms without sacrificing the corresponding time complexity. We present new algorithms for computing the constrained ordered tree edit distance and the alignment of (ordered) trees. The techniques can also be applied to other related problems.

keywords: Space efficient algorithms, Constrained tree edit distance, Alignment of trees.

1 Introduction

Ordered labeled trees are trees whose nodes are labeled and in which the left to right order among siblings is significant. Comparing such trees with distance and/or similarity measures has applications in several diverse areas such as computational molecular biology [8, 5], computer vision, pattern recognition, programming compilation and natural language processing.

Algorithms have been developed for ordered labeled tree comparisons [10, 9, 14, 5, 12]. The degree-one edit distance was introduced by Selkow [9] in which insertions and deletions are restricted to the leaves of the trees. The degree-two edit distance was introduced by Zhang et al. [13, 16] in which insertions and deletions are restricted to tree nodes with zero or one child. Zhang [12] introduced the constrained edit distance which is a bit more general than degree-two edit distance. The (general) edit distance between ordered labeled trees was introduced by Tai [10]. His algorithm has been improved by several authors [14, 6, 3]. The alignment of trees was introduced by Jiang et al. in [5].

Given two ordered trees T_1 and T_2, the degree-one edit distance, the degree-two edit distance, and the constrained edit distance can all be computed in $(|T_1||T_2|)$ time and space [9, 12, 13], where $|\cdot|$ is the number of nodes in the tree. Richter [7] presented an algorithm for the constrained edit distance with $O(|T_1||T_2|deg(T_1)deg(T_2))$ time and $O(|T_1|dep(T_2)deg(T_2))$ space, where $deg(\cdot)$ is the degree of the tree and $dep(\cdot)$ is the depth of the tree. For small degree and low depth trees, this is a space improvement. According to a recent survey

on tree edit distance by Bille [1], the algorithms of Zhang [12] and Richter [7] are currently the best for the constrained tree edit distance. In this paper, we present an algorithm for the constrained tree edit distance with $O(|T_1||T_2|)$ time and $O(\log(|T_1|)|T_2|)$ space. The techniques used can also be used for the degree-one and the degree-two edit distance to achieve the same time and space complexities.

For alignment of trees, the algorithm in [5] runs in $O(|T_1|T_2|(deg(T_1) + deg(T_2))^2)$ time and needs $O(|T_1||T_2|(deg(T_1) + deg(T_2))$ space. Recently, there is a strike to reduce the space [11]. The space required for the new algorithm is $O(\log(|T_1|)|T_2|(deg(T_1) + deg(T_2))deg(T_1))$. However, the running time is increased to $O(|T_1|^2|T_2|(deg(T_1) + deg(T_2))^2)$. In this paper, we proposed a new algorithm that keeps the same space complexity and reduces the run timep by a factor of $O(|T_1|)$.

2 Constrained edit distance between ordered trees

The nodes in an ordered tree of size n are numbered from 1 to n according to the postorder, where the left siblings are ordered before the right siblings. Given an ordered labeled tree T, the ith node of tree T is represented as $t[i]$, the subtree of T rooted at node $t[i]$ is represented as $T[i]$, and the subforest obtained by deleting $t[i]$ from $T[i]$ is represented as $F[i]$. The parent of node $t[i]$ in T is denoted as $t[p(i)]$. The number of nodes in a tree T is denoted as $|T|$.

Let θ be the empty tree, and λ a inserted space. $\gamma(a, \lambda)$, $\gamma(\lambda, a)$, and $\gamma(a, b)$ denote the cost of deleting a, inserting a, and substituting a with b, respectively.

Consider two trees T_1 and T_2 to compare. Suppose that the degrees of node $t_1[i]$ and node $t_2[j]$ are m_i and n_j, respectively. Denote the children of $t_1[i]$ from left to right as $t_1[i_1], ..., t_1[i_{m_i}]$ and children of $t_2[j]$ from left to right as $t_2[j_1], ..., t_2[j_{n_j}]$.

The constrained edit distance metric is based on a constrained edit mapping allowed between two trees. We will first give the definition of constrained editing mapping and then use it to define the constrained editing distance metric.

Formally we define a triple (M, T_1, T_2) to be a constrained edit mapping from T_1 to T_2, where M is any set of pairs of integers (i, j) satisfying:

(1) $1 \leq i \leq |\ T_1\ |$, $1 \leq j \leq |\ T_2\ |$;
(2) For any pair of (i_1, j_1) and (i_2, j_2) in M,
   (a) $i_1 = i_2$ iff $j_1 = j_2$ (one-to-one)
   (b) $t_1[i_1]$ is to the left of $t_1[i_2]$ iff $t_2[j_1]$ is to the left of $t_2[j_2]$ (sibling order preserved);
   (c) $t_1[i_1]$ is an ancestor of $t_1[i_2]$ iff $t_2[j_1]$ is an ancestor of $t_2[j_2]$ (ancestor order preserved);
(3) For any triple (i_1, j_1), (i_2, j_2) and (i_3, j_3) in M, let $lca()$ represent least common ancestor function,
   $t_1[lca(i_1, i_2)]$ is a proper ancestor of $t_1[i_3]$ iff $t_2[lca(j_1, j_2)]$ is a proper ancestor of $t_2[j_3]$.

This definition adds constraint (3) to the definition of the edit mapping [14]. This is why we call it a constrained edit mapping. The intuitive idea behind this definition is that two separate subtrees of T_1 should be mapped to two subtrees of T_2 and vice versa.

We will use M instead of (M, T_1, T_2) if there is no confusion. Let M be a constrained editing mapping from T_1 to T_2. We can define the cost of M:

$$\gamma(M) = \sum_{(i,j)\in M} \gamma(t_1[i], t_2[j]) + \sum_{i\notin M} \gamma(t_1[i], \lambda) + \sum_{j\notin M} \gamma(\lambda, t_2[j])$$

We can now define the constrained edit distance between T_1 and T_2 as:

$$D(T_1, T_2) = \min_M \{\gamma(M) \mid M \text{is a constrained edit mapping between } \mathrm{T_1toT_2}\}$$

2.1 A simple algorithm

We now present an algorithm for computing the constrained edit distance between two ordered trees. This algorithm is given in [12] and is the basis of our new space efficient algorithm.

The algorithm in [12] for the constrained edit distance between two ordered trees is given in Figure 1. While we do not show the details of the correctness of the algorithm, we now explain the ideas of the algorithm.

The computation of $D(T_1[i], T_2[j])$ has several cases. In one case, $T_1[i]$ is matched to a child subtree of $T_2[j]$. In a symmetric case, $T_2[j]$ is matched to a child subtree of $T_1[i]$. In the last case, $t_1[i]$ is matched with $t_2[j]$ and therefore $F_1[i]$ is matched with $F_2[j]$ which means that we need to know $D(F_1[i], F_2[j])$.

Similarly, the computation of $D(F_1[i], F_2[j])$ has several cases. In the first case, $F_1[i]$ is matched to $F_2[j_k]$, $1 \le k \le n_j$, a subforest of a child of $F_2[j]$. In the second case, $F_2[j]$ is matched to $F_1[i_k]$, $1 \le k \le m_i$, a subforest of a child of $F_1[i]$. In the third case, there is a matching between $T_1[i_1], ..., T_1[i_{m_i}]$ and $T_2[j_1], ..., T_2[j_{n_j}]$. We uae $E(m_i, n_j)$ to denote the optimal matching between $T_1[i_1], ..., T_1[i_{m_i}]$ and $T_2[j_1], ..., T_2[j_{n_j}]$. $E(m_i, n_j)$ can be computed using the sequence edit distance algorithm treating each subtree as an unit. The recurrence equations are shown in Figure 1. All the values can be computed in a bottom up fashion.

The time and space complexity for computing $E(m_i, n_j)$ is obviously $O(m_i \times n_j)$. The complexity of computing $D(T_1[i], T_2[j])$ is bounded by $O(m_i + n_j)$. The complexity of computing $D(F_1[i], F_2[j])$ is bounded by the $O(m_i + n_j)$.

Hence for any pair i and j, the complexity of computing $D(T_1[i], T_2[j])$ and $D(F_1[i], F_2[j])$ is bounded by $O(m_i \times n_j)$. Therefore the complexity of the algorithm is

$$\sum_{i=1}^{|T_1|}\sum_{j=1}^{|T_2|} O(m_i \times n_j) = O(\sum_{i=1}^{|T_1|} m_i \times \sum_{j=1}^{|T_2|} n_j) = O(|T_1| \times |T_2|)$$

$$E(s,t) = \min \begin{cases} E(s,t-1) + D(\theta, T_2[j_t]) \\ E(s-1,t) + D(T_1[i_s], \theta) \\ E(s-1,t-1) + D(T_1[i_s], T_2[j_t]), \end{cases}$$

$$D(F_1[i], F_2[j]) = \min \begin{cases} D(F_1[i], \theta) + \min\limits_{1 \le s \le m_i} \{D(F_1[i_s], F_2[j]) - D(F_1[i_s], \theta)\} \\ D(\theta, F_2[j]) + \min\limits_{1 \le t \le n_j} \{D(F_1[i], F_2[j_t]) - D(\theta, F_2[j_t])\} \\ E(n_i, n_j), \end{cases}$$

$$D(T_1[i], T_2[j]) = \min \begin{cases} D(T_1[i], \theta) + \min_{1 \le s \le m_i} \{D(T_1[i_s], T_2[j]) - D(T_1[i_s], \theta)\} \\ D(\theta, T_2[j]) + \min_{1 \le t \le n_j} \{D(T_1[i], T_2[j_t]) - D(\theta, T_2[j_t])\} \\ D(F_1[i], F_2[j]) + \gamma(t_1[i] \to t_2[j]). \end{cases}$$

Fig. 1. The equations for the simple algorithm.

2.2 Reducing the space complexity

In practice, the space required for the simple algorithm maybe a bottleneck. In this section, we propose a method to modify Algorithm 1 so that the space required is reduced to $O(\log(|T_1|) \cdot |T_2|)$.

The basic idea is straightforward. In order to compute the constrained edit distance, some computed values are no longer useful after they were used. We simple release the space that is no longer useful. To achieve our goal, we need to be more careful about the computational order of nodes of tree T_1, not just an arbitrary postorder. We also modify the computation of node $t_1[i]$ slightly. When the computation for node $t_1[i_1]$ is completed, we immediately start the computation of $E[i, j]$ without waiting for the computation of $t_1[i_2]$.

Here we give a more restricted order for the nodes in T_1. Consider a node $t_1[i]$ in T_1. Let $t_1[i_1]$, $t_1[i_2]$, ..., $t_1[i_{m_i}]$ be the children of $t_1[i]$ in T_1 and $|T_1[i_k]|$ be the number of nodes in the subtree $T_1[i_k]$. The *computational order* of the children of $t_1[i]$ is that the child with largest number of nodes, which we refer to as the favorable child, would be the first one and then from left to right for the rest of children. This order is applied to all the nodes and if $t_1[u]$ and $t_1[v]$ are siblings in T_1 and $t_1[u]$ is ordered before $t_1[v]$ then all nodes in $T_1[u]$ are ordered before any node in $T_1[v]$. Figure 2 gives an example of the new order.

The modified algorithm will use this new order for T_1 and the same post order as in the simple algorithm for T_2. For each node $t_1[i]$, we will compute $D(F_1[i], F_2[j])$ and $D(T_1[i], T_2[j])$ for all nodes $t_2[j]$ in T_2. Now suppose that $t_1[p(i)]$ is the parent of $t_1[i]$, then we can immediately use $D(F_1[i], F_2[j])$ and $D(T_1[i], T_2[j])$ for the computation of $D(F_1[p(i)], F_2[j])$ and $D(T_1[p(i)], T_2[j])$. There are two cases. One case is that $t_1[i]$ is the favorable child of $t_1[p(i)]$ and it is not the leftmost child of $t_1[p(i)]$. In this case, we need to keep the values of $D(F_1[i], F_2[j])$ and $D(T_1[i], T_2[j])$ since the computation of $E(p(i), j)$ is from left to right. The other case is that $t_1[i]$ is not the favorable child or it is the favorable child and also the leftmost child. In this case, we can use it immediately for the computation of $E(p(i), j)$. Therefore for each node $t_1[i]$ in T_1 such that at least its favorable child's computation is completed, we at most will need to keep two sets of values: the values for its favorable child and the partial computation of

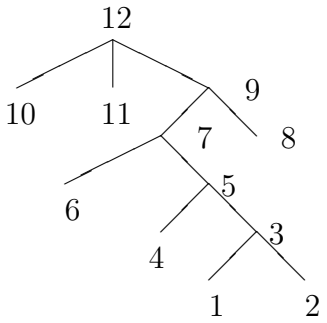

Fig. 2. An example of the new order.

$E(m_i, n_j)$. Notice that for any node $t_1[i]$ in T_1, if the computation of its favorable child is not completed, we do not need to store any value and if the computation of all its children have been completed, then in next step the computation for node $t_1[i]$ is completed and we can release the space used.

By using such a new order for T_1 and the above straightforward idea we now have an algorithm with much less space. Assuming the initial values are set, the modified algorithm is given in Figure 3 as Algorithm 2.

The following lemma shows that the space complexity of Algorithm 2 is reduced to $O(\log(|T_1|)|T_2|)$.

Lemma 1. *The time and space complexities of Algorithm 2 are* $O(|T_1||T_2|)$ *and* $O(\log(|T_1|)|T_2|)$.

Proof. From the algorithm, it is clear that for each i of T_1, the number of steps is bounded by $O(\sum_{j=1}^{|T_2|} n_j)$. Therefore the time complexity of Algorithm 2 is $O(\sum_{i=1}^{|T_1|} \sum_{j=1}^{|T_2|} n_j) = O(|T_1||T_2|)$.

For the space complexity, let us consider the status of a node in T_1 with respect to the space requirement. We say a node $t_1[i]$ in T_1 is active if the computation of its favorable child is completed and the computation for node $t_1[i]$ itself has not been completed. Therefore a node is inactive if the computation of its favorable child has not been completed or if the computation for node i is completed.

If a node is inactive because its computation is completed and this node is the favorable child of its parent, then we cannot immediately release the space used since these values will be used for the computation of its parent. However in this situation, we can contribute the space requirement of the favorable child to its parent, which is active, when counting the space requirement. Therefore, we can assume that if a node is inactive, then we can release the space used for that node. This means that we only have to check maximum space used by active nodes during the execution of the algorithm.

For any active node $t_1[p]$, assuming that its favorable child is $t_1[p_f]$, then we need $O(|T_2|)$ to store $D(F_1[p_f], F_2[j])$ and $D(T_1[p_f], T_2[j])$ for $1 \leq j \leq |T_2|$, and

Input: T_1 and T_2
Output: $D(T_1, T_2[j])$, where $1 \leq j \leq |T_2|$

for $i = 1$ **to** $|T_1|$ (the new order)
  for $j = 1$ **to** $|T_2|$

$$D(F_1[i], F_2[j]) = \min \begin{cases} D(\theta, F_2[j]) + \min\limits_{1 \leq t \leq n_j} \{D(F_1[i], F_2[j_t]) - D(\theta, F_2[j_t])\} \\ F_i \\ E_i(n_j); \end{cases}$$

$$D(T_1[i], T_2[j]) = \min \begin{cases} D(\theta, T_2[j]) + \min_{1 \leq t \leq n_j} \{D(T_1[i], T_2[j_t]) - D(\theta, T_2[j_t])\} \\ G_i \\ D(F_1[i], F_2[j]) + \gamma(t_1[i], t_2[j]); \end{cases}$$

Let $t_1[p(i)]$ be the parent of $t_1[i]$ and $t_1[p_f]$ be the favorable child of $t_1[p(i)]$
if $t_1[i]$ is the favorable child of $t_1[p(i)]$ **then**
  $E_p(0) = 0$;
  $F_p = D(F_1[p(i)], \theta) + D(F_1[i], F_2[j]) - D(F_1[i], \theta)$;
  $G_p = D(T_1[p(i)], \theta) + D(T_1[i], T_2[j]) - D(T_1[i], \theta)$;
  for $t = 1$ **to** n_j
    $E_p(t) = E(t-1) + D(\theta, T_2[j_t])$;
if $t_1[i]$ is not the favorable child of $t_1[p(i)]$ or $t_1[i]$ is the leftmost child of $t_1[p(i)]$ **then**
  $F_p = \min\{F_p, D(F_1[p(i)], \theta) + D(F_1[i], F_2[j]) - D(F_1[i], \theta)\}$;
  $G_p = \min\{G_p, D(T_1[p(i)], \theta) + D(T_1[i], T_2[j]) - D(T_1[i], \theta)\}$;
  for $t = 0$ **to** n_j
    $E_p^0(t) = E_p(t)$;
  $E_p(0) = E_p^0(0) + D(T_1[i], \theta)$;
  for $t = 1$ **to** n_j

$$E_p(t) = \min \begin{cases} E_p(t-1) + D(\theta, T_2[j_t]) \\ E_p^0(t) + D(T_1[i], \theta) \\ E_p^0(t-1) + D(T_1[i], T_2[j_t]); \end{cases}$$

if $t_1[i]$ is the left sibling of the favorable child $t_1[p_f]$ of $t_1[p(i)]$ **then**
  $F_p = \min\{F_p, D(F_1[p(i)], \theta) + D(F_1[p_f], F_2[j]) - D(F_1[p_f], \theta)\}$;
  $G_p = \min\{G_p, D(T_1[p(i)], \theta) + D(T_1[p_f], T_2[j]) - D(T_1[p_f], \theta)\}$;
  for $t = 0$ **to** n_j
    $E_p^0(t) = E_p(t)$;
  $E_p(0) = E_p^0(0) + D(T_1[p_f], \theta)$;
  for $t = 1$ **to** n_j

$$E_p(t) = \min \begin{cases} E_p(t-1) + D(\theta, T_2[j_t]) \\ E_p^0(t) + D(T_1[p_f], \theta) \\ E_p^0(t-1) + D(T_1[p_f], T_2[j_t]). \end{cases}$$

Fig. 3. A more space efficient algorithm, Algorithm 2

$O(\sum_{j=1}^{|T_2} n_j) = O(|T_2|)$ to store $E_p(t)$ where $1 \leq t \leq n_j$ for all j in T_2. Therefore for each active node, the space required is $O(|T_2|)$. Because of the new order, if two nodes are active at the same time, then one has to be the ancestor of the other. Therefore all active nodes of T_1 are on a path in T_1. Let $t_1[s]$ and $t_1[t]$ be two neighboring active nodes on this path and $t_1[s]$ is an ancestor of $t_1[t]$, then $|T_1[s]| \geq 2|T_1[t]|$ since the computation of $t_1[s_f]$, the favorable child of $t_1[s]$ with maximum size, is already completed and therefore $t_1[t]$ is not a descendant of $t_1[s_f]$.

This means that there are at most $O(\log(|T_1|))$ active nodes. Therefore the space complexity is $O(\log(|T_1|)|T_2|)$.

2.3 Finding the optimal constrained edit mapping between two trees

In previous section we show that $D(T_1, T_2)$ can be computed in $O(|T_1||T_2|)$ time and $O(\log(|T_1|)|T_2|)$ space. However in real application the optimal mapping that achieves $D(T_1, T_2)$ maybe required.

Finding the optimal mapping in $O(\log(|T_1|)|T_2|)$ space is in fact a more difficult task. This is similar to the situation of computing edit distance between two sequences. Computing the edit distance in linear space is easy, but finding the optimal edit script in linear space is more involved. Hirschberg [4] presented a clever way to do this.

In this section we will present a method that can find the optimal mapping between two ordered trees in $O(|T_1||T_2|)$ time and $O(\log(|T_1|)|T_2|)$ space. When the input is two sequences (trees such that each node only has one child), then our method produces the optimal edit script for sequence edit distance in $O(|T_1||T_2|)$ time and $O(|T_2|)$ space.

The main idea is as the follows. Given T_1 and T_2, there is a unique node, called key node, $t_1[k]$ in T_1 such that in $O(|T_1||T_2|)$ time and $O(\log(|T_1|)|T_2|)$ space we can determine, in the optimal mapping, what subtree and subforest in T_2 that $T_1[k]$ and $F_1[k]$ would match to produce $D(T_1, T_2)$. With this information, we then decompose the optimal mapping into several components mapping such that for each component the subtree or subforest of T_1 involved has a size less than or equal to half of $|T_1$. This means that in the next step if we repeat the same process for each component then the total cost for all the components is $O(0.5|T_1||T_2|)$ time using $O(\log(|T_1|)|T_2|)$ space. Therefore, repeating this process at most $\log(|T_1|)$ times, we can compute the optimal mapping in $O(|T_1||T_2|)$ time and $O(\log(|T_1|)|T_2|)$ space.

Given T_1, the unique key node $t_1[k]$, with children $t_1[k_1], t_2[k_2], ..., t_1[k_{m_k}]$, is a node satisfying the following properties.

- $|T_1[k]| > 0.5|T_1|$,
- $|T_1[k_s]| \leq 0.5|T_1|$ for $1 \leq s \leq m_k$.

With a small modification of algorithm 2, when computing $D(T_1, T_2)$, we can also compute two integers t_1 and t_2 for subtree $T_1[k]$ and two integers f_1 and f_2 for subforest $F_1[k]$.

We now define the meaning of t_1 and t_2. If in the optimal mapping $T_1[k]$ is deleted, then $t_1 = t_2 = 0$. If in the optimal mapping subtree $T_1[k]$ is matched with subtree $T_2[l]$ such that in this mapping $T_1[k_s]$ is matching with $T_2[l]$ and the rest of $T_1[k]$ is deleted, then $t_1 = k_s$ and $t_2 = l$. If in the optimal mapping node $t_1[k]$ is matched with node $t_2[l]$, then $t_1 = k$ and $t_2 = l$. If in the optimal mapping $t_1[k]$ is deleted and $F_1[k]$ is matched with $F_2[l]$, then $t_1 = k + 1$ and $t_2 = l$.

f_1 and f_2 are defined similarly. Note that f_1 and f_2 are only meaningful when $t_1 = k$ or $t_1 = k+1$, Otherwise $f_1 = f_2 = -1$. If in the optimal mapping $F_1[k]$ is deleted, then $f_1 = f_2 = 0$. If in the optimal mapping $F_1[k]$ is matched with $F_2[l]$ such that in this mapping $F_1[k_s]$ is matching with $F_2[l]$ and the rest of $F_1[k]$ are deleted, then $f_1 = k_s$ and $f_2 = l$. If in the optimal mapping $F_1[k]$ is matched with $F_2[l]$ and $D(F_1[k], F_2[l]) = E(m_k, n_l)$, then $f_1 = k$ and $f_2 = l$.

We now give the formulae for computing t_1, t_2, f_1, and f_2. We use $t\_T(i, j)$ to represent (t_1, t_2) in the optimal mapping of $D(T_1[i], T_2[j])$. We use $f\_T(i, j)$ to represent (f_1, f_2) in the optimal mapping of $D(T_1[i], T_2[j])$. We use $t\_F(i, j)$ to represent (t_1, t_2) in the optimal mapping of $D(F_1[i], F_2[j])$. We use $f\_F(i, j)$ to represent (f_1, f_2) in the optimal mapping of $D(F_1[i], F_2[j])$.

Lemma 2. *Let $t_1[k]$ be the key node of T_1, then*

$$f\_F(k,j)=\begin{cases}(k,j) & \text{if } D(F_1[k],F_2[j])=E(m_k,n_j)\\ (k_s,j) & \text{if } D(F_1[k],F_2[j])=D(F_1[k],\theta)+D(F_1[k_s],F_2[j])-D(F_1[k_s],\theta)\\ f\_F(k,j_t) & \text{if } D(F_1[k],F_2[j])=D(\theta,F_2[j])+D(F_1[k],F_2[j_t])-D(\theta,F_2[j_t]),\end{cases}$$

$$t\_T(k,j)=\begin{cases}(k,j) & \text{if } D(T_1[k],T_2[j])=D(F_1[k],F_2[j])+\gamma(t_1[k],t_2[j])\\ (k_s,j) & \text{if } D(T_1[k],T_2[j])=D(T_1[k],\theta)+D(T_1[k_s],T_2[j])-D(T_1[k_s],\theta)\\ t\_T(k,j_t) & \text{if } D(T_1[k],T_2[j])=D(\theta,T_2[j])+D(T_1[k],T_2[j_t])-D(\theta,T_2[j_t]),\end{cases}$$

$$f\_T(k,j)=\begin{cases}f\_F(k,j) & \text{if } D(T_1[k],T_2[j])=D(F_1[k],F_2[j])+\gamma(t_1[k],t_2[j])\\ (-1,-1) & \text{if } D(T_1[k],T_2[j])=D(T_1[k],\theta)+D(T_1[k_s],T_2[j])-D(T_1[k_s],\theta)\\ f\_T(k,j_t) & \text{if } D(T_1[k],T_2[j])=D(\theta,T_2[j])+D(T_1[k],T_2[j_t])-D(\theta,T_2[j_t]).\end{cases}$$

Lemma 3. *Let $t_1[i]$ be the parent of $t_1[k]$. Let x be t or f in the following formula for $x\_T(i, j)$.*

$$t\_F(i,j)=\begin{cases}(0,0) & \text{if } D(F_1[i],F_2[j])=E(m_i,n_j) \text{ and } T_1[k] \text{ is deleted}\\ t\_T(k,j_t) & \text{if } D(F_1[i],F_2[j])=E(m_i,n_j) \text{ and } T_1[k] \text{ is matched to } T_2[j_t]\\ (0,0) & \text{if } D(F_1[i],F_2[j])=D(F_1[i],\theta)+D(F_1[i_s],F_2[j])-D(F_1[i_s],\theta)\\ & i_s\neq k\\ (k+1,j) & \text{if } D(F_1[i],F_2[j])=D(F_1[i],\theta)+D(F_1[k],F_2[j])-D(F_1[k],\theta)\\ t\_F(i,j_t) & \text{if } D(F_1[i],F_2[j])=D(\theta,F_2[j])+D(F_1[i],F_2[j_t])-D(\theta,F_2[j_t]),\end{cases}$$

$$f\_F(i,j)=\begin{cases}(0,0) & \text{if } D(F_1[i],F_2[j])=E(m_i,n_j) \text{ and } T_1[k] \text{ is deleted}\\ f\_T(k,j_t) & \text{if } D(F_1[i],F_2[j])=E(m_i,n_j) \text{ and } T_1[k] \text{ is matched to } T_2[j_t]\\ (0,0) & \text{if } D(F_1[i],F_2[j])=D(F_1[i],\theta)+D(F_1[i_s],F_2[j])-D(F_1[i_s],\theta)\\ & i_s\neq k\\ f\_F(k,j) & \text{if } D(F_1[i],F_2[j])=D(F_1[i],\theta)+D(F_1[k],F_2[j])-D(F_1[k],\theta)\\ f\_F(i,j_t) & \text{if } D(F_1[i],F_2[j])=D(\theta,F_2[j])+D(F_1[i],F_2[j_t])-D(\theta,F_2[j_t]),\end{cases}$$

$$x\_T(i,j)=\begin{cases} x\_F(i,j) & \text{if } D(T_1[i],T_2[j]) = D(F_1[i],F_2[j]) + \gamma(t_1[i],t_2[j]) \\ (0,0) & \text{if } D(T_1[i],T_2[j]) = D(T_1[i],\theta) + D(T_1[i_s],T_2[j]) - D(T_1[i_s],\theta) \\ & i_s \neq k \\ x\_T(k,j) & \text{if } D(T_1[i],T_2[j]) = D(T_1[i],\theta) + D(T_1[k],T_2[j]) - D(T_1[k],\theta) \\ x\_T(i,j_t) & \text{if } D(T_1[i],T_2[j]) = D(\theta,T_2[j]) + D(T_1[i],T_2[j_t]) - D(\theta,T_2[j_t]). \end{cases}$$

Lemma 4. *Let $t_1[i]$ be a proper ancestor of $t_1[p(k)]$ and $t_1[i_k]$ be the child of $t_1[i]$ which is an ancestor of $t_1[k]$. Let x be t or f in the following formulae for $x\_F(i,j)$ and $x\_T(i,j)$.*

$$x\_F(i,j)=\begin{cases} (0,0) & \text{if } D(F_1[i],F_2[j]) = E(m_i,n_j) \text{ and } T_1[i_k] \text{ is deleted} \\ x\_T(i_k,j_t) & \text{if } D(F_1[i],F_2[j]) = E(m_i,n_j) \text{ and } T_1[i_k] \text{ is matched to } T_2[j_t] \\ (0,0) & \text{if } D(F_1[i],F_2[j]) = D(F_1[i],\theta)+D(F_1[i_s],F_2[j])-D(F_1[i_s],\theta) \\ & s \neq k \\ x\_F(i_k,j) & \text{if } D(F_1[i],F_2[j]) = D(F_1[i],\theta)+D(F_1[i_k],F_2[j])-D(F_1[i_k],\theta) \\ x\_F(i,j_t) & \text{if } D(F_1[i],F_2[j]) = D(\theta,F_2[j])+D(F_1[i],F_2[j_t])-D(\theta,F_2[j_t]), \end{cases}$$

$$x\_T(i,j)=\begin{cases} x\_F(i,j) & \text{if } D(T_1[i],T_2[j]) = D(F_1[i],F_2[j]) + \gamma(t_1[i],t_2[j]) \\ (0,0) & \text{if } D(T_1[i],T_2[j]) = D(T_1[i],\theta) + D(T_1[i_s],T_2[j]) - D(T_1[i_s],\theta) \\ & s \neq k \\ x\_T(i_k,j) & \text{if } D(T_1[i],T_2[j]) = D(T_1[i],\theta) + D(T_1[i_k],T_2[j]) - D(T_1[i_k],\theta) \\ x\_T(i,j_t) & \text{if } D(T_1[i],T_2[j]) = D(\theta,T_2[j]) + D(T_1[i],T_2[j_t]) - D(\theta,T_2[j_t]). \end{cases}$$

From these formulae, it is clear that we can computer $t\_T(|T_1|,|T_2|)$ and $f\_T(|T_1|,|T_2|)$ with a small modification of Algorithm 2.

We now consider how to use $t\_T(|T_1|,|T_2|) = (t_1,t_2)$ and $f\_T(|T_1|,|T_2|) = (f_1,f_2)$ to decompose the optimal mapping into smaller component mappings.

case 1: $t_1 = t_2 = 0$. In this case, since $T_1[k]$ is deleted, it is not in the optimal mapping for $D(T_1,T_2)$. Therefore the optimal mapping for $D(T_1,T_2)$ would be the optimal mapping for $D(T_1/T_1[k],T_2)$ where $T_1/T_1[k]$ is T_1 with $T_1[k]$ deleted. Note that $|T_1/T_1[k]| < 0.5|T_1|$.

case 2: $t_1 = k_s$ and $t_2 = l$. In this case, $T_1[k]$ is matched with $T_2[l]$ and in this matching, $T_1[k_s]$ is matched with $T_2[l]$ and the rest of $T_1[k]$ is deleted. Therefore the optimal mapping for $D(T_1,T_2)$ has two component mappings. One component is the optimal mapping for $D(T_1/F_1[k],T_2/F_2[l])$ with additional condition that $t_1[k]$ has to match with $t_2[l]$ though (k,l) is not in the optimal mapping of $D(T_1,T_2)$ since $t_1[k]$ is deleted. The other component is the optimal mapping of $D(T_1[k_s],T_2[l]$. Note that $|T_1/F_1[k]| \leq 0.5|T_1|$ and $|T_1[k_s]| \leq 0.5|T_1|$.

Computing $D(T_1/F_1[k],T_2/F_2[l])$ with condition that some of the leaves of the two trees are forced to match would not be more difficult then computing $D(T_1/F_1[k],T_2/F_2[l])$ without any condition. In fact the condition will force the ancestors of these matched leaves of the two trees to match making the computation less expensive.

case 3: $t_1 = k$ and $t_2 = l$, $f_1 = f_2 = 0$. In case 3, case 4 and case 5, $t_1 = k$ and $t_2 = l$. This means that in the optimal mapping for $D(T_1, T_2)$, $t_1[k]$ is matched with $t_2[l]$. Therefore one component mapping (for case 3-5) is the optimal mapping for $D(T_1/F_1[k], T_2/F_2[l])$ with additional condition that $t_1[k]$ matched with $t_2[l]$. Note that $|T_1/T_1[k]| < 0.5|T_1|$.

In this case, since $f_1 = f_2 = 0$ there is no additional component.

case 4: $t_1 = k$ and $t_2 = l$, $f_1 = k_s$ and $f_2 = l$. In this case, one component mapping is the same as that in case 3 and the other component mapping is the optimal mapping for $D(F_1[k_s], F_2[l])$. Note that $|F_1[k_s]| \leq 0.5|T_1|)$.

case 5: $t_1 = k$ and $t_2 = l$, $f_1 = k$ and $f_2 = l$. In this case, one component mapping is the same as that in case 3 and the other component mapping is the optimal mapping for $D(F_1[k], F_2[l])$. Since in this case, $D(F_1[k], F_2[l]) = E(m_k, n_l)$, the optimal mapping for $D(F_1[k], F_2[l])$ can be further decomposed into the optimal mappings of the matching subtree pairs of $E(m_k, n_l)$. Note that $|T_1[k_s]| \leq 0.5|T_1|)$ for $1 \leq s \leq m_k$.

case 6-8: $t_1 = k + 1$ and $t_2 = l$. This means that in the optimal mapping for $D(T_1, T_2)$, $t_1[k]$ is deleted and $F_1[k]$ is matched with $F_2[l]$. Therefore one component mapping (for case 6-8) is the optimal mapping for $D(T_1/F_1[k], T_2/F_2[l])$ with additional condition that $t_2[l]$ matched with a proper ancestor of $t_1[k]$. Note that $|T_1 - T_1[k]| < 0.5|T_1|$.

There is no additional component for case 6. The other component for case 7 is exactly the same as that of case 4. The other components for case 8 are exactly the same as that of case 5.

In all the above cases, with t_1, t_2, f_1, and f_2, we can determine components needed except matching subtree pairs of $E(m_k, n_l)$ for case 5 and case 8. Finding the matching subtree pairs of $E(m_k, n_l)$ can be done in $O(|F_1[k]||F_2[l]|)$ time and $O(\log(|F_1[k]|)|F_2[l]|)$ space. First we find $T_1[k_s]$ such that $\sum_{i=1}^{s-1} |T_1[k_i]| \leq 0.5|F_1[k]|$ and $\sum_{i=1}^{s} |T_1[k_i]| > 0.5|F_1[k]|$. We then determine $T_2[l_t]$, in $O(|F_1[k]||F_2[l]|)$ time and $O(\log(|F_1[k]|)|F_2[l]|)$ space, such that either $T_1[k_s]$ matched with $T_2[l_t]$ in $E(m_k, n_l)$ or $T_1[k_s]$ is deleted, $T_1[k_1], ..., T_1[k_{s-1}]$ are matched with $T_2[l_1], ..., T_2[l_t]$, and $T_1[k_{s+1}], ..., T_1[k_{m_k}]$ are matched with $T_2[l_{t+1}], ..., T_2[l_{n_l}]$ in $E(m_k, n_l)$. In both cases, since $\sum_{i=1}^{s-1} |T_1[k_i]| \leq 0.5|F_1[k]$ and $\sum_{i=s+1}^{m_k} |T_1[k_i]| \leq 0.5|F_1[k]$, we can repeat and get all the subtree matching pairs for $E(m_k, n_l)$ in $O(|F_1[k]||F_2[l]|)$ time and $O(\log(|F_1[k]|)|F_2[l]|)$.

Therefore using $c|T_1||T_2|$ time for some constant c and $O(\log(|T_1|)|T_2|)$ space we can decompose the optimal mapping for $D(T_1, T_2)$ in to some components such that for each component the involved subtree or subforest of T_1 has a size less or equal to $0.5|T_1|$.

In the next step we will use the same algorithm for all the components and the total time is bounded by $c0.5|T_1||T_2|$.

The above analysis means that for our algorithm the space complexity is bounded by $O(\log(|T_1|)|T_2|)$ and the time complexity is bounded by $c|T_1||T_2| + c\frac{1}{2}|T_1||T_2| + c\frac{1}{4}|T_1||T_2| + ... \leq 2c|T_1|T_2| = O(|T_1||T_2|)$.

Theorem 1. *Given two ordered trees T_1 and T_2, the constrained edit distance and the optimal constrained mapping between them can be computed in $O(|T_1||T_2|)$ time and $O(\log(|T_1|)|T_2|)$ space.*

Proof. Immediately from the above analysis.

3 Alignment of trees

The alignment of tree is another measure for comparison of ordered trees [5]. The definition of *alignment of trees* is as follows: Inserting a node u into T means that for some node v (could be a leaf) in T, we make u the parent of the consecutive subsequence of the children of v (if any) and then v the parent of u. We also allow to directly add/insert a node as a child of a leaf in the tree. Given two trees T_1 and T_2, an alignment of the two trees can be obtained by first inserting nodes labeled with spaces into T_1 and T_2 such that the two resulting tree T_1' and T_2' have the same structure, (i.e., they are identical if labels are ignored,) and then overlaying T_1' and T_2'. A score is defined for each pair of labels. The value of an alignment is the total score of all the opposing labels in the alignment. The problem here is to find an alignment with the optimal (maximum or minimum) value.

Theorem 2. *There exists an algorithm for alignment of trees that runs in $O(|T_1||T_2|(deg(T_1) + deg(T_2))^2)$ time and requires $O(\log(|T_1|)|T_2|(deg(T_1)+ deg(T_2))deg(T_1))$ space.*

The basic ideas of the algorithm are similar to that for constrained edit distance. We first use the method in [11] to compute the cost of an optimal alignment. The remaining task is to get the alignment.

Let q be the node in T_1 such that $|T_1[q]| \geq 0.5|T_1|$ and for any child q_i of node q, $|T_1[q_i]| < 0.5|T_1|$. q is refereed to as a *cutting* point in T_1. By definition, there always exists a cutting point for any T_1.

By the definition of the alignment, node q is either aligned with a node $T_2[j]$ or aligned with an inserted node (labeled with empty). We can modify the algorithm for computing the cost of an optimal alignment so that when the cost of an optimal alignment is computed, we immediately know the configuration of node q in the alignment. Thus, we can decompose the alignment of the whole two tree T_1 and T_2 into two parts (1) the alignment of subtree $T_1[q]$ with a subtree or subforest of T_2, and (2) the alignment of the remaining parts of T_1 and T_2.

Repeating the process, we can get the alignment. We can show that the time required is still $O(|T_1||T_2|(deg(T_1) + deg(T_2))^2)$.

4 Conclusion

We have presented space efficient algorithms for the computation of constrained edit distance and tree alignment for ordered rooted trees. The techniques can also

be applied to other tree comparison problems such as degree-one and degree-two edit distance between ordered trees.

Acknowledgements

This work is supported by a grant from City University of Hong Kong (Project No. 7001696) and the Natural Sciences and Engineering Research Council of Canada under Grant No. OGP0046373.

References

1. P. Bille, "A survey on tree edit distance and related problems", *Theoretical Computer Science*, no. 337, pp. 217-239, 2005.
2. Y. C. Cheng and S. Y. Lu, "Waveform correlation by tree matching", *IEEE Trans. PAMI*, vol. 7, pp.299-305, 1985
3. S. Dulucq and H. Touzet, "Decomposition algorithm for tree editing distance", *Journal of Discrete Algorithms*, 2004.
4. D.S. Hirschberg, "A linear space algorithm for computing maximal common subsequences", *Communications of the ACM*, no. 18, pp. 341-343, 1975.
5. T. Jiang, L. Wang and K. Zhang, 'Alignment of trees - an alternative to tree edit', *Theoretical Computer Science* vol. 143, no. 1, pp. 137-148, 1995.
6. P. Klein, "Computing the edit-distance between unrooted ordered trees", *Proceedings of 6th European Symposium on Algorithms*, pp. 91-102, 1998.
7. T. Richter, "A new measure of the distance between ordered trees and its applications", *Technical Report 85166-cs*, Department of Computer Science, University of Bonn, 1997.
8. B. Shapiro and K. Zhang, 'Comparing multiple RNA secondary structures using tree comparisons', *Comput. Appl. Biosci* vol. 6, no.4, pp.309-318, 1990
9. S.M. Selkow, "The tree-to-tree editing problem", *Information Processing Letters*, vol. 6, pp.184-186, 1977.
10. K.C. Tai, "The tree-to-tree correction problem", *J. ACM*, vol. 26, pp.422-433, 1979.
11. L. Wang and J. Zhao, "Parametric alignment of ordered trees", *Bioinformatics*, vol. 19, pp. 2237-2245, 2003.
12. K. Zhang, "Algorithms for the constrained editing distance between ordered labeled trees and related problems". *Pattern Recognition*, vol.28, no. 3, pp. 463-474, 1995.
13. K. Zhang, "Efficient parallel algorithms for tree editing problems", *Proceedings of the Seventh Symposium on Combinatorial Pattern Matching*, Laguna Beach, California, June 1996, *Springer-Verlag's Lecture Notes in Computer Science 1075*, pp 361-372.
14. 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
15. K. Zhang, L. Wang, and B. Ma, 'Computing similarity between RNA structures', *Proceedings of the Tenth Symposium on Combinatorial Pattern Matching. LNCS 1645*, pp. 281-293, 1999.
16. K. Zhang, J.T.L. Wang and D. Shasha, 'On the editing distance between undirected acyclic graphs', *International Journal of Foundations of Computer Science*, vol. 7, no. 1, pp. 43-57, 1996.