Abstract. This paper considers the problem of computing the editing distance between unordered, labeled trees. We give efficient polynomial-time algorithms for the case when one tree is a string or has a bounded number of leaves. By contrast, we show that the problem is \(NP\)-complete even for binary trees having a label alphabet of size two.

**keywords**: Computational Complexity, Unordered trees, \(NP\)-completeness.

1 Introduction

Unordered labeled trees are trees whose nodes are labeled and in which only ancestor relationships are significant (the left-to-right order among siblings is not significant). Such trees arise naturally in genealogical studies, for example, the genetic study of the tracking of diseases. For many such applications, it would be useful to compare unordered labeled trees by some meaningful distance metric. The editing distance metric, used with some success for ordered labeled trees [1], is a natural such metric. This paper presents algorithms and complexity results for a wide spectrum of assumptions concerning this problem.

2 Definitions

The definitions below are similar in style to those for ordered labeled trees in [5]. We will omit the proofs for Lemma 1 and Theorem 1 since they are similar to those in [5] and can be found in [6].
2.1 Editing operations and editing distance between unordered labeled trees

We consider three kinds of operations. Changing a node \( n \) means changing the label on \( n \). Deleting a node \( n \) means making the children of \( n \) become the children of the parent of \( n \) and then removing \( n \). Inserting is the complement of deleting. This means that inserting \( n \) as the child of \( m \) will make \( n \) the parent of a subset (as opposed to a consecutive subsequence [5]) of the current children of \( m \).

We represent an edit operation as \( a \rightarrow b \), where \( a \) is either \( \Lambda \) or a label of a node in tree \( T_1 \) and \( b \) is either \( \Lambda \) or a label of a node in tree \( T_2 \). We call \( a \rightarrow b \) a change operation if \( a \neq \Lambda \) and \( b \neq \Lambda \); a delete operation if \( b = \Lambda \); and an insert operation if \( a = \Lambda \).

Let \( S \) be a sequence \( s_1, \ldots, s_k \) of edit operations. An \( S \)-derivation from \( A \) to \( B \) is a sequence of trees \( A_0, \ldots, A_k \) such that \( A = A_0, B = A_k, \) and \( A_{i-1} \rightarrow A_i \) via \( s_i \) for \( 1 \leq i \leq k \). Let \( \gamma \) be a cost function which assigns to each edit operation \( a \rightarrow b \) a nonnegative real number \( \gamma(a \rightarrow b) \).

We constrain \( \gamma \) to be a distance metric. That is, i) \( \gamma(a \rightarrow b) \geq 0, \gamma(a \rightarrow a) = 0 \); ii) \( \gamma(a \rightarrow b) = \gamma(b \rightarrow a) \); and iii) \( \gamma(a \rightarrow c) \leq \gamma(a \rightarrow b) + \gamma(b \rightarrow c) \).

We extend \( \gamma \) to the editing operations sequence \( S \) by letting \( \gamma(S) = \sum_{i=1}^{\left| S \right|} \gamma(s_i) \). Formally the distance between \( T_1 \) and \( T_2 \) is defined as:

\[
\delta(T_1, T_2) = \min \{ \gamma(S) \mid S \text{ is an edit operation sequence taking } T_1 \text{ to } T_2 \}.
\]

\( \delta \) is also a distance metric according to the definition of \( \gamma \).

2.2 Mappings

The edit operations give rise to a mapping which is a graphical specification of what edit operations apply to each node in the two unordered labeled trees.

Suppose that we have a numbering for each tree. Let \( T[i] \) be the \( i \)th node of tree \( T \) in the given numbering. Formally we define a triple \((M, T_1, T_2)\) to be a mapping from \( T_1 \) to \( T_2 \), where \( M \) is any set of pair 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 an ancestor of \( T_1[i_2] \) iff \( T_2[j_1] \) is an ancestor of \( T_2[j_2] \) (ancestor order preserved)

We will use \( M \) instead of \((M, T_1, T_2)\) if there is no confusion. Let \( M \) be a mapping from \( T_1 \) to \( T_2 \). Let \( I \) and \( J \) be the sets of nodes, in \( T_1 \) and \( T_2 \), not in \( M \). Then we can define the cost of \( M \):

\[
\gamma(M) = \sum_{(i,j) \in M} \gamma(T_1[i] \rightarrow T_2[j]) + \sum_{i \in I} \gamma(T_1[i] \rightarrow \Lambda) + \sum_{j \in J} \gamma(\Lambda \rightarrow T_2[j])
\]
The relation between a mapping and a sequence of editing operations is as follows:

**Lemma 1** Given $S$, a sequence $s_1, \ldots, s_k$ of edit operations from $T_1$ to $T_2$, there exists a mapping $M$ from $T_1$ to $T_2$ such that $\gamma(M) \leq \gamma(S)$. Conversely, for any mapping $M$, there exists a sequence of editing operations such that $\gamma(S) = \gamma(M)$.

**Theorem 1** $\delta(T_1, T_2) = \min_M \{ \gamma(M) \mid M$ is a mapping from $T_1$ to $T_2 \}$

3 Algorithmic results

In this section, we first study the problem of finding the editing distance (a minimum cost mapping) between a string and an unordered tree; and then between a tree with $k$ leaves and an unordered tree.

For the purpose of numbering the nodes in an unordered tree, we can take any ordering of the tree. That is, we first fix an arbitrary order among the children of each interior node of the unordered tree, yielding an ordered tree $T$. We will then use a postorder numbering of the nodes in $T$. In the intermediate steps of our algorithm, we may have to consider unordered forests. We note that the definition of mappings for unordered forests is the same as for unordered trees.

Define the out-degree of a node in a given tree to be the number of children of the node. We then define the degree of a tree to be the maximum out-degree of any of its nodes. A string can be considered to be a degree one tree.

3.1 Notation

Let $T[i]$ be the $i$th node in the tree according to the left-to-right postorder numbering of tree $T$. Let $l(i)$ be the number of the leftmost leaf descendant of the subtree rooted at $T[i]$. When $T[i]$ is a leaf, $l(i) = i$. $T[l(i)..i]$ is the unordered forest resulting from removing $T[i]$ from the subtree rooted at $T[i]$. The distance between $T_1[l(i)..i]$ and $T_2[l(j)..j]$ is denoted by $\text{dist}(i, j)$. If $i$ or $j$ is zero it represents the empty tree. We use $d(i, 0)$ to represent $\gamma(S[i] \rightarrow \Lambda)$, $d(0, j)$ to represent $\gamma(\Lambda \rightarrow T[j])$, and $d(i, j)$ to represent $\gamma(S[i] \rightarrow T[j])$.

3.2 Algorithm for the Editing Distance between a String and a Tree

The tree will be denoted by $T$. We will refer to the string as $S$ and assume that it represents the sequence $a_1 a_2 \ldots a_n$. In the postorder representation, $S[1]$ represents $a_n$, $S[n]$ represents $a_1$, and in general $S[i]$ represents $a_{n-i+1}$. That is, the postorder numbering reverses the order of the string. This implies that $S[l(i)..i] = S[1..i]$ represents $a_n a_{n-1} \ldots a_{n-i+1}$.
With these preliminaries out of the way, we can now present a few lemmas and then the algorithm.

**Lemma 2** The relative order among siblings of $T$ does not influence the editing distance between $S$ and $T$.

**Proof:** By the ancestor-descendant constraint on mappings, we know that, in the best mapping between $S$ and $T$, $S$ can be mapped only to a path of $T$ from the root to a leaf. All other nodes of the tree must be inserted. Since leaf-to-root paths are preserved by different orderings of $T$, the order of $T$ does not matter. □

This lemma implies that we can solve this problem by the algorithm of Zhang and Shasha [5] for ordered trees by fixing an arbitrary order for $T$. The time complexity will be $O(|S| \cdot |T| \cdot \text{depth}(T))$.

In the following we show that this can be improved to $O(|S| \cdot |T|)$.

Assume that $T[j]$ has $k \geq 0$ children, $T[j_1], T[j_2], ... T[j_k]$.

**Lemma 3**

1) $\text{dist}(0,0) = 0$

2) $\text{dist}(i,0) = d(i,0) + \text{dist}(i-1,0)$

3) $\text{dist}(0,j) = d(0,j) + \sum_{i=1}^{k} \text{dist}(0,t_i)$

**Proof:** Statements (1) and (2) are obvious. As for (3), insertion of $\text{subtree}(j)$ is equivalent to the insertions of $\text{subtree}(j_1)$, ... $\text{subtree}(j_k)$, and then the insertion of root $T[j]$. □

**Lemma 4** If $T[j]$ has no children, then

$$\text{dist}(i,j) = \min \left\{ \begin{array}{ll} \text{dist}(i,0) + d(0,j) \\
\text{dist}(i-1,j) + d(i,0) \\
\text{dist}(i-1,0) + d(i,j) \end{array} \right. $$

**Proof:** Consider the best mapping between $S[1..i]$ and $T[j]$. We have three cases.

1. $S[i]$ is not in the best mapping, then $\text{dist}(i,j) = \text{dist}(i-1,j) + d(i,0)$.

2. $T[j]$ is not in the best mapping. Since $T[j]$ has no child, $\text{dist}(i,j) = \text{dist}(i,0) + d(0,j)$.

3. Both $S[i]$ and $T[j]$ are in the best mapping, then by the definition of mapping $S[i]$ and $T[j]$ must mapped to each other. Therefore $\text{dist}(i,j) = \text{dist}(i-1,0) + d(i,j)$.

Lemma follows. □
If $S[i]$ maps to $T[j]$, and $S[1..i-1]$ maps to subtree rooted at $T[j_i]$. The cost of such a mapping includes the cost of inserting all subtrees of $T[j]$ aside from $T[j_i]$.

Lemma 5 If $T[j]$ has $k \geq 1$ children, then

\[
\text{dist}(i, j) = \min \left\{ \begin{array}{l}
\text{dist}(i-1, j) + d(i, 0) \\
\text{dist}(0, j) + \min_{t=1}^{k} \{\text{dist}(i, j_t) - \text{dist}(0, j_t)\} \\
\text{dist}(i, j) + \text{dist}(0, j) - d(0, j) + \min_{t=1}^{k} \{\text{dist}(i - 1, j_t) - \text{dist}(0, j_t)\} 
\end{array} \right. 
\]

Proof: Consider the best mapping between $S[1..i]$ and $T[l(j)..j]$. Again we have three cases.

1. $S[i]$ is not in the mapping, then $\text{dist}(i, j) = \text{dist}(i - 1, j) + d(i, 0)$.

2. $T[j]$ is not in the mapping, then assume that $j_1, j_2, \ldots j_k$ are the children of $T[j]$. The best mapping must map $S[1..i]$ to one of the subtrees rooted at $T[j]$'s children. Suppose that $S[1..i]$ is mapped to the subtree rooted at $T[j_t]$. Then the distance is the sum of the cost of mapping $S[1..i]$ to $T[l(j_t)..j_t]$, the cost of mapping null to $T[j]$ and the costs of inserting all the other subtrees of $T[j]$. Therefore $\text{dist}(i, j) = \text{dist}(i, j_t) + d(0, j) + \sum_{t=1}^{k} \text{dist}(0, j_t) + \sum_{t=t+1}^{k} \text{dist}(0, j_t)$

   By lemma 3 $\text{dist}(0, j) = d(0, j) + \sum_{t=1}^{k} \text{dist}(0, j_t)$. So $\text{dist}(i, j) = \text{dist}(i, j_t) + \text{dist}(0, j) - \text{dist}(0, j_t)$. Hence we have following: $\text{dist}(i, j) = \text{dist}(0, j) + \min_{t=1}^{k} \{\text{dist}(i, j_t) - \text{dist}(0, j_t)\}$.

3. $S[i]$ and $T[j]$ are both in the best mapping then they must map to each other, by the ancestor-descendant constraints on mappings. In this case $S[1..i - 1]$ must mapped to a subtree rooted at a child of $T[j]$. Suppose that it is mapped to subtree $T[j_t]$, then $\text{dist}(i, j) = d(i, j) + \sum_{t=1}^{i-1} \text{dist}(0, j_t) + \sum_{t=t+1}^{k} \text{dist}(0, j_t) = d(i, j) + \sum_{t=1}^{i-1} \text{dist}(0, j_t) + \sum_{t=t+1}^{k} \text{dist}(0, j_t) - \text{dist}(0, j_t) - d(0, j)$. Therefore we have following: $\text{dist}(i, j) = d(i, j) + \text{dist}(0, j) - d(0, j) + \min_{t=1}^{k} \{\text{dist}(i - 1, j_t) - \text{dist}(0, j_t)\}$

Lemma follows by combining the above three formulas. \(\square\)

We are now ready to give our algorithm.

$\text{dist}(0, 0) = 0$;
For \( i = 1 \) to \( n \)
\[
\text{dist}(i, 0) = d(i - 1, 0) + d(i, 0);
\]
For \( j = 1 \) to \( m \) (* \( T[j] \) has \( k \) \( \geq 0 \) children \( j_1, ..., j_k \) *)
\[
\text{dist}(0, j) = d(0, j) + \sum_{t=1}^{k} \text{dist}(0, j_t)
\]
For \( i = 1 \) to \( n \)
For \( j = 1 \) to \( m \)
\[
\text{if } l(j) = j \text{ then (* } T[j] \text{ has no child *)}
\]
\[
\text{dist}(i, j) = \min \left\{ \begin{array}{l}
\text{dist}(i, 0) + d(0, j), \\
\text{dist}(i - 1, j) + d(i, 0), \\
\text{dist}(i - 1, 0) + d(i, j)
\end{array} \right. 
\]
\[
\text{else (* } T[j] \text{ has } k \geq 1 \text{ children } j_1, ..., j_k *)
\]
\[
\text{dist}(i, j) = \min \left\{ \begin{array}{l}
\text{dist}(i - 1, j) + d(i, 0), \\
\text{dist}(0, j) + \min_{t=1}^{k}\{\text{dist}(i, j_t) - \text{dist}(0, j_t)\}, \\
\text{dist}(0, j) + \text{dist}(i, j) - \text{dist}(0, j) + \min_{t=1}^{k}\{\text{dist}(i - 1, j_t) - \text{dist}(0, j_t)\}
\end{array} \right. 
\]

**Theorem 2** The algorithm correctly calculates the minimum distance between \( S \) and \( T \) in \( O(|S| \cdot |T|) \) time.

**Proof:** Correctness is immediate from the preceding lemmas, Lemma 3 to 5.

Consider the time complexity. To compute \( \text{dist}(i, j) \), the time is bounded by \( O(\text{degree}(T[j])) \).

Therefore the time of the algorithm is bounded by \( \sum_{i=1}^{|S|} \sum_{j=1}^{|T|} (\text{degree}(T[j]) + 1) = (\sum_{i=1}^{|S|} 1) \times (\sum_{j=1}^{|T|} O(\text{degree}(T[j]))) = O(|S| \cdot |T|) \) \( \square \)

Strings are trees with a single leaf. One might wonder how the complexity grows when comparing an unordered tree with \( k \) leaves to a general unordered tree. Let us first consider the case where \( k = 2 \). That is there are two leaves in \( T_1 \) and an arbitrary tree \( T_2 \). Let \( i_0 \) be the node in \( T_1 \) with two children. Let its two children be \( i_1 \) and \( i_2 \). We can use the string-to-tree algorithm to compute their distance to tree \( T_2 \). Let us consider what happens when we compute distance \( T_1[l(i_0) .. i_0] \) to \( T_2[l(j) .. j] \).

As usual, we have three cases:

1. Node \( i_0 \) is not in the best mapping. Let us consider \( T[j] \),

   (a) \( T[j] \) is in the mapping, then it can map to either \( T[i_1] \) or \( T[i_2] \). This yields \( \text{dist}(i_0, j) = d(i_0, 0) + \min\{\text{dist}(i_1, j) + \text{dist}(i_2, 0), \text{dist}(i_2, j) + \text{dist}(i_1, 0)\} = \text{dist}(i_0, 0) + \min\{\text{dist}(i_1, j) - \text{dist}(i_1, 0), \text{dist}(i_2, j) - \text{dist}(i_2, 0)\} \).

   (b) \( T[j] \) is not in the best mapping. This yields \( \text{dist}(i_0, j) = \text{dist}(l(i_0) .. i_2, l(j) .. j - 1) + d(i_0, 0) + d(0, j) \). (This turns out to be larger than case 3, so we don’t have to consider it.)
2. $T[j]$ is not in the best mapping. The reasoning is similar to case 1, yielding $dist(i_0, j) = dist(0, j) + \min_{t=1}^{k} \{dist(i_{0}, j_{t}) - dist(0, j_{t})\}$.

3. $T_1[i_0]$ and $T_2[j]$ are both in the best mapping. $dist(i_0, j) = dist(l(i_0)...i_2, l(j)...j - 1) + d(i_0, j)$. Unfortunately, the first term is unknown since we must consider the mapping of both strings to $T_2$. There are two subcases pertaining to the calculation of that term.

(a) Only one string is in the best mapping. $dist(l(i_0)...i_2, l(j)...j - 1) = \min\{dist(i_1, l(j)...j - 1) + dist(i_2, 0), dist(i_2, j - 1) + dist(i_1, 0)\}$.

(b) Both strings are in the best mapping.
- The two strings map to one subtree. $dist(l(i_0)...i_2, l(j)...j - 1) = dist(0, j) - d(0, j) + \min_{t=1}^{k} \{dist(l(i_0)...i_2, j_{t}) - dist(0, j_{t})\}$.
- Each string maps to a different tree, so we use bipartite matching to find the best mapping.

From the above discussion, if we know all $dist(l(i_0)...i_2, l(j)...j)$, where $1 \leq j \leq T_2$, then we can proceed as in the string to tree algorithm. For fixed $j$, to compute $dist(l(i_0)...i_2, l(j)...j)$, we need $2 + degree(T_2[j]) + 2! * 3 * (degree(T_2[j]) + 1) = O(2! * 3 * degree(T_2[j]) + 1)$. Therefore the total time for $dist(l(i_0)...i_2, l(j)...j)$, where $1 \leq j \leq T_2$, is $\sum_{j=1}^{T_2} 2! * 3 * degree(T_2[j]) = 2! * 3 * T_2$. Hence the time complexity of a two leaf tree to a general tree is $O(T_1 * T_2 + 2! * 3 * T_2)$.

If $T_1$ has $k$ leaves, we have to consider the case where we have a subforest in that tree. The subforest consists of trees each of which has less than $k$ leaves. There are at least $2^k$ such forests.

Extending the matching techniques to groups of forests, we use bipartite matching to determine which group in $T_1$ maps to which different tree in $T_2$. The resulting complexity is $O(T_1 * T_2 + k! * 2^k * (k^3 + degree(T_2) * T_2))$.

4 \textbf{NP-completeness}

In this section we show that the problem of computing the editing distance between unordered labeled fixed-degree $k \geq 2$ tree is \textit{NP}-complete. For details of this section, please refer to [4] and [6].

We will reduce Exact Cover by 3-Sets to computing unordered labeled tree editing distance.

\textbf{Exact Cover by 3-Sets}

\textbf{INSTANCE}: A finite set $S$ with $|S| = 3q$ and a collection $T$ of 3-element subsets of $S$.
QUESTION: Does $T$ contain an exact cover for $S$, that is, a subcollection $T' \subseteq T$ such that every element of $S$ occurs in exactly one member of $T'$?

Given an instance of the exact 3-cover problem, let the set $S = \{ s_1, s_2, \ldots, s_m \}$, where $m = 3q$. Let $T = \{ C_1, C_2, \ldots, C_n \}$. Here each $C_i = \{ t_{i1}, t_{i2}, t_{i3} \}$, where $t_{ij} \in S$. Without loss of generality, we assume that $n > q$.

We construct two trees as in Figure 2. The top portion of each tree is represented by a triangle. Below the triangle are $n$ subtrees for each tree. We use the triangle at the top to connect the $n$ subtrees into a tree. It has ceiling of $\log_3(n)$ levels. It plays no further role in the construction. The total number of nodes in the triangle is less than $n$. We assume that all the nodes in the triangle have the same label, which may be arbitrary. We call a sequence of nodes with label $\phi$, $\lambda$, $s_i$ or $t_{ij}$ a segment. The length of each segment is $f_3(n) = 4n$.

Inspection of the figure shows that $T_1$ and $T_2$ can be constructed from an instance of exact
cover by 3-sets in polynomial time.

In the following we will assume that each edit operation has unit cost, i.e. \( \gamma(a \rightarrow b) = 1 \) for all \( a, b \) such that \( a \neq b \), and that \( T_1 \) and \( T_2 \) are trees as in Figure 2.

**Lemma 6** Let \( M \) be a mapping between \( T_1 \) and \( T_2 \), if there are \( d \geq 0 \) nodes of \( T_2 \) not in mapping \( M \), then \( \gamma(M)(T_1, T_2) \geq 3(n - k)f_3(n) + k(f_3(n) - 1) + d \).

**Lemma 7** If there is an exact cover by 3-set, then \( \delta(T_1, T_2) < 3(n - k)f_3(n) + k(f_3(n) - 1) + f_3(n) \).

**Lemma 8** If \( \delta(T_1, T_2) < 3(n - k)f_3(n) + k(f_3(n) - 1) + f_3(n) \), then there is an exact cover by 3-set.

**Theorem 3** Computing the editing distance between unordered labeled degree three trees is \( NP \)-complete.

For the editing distance between degree \( k > 3 \) trees, we can reduce the Exact Cover by \( k \)-sets to it. However for degree two trees we can not use the Exact cover by 2-sets problem since it is in P. We can still use exact cover by 3-sets with a minor modification of the trees in Figure 2. For any node with out-degree 3, let the node be \( a \) and its children be \( b, c, \) and \( d \), that is \( (a(b)(c)(d)) \). We change it to two out-degree 2 nodes, namely \( (a(b)(a(c)(d))) \). Now if we replace \( f_3(n) = 4n \) in Lemma 6 to Lemma L5 by \( f_2(n) = 6n \) We can prove the same results as in Lemma 6 to Lemma L5 and Theorem 3. For the problem of computing the editing distance between a degree \( d_1 \geq 2 \) tree and a degree \( d_2 \geq 2 \) tree, we can similarly prove that it is \( NP \) complete [6].
The previous results are based on the assumption that the size of the alphabet of the labels are not bounded. However in applications the size of the alphabet is always a constant. When we fix the size of alphabet, we can use different strings to encode different symbols. Therefore even if the size of alphabet is two all the previous results are still true [6].

5 Conclusion

We have presented the following results concerning the calculation of the editing distance between unordered trees:

- An algorithm to compare a string \( S \) with an unordered tree \( T \) in \( O(|S| \cdot |T|) \) time.

- The essential details of an algorithm to compare an unordered tree \( T_1 \) having \( k \) leaves with a general unordered tree \( T_2 \) in \( O(T_1 \cdot T_2 + k! \cdot 2^k \cdot (k^3 + \text{degree}(T_2)^2) \cdot T_2) \) time.

- A construction and proof sketch to show that the problem is \( NP \)-complete even when the two trees are binary and the alphabet is of size two.

References


