Title	Approximating Tree Edit Distance through String Edit Distance
Author(s)	Akutsu, Tatsuya; Fukagawa, Daiji; Takasu, Atsuhiro
Citation	Algorithmica (2010), 57(2): 325-348
Issue Date	2010-06
URL	http://hdl.handle.net/2433/113886
Right	The original publication is available at www.springerlink.com
Type	Journal Article
Textversion	author

Approximating Tree Edit Distance through String Edit Distance*

Tatsuya Akutsu†
Bioinformatics Center, Institute for Chemical Research, Kyoto University,
Uji, Kyoto 611-0011, Japan.
e-mail: takutsu@kuicr.kyoto-u.ac.jp

Daiji Fukagawa Atsuhiro Takasu
National Institute of Informatics,
Chiyoda-ku, Tokyo 101-8430, Japan.
e-mail: {daiji,takasu}@nii.ac.jp

Abstract

We present an algorithm to approximate edit distance between two ordered and rooted trees of bounded degree. In this algorithm, each input tree is transformed into a string by computing the Euler string, where labels of some edges in the input trees are modified so that structures of small subtrees are reflected to the labels. We show that the edit distance between trees is at least 1/6 and at most $O(n^{3/4})$ of the edit distance between the transformed strings, where n is the maximum size of two input trees and we assume unit cost edit operations for both trees and strings. The algorithm works in $O(n^2)$ time since computation of edit distance and reconstruction of tree mapping from string alignment takes $O(n^2)$ time though transformation itself can be done in $O(n)$ time.

Key Words. tree edit distance, string matching, approximation algorithms, embedding, Euler string

*A preliminary version of this paper appeared in the *Proceedings of the 17th Annual International Symposium on Algorithms and Computation* (*ISAAC* '06), Lecture Notes in Computer Science, 4288, pp. 90-99, 2006. This work is partially supported by Grants-in-Aid on Scientific Research on Priority Areas "Cyber Infrastructure for the Information-explosion Era" and "Systems Genomics", and Grant-in-Aid #16300092 from MEXT, Japan.

†Corresponding author. TEL: +81-774-38-3015, FAX: +81-774-38-3022

1 Introduction

Recently, comparison of tree-structured data is becoming important in several diverse areas such as computational biology, XML databases and image analysis [4, 12, 20]. Though various measures have been proposed for comparison of trees [4], the *edit distance* between rooted and ordered trees is well-studied and widely-used [14, 18, 19, 21]. This tree edit distance is a generalization of the edit distance for two strings [2, 17], which is also well-studied and widely-used for measuring the similarity between two strings. In this paper, we use *tree edit distance* and *string edit distance* to denote the distance between rooted and ordered trees and the distance between strings, respectively.

It is well-known that the string edit distance can be computed in $O(n^2)$ time by a simple dynamic programming algorithm, where n is the maximum length of input strings. Recently, extensive studies have been done on efficient (quasi linear time) approximation and low distortion embedding of string edit distances [2, 3, 13, 15, 17].

For the tree edit distance problem, Tai [18] first developed a polynomial time algorithm, from which several improvements followed [5, 9, 14, 21]. Among these, a recent algorithm by Demaine et al. [9] is the fastest in the worst case and works in $O(n^3)$ time where n is the maximum size of input trees. They also proved an $\Omega(n^3)$ lower bound for the class of decomposition strategy algorithms. Therefore, it is quite difficult to develop an $o(n^3)$ time exact algorithm for the tree edit distance problem.

Garofalakis and Kumar developed an algorithm for efficient embedding of rooted and ordered trees [10]. Their algorithm provides an approximate tree edit distance with a guaranteed $O(\log^2 \log^* n)$ factor in $O(n \log^* n)$ time, where $\log^* n$ denotes the number of log applications required to reduce n to 1 or less. *However, the distance considered there is not the same as the tree edit distance: move operations are allowed in their distance.* Several practical algorithms have been developed for efficient computation of lower bounds of tree edit distances [12, 20], but these algorithms do not guarantee upper bounds of tree edit distances. Therefore, it is required to develop algorithms for efficient approximation and/or low distortion embedding for trees in terms of the original definition of the tree edit distance. It should be noted that for the case of strings, an efficient approximation/embedding algorithm was first proposed for edit distance with block copies, block uncopies and block moves [7, 16], which was soon modified to take care of string edit

distance with block moves only [6], and then extensive studies followed for edit distance without moves [2, 3, 13, 15, 17].

In order to approximate the tree edit distance, we studied a relation between the tree edit distance and the sting edit distance for the *Euler strings* [1]. It was shown that the tree edit distance is at least half and at most $2h + 1$ of the edit distance for the Euler strings, where h is the minimum height of two trees. This result gives good approximation if the heights of input trees are low. However, it does not guarantee any upper bounds of tree edit distances if the heights of input trees are $O(n)$. In this paper, we improve this result by modifying the Euler string. Modification is done by changing labels of some edges in the trees so that structures of small subtrees are reflected to the labels. Though the modification is slight, a novel idea is introduced and much more involved analysis is performed. We show that the unit cost edit distance between trees is at least $1/6$ and at most $O(n^{3/4})$ of the unit cost edit distance between the modified Euler strings, where we assume that the maximum degree of trees is bounded by a constant. This result leads to the first $O(n^{3-\epsilon})$ time algorithm for computing the unit cost tree edit distance with a guaranteed approximation ratio (for bounded degree trees). Though this result is not practical, it would stimulate further developments. It should be noted that the current best approximation ratio within near linear time algorithms for string edit distance is around $O(n^{1/3})$ [3] even though extensive studies have been done in recent years. Though we consider the *unit cost* edit distances in this paper, the result can be extended as in [1] for more general distances for which the ratio of the maximum cost of an edit operation to the minimum cost of an edit operation is bounded by a constant for both strings and trees.

2 String Edit Distance and Tree Edit Distance

Here we briefly review the string edit distance and the tree edit distance. We consider strings over a finite or infinite alphabet Σ_S. For string s and integer i, $s[i]$ denotes the ith character of s, $s[i \dots j]$ denotes $s[i] \dots s[j]$, and $|s|$ denotes the length of s. We may use $s[i]$ to denote both the character itself and the position. An *edit operation on a string* s is either a *deletion*, an *insertion*, or a *substitution* of a character of s. The *edit distance between two strings* s_1 and s_2 is defined as the minimum number of operations

to transform s_1 into s_2, where only unit cost operations are considered in this paper. We use $ED_S(s_1, s_2)$ to denote the edit distance between s_1 and s_2.

We also define an *alignment* between two strings. An alignment between two strings s_1 and s_2 is obtained by inserting *gap symbols* (denoted by '-' where '-' $\notin \Sigma_S$) into or at either end of s_1 and s_2 such that the resulting strings s'_1 and s'_2 are of the same length l, where it is not allowed for each $i = 1, \ldots, l$ that both $s'_1[i]$ and $s'_2[i]$ are gap symbols. The *cost* of alignment is given by $cost(s'_1, s'_2) = \sum_{i=1}^{l} f(s'_1[i], s'_2[i])$, where $f(x, y) = 0$ if $x = y \neq$ '-', otherwise $f(x, y) = 1$. Then, an optimal alignment is an alignment with the minimum cost. It is straight-forward to see that the cost of an optimal alignment is equal to the edit distance. For example, consider strings s_1 ="TCGTGCAT" and s_2="CGATCCT". Then, the following is an optimal alignment.

```
T   C   G   -   T   G   C   A   T
-   C   G   A   T   C   C   -   T
(a)         (b)     (c)     (d)
```

In this case, (a) and (d) correspond to deletions, (b) corresponds to an insertion, and (c) corresponds to a substitution. Thus, we have $ED_S(s_1, s_2) = 4$.

Next, we define edit distance between trees (for the details, see [4]), where we only consider the unit cost case. Let T be a rooted ordered tree, where "ordered" means that a left-to-right order among siblings is given in T. Moreover, we assume that each node v has a label $label(v)$ from a finite alphabet Σ_T. $|T|$ denotes the size (the number of nodes) of T. An *edit operation on a tree* T is either a deletion, an insertion, or a substitution (see also Fig. 1):

Deletion: Delete a non-root node v in T with parent u, making the children of v become the children of u. The children are inserted in the place of v as a subsequence in the left-to-right order of the children of u.

Insertion: Complement of delete. Insert a node v as a child of u in T making v the parent of a consecutive subsequence of the children of v'.

Substitution: Change the label of a node v in T.

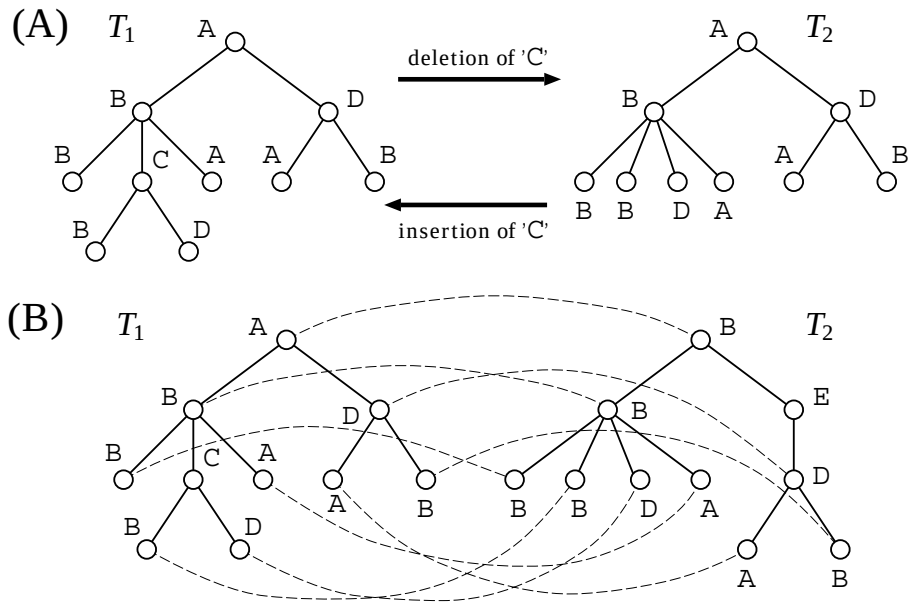

Figure 1: (A) Insertion and deletion operations. (B) T_2 is obtained by deletion of a node with label 'C', insertion of a node with label 'E' and substitution for the root node. A mapping corresponding to this edit sequence is also shown by dashed curves.

The *edit distance between two trees* T_1 and T_2 is defined as the minimum number of operations to transform T_1 into T_2. We use $ED_T(T_1, T_2)$ to denote the edit distance between T_1 and T_2.

It is known that there exists a close relationship between the edit distance and the *ordered edit distance mapping* (or just a *mapping*) [4]. $M \subseteq V(T_1) \times V(T_2)$ is called a mapping if the following conditions are satisfied for any pair (v_1, w_1), $(v_2, w_2) \in M$: (i) $v_1 = v_2$ iff. $w_1 = w_2$, (ii) v_1 is an ancestor of v_2 iff. w_1 is an ancestor of w_2, (iii) v_1 is to the left of v_2 iff. w_1 is to the left of w_2. Let $ID(M)$ be the number of pairs having identical labels in M. It is well-known that the mapping M maximizing $ID(M)$ corresponds to the edit distance, for which $ED_T(T_1, T_2) = |T_1| + |T_2| - |M| - ID(M)$ holds.

3 Euler String

Our transformation from a tree to a string is based on the *Euler string* [14], which is obtained by traversing a tree using the Euler tour. In this section, we review the Euler string and our previous result on the Euler string [1].

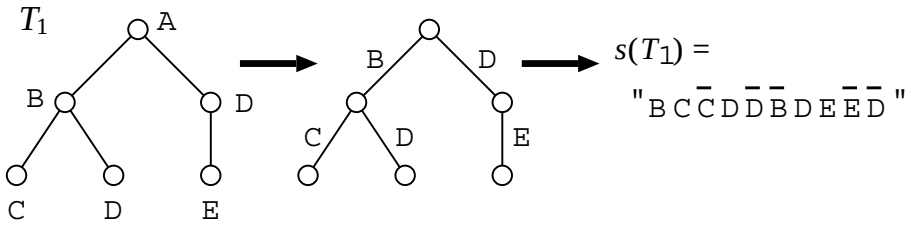

Figure 2: Construction of an Euler string.

For simplicity, we treat each tree T as an edge labeled tree: the label of each non-root node v in the original tree is assigned to the edge $\{u, v\}$ where u is the parent of v. It should be noted that information on the label on the root is lost in this case. But, it is not a problem because the roots are not deleted or inserted. In what follows, we assume that the roots of two input trees have identical labels (otherwise, we just need to add 1 to the distance).

The depth-first search traversal of T (i.e., visiting children of each node according to their left-to-right order) defines an Euler tour of a tree T. That is, the depth-fist search gives an Euler path beginning from the root and ending at the root where each edge $\{w, v\}$ is traversed twice in the opposite directions. We use $EE(T)$ to denote the set of directed edges in the Euler tour of T. Let $\Sigma_S = \{a, \overline{a} | a \in \Sigma_T\}$, where $\overline{a} \notin \Sigma_T$. Let $(e_1, e_2, \ldots, e_{2n-2})$ be the sequence of directed edges in the Euler path of a tree T with n nodes. From this, we create the Euler string $s(T)$ of length $2n - 2$. Let $e = \{u, v\}$ be an edge in T, where u is the parent of v. Suppose that $e_i = (u, v)$ and $e_j = (v, u)$ (clearly, $i < j$). We define $i_1(e)$ and $i_2(e)$ by $i_1(e) = i$ and $i_2(e) = j$, respectively. That is, $i_1(e)$ and $i_2(e)$ denote the first and second positions of e in the Euler tour, respectively. Then, we define $s(T)$ by letting $s(T)[i_1(e)] = L(e)$ and $s(T)[i_2(e)] = \overline{L(e)}$, where $L(e)$ is the label of e (see also Fig. 3).

Proposition 1 *[1, 19]* $s(T_1) = s(T_2)$ *if and only if* $ED_T(T_1, T_2) = 0$. *Moreover, we can reconstruct* T *from* $s(T)$ *in linear time.*

Lemma 1 *[1]* $ED_S(s(T_1), s(T_2)) \leq 2 \cdot ED_T(T_1, T_2)$.

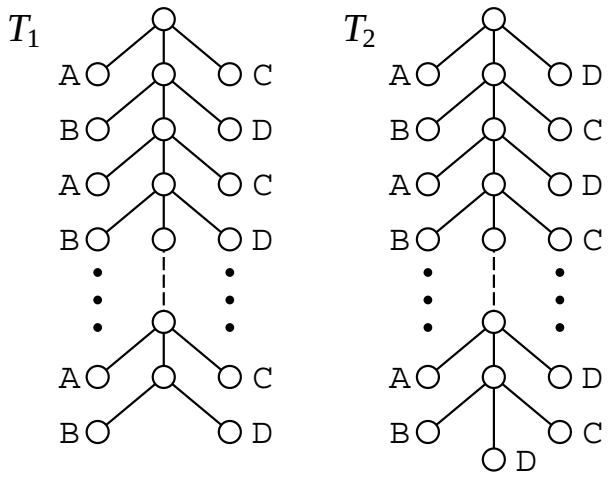

Figure 3: Example for the case of $ED_T(T_1, T_2) = \Theta(h) \cdot ED_S(s(T_1), s(T_2))$ [1].

Proof. We associate each edit operation on T_1 with two edit operations on $s(T_1)$. For a deletion of e (i.e., deletion of the deeper node of the endpoints of e), we associate deletions of $L(e)$ and $\overline{L(e)}$. For an insertion of e, we associate insertions of $L(e)$ and $\overline{L(e)}$. For a substitution of e to e', we associate substitutions of $L(e)$ and $\overline{L(e)}$ to $L(e')$ and $\overline{L(e')}$, respectively. Clearly, the resulting sequence transforms $s(T_1)$ to $s(T_2)$ and the cost is $2 \cdot ED_T(T_1, T_2)$. □

Lemma 2 *[1]* $ED_T(T_1, T_2) \leq (2h + 1) \cdot ED_S(s(T_1), s(T_2))$*, where* h *is the minimum height of two input trees.*

It was shown in [1] that this bound is tight up to a constant factor. Fig. 3 gives an example such that $ED_S(s(T_1), s(T_2)) = 4$ and $ED_T(T_1, T_2) = \Theta(h)$.

4 Modified Euler String

As shown in the above, the approximation ratio of the tree edit distance through the edit distance between the Euler strings is not good if the minimum height of input trees is high. In order to improve the worst case approximation ratio, we modify labels of some edges in the input trees so that structures of small subtrees are reflected to the labels. For example, we consider trees shown in Fig. 3. Suppose that label "AC" is assigned

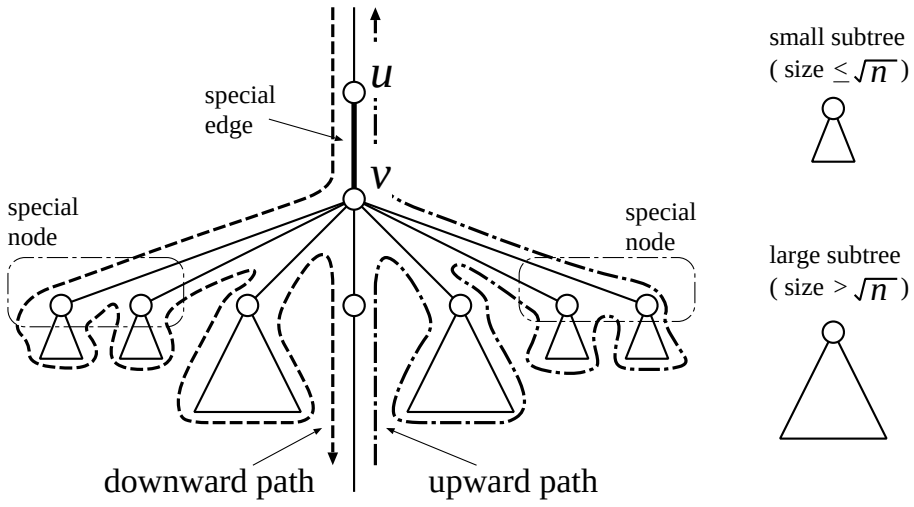

Figure 4: Special nodes, special edges, and large subtrees.

to each edge just above each node having children with labels 'A' and 'C'. Similarly, suppose that labels "BD", "AD" and "BC" are assigned to appropriate edges. Then, $ED_S(s(T_1), s(T_2)) = \Theta(h)$ should hold. But, changes of labels should be performed carefully in order to keep distance distortion not too large.

Before explaining changes of labels, we need several definitions (see also Fig. 4). Let $size(v)$ be the size (the number of nodes) of the subtree induced by v and its descendants. A subtree rooted at v is called *large* if $size(v) > \alpha$, where α is a parameter defined as $\alpha = n^{1/2}$. Otherwise, it is called *small*. We call w_i a *special node* if $size(w_i) \leq \alpha$ and $size(v) > \alpha$ where v is the parent of w_i. Then, we have the following proposition, where $depth(v)$ denotes the depth of v (i.e., the length of the path from the root to v).

Proposition 2 *For each node v in T, there exists at most one special node in the path from the root to v. Moreover, if v is a leaf and $depth(v) \geq \alpha$, there exists exactly one special node in the path.*

Proof. Let $(v_0 = r, v_1, v_2, \cdots, v_k = v)$ be the path from the root to v. Since $size(v_0) > size(v_1) > size(v_2) > \cdots > size(v_k)$ holds, at most one v_i can satisfy $size(v_{i-1}) > \alpha \geq size(v_i)$.

If v is a leaf and $depth(v) \geq \alpha$, there must exist v_i satisfying $size(v_{i-1}) > \alpha \geq size(v_i)$ because $size(v_0) \geq \alpha + 1 > \alpha$ and $size(v_k) = 1$. □

For a node v in T_1 or T_2, $id(v)$ is an integer such that $id(v) = id(v')$ if and only if the subtree induced by v and its descendants is isomorphic (including labels) to the subtree induced by v' and its descendants. Since we only consider subtrees induced by some node v in T_1 or T_2 and its descendants and we assume that Σ_T is a finite alphabet (i.e., $|\Sigma_T|$ is a constant), all $id(v)$'s can be computed in $O(n)$ time [11], where $n = \max\{|T_1|, |T_2|\}$. Furthermore, each $id(v)$ can be an integer between 1 and $2n$ and thus can be stored in a word (i.e., $O(\log n)$ bits). In the following, we briefly review the algorithm for computing $id(v)$. For details, refer to [11].

We construct a *suffix tree* $\mathcal{T}$ for a string $s(T_1)\cdot$"\$"$\cdot s(T_2)\cdot$ "#", where '\$' and '#' are letters not appearing in Σ_S, and $x \cdot y$ denotes the concatenation of x and y. Then, each suffix staring from a letter in Σ_T corresponds to a subtree in T_1 or T_2. For each leaf l in $\mathcal{T}$ corresponding to such a suffix, let $sz(l)$ be the size of the corresponding subtree in T_1 or T_2. Let $a(l)$ be the first node v, encountered along the path from the root of $\mathcal{T}$ to the leaf l, such that the length of the substring corresponding to the path from the root to $a(l)$ is no less than $2 \cdot sz(l)$. Then, it is shown in [11] that the subtrees corresponding to $l_1, l_2, \ldots, l_k$ are isomorphic to each other if and only if $a(l_1) = a(l_2) = \cdots = a(l_k)$. Thus, by identifying all $a(l)$'s, we can partition all subtrees into the equivalent classes. Identification of all $a(l)$'s can be done by using the depth first search traversal of $\mathcal{T}$. When the first leaf l corresponding to a subtree is found, we find the node $a(l)$ by coming back from l to the root of $\mathcal{T}$. Next, we visit all descendants of $a(l)$ and put leaves corresponding to subtrees in T_1 and T_2 into the same class. Then, we delete $a(l)$ and all of its descendants and resume the depth first search traversal. It is shown in [11] that this algorithm works in $O(n)$ time for a finite alphabet, including the construction of a suffix tree. After the set of equivalent classes is obtained, we can assign different integers to different classes in $O(n)$ time. Therefore, we can assign $id(v)$ to all nodes in $O(n)$ time.

Using $id(v)$, we define edge labels, with which the modified Euler strings are constructed. Let v be a node in T_1. Let u be the parent of v and $w_1, \ldots, w_k$ be the children of v (Similarly, we define v', u', and $w'_1, \ldots$ for T_2). If at least one of w_i's is special, $\{u, v\}$ is called a *special edge*. Otherwise, $\{u, v\}$ is not special and the original label (i.e., label in Σ_T) of v is assigned to $\{u, v\}$. For a special edge $\{u, v\}$, let $w_{i_1}, \ldots, w_{i_h}$ be the special children of v. Let $id'(v, w_{i_1}, \ldots, w_{i_h})$ be an integer number such that

$id'(v, w_{i_1}, \ldots, w_{i_h}) = id'(v', w'_{i_1}, \ldots, w'_{i_l})$ if and only if $h = l$, $label(v) = label(v')$, and $id(w_{i_j}) = id(w'_{i_j})$ holds for all $j = 1, \ldots, h$. Then, we assign $id'(v, w_{i_1}, \ldots, w_{i_h})$ to $\{u, v\}$ where we assume w.l.o.g. (without loss of generality) that $id'(\ldots) \notin \Sigma_T$. It should be noted that if v has at least one special children, information of the subtrees of the special children is reflected to the label of $\{u, v\}$.

These indices can be computed in $O(n)$ time as follows. We simply sort all tuples (i.e., all $(label(v), id(w_{i_1}), \ldots, id(w_{i_h}))$'s) in lexicographic order. Since we assume that the maximum degree is bounded, each tuple is a sequence of at most constant number of integers between 1 and $2n + |\Sigma_T|$. Furthermore, each label of a special edge does not depend on labels of other special edges. Therefore, we can obtain the sorted list of tuples in $O(n)$ time by performing radix sort only once.

Using the above labeling of edges, we create a modified Euler string $ss(T)$ as in $s(T)$. It should be noted that $ss(T)$ and $s(T)$ differ only on labels of special edges. It is also worthy to note that $ss(T_1)$ and $ss(T_2)$ can be constructed in $O(n)$ time from T_1 and T_2. In what follows, we consider editing operations on T_1 and T_2, by which the number of nodes in trees may change. However, α is fixed to $\alpha = n^{1/2} = (\max\{|T_1|, |T_2|\})^{1/2}$ throughout editing operations.

Proposition 3 *Substitution, insertion or deletion of a node in T_1 or T_2 affects the label of at most two special edges.*

Proof. Since there exists at most one special node in the path from the root to each node v (though there may exist many special edges), each edit operation can change the label of at most one existing special edge, including the case where a special edge becomes non-special. In addition, at most one non-special edge in the path may become special. Therefore, each edit operation affects the label of at most two special edges. □

5 Analysis

In this section, we show the following main theorem using several propositions and lemmas, where we assume that the maximum degree of input trees are bounded by a constant. In what follows, we may identify a directed edge (u, v), a node v and the corresponding letter in $ss(T)$ if there is no confusion.

Theorem 1 $\frac{1}{O(n^{3/4})} \cdot ED_T(T_1, T_2) \leq ED_S(ss(T_1), ss(T_2)) \leq 6 \cdot ED_T(T_1, T_2)$.

5.1 Upper Bound of String Edit Distance

First, we prove the latter half of Theorem 1, which is easily done as in Lemma 1.

Lemma 3 $ED_S(ss(T_1), ss(T_2)) \leq 6 \cdot ED_T(T_1, T_2)$.

Proof. As in the proof of Lemma 1, we associate each edit operation on T_1 with two edit operations on $ss(T_1)$. But, in this case, additional substitutions are required because labels of some edges may change. From Proposition 3, it is seen that labels of at most two edges change per edit operation, which correspond to substitutions of 4 letters in $ss(T_1)$. □

It is to be noted that both Lemma 3 and Proposition 3 hold for any $\alpha \geq 1$. Therefore, the above lemma holds even when $ED_T(T_1, T_2)$ is $O(n)$. In order to prove the former half of Theorem 1, it is enough to consider the case where $|T_1|$ and $|T_2|$ are $\Theta(n)$. Otherwise $||T_1| - |T_2|| > cn$ would hold for some constant c and thus the former half obviously holds since $ED_S(ss(T_1), ss(T_2)) > cn$ holds. Therefore, we can assume w.l.o.g. that α is $\Theta(|T_1|^{1/2}) = \Theta(|T_2|^{1/2})$.

5.2 Construction of Tree Mapping from String Alignment

In order to prove the rest half inequality, we show a procedure for obtaining a mapping between T_1 and T_2 from an (not necessarily optimal) alignment AL_S between $ss(T_1)$ and $ss(T_2)$ with cost d. Before showing details of the procedure, we describe an outline. We first create a mapping M_1 that is induced by corresponding downward paths, where downward (and upward) paths are to be defined later. Next, we modify M_1 to M_2 so that labeling information on special edges is reflected (i.e., mapping pairs for right subtrees rooted at special children are added to M_1). However, such mappings (both M_1 and M_2) may contain pairs violating ancestor-descendant relations. Thus, we delete inconsistent pairs from M_2 (the resulting mapping is denoted as M_3). Finally, we add large subtrees included in upward paths, then delete some inconsistent pairs from M_3, and get the desired mapping M_4.

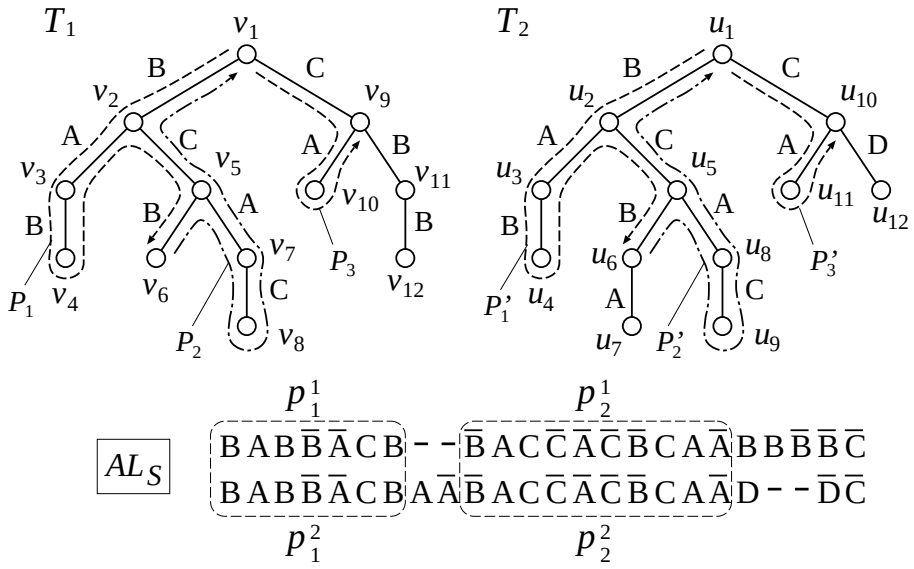

Figure 5: Construction of M_1 from AL_S. Regions surrounded by dashed lines in AL_S correspond to maximal substring pairs. The path corresponding to p_2^1 (resp. p_2^2) is divided into P_2 and P_3 (resp. P_2' and P_3'), where P_2 and P_2', and P_3 and P_3' are twins respectively. The path corresponding to p_1^1 (resp. p_1^2) is divided into P_1 (resp. P_1') and an empty path. P_1, P_1', P_3 and P_3' are downward paths, whereas P_2 and P_2' are upward paths. P_1 contains central edges of $(v_1, v_2), (v_2, v_5), (v_5, v_6)$, and P_2 contains central edges of $(v_6, v_5), (v_5, v_2), (v_2, v_1)$. $M_1 = \{(v_2, u_2), (v_3, u_3), (v_4, u_4), (v_5, u_5), (v_6, u_6), (v_9, u_{10}), (v_{10}, u_{11})\}$.

5.2.1 Construction of M_1

In this first phase, we create a mapping M_1 from AL_S. M_1 acts as a backbone for the whole mapping. In order to explain the construction of M_1, we define downward paths and related terms.

An edge (u, v) is called a *downward edge* if v is a child of u. Otherwise, (u, v) is called an *upward edge*. For each downward edge e, $\overline{e}$ denotes the upward edge corresponding to e (i.e., $\overline{e} = (v, u)$ if $e = (u, v)$). For each edge $e = (u, v)$, we let $src(e) = u$ and $dst(e) = v$.

Let $\{(p_1^1, p_1^2), (p_2^1, p_2^2), \ldots\}$ be the set of maximal *substring* pairs (p_i^1 from $ss(T_1)$ and p_i^2 from $ss(T_2)$), each of which corresponds to a maximal consecutive region (with length at least 2) in AL_S without insertions, deletions or substitutions. Note that p_i^1 and p_i^2 correspond to isomorphic paths in T_1 and T_2. We divide each path corresponding to p_i^j into two parts (see also Fig. 4 and Fig. 5): upward path and downward path, where

one path may be empty. Let $p_i^1[k]$ be the first letter corresponding to a downward edge (u, v) such that a letter corresponding to (v, u) does not appear in p_i^1. Then, $(p_i^1[1 \ldots k-1], p_i^2[1 \ldots k-1])$ and $(p_i^1[k \ldots], p_i^2[k \ldots])$ are the *upward segment pair* and the *downward segment pair*, respectively. Two paths P in T_1 and P' in T_2 corresponding to an upward segment pair are called *upward paths*, and P (resp. P') is called a *twin* of P' (resp. P). Two paths corresponding to a downward segment pair are called *downward paths*, and twins are defined in the same way as above. A subtree that is fully included in a downward (resp. upward) path is called a *left subtree* (resp. *right subtree*). An edge (u, v) in a downward (resp. upward) path is called a *central edge* if (v, u) does not appear in the same path. Thus, central edges are the edges (in downward paths and upward paths) not appearing in any left or right subtrees. For a downward path (resp. an upward path) P, a *downward central path* (resp. *upward central path*) denotes the path consisting of central edges in P and $height(P)$ denotes the number of edges in this central path. It is possible that there is no central edge in an upward path (i.e., $height(P) = 0$), where a downward path must contain at least one central edge from the definition. In such a case, the path consists of a large or small subtree. In the following, such a subtree is regarded as a left subtree and the path corresponding to the subtree is regarded as a downward path. It is to be noted that the roots of such subtrees are not included M_1 since these roots are not destinations of downward edges (see also P_2 and P_2' in Fig. 9).

Suppose that $ss(T_1)[i]$ corresponds to $ss(T_2)[i']$ in any downward segment pair by means of AL_S, and downward edges (u, v) and (u', v') correspond to $ss(T_1)[i]$ and $ss(T_2)[i']$, respectively. Then, we let M_1 be the set of such (v, v')'s defined as above (see Fig. 5 for an example). If there exist pairs of twin downward paths consisting of only subtrees, pairs of the roots of the corresponding subtrees are added to M_1.

It should be noted that in our previous work [1], we construct a mapping between T_1 and T_2 from all corresponding pairs of left and right subtrees, which is shown to be a valid mapping. From that result, it is seen that mapping pairs in M_1 corresponding to left subtrees are consistent with each other. Thus, the inconsistency will be caused by central edges and (small and large) right subtrees that are to be added in constructions of M_2 and M_4. The following proposition implies that we only need to take care of nodes in downward edges, left subtrees and right subtrees since the order of approximation ratio

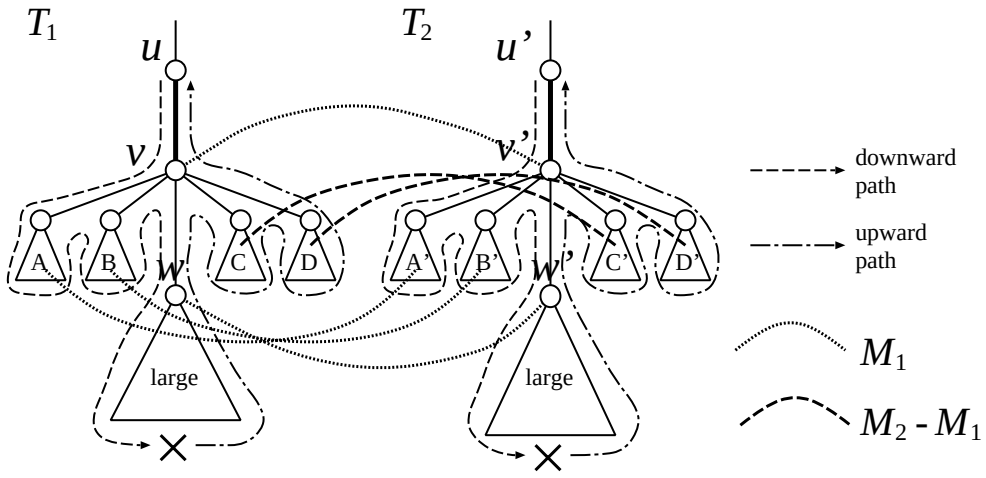

Figure 6: Addition of mapping pairs to M_2. In this case, mapping pairs between C and C', and between D and D' are added to M_2. Each cross means that there exist insertion(s), deletion(s) or substitution(s) around the point.

does not change if we ignore the other $O(d)$ nodes.

Proposition 4 *The number of nodes not appearing in destinations (i.e., $dst(e)$) of downward central edges, left or right subtrees is $O(d)$.*

Proof. Since there are $O(d)$ insertions, deletions and substitutions, the number of edges not appearing in downward paths or upward paths is $O(d)$. The nodes not on these edges must be on downward central edges or in left or right subtrees. □

5.2.2 Construction of M_2

M_1 gives a mapping for left subtrees and central edges, but does not give a mapping for right subtrees. In the construction of M_2, mapping pairs for small right subtrees are added, whereas mapping pairs for large right subtrees are added in the construction of M_4.

Let $e = (u, v)$ and $e' = (u', v')$ be a pair of corresponding central special edges in M_1 (i.e., $(v, v') \in M_1$). Suppose that there exists a node w in T_1 satisfying the following conditions (see Fig. 6):

(i) (u, v) and (v, w) belong to the same downward path, (v, u) and (w, v) belong to the same upward path,

(ii) the subtree rooted at w is large.

Then, there must exist a node w' in T_2 satisfying analogous conditions because (u, v) and (u', v') are special edges having the same label. It is to be noted that from condition (i), there must exist insertion(s), deletion(s) or substitution(s) in the large subtree rooted at w. We add mapping pairs to M_1 that are induced by the small right subtrees rooted at the special children of v and v', where the correspondence between right subtrees in T_1 and T_2 are given by *id*'s of the special children. Since we assume that the maximum degree is bounded, detection of corresponding special children can be done in a constant time per special edge. We let the resulting mapping be M_2.

Some examples explaining the above conditions are given in Fig. 7. In case (a) and case (b), mapping pairs are added. In case (c), mapping pairs are not added because w is a root of a small subtree. In case (d), mapping pairs are not added because (w_1, v) and (v, u) belong to different upward paths and (u, v) and (v, w_2) belong to different downward paths. As shown below, the number of central edges such as in (c) and (d) is not so large.

Proposition 5 *The number of central edges which do not satisfy the condition in the construction of M_2 is $O(d)$.*

Proof. We prove the proposition for central edges in T_1. The proposition can be proven for central edges in T_2 in an analogous way and thus the total number should still be $O(d)$.

First, we consider the case that condition (i) is violated for a central edge (u, v) (see also Fig. 7 (d)). Then, v must have at least two subtrees in each of which there exist insertion(s), deletion(s) or substitution(s). Since the total number of insertions, deletions and substitutions is $O(d)$, we can have $O(d)$ central edges that do not satisfy condition (i) (recall that a tree such that every internal node has at least two children can have at most $2l - 1$ nodes where l is the number of leaves).

Next, we consider condition (ii). For a central edge (u, v) that violates condition (ii), we associate the small (left or right) subtree rooted at a special child of v in which there exist insertion(s), deletion(s) or substitution(s) (see also Fig. 7 (c)). This small subtree can be associated with at most one central edge. Since the total number of insertions, deletions and substitutions is $O(d)$, we can have $O(d)$ central edges that do not satisfy condition (ii). □

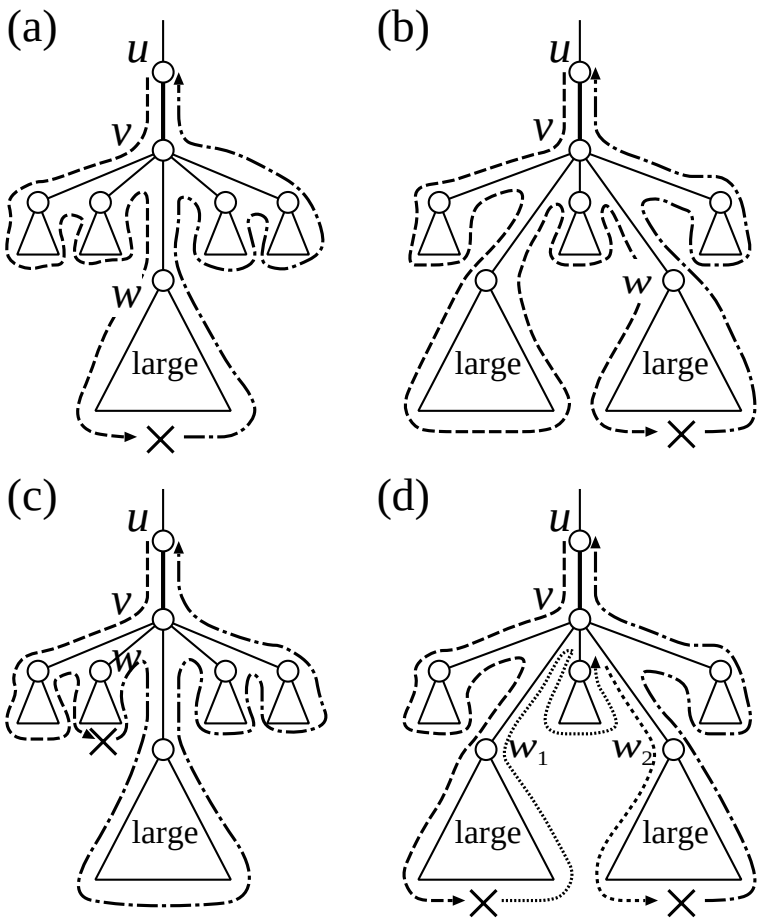

Figure 7: Examples for explaining the conditions used in the construction of M_2. Mapping pairs for right subtrees are added to M_2 in case (a) and case (b), whereas these are not added in case (c) or case (d).

5.2.3 Construction of M_3

In the construction of M_3, we delete inconsistent pairs. Since M_1 is constructed from downward paths obtained from string alignment, which preserve left-right relationships, and M_2 is constructed from central special edges, left-right relationships between mapping pairs are preserved. Thus, we focus on violation of ancestor-descendant relationships.

Let $(v, v') \in M_1$ be a pair of the highest nodes in a pair of twin downward paths, where the highest node in any downward path is determined uniquely. Let $\hat{P}_v$ (resp. $\hat{P}_{v'}$) be the set of nodes in the path from the root to v (resp. v'). We delete any $(u, u') \in M_1$ from M_2 if either ($u \in \hat{P}_v$ and $u' \notin \hat{P}_{v'}$) or ($u \notin \hat{P}_v$ and $u' \in \hat{P}_{v'}$) holds. We also delete

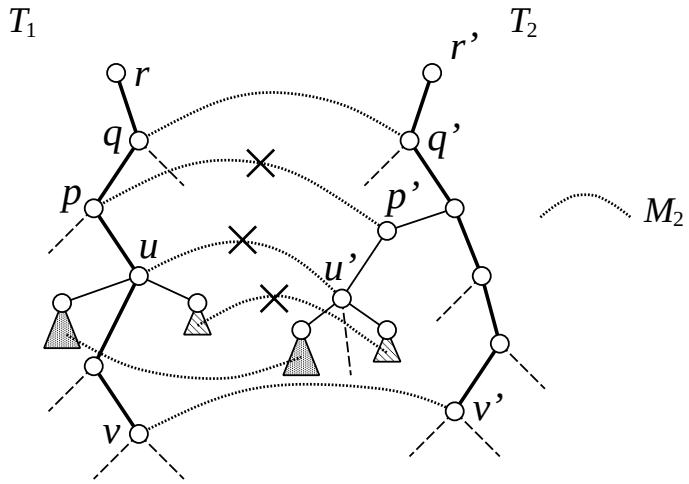

Figure 8: Deletion of inconsistent mapings from M_2 in the construction of M_3. Bold lines correspond to $\hat{P}_v$ and $\hat{P}_{v'}$.

small right subtrees rooted at children of u and u' (see Fig. 8). In this paper, *deletion of a subtree* (resp. a region or a node) means that all mapping pairs containing nodes in the subtree are deleted from the current mapping set, but does not mean that the subtree is deleted from the tree. *Addition of a subtree* is defined analogously. We execute this deletion procedure for all downward segment pairs in an arbitrary order, where we allow that the same mapping pairs are deleted multiple times. Then, the resulting mapping M_3 is determined uniquely. M_3 is a valid mapping as shown below.

Proposition 6 *M_3 is a valid mapping between T_1 and T_2.*

Proof. Since mapping pairs between small right subtrees are obtained from central special edges, these are consistent with other mapping pairs if the corresponding central special edges are consistent. Therefore, in the following, we focus on inconsistency caused by M_1.

Mapping pairs created by a single pair of twin downward paths are consistent with each other since twin downward paths are isomorphic. Therefore, inconsistency is caused by $(u, u') \in M_1$ and $(w, w') \in M_1$ belonging to different pairs of twin downward paths.

Suppose that u is an ancestor of w but u' is not an ancestor of w'. Then, u must be an ancestor of the node v which corresponds to the highest node of the downward path containing w, but u' cannot be an ancestor of the node v' which corresponds to the highest

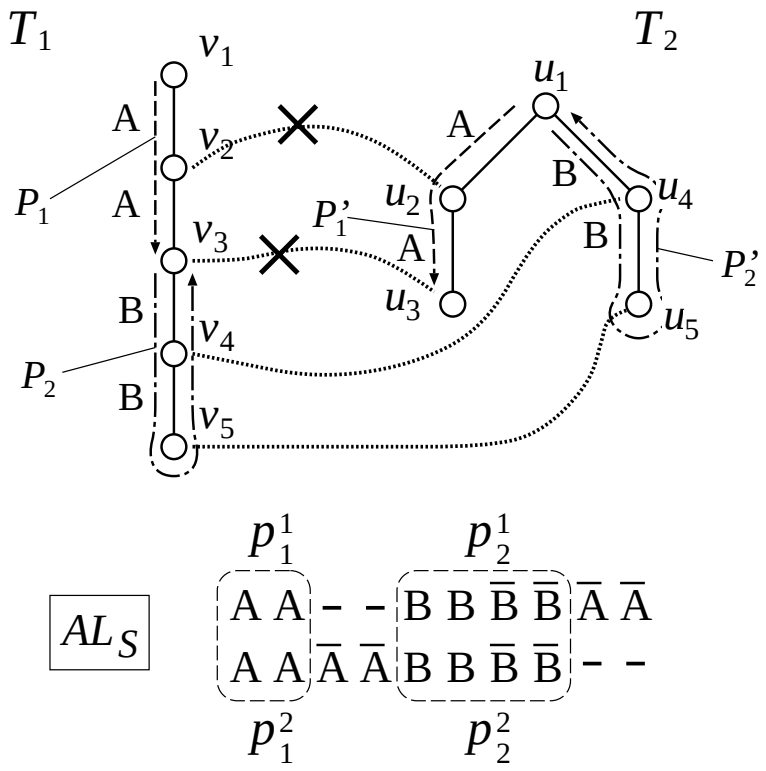

Figure 9: An example of construction of M_3. P_1 and P_1' are downward paths obtained from (p_1^1, p_1^2), and P_2 and P_2' are downward paths obtained from (p_2^1, p_2^2). $M_1 = M_2 = \{(v_2, u_2), (v_3, u_3), (v_4, u_4), (v_5, u_5)\}$ is shown by bold dotted lines, from which $M_3 = \{(v_4, u_4), (v_5, u_5)\}$ is obtained by deleting $(v_2, u_2), (v_3, u_3)$.

node of the downward path containing w' (see also Fig. 8). Therefore, it is enough to examine consistency between all pairs (u, u') in M_1 and all pairs (v, v') of highest nodes in twin downward paths. Since the other cases can be proven in an analogous way, the proposition holds. □

It should be noted that left subtrees are never deleted since any node in a left subtree cannot be an ancestor of nodes not in the subtree. Mapping pairs between left subtrees will be consistent with other pairs in M_3 (and in M_4).

Fig. 9 gives an example of construction of M_3. In this case, (v_2, u_2) is deleted from M_2 since $v_2 \in \hat{P}_{v_4}$ but $u_2 \notin \hat{P}_{u_4}$. Similarly, (v_3, u_3) is deleted from M_2. The readers may think that too many mapping pairs are deleted if the paths corresponding to AA are much longer than the paths corresponding to BB. However, in such a case, AL_S should

have many unaligned $\overline{\mathrm{A}}$'s and thus d should have been so large that $d \cdot O(n^{4/3}) = O(n)$ is satisfied (recall that the tree edit distance is always $O(n)$). This intuition is to be proven concretely in Lemma 10.

5.2.4 Construction of M_4

Finally, we add all large subtrees (i.e., subtrees with more than $n^{1/2}$ nodes) that are fully included in upward paths, and then delete inconsistent mapping pairs.

In the following, we assume that the number of large subtrees attached to a node is at most one. If multiple large subtrees are attached to a node, we can regard these large subtrees as one large subtree. In this final phase, we consider the following two cases (see Fig. 10):

(A) the number of edges in an upward path is at most $2d$,

(B) the number of edges in an upward path is greater than $2d$, where we assume w.l.o.g. that all the central edges of an upward path are shared by the central edges in a downward path (otherwise, we can cut the upward path into multiple upward paths without affecting the order of the approximation ratio since the total number of upward paths remains $O(d)$).

As to be shown later, there is some periodicity in case (B), which plays an important role in both construction and analysis of M_4. In all cases, mapping pairs between large right subtrees, which are induced by twin upward paths, are added to M_4. None of these added pairs is deleted. Instead, mapping pairs between some nodes in central paths and small right subtrees are deleted. Therefore, we only describe deletion procedures in the following.

For case (A), we only show the procedure for the case where only one large subtree is included in an upward path (see Fig. 11). This procedure is referred as the *cleanup procedure*. Extension to the other cases is straight-forward since it is enough to repeat the same procedure. Let $z \dots \overline{z}$ (resp. $z' \dots \overline{z'}$) be the sequence of directed edges corresponding to the large subtree of T_1 (resp. T_2). Let $x = (u, v) \in EE(T_1)$ be a parent of z, and let $y' = (u', v') \in EE(T_2)$ be a parent of z' (recall that $EE(T)$ denotes the set of directed edges in the Euler tour of T). Let x' and y correspond to x and y' in downward paths,

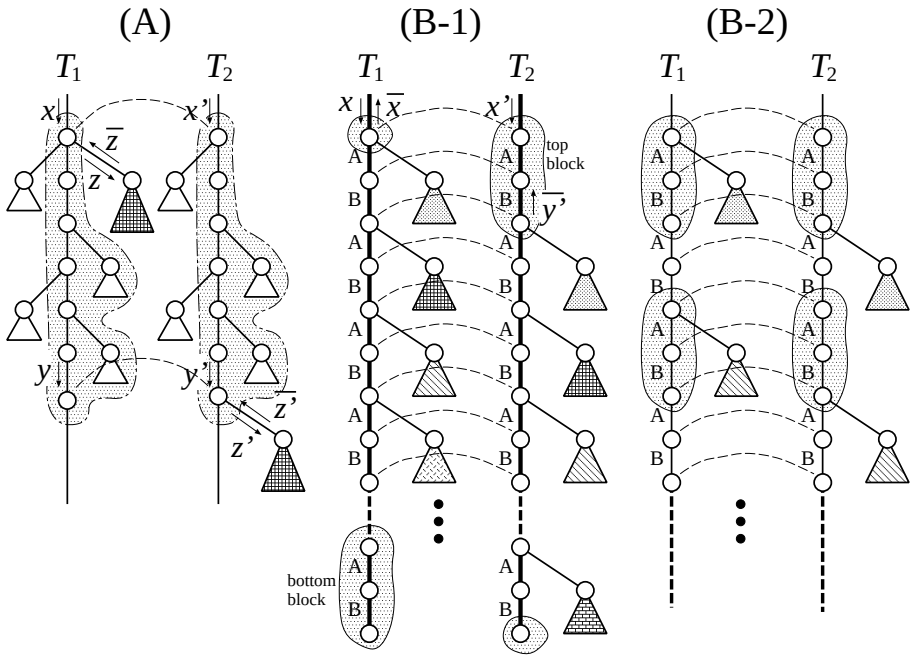

Figure 10: Construction of M_4. Gray triangles denote large subtrees, and thin dashed lines denote (parts of) M_3. (A) Gray regions are deleted. (B-1) Gray regions and central edges (shown by bold lines) are deleted, and mappings between small right subtrees are modified. (B-2) For each pair of large right subtrees, gray regions are deleted as in case (A). Different from case (A), the size of each deleted region is $O(dn^{1/4})$.

respectively. We assume w.l.o.g. that y' is a descendant of x'. Then, we delete v' and ith ancestors of v' for $i = 1, 2, \ldots, d$ along with attached small right subtrees (see Fig. 11). Though the cleanup procedure is very simple, analysis is a bit involved. In the following, we show the correctness of the procedure.

For two strings s_1 and s_2, $s_1 \cdot s_2$ denotes the concatenation of s_1 and s_2. For a string s, $\#_0(s)$ and $\#_1(s)$ denote the number of letters corresponding to downward edges and upward edges, respectively. Let $\Delta(s) = \#_1(s) - \#_0(s)$.

Proposition 7 *$ED_S(s_1, s_2) \geq |\Delta(s_1) - \Delta(s_2)|$.*

Lemma 4 *Let $s_2 = s_2^1 \cdot s_2^2$. Suppose that $\Delta(s_2^1) = h > 0$ holds, and $\Delta(s_1^1) \leq 0$ holds for any prefix s_1^1 of s_1. Then, $ED_S(s_1, s_2) \geq h$ holds.*

Proof. From the definition of string edit distance,

$$ED_S(s_1, s_2) \;=\; \min\{ED_S(s_1^1, s_2^1) + ED_S(s_1^2, s_2^2)\}$$

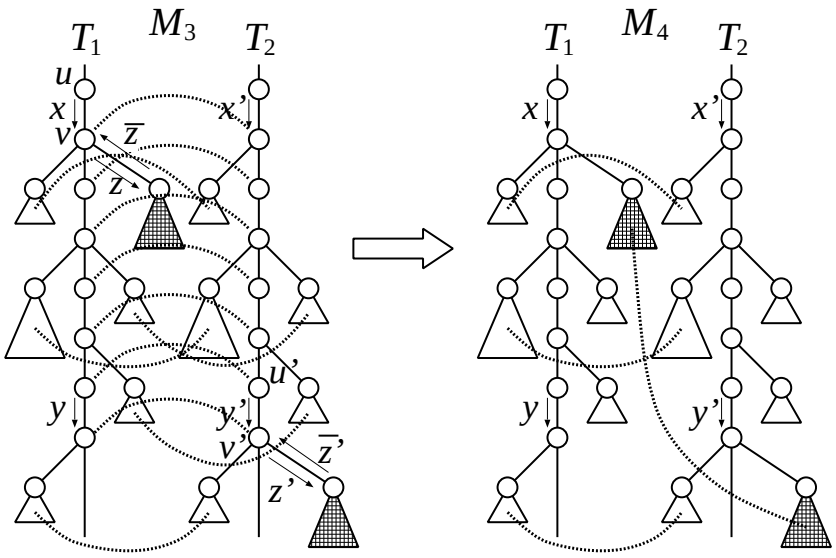

Figure 11: Details of case (A) in M_4. In this case, mapping pairs between large checked triangles are added and mapping pairs between nodes in central edges and small right subtrees are deleted.

where the minimum is taken over all partitions $s_1 = s_1^1 \cdot s_1^2$ of s_1. Since $ED_S(s_1^1, s_2^1) \geq h$ holds for any partition $s_1 = s_1^1 \cdot s_1^2$ from the assumption and Proposition 7, we have

$$ED_S(s_1, s_2) \ \geq \ h + ED_S(s_1^2, s_2^2) \ \geq \ h.$$

□

Lemma 5 *Suppose that only a pair of identical large subtrees is added in the construction of M_4 and the cleanup operation is performed. Then, the resulting mapping is valid.*

Proof. It is to be noted that mapping pairs corresponding to left subtrees and large right subtree are consistent with each other [1]. Thus, inconsistency is caused only when ancestors of v (resp. v') (along with attached small right subtrees) are mapped to non-ancestors of v' (resp. v). In this proof, we consider mapping pairs only for ancestors of v. Mapping pairs for ancestors of v' can be treated in an analogous way. We need to consider two cases (see also Fig. 12): (A-1) ancestors of v are mapped to descendants of v', (A-2) ancestors of v are mapped to non-descendants of v'.

First we consider the case of (A-1). Let p and q be the $(d+1)$th and dth ancestors of v, respectively. Suppose that p is mapped to v' in M_3. Suppose also that q is mapped to

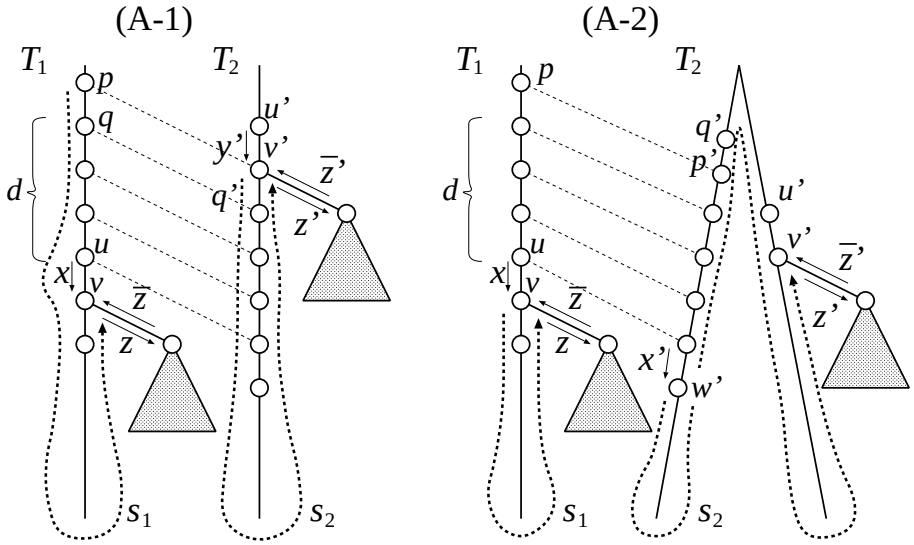

Figure 12: Illustration of the proof of Lemma 5.

a child q' of v' in M_3. Modification of the proof for the other cases (e.g., p is mapped to a descendant of v') is straight-forward.

Let s_1 be a substring of $ss(T_1)$ starting from (p, q) and ending just before z. Let s_2 be a substring of $ss(T_2)$ starting from (v', q') and ending just before z'. Then, $\Delta(s_1) = -(d+1)$ and $\Delta(s_2) = 0$ hold. Since (p, q) and z correspond to (v', q') and z' respectively in AL_S, we have from Proposition 7:

$$d \ \geq \ ED_S(s_1, s_2) \ \geq \ |-(d+1) - 0| = d + 1.$$

This is a contradiction. Thus, p cannot be mapped to v' (or its descendant). Since at most d ancestors of v can be mapped to v' or its descendants, the cleanup operation removes the inconsistency in the case of (A-1).

Next we consider the case of (A-2). In this case, we assume w.l.o.g. that v is mapped to w' which is not an ancestor or descendant of v', or v'. Let x' be the incoming edge to w'. Let p be the $(d+1)$th ancestor of v. Suppose that p is mapped to a node p' which is an ancestor w' but is not an ancestor of v'. Modification of the proof for the other cases is straight-forward.

Let q' be the parent of p'. Let s_1 be a substring of $ss(T_1)$ starting just after x and ending just before z. Let s_2 be a substring of $ss(T_2)$ starting just after x' and ending just

before z'. Let s_2^1 be the prefix of s_2 ending at (p', q'). Then, $\Delta(s_1^1) \leq 0$ holds for all prefix s_1^1 of s_1, and $\Delta(s_2^1) \geq d+1$ holds. Since x and z correspond to x' and z' respectively in AL_S, we have from Lemma 4:

$$d \ \geq \ ED_S(s_1, s_2) \ \geq \ d+1.$$

This is a contradiction. Thus, p cannot be mapped to a non-ancestor of v'. Therefore, the cleanup operation removes the inconsistency also in the case of (A-2). □

Before considering case (B), we need the following proposition (see also Fig. 10 (B-1)), which can be shown by counting the numbers of downward edges and upward edges in AL_S, where $depth(e)$ denotes the depth of v for a downward edge $e = (u, v)$ (resp. an upward edge $e = (v, u)$).

Proposition 8 *Suppose that a downward edge $x \in EE(T_1)$ corresponds to a downward edge $x' \in EE(T_2)$ in AL_S. Then, $|depth(x) - depth(x')| \leq d$ holds. Furthermore, suppose that $\overline{x}$ corresponds to an upward edge $\overline{y'} \in EE(T_2)$ in AL_S. Then,*

$$|depth(x') - depth(\overline{y'})| \ \leq \ d$$

holds.

For case (B), central paths should have *periodicity* because upward central paths match with identical downward central paths at different positions. From Proposition 8, we can see that the length of a period is at most d. That is, central paths are repetitions of a chain of length at most d. Suppose that x and $\overline{x}$ correspond to x' and $\overline{y'}$ in AL_S, respectively (see also Fig. 10). We assume w.l.o.g. that the downward paths begin with x and x' in T_1 and T_2 respectively, and y' is a descendant of x'. Then, the subtree consisting of the nodes in the central path between $dst(x')$ and $dst(y')$ and the nodes in their small right subtrees is called a *block*. A subtree isomorphic to the block is also called a block. It can be seen that blocks appear repeatedly in the vertical direction in both T_1 and T_2. We consider the following two cases:

(B-1) the size of a block is greater than β, where $\beta = dn^{1/4}$.

In this case, the number of the central edges in an upward path is $O(n^{3/4})$ because

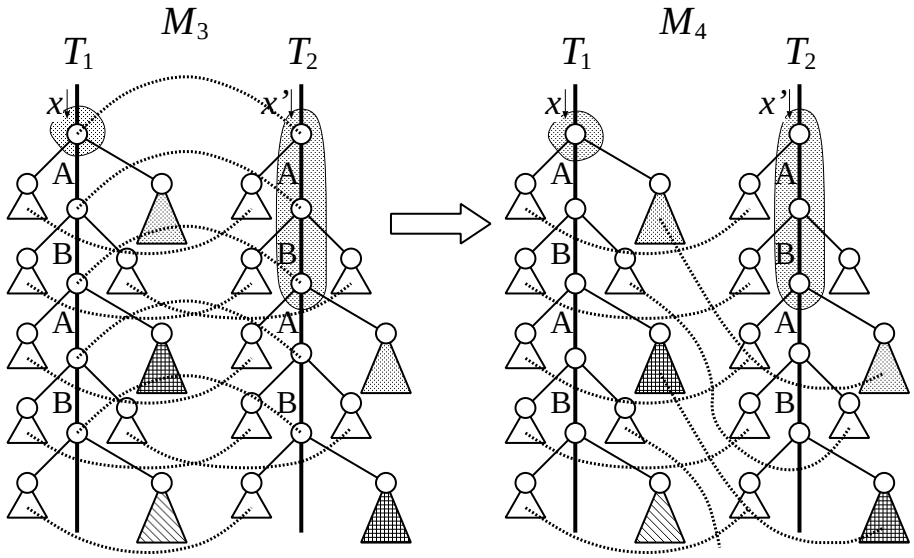

Figure 13: Details of case (B1) in M_4, where the upper half part is only shown in this figure. Bold dotted lines denote mappings, and gray triangles denote large right subtrees. Gray regions and central edges are deleted. Furthermore, mappings between small right subtrees are modified.

the height of each block is at most d, the height of an upward path is at most n, and

$$\beta \cdot \frac{\#(\text{central edges in the upward path})}{d} \leq n$$

holds from the definition of the block. We delete the top block from T_2, the bottom block from T_1, the central edges in the current downward paths from T_1 and T_2, small right subtrees attached to the top node in the top block of T_1 and small right subtrees attached to the bottom node in the bottom block of T_2 (see Fig. 13). Furthermore, we modify the mapping between the remaining small right subtrees so that mapping pairs between small right subtrees are consistent with those between large right subtrees. In other wards, mapping pairs between left subtrees are induced by the twin downward paths, whereas mapping pairs between right subtrees are induced by the twin upward paths. It is to be noted that in Fig. 13, the left subtree attached to the top 'A' in T_1 is mapped to the left subtree attached to the top 'A' in T_2, whereas the small right subtree attached to the top 'A' in T_1 is mapped to the small right subtree attached to the second top 'A' in T_2. However, there is no inconsistency because central edges are deleted.

(B-2) the size of a block is at most β.

In this case, for each pair of corresponding large right subtrees, we perform the cleanup procedure as in case (A). However, different from case (A), $O(\beta) = O(dn^{1/4})$ nodes are deleted per pair since the nodes in at most two consecutive blocks are deleted per large right subtree.

As in the case of (A), we can see that the resulting mapping is valid. Therefore, we have:

Proposition 9 *M_4 is a valid mapping between T_1 and T_2.*

5.3 Analysis of Lower Bound of String Edit Distance

Now we analyze the lower bound of $ED_S(ss(T_1), ss(T_2))$. We estimate the cost (i.e., the number of corresponding edit operations) of M_4, assuming that the cost of AL_S is $d = ED_S(ss(T_1), ss(T_2))$. For that purpose, we estimate the number of mapping pairs deleted or ignored in the construction. For two edges $x = (u, u')$ and $y = (v, v')$, $dist(x, y)$ denotes the length of the shortest path between u and v. For two nodes u and v, $dist(u, v)$ denotes the length of the shortest path between u and v.

It is straight-forward to see the following propositions (see also Proposition 4).

Proposition 10 *The number of nodes not appearing in any downward or upward path is $O(d)$.*

Proposition 11 *The number of downward (resp. upward) paths is $O(d)$.*

Proposition 12 *The total number of nodes in downward paths that do not appear in M_1 is $O(d)$.*

Due to the above propositions, we only need to consider hereafter upward paths and deleted mapping pairs.

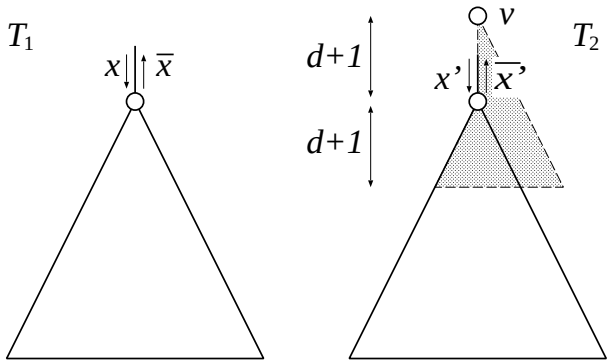

Figure 14: Explanation of Lemma 7. $\overline{y'}$ should be located in the gray region.

Lemma 6 *The number of nodes in the small subtrees in the upward paths that are not included in M_2 is $O(dn^{1/2})$.*

Proof. It is seen from Proposition 5 that the number of central special edges that are not taken into account for M_2 is $O(d)$. Since we assume that the maximum degree is bounded by a constant, the number of nodes in the small right subtrees rooted at special children of a special edge is $O(n^{1/2})$. There may exist small subtrees that are fully included in the upward paths but are not attached to special edges. But, the total size of such small subtrees per upward path is $O(n^{1/2})$. □

Next, we estimate the number of deleted mapping pairs in constructing M_3. For that purpose, we show using some lemmas that not so many nodes are deleted from central edges of each downward path.

Lemma 7 *Suppose that a downward edge $x \in EE(T_1)$ corresponds to a downward edge $x' \in EE(T_2)$ in AL_S. Suppose also that an upward edge $\overline{x} \in EE(T_1)$ corresponds to an upward edge $\overline{y'} \in EE(T_2)$. Then, $dist(\overline{x'}, \overline{y'}) \leq 3d + 3$.*

Proof. From Proposition 8, $|depth(x) - depth(\overline{y'})| \leq d$ holds. Let v be the ancestor of $src(x')$ such that $dist(v, src(x')) = d + 1$ (If there does not exist such a node, we let v be the root). From the above fact, it is seen that $\overline{y'}$ is in the subtree rooted at v and $depth(\overline{y'}) - depth(v) \leq 2d + 2$ holds (see Fig. 14). Since $dist(v, src(x')) \leq d + 1$ holds, we have the lemma. □

Lemma 8 *Let $x \in EE(T_1)$ and $y \in EE(T_1)$ be downward edges belonging to the same downward central path where x is an ancestor of y. Suppose that $\overline{x}$ and $\overline{y}$ belong to the same upward central path. Let x', y', $\overline{z'}$ and $\overline{w'}$ in $EE(T_2)$ correspond in AL_S to x, y, $\overline{y}$ and $\overline{x}$, respectively. Moreover, suppose that $depth(y) - depth(x) > 100d$. Then, at most $20d$ nodes are deleted from the downward central (sub)path beginning from x and ending at y in the construction of M_3.*

Proof. From Lemma 7, we see that the central upward path from $\overline{z'}$ to $\overline{w'}$ must have an overlap with the central downward path from x' to y' with the amount of at least $dist(x, y) - 20d$ nodes (see Fig. 15).

Let v' be the lowest common ancestor of y' and $\overline{z'}$. Let v be the node in T_1 such that $(v, v') \in M_2$, which also means $(v, v') \in M_1$ since these nodes belong to central edges. Let $(u, u') \in M_1$ be a pair of beginning nodes of twin downward paths such that u is a descendant of $dst(y)$. Then, u' must be a descendant of v' because u' is in the part of Euler path beginning from y' and ending at $\overline{z'}$. Therefore, (u, u') cannot delete any node in the consecutive part of a downward central path beginning from $dst(x)$ and ending at v.

Since $\overline{w'}$ and $\overline{z'}$ belong to the same upward central path, both $src(\overline{w'})$ and $dst(\overline{w'})$ are ancestors of $dst(\overline{z'})$. Thus, we consider two cases: (i) $dst(w')$ is a descendant of $dst(x')$, (ii) $dsp(w')$ is a ancestor of $dst(x')$ or w' coincides with x'. We let $t' = src(\overline{w'})$ in case of (i), otherwise we let $t' = dst(x')$. Let t be the node in T_1 such that $(t, t') \in M_1$. Then, any downward path in T_1 (resp. T_2) begins either from a descendant of $dst(y)$ (resp. v') or from a non-descendant of $dst(x)$ (resp. t'). In the former case, any central edge between t and v (resp. between t' and v') cannot be deleted as explained before. In the latter case, neither t or t' is an ancestor of the beginning node of a downward path. Therefore, any node in the central edges between t and v (resp. between t' and v') cannot be deleted.

Since the number of nodes between t and v (resp. t' and v') is at least $dist(x, y) - 20d$, the lemma holds. □

It should be noted that $100d$ and $-20d$ are determined with large margin since we discuss in this paper the order of the approximation ratio.

Next, we bound the number of nodes appearing in M_1 which may be deleted in the construction of M_3. Let $P = (v_{i_1}, v_{i_2}, \ldots, v_{i_k})$ be a consecutive part of a downward central

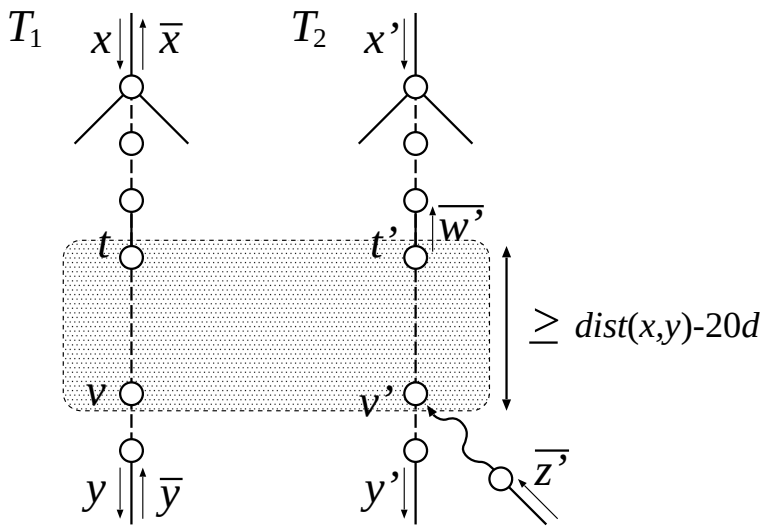

Figure 15: Explanation of Lemma 8.

path in T_1. Let $P' = (u_{i_1}, u_{i_2}, \ldots, u_{i_k})$ be the corresponding part of a downward central path in T_2. P (resp. P') is called a *maximal central subpath* if (P, P') is maximal (i.e., cannot be extended) under the condition that $(v_{i_k}, v_{i_{k-1}}, \ldots, v_{i_1})$ is a consecutive part of an upward central path of T_1 and $(u_{i_k}, u_{i_{k-1}}, \ldots, u_{i_1})$ is a consecutive part of an upward central path of T_2. Furthermore, P (resp. P') is called a *maximal conserved central subpath* if P (resp. P') is a maximal central subpath and $(v_{i_k}, v_{i_{k-1}}, \ldots, v_{i_1})$ and $(u_{i_k}, u_{i_{k-1}}, \ldots, u_{i_1})$ are in twin upward central paths. It is to be noted that $(v_{i_h}, v_{i_{h-1}})$ and $(u_{i_h}, u_{i_{h-1}})$ do not necessarily correspond to each other in AL_S. See Fig. 16 for an example.

Lemma 9 *The number of nodes that appear in central downward paths but do not belong to maximal conserved central subpaths of length greater than* $100d$ *is* $O(d^2)$.

Proof. Since there are $O(d)$ downward paths and $O(d)$ upward paths, the number of maximal central subpaths is $O(d)$. The number of edges in central downward paths but not in maximal central subpaths is also $O(d)$.

The number of nodes in maximal central subpaths whose length is at most $100d$ is $O(d) \times O(d) = O(d^2)$. The number of nodes in maximal central paths but not in maximal conserved central subpaths is $O(d) \times O(d) = O(d^2)$ because the total number of downward central subpaths for which corresponding upward central paths are not twins is $O(d)$ and the length of such a subpath is $O(d)$.

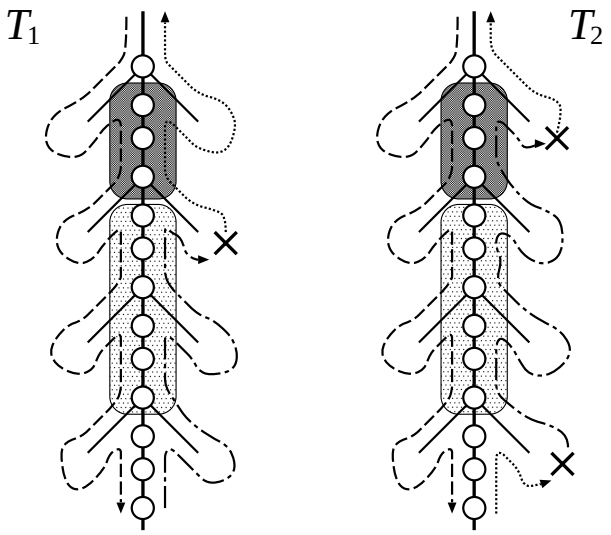

Figure 16: Explanation of a maximal conserved central subpath. Each of the light-gray regions is a maximal conserved central subpath. Each of the dark-gray regions is a maximal central subpath but is not a maximal conserved central subpath since upward paths in these regions are not twins. It should be noted that the lines of the same type correspond to twins.

By summing up all of the above, we have the lemma. □

Lemma 10 *The number of deleted mapping pairs in the construction of M_3 is $O(d^2n^{1/2})$.*

Proof. From Lemma 9, it is seen that $O(d^2)$ nodes in downward paths are deleted. For each deleted node, $O(n^{1/2})$ nodes in the attached small subtrees may also be deleted (recall that the maximum degree is bounded by a constant). Therefore, $O(d^2n^{1/2})$ mapping pairs are deleted in total. □

Then, we estimate the number of deleted mapping pairs in constructing M_4.

Lemma 11 *The number of deleted mapping pairs in the construction of M_4 is $O(d^2n^{1/2} + dn^{3/4})$.*

Proof. The total number of deleted nodes for upward paths of type (A) is

$$O(d) \cdot O(dn^{1/2}) \;=\; O(d^2n^{1/2})$$

because the length of each upward path is at most $2d$ and thus $O(dn^{1/2})$ nodes are deleted per upward path, and there exist $O(d)$ upward paths of type (A).

The total number of deleted nodes for upward paths of type (B-1) is

$$O(d) \cdot \left\{ O(n^{3/4}) + O(dn^{1/2}) \right\} \;=\; O(dn^{3/4} + d^2 n^{1/2})$$

because $O(n^{3/4})$ central edges and $O(dn^{1/2})$ nodes in the top and bottom blocks are deleted, and there exist $O(d)$ upward paths of type (B-1).

The total number of deleted nodes for upward paths of type (B-2) is

$$O(dn^{1/4}) \cdot O(n^{1/2}) \;=\; O(dn^{3/4})$$

because there exist $O(n^{1/2})$ large subtrees in total and at most $O(dn^{1/4})$ nodes are deleted per large subtree in this case.

□

Here, we can assume $d = O(n^{1/4})$, otherwise $\frac{1}{O(n^{3/4})} \cdot ED_T(T_1, T_2) \leq ED_S(ss(T_1), ss(T_2))$ is always satisfied since $ED_T(T_1, T_2)$ is at most $2n$. Therefore, by summing up all the costs, we can see that the cost of M_4 is

$$O(d) + O(dn^{1/2}) + O(d^2 n^{1/2}) + O(d^2 n^{1/2} + dn^{3/4}) \;=\; O(dn^{3/4}),$$

which completes the proof of Theorem 1.

Finally, we consider the time complexity. Transformation of trees to the modified Euler strings can be done in $O(n)$ time since all id's can be computed in $O(n)$ time and we assume that the maximum degree is bounded by a constant. The string edit distance can be computed in $O(n^2)$ time using a simple dynamic programming algorithm. Construction of M_1 can be clearly done in $O(n)$ time. Construction of M_2 can be done in $O(n)$ time since the total number of nodes in small right subtrees is $O(n)$ and we assume that the maximum degree is bounded by a constant. Construction of M_3 can be done in $O(dn)$ time because we only need to examine $O(d)$ pairs of the highest nodes for each of which $O(n)$ time is enough. Construction of M_4 can also be done in $O(dn)$ time since the period of a string can be computed in linear time even for a general alphabet [8] and thus $O(dn)$ time is enough for classifying all upward paths and for identifying periods, and $O(dn^{3/4})$ time is enough for deleting nodes. Therefore, we have:

Corollary 1 *The unit cost edit distance for trees of bounded degree can be approximated within a factor of* $O(n^{3/4})$ *in* $O(n^2)$ *time.*

6 Concluding Remarks

In this paper, we have presented an algorithm for approximating the tree edit distance efficiently using the string edit distance. Though we have modified the Euler strings in order to guarantee the worst case distortion bound, simple use of the Euler strings may work well in practice. Furthermore, string embedding techniques [3, 17] would be directly applied if we simply use the Euler strings. However, the modified Euler strings cannot be directly combined with string embedding techniques because id's depend on two input trees. Thus, development of algorithms for low-distortion embedding for tree edit distance is left as an open problem.

We have assumed that the labels of nodes in trees come from a finite alphabet (i.e., $|\Sigma_T|$ is assumed to be a constant). However, this property was only used to guarantee that $id(v)$'s can be computed in $O(n)$ time. If a general alphabet is used, $id(v)$'s can be computed in $O(n \log n)$ time [11]. Thus, even for the case of a general alphabet, we can obtain the same results except that the time complexity of transformation of trees into strings increases from $O(n)$ to $O(n \log n)$.

We have also assumed that the maximum degree is bounded by a constant. However, it seems difficult to remove this assumption. Thus, removal of this assumption is left as an open problem as well as improvement of the approximation ratio for bounded degree trees.

Acknowledgements

We would like to thank Tetsuji Kuboyama in the University of Tokyo for suggestions of several references and for helpful discussions. We also thank anonymous reviewers for helpful comments that improved the presentation of the paper.

References

[1] T. Akutsu, A relation between edit distance for ordered trees and edit distance for Euler strings, *Information Processing Letters*, 100 (2006), 105–109.

[2] Z. Bar-Yossef, T. Jayram, R. Krauthgamer, and R. Kumar, Approximating edit distance efficiently, *Proc. 45th IEEE Symp. Foundations on Computer Science* (2004), pp. 550–559.

[3] T. Batu, F. Ergun, and S. C. Sahinalp, Oblivious string embeddings and edit distance approximations, *Proc. 17th ACM-SIAM Symp. Discrete Algorithms* (2006), pp. 792–801.

[4] P. Bille, A survey on tree edit distance and related problem, *Theoretical Computer Science*, 337 (2005), 217–239.

[5] W. Chen, New algorithm for ordered tree-to-tree correction problem, *Journal of Algorithms*, 40 (2001), 135–158.

[6] G. Cormode and S. Muthukrishnan, The string edit distance matching problem with moves, *Proc. 13th ACM-SIAM Symp. Discrete Algorithms* (2002), pp. 667–676.

[7] G. Cormode, M. Paterson, S. C. Sahinalp and U. Vishkin, Communication complexity of document exchange, *Proc. 11st ACM-SIAM Symp. Discrete Algorithms* (2000), pp. 197–206.

[8] A. Czumaj and L. Gasieniec, On the complexity of determining the period of a string, *Proc. 11th Symp. Combinatorial Pattern Matching* (2000), pp. 412–422.

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

[10] M. Garofalakis and A. Kumar, XML stream processing using tree-edit distance embedding, *ACM Trans. Database Systems*, 30 (2005), 279–332.

[11] R. Grossi, On finding common subtrees, *Theoretical Computer Science*, 108 (1993), 345–256.

[12] S. Guha, H. V. Jagadish, N. Koudas, D. Srivastava, and T. Yu, Approximate XML joins, *Proc. ACM SIGMOD* (2002), pp. 287–298.

[13] S. Khot and A. Naor, Nonembeddability theorems via Fourier analysis, *Proc. 46th IEEE Symp. Foundations on Computer Science* (2005), pp. 101–110.

[14] P. N. Klein, Computing the edit-distance between unrooted ordered trees, *Proc. 6th European Symp. Algorithms* (1998), pp. 91–102.

[15] R. Krauthgamer and Y. Rabani, Improved lower bounds for embeddings into L_1, *Proc. 17th ACM-SIAM Symp. Discrete Algorithms* (2006), pp. 1010–1017.

[16] S. Muthukrishnan, S. C. Sahinalp, Approximate nearest neighbors and sequence comparison with block operations, *Proc. 32nd ACM Symp. Theory of Computing* (2000), pp. 416–412.

[17] R. Ostrovsky and Y. Rabani, Low distortion embeddings for edit distance, *Proc. 37th ACM Symp. Theory of Computing* (2005), pp. 218–224.

[18] K-C. Tai, The tree-to-tree correction problem, *J. ACM*, 26 (1979), 422–433.

[19] G. Valiente, Algorithms on Trees and Graphs, Springer-Verlag, Berlin Heidelberg, 2002.

[20] R. Yang, P. Kalnis, and A. K. H. Tung, Similarity evaluation on tree-structured data, *Proc. ACM SIGMOD* (2005), pp. 754–765.

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