IEICE TRANS. INF. & SYST., VOL.E93–D, NO.2 FEBRUARY 2010

INVITED PAPER Special Section on Foundations of Computer Science

Tree Edit Distance Problems: Algorithms and Applications to Bioinformatics

Tatsuya AKUTSU[†a)], ***Member***

SUMMARY Tree structured data often appear in bioinformatics. For example, glycans, RNA secondary structures and phylogenetic trees usually have tree structures. Comparison of trees is one of fundamental tasks in analysis of these data. Various distance measures have been proposed and utilized for comparison of trees, among which extensive studies have been done on tree edit distance. In this paper, we review key results and our recent results on the tree edit distance problem and related problems. In particular, we review polynomial time exact algorithms and more efficient approximation algorithms for the edit distance problem for ordered trees, and approximation algorithms for the largest common sub-tree problem for unordered trees. We also review applications of tree edit distance and its variants to bioinformatics with focusing on comparison of glycan structures.

key words: *tree edit distance, dynamic programming, largest common subtree, tree alignment*

1. Introduction

Comparison of two or more objects is one of fundamental tasks in information processing. Extensive studies have been done on comparison of string data and graph structured data. Though a lot of efficient and practical algorithms have been developed for comparison of strings, it is quite difficult to efficiently compare graph structured data because of NP-completeness of subgraph isomorphism [15]. Therefore, subclasses of graphs, for which comparison can be efficiently done, have been studied. Among various classes, extensive studies have been done on trees because tree structured data often appear in bioinformatics and other areas. For example, glycans, RNA secondary structures and phylogenetic trees usually have tree structures. Therefore, comparison of tree structured data is important both from a theoretical viewpoint and from a practical viewpoint.

Various measures have been proposed and studied for comparison of two trees. Among such measures, *tree edit distance* is the most common and well-studied. In particular, extensive studied have been done on tree edit distance between rooted and ordered trees. Beginning from an $O(n^6)$ time algorithm developed by Tai in 1979 [26], several improvements follow, and Demaine et al. developed an $O(n^3)$ time algorithm and showed that it is optimal under a reasonably wide class of algorithms in 2007 [12], where n is the number of nodes in an larger input tree. In order to cope with this lower bound, the author and collaborator have been developed approximation algorithms that works in $o(n^3)$ time [4]. On the other hand, it is known that the edit distance problem for unordered trees is NP-hard [29]. Therefore, several variants of edit distance have been proposed. Among them, tree alignment [20] and tree inclusion [22] are important because these problems can be solved in polynomial time even for unordered trees if input trees are of bounded degree.

Tree edit distance and its variants have been applied to analysis of real data in bioinformatics. Several systems were developed for comparison and similarity search of RNA secondary structures [21] and the author and collaborators developed a web-based tool for similarity search for glycan structures [7].

Motivated by the above mentioned theoretical and practical developments, this paper gives a survey on algorithms and bioinformatics applications of tree edit distance problems. However, it is almost impossible to make a comprehensive survey because a number of variants and special cases have been studied. Therefore, this paper tries to explain fundamental results and algorithms concisely. Of course, an excellent survey has already been done in [10], which gives concise descriptions of major algorithms and their variants. Furthermore, more comprehensive survey and classification were done in [24]. However, in these works, not so much attentions were paid to approximation algorithms or applications to bioinformatics. Since this paper gives more weights to these two aspects, it can be thought as a complement of these two works.

2. Edit Distance

In this section, we briefly review the string edit distance and the tree edit distance. The reason for reviewing the string edit distance is that it is more fundamental, is easier to understand, and has been widely applied to bioinformatics.

2.1 String 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, $s[i \ldots j]$ denotes $s[i] \ldots s[j]$, and $|s|$ denotes the length of s. We may use $s[i]$ to denote both the character and its 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

Manuscript received March 23, 2009.
Manuscript revised June 25, 2009.

[†]The author is with Bioinformatics Center, Institute for Chemical Research, Kyoto University, Kyoto-shi, 611–0011 Japan.
a) E-mail: takutsu@kuicr.kyoto-u.ac.jp

DOI: 10.1587/transinf.E93.D.208

Copyright © 2010 The Institute of Electronics, Information and Communication Engineers

only unit cost operations are considered for the string edit distance in this paper. We use $d_S(s_1, s_2)$ to denote the edit distance between two strings s_1 and s_2.

While edit distance is well-known in computer science, *alignment* is much more common in bioinformatics. 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 *score* of an alignment is given by $score(s'_1, s'_2) = \sum_{i=1}^{l} f(s'_1[i], s'_2[i])$.

An *optimal alignment* is an alignment with the minimum score [†]. Under the unit cost model, $f(x, y) = 0$ if $x = y \neq$ '-', $f(x, y) = 1$ otherwise. It is straight-forward to see that the score of an optimal alignment is equal to the edit distance. For example, consider strings s_1 ="ACGTGGA" and s_2="CGTTCGTA". Then, the following is an optimal alignment.

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

In this case, (a) corresponds to deletion, (b) and (d) correspond to insertions, and (c) corresponds to a substitution. Thus, we have $score(s_1, s_2) = d_S(s_1, s_2) = 4$. It is to be noted that an optimal alignment is not necessarily unique. In the above case, 'T' in s_1 can match to the first 'T' in s_2 in place of the second 'T'. It is well-known that $d_S(s_1, s_2)$ can be computed in $O(|s_1| \cdot |s_2|)$ time by a simple dynamic programming algorithm.

In the above, an optimal alignment was defined as an alignment with the minimum score. However, in bioinformatics, an optimal alignment is usually defined as an alignment with the maximum score [13]. In such a case, the score function is defined in an opposite way: $f(x, x)$ takes a larger value than $f(x, y)$ for $x \neq y$.

2.2 Tree Edit Distance

Let T be a rooted unordered tree where "ordered" means that a left-to-right order among siblings is given in T. We assume that each node v has a label $\ell(v)$ over an alphabet Σ_T. $V(T)$ denotes the set of nodes in T. The size of T is defined as the number of nodes in T and is denoted by $|T|$. For a node $v \in V(T)$, $T - v$ denotes the tree obtained by deleting v from T, $p(v)$ denotes the parent of v, and $chd(v)$ denotes the set of children of v. $T(v)$ denotes the subtree induced by v and its descendants, and $r(T)$ denotes the root of T. For a subtree T' of T, $T - T'$ denotes the tree obtained by deleting all nodes in T' from T. The *depth* of a node v is the length of the path from the root to v and is denoted by $depth(v)$. The *height* of a tree T is the depth of the deepest node in T.

As in the string edit distance, an *edit operation on a tree* T is either a deletion, an insertion, or a substitution (see also Fig. 1):

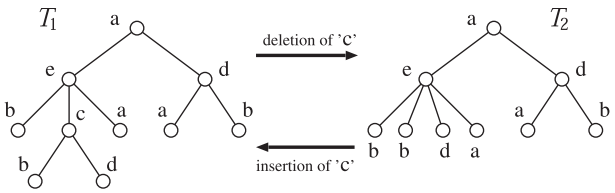

Fig. 1 Insertion and deletion operations.

Deletion: Delete a non-root node v in T with parent u, making the children of v become 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.

We assign a *cost* for each edit operation: $\gamma(a, b)$ denotes the cost of substituting a node with label a to label b, $\gamma(a, \epsilon)$ denotes the cost of deleting a node labeled with a, and $\gamma(\epsilon, a)$ denotes the cost of inserting a node labeled with a.

The *edit distance between two ordered trees* T_1 and T_2 is defined as the cost of the minimum cost sequence of edit operations that transforms T_1 to T_2. We use $d_T(T_1, T_2)$ to denote the edit distance between T_1 and T_2. In this paper, we adopt the following standard assumption so that $d_T(T_1, T_2)$ becomes a distance metric [10]: $\gamma(a, b) \geq 0$ for any $(a, b) \in \Sigma' \times \Sigma'$, $\gamma(a, a) = 0$ for any $a \in \Sigma'$, $\gamma(a, b) = \gamma(b, a)$ for any $(a, b) \in \Sigma' \times \Sigma'$, $\gamma(a, c) \leq \gamma(a, b) + \gamma(b, c)$ for any $(a, b, c) \in \Sigma' \times \Sigma' \times \Sigma'$, where $\Sigma' = \Sigma_T \cup \{\epsilon\}$. We call T_2 a *sub-tree* of T_1 if T_2 is obtained from T_1 only by deletion operations [††].

There exists a close relationship between the edit distance and the *ordered edit distance mapping* (or just *mapping*) [10]. $M \subseteq V(T_1) \times V(T_2)$ is called a mapping if the following conditions are satisfied for any two pairs (u_1, v_1), $(u_2, v_2) \in M$:

(i) $u_1 = u_2$ iff $v_1 = v_2$,
(ii) u_1 is an ancestor of u_2 iff v_1 is an ancestor of v_2,
(iii) u_1 is to the left of u_2 iff v_1 is to the left of v_2.

Let I_1 and I_2 be the sets of nodes in $V(T_1)$ and $V(T_2)$ not appearing in M, respectively. Then, the following relation holds:

$$d_T(T_1, T_2) = \min_M \left\{ \sum_{u \in I_1} \gamma(\ell(u), \epsilon) + \sum_{v \in I_2} \gamma(\epsilon, \ell(v)) \right.$$

[†]We mainly use 'score' for maximization problems in this paper. We used 'score' here because, as mentioned later, string alignment is usually defined as a maximization problem in bioinformatics.

[††]We use 'subtree' for denoting a usual subtree (i.e., a connected subgraph of a tree).

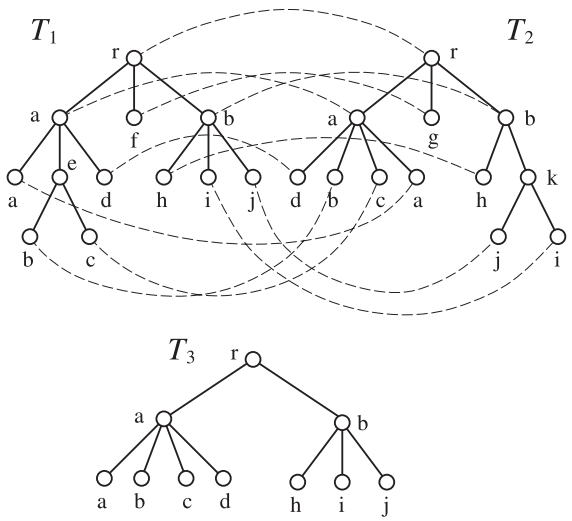

Fig. 2 Mapping (M_{OPT}) with the maximum score between T_1 and T_2 is shown by dashed curves, which correspond to the tree edit distance. T_3 is the largest common sub-tree of T_1 and T_2. It is to be noted that M_{OPT} does not give T_3 because M_{OPT} contains a pair of non-identical labels.

$$+\sum_{(u,v)\in M}\gamma(\ell(u),\ell(v))\Bigg\}.$$

Here we define a *score function* $f(u,v)$ for $(u,v) \in V(T_1) \times V(T_2)$ by $f(u,v) = \gamma(\ell(u),\epsilon) + \gamma(\epsilon,\ell(v)) - \gamma(\ell(u),\ell(v))$. It is seen that $f(u,v) = f(v,u) \geq 0$ holds. It should also be noted that under the unit cost model (i.e., $\gamma(a,b) = 1$ for all $a \neq b$), $f(u,v) = 2$ if $\ell(u) = \ell(v)$, $f(u,v) = 1$ otherwise. Let $score(M)$ be the score of a mapping M defined by $score(M) = \sum_{(u,v)\in M} f(u,v)$. Let M_{OPT} be the mapping with the maximum score. Then, we can see from the definition that the following property holds:

$$d_T(T_1,T_2) = \sum_{u\in V(T_1)}\gamma(\ell(u),\epsilon) + \sum_{v\in V(T_2)}\gamma(\epsilon,\ell(v)) - score(M_{OPT}).$$

If M consists of pairs of nodes with identical labels, the sub-tree obtained by deleting nodes not appearing in M from T_1 is isomorphic to the sub-tree obtained by deleting nodes not appearing in M from T_2. Such a tree is called a *common sub-tree* between T_1 and T_2. The *largest common sub-tree* (LCST, in short) is defined as the common sub-tree with the maximum score [†]. For example, T_3 in Fig. 2 is a sub-tree of T_1 because it is obtained by deleting nodes labeled with 'e' and 'f' from T_1. However, T_3 is not a subtree of T_1 because it is not a part of T_1. LCST can be obtained from M_{OPT} if $f(u,v)$ is appropriately defined [††].

For unordered trees, edit operations are defined in the same way as for ordered trees except that the left-right order need not be preserved. $M \subseteq V(T_1) \times V(T_2)$ is called an *unordered edit distance mapping* (or just a mapping) if only (i) and (ii) of the above mentioned conditions are satisfied for any pair $(u_1,v_1), (u_2,v_2) \in M$. Figure 3 illustrates the difference between the ordered tree mapping and the unordered tree mapping. Suppose that the root of T_1 has two subtrees A and B and the root of T_2 has two subtrees B' and A' in these orders, where A and A' (resp. B and B') are similar to each other, A and B do not have nodes of identical labels, and A is much larger than B. If two trees are regarded as ordered, only a mapping between A and A' is taken into account. Otherwise, mappings between A and A', and between B and B' are taken into account. In general, unordered tree edit distance gives more flexible measure between two trees than ordered tree edit distance.

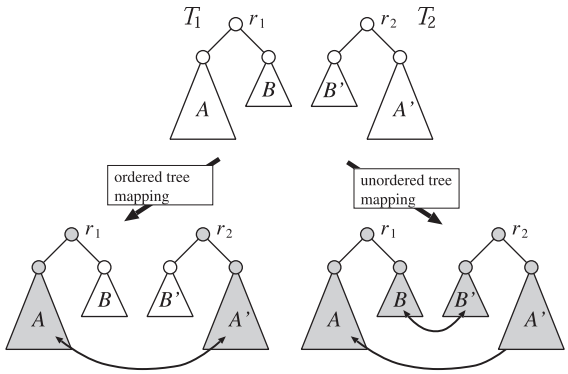

Fig. 3 Difference between ordered tree mapping and unordered tree mapping.

3. Dynamic Programming Algorithms for Ordered Tree Edit Distance

Almost all algorithms for computing the tree edit distance for ordered trees are based on *dynamic programming*. Here we briefly review the general scheme of dynamic programming for tree edit distance, following to [10], [12]. Let $n = |T_1|$ and $m = |T_2|$. We assume w.l.o.g. (without loss of generality) that $n \geq m$. A *forest* is a set of rooted ordered trees and we assume that trees are ordered from left-to-right. For a forest F which is a (not necessarily connected) subgraph of T and a node v in F, $F - v$ and $F - T(v)$ denote the forests obtained by deleting v from F and by deleting $T(v)$ from F, respectively. Then, the tree edit distance can be computed by a dynamic programming algorithm (called Zhang and Shasha's algorithm [28]) based on the following recursion (see also Fig. 4):

$$\delta(F_1,\epsilon) = \sum_{u\in V(F_1)}\gamma(\ell(u),\epsilon),$$

$$\delta(\epsilon,F_2) = \sum_{v\in V(F_2)}\gamma(\epsilon,\ell(v)),$$

$$\delta(F_1,F_2) = \min\begin{cases}\delta(F_1-u,F_2)+\gamma(\ell(u),\epsilon),\\ \delta(F_1,F_2-v)+\gamma(\epsilon,\ell(v)),\\ \delta(F_1-T_1(u),F_2-T_2(v))\\ \quad+\delta(T_1(u)-u,T_2(v)-v)\\ \quad+\gamma(\ell(u),\ell(v)),\end{cases}$$

[†]The largest common subtree problem based on a usual definition of the subtree has also been studied [1].

[††]LCST for ordered trees is generalized for multiple trees in [5].

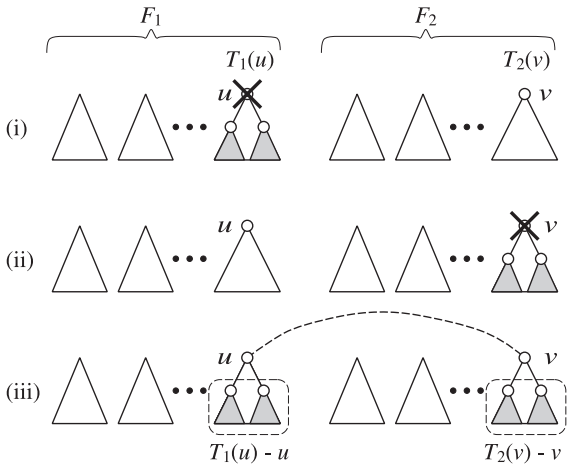

Fig. 4 Illustration for dynamic programming algorithm for ordered edit distance.

where u and v in the third recursion are the roots of the rightmost trees in F_1 and F_2, respectively. The meaning of the third recursion can be seen from Fig. 4. It is straightforward to see that $d_T(T_1, T_2) = \delta(T_1, T_2)$ holds [10], [12], [16]. Once $\delta(F_1, F_2)$s are computed, an optimal mapping can be obtained using a standard *traceback* technique.

Here we briefly analyze the time complexity of the above algorithm, following to [10], [12]. We estimate an upper bound on the number of relevant forests F_1 and F_2 and ignore other details, where we call that a forest is *relevant* if it appears in the recursion. It is seen that every F_1 is obtained by deleting rightmost roots repeatedly from $T_1(v)$ for some v. Since the number of vs is n and the number of forests obtained by repeatedly deleting rightmost nodes is at most $|T_1(v)| \leq n$, we can see that the number of relevant forests F_1 is n^2. Similarly, we can see that the number of relevant forests F_2 is m^2. Since the number of relevant forest pairs is $O(n^2m^2)$, the algorithm works in $O(n^2m^2)$ time.

In the above estimation, many of the same subforests are counted multiple times. Suppose that T_1 contains 7 nodes: r, v_1, v_2, w_1, w_2, w_3, w_4, where v_1 and v_2, w_1 and w_2, and w_3 and w_4 are the children of r, v_1 and v_2, respectively. Then, the subforest consisting of w_1 and w_2 is obtained from both $T_1(v_1)$ and $T_1(r)$ by repeatedly deleting rightmost roots.

3.1 Zhang and Shasha's Algorithm

Zhang and Shasha showed the above mentioned algorithm works in $O(nm \min\{h_1, l_1\} \min\{h_2, l_2\})$ time [28], where h_i and l_i denote the height and the number of leaves of a tree T_i, respectively. In order to prove this bound, they introduced the concept of *keyroots*. The set of keyroots of a tree T is defined by $keyroots(T) = \{v \in V(T) \mid v \text{ has a left sibling}\} \cup \{r(T)\}$.

It is straight-forward to see that every relevant subforest of T_1 is obtained by repeatedly deleting rightmost roots from $T_1(v)$ only for some $v \in keyroots(T_1)$. For a node $v \in V(T)$, the *collapsed depth* of v, denoted by $cdepth(v)$, is defined as the number of keyroot ancestors of v. Let $cdepth(T) = \max_{v \in V(T)} cdepth(v)$. Then, the number of relevant subforests of T_1 is bounded by

$$\sum_{v \in keyroots(T_1)} |T_1(v)| = \sum_{v \in V(T_1)} cdepth(v) \leq |T_1| cdepth(T_1).$$

An analogous result holds for T_2. Since $cdepth(T_i) \leq \min\{h_i, l_i\}$ holds, Zhang and Shasha's algorithm works in $O(nm \min\{h_1, l_1\} \min\{h_2, l_2\})$ time, It is to be noted that this bound is still $O(n^2m^2)$ in the worst case.

3.2 Klein's Algorithm

Klein developed an improved algorithm which works in $O(n^2 m \log m)$ time [23]. The algorithm is based on a recursion similar to the above mentioned one. However, instead of recursing always on the rightmost roots, his algorithm recurses on the leftmost roots if the leftmost subtree of F_1 is smaller than the rightmost subtree of F_1. The correctness of the algorithm is almost obvious because left and right play equivalent roles in the definition of the ordered tree edit distance. Klein showed that the number of relevant forests of T_1 is $O(n \log n)$ whereas the number of relevant forests of T_2 remains $O(m^2)$. We can see that the number of relevant forests of T_2 is $O(m^2)$ because every relevant forest of T_2 can be obtained by deleting leftmost nodes repeatedly and then deleting rightmost nodes repeatedly.

Klein's analysis is based on a technique called *heavy path decomposition*. First, the root is marked as *light*. For each node $v \in V(T_1)$, one of v's children with the maximum number of descendants is chosen and is marked as heavy (tie is broken arbitrary) and all other children are marked as light. For a node v, $ldepth(v)$ denotes the number of light nodes that are proper ancestors of v. For a forest F, $lights(F)$ denotes the set of light nodes in F. It is easy to verify that $ldepth(v) \leq \log |V(F)| + O(1)$ holds for any forest F and node v.

Since lighter subtrees are always chosen in Klein's algorithm, we can see that every relevant subforest is obtained from $T_1(v)$ for some light node v by consecutive deletions. Therefore, the number of relevant forests of T_1 is bounded by

$$\begin{aligned}\sum_{v \in light(T_1)} |T_1(v)| &\leq \sum_{v \in V(T_1)} 1 + ldepth(v) \\ &\leq \sum_{v \in V(T_1)} \log |T_1| + O(1) \\ &= O(|T_1| \log |T_1|) = O(n \log n).\end{aligned}$$

Therefore, Klein's algorithm works in $O(nm^2 \log n)$ time.

3.3 Optimal Decomposition Algorithm

Zhang and Shasha's algorithm recurses always on the rightmost roots, whereas Klein's algorithm recurses on the leftmost roots if the leftmost tree is smaller than the rightmost

tree. These strategies can be generalized: the algorithm can choose left or right at each recursive step (i.e., a strategy is a function from a set of pairs of forests to the set of left and right). Such an algorithm is called a *decomposition algorithm*. Though the strategy of Klein's algorithm depends only on T_1, faster algorithms might be obtained by using a strategy depending on both T_1 and T_2. Demaine et al. developed such an algorithm, which works in $O(nm^2(1 + \log \frac{n}{m}))$ time (it is $O(n^3)$ since $m \leq n$ is assumed) [12]. Furthermore, they showed the matching lower bound for the class of decomposition algorithms.

Theorem 1: [12] The tree edit distance problem for ordered trees can be solved in $O(nm^2(1 + \log \frac{n}{m}))$ time, which is optimal under the class of decomposition algorithms.

4. Approximation of Tree Edit Distance

As mentioned in Sect. 3, the tree edit distance problem takes $\Omega(n^3)$ time for ordered trees under the class of decomposition algorithms. Furthermore, it is NP-hard for unordered trees [29]. Therefore, it is reasonable to try to develop approximation algorithms for these problems. For ordered edit distance, algorithms using reduction to the string edit distance problem were proposed [2], [4], [8]. In the first part of this section, we briefly review these algorithms. For unordered edit distance, to my knowledge, no approximation algorithm with a guaranteed approximation ratio has been developed for the general case [14]. Instead, some approximation algorithms have been proposed for the LCST problem [3], [16]. In the second part of this section, we briefly review these algorithms. In this section, we only consider the unit cost model, where the results are generalized to some extent. For terminologies on approximation algorithms, refer to [27].

4.1 Approximation of Ordered Tree Edit Distance

As mentioned above, approximation algorithms for ordered tree edit distance are based on a reduction to string edit distance. The idea of the reduction is quite simple. We transform input trees T_1 and T_2 into strings $s(T_1)$ and $s(T_2)$, respectively, and then we compute the string edit distance between $s(T_1)$ and $s(T_2)$. In order to transform a tree into a string, we employ *Euler string* [23], which is obtained by traversing a tree using the Euler path and is defined as below.

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 $u = p(v)$. Though the root labels are lost in this case, it does not cause a problem because the roots are not deleted or inserted. In what follows, we assume w.l.o.g. that the roots of two input trees have identical labels.

The depth-first search traversal of T (i.e., visiting children of each node according to their left-to-right order) gives an Euler path beginning from the root and ending at the root where each edge is traversed twice in the opposite directions. 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 sequence, we create the Euler string $s(T)$ of length $2n - 2$. Let $e = \{u, v\}$ be an edge in T, where $u = p(v)$. Suppose that $e_i = (u, v)$ and $e_j = (v, u)$ where $i < j$ should hold. Let $i_1(e) = i$ and $i_2(e) = j$. That is, $i_1(e)$ and $i_2(e)$ denote the first and second positions of e in the Euler path, respectively. Then, we define $s(T)$ by letting $s(T)[i_1(e)] = \ell(e)$ and $s(T)[i_2(e)] = \overline{\ell(e)}$, where $\ell(e)$ is the label of e (see also Fig. 5).

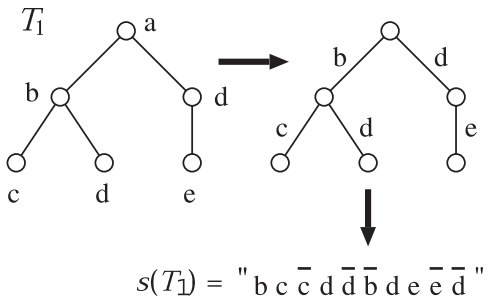

Fig. 5 Construction of an Euler string.

Clearly, it is seen that $s(T)$ can be constructed from T in linear time. Conversely, we can reconstruct T from $s(T)$ in linear time, and $s(T_1) = s(T_2)$ holds if and only if $d_T(T_1, T_2) = 0$ holds [2].

Since the string edit distance can be computed in $O(nm)$ time, $d_S(s(T_1), s(T_2))$ can be computed in $O(nm)$ time including the time for construction of Euler strings. The following theorem estimates the distortion due to this reduction.

Theorem 2: [2] $\frac{d_T(T_1,T_2)}{2h+1} \leq d_S(s(T_1), s(T_2)) \leq 2d_T(T_1, T_2)$ holds under the unit cost model where h is the minimum height of two input trees.

Since the proof of the latter inequality is simple, we show it here. 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 endpoint of e), we associate deletions of $\ell(e)$ and $\overline{\ell(e)}$. For an insertion of e, we associate insertions of $\ell(e)$ and $\overline{\ell(e)}$. For a substitution of e to e', we associate substitutions of $\ell(e)$ and $\overline{\ell(e)}$ to $\ell(e')$ and $\overline{\ell(e')}$, respectively. Then, it is straight-forward to see that the resulting sequence of edit operations transforms $s(T_1)$ to $s(T_2)$ and the cost is $2 \cdot d_T(T_1, T_2)$.

It is known that the above bound is tight up to a constant factor [2] although some improvement of a constant factor is possible by using another reduction [8]. Fig. 6 gives an example such that $d_S(s(T_1), s(T_2)) = 4$ and $d_T(T_1, T_2) = \Theta(h)$.

Theorem 2 means that the tree edit distance can be approximated within a factor of $4h + 2$ in $O(nm)$ time. This approximation ratio is good when the minimum height of two input trees is low (e.g., the ratio is $O(\log n)$ if the height is $O(\log n)$). However, it does not give any meaningful approximation ratio if the height is $O(n)$.

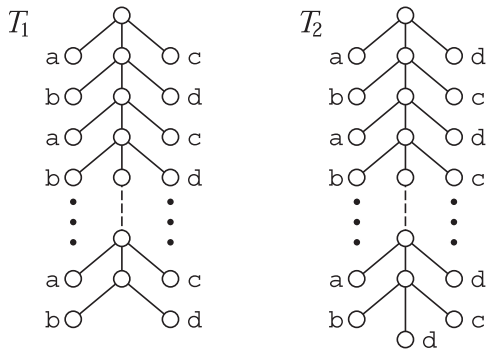

Fig. 6 Example for the case of $ED_T(T_1, T_2) = \Theta(h) \cdot ED_S(s(T_1), s(T_2))$ [2].

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. 6. Suppose that label "ac" is assigned to each edge just above each node having children labeled 'a' and 'c'. Similarly, suppose that labels "bd", "ad", and "bc" are assigned to appropriate edges. Then, $d_S(s(T_1), s(T_2)) = \Theta(h)$ should hold. However, changes of labels should be performed carefully so that the distortion does not become too large. Here, we briefly review such a method of changing labels [4].

A subtree rooted at v is called *large* if $|T(v)| > \alpha$, where $\alpha = n^{1/2}$. Otherwise, it is called *small*. We call w_i a *special node* if $|T(w_i)| \leq \alpha$ and $|T(v)| > \alpha$ where $v = p(w_i)$. Let v be a node in T, $u = p(v)$, and $chd(v) = \{w_1, \ldots, w_k\}$. If none of w_is are special, the original label of v is assigned to edge $\{u, v\}$. Otherwise, $\{u, v\}$ is called a *special edge*, to which a new integer label is given in the following way. Let $w_{i_1}, \ldots, w_{i_h}$ (resp. $w'_{i_1}, \ldots, w'_{i_l}$) be the special children of v in T_1 (resp. v' in T_2). Then, we give integer labels to special edges so that v and v' have an identical label if and only if $\ell(v) = \ell(v')$, $h = l$, and $T_1(w_{i_j})$ is isomorphic to $T_2(w'_{i_j})$ for all $j = 1, \ldots, h$. 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 $\{p(v), v\}$. Let $ss(T_1)$ and $ss(T_2)$ be the resulting Euler strings. The following relation was proven in [4], which gave an $O(nm)$ time $O(n^{3/4})$ approximation algorithm for the edit distance problem between ordered trees of bounded degree.

Theorem 3: [4] $\frac{1}{cn^{3/4}} \cdot d_T(T_1, T_2) \leq d_S(ss(T_1), ss(T_2)) \leq 6 \cdot d_T(T_1, T_2)$ holds for ordered trees of bounded degree with a fixed Σ_T under the unit cost model, where c is some positive constant.

4.2 Approximation of Largest Common Sub-tree

Though the edit distance problem can be solved in $O(n^3)$ time for ordered trees, it is NP-hard for unordered trees. Furthermore, it was proven to be MAX SNP-hard [29] even for unordered trees of bounded degree. To my knowledge, no algorithm with a guaranteed approximation ratio has been developed for the general case of unordered trees [14].

It is to be noted that the recursion presented in Sect. 3 can be modified for unordered trees as below.

$$\delta(F_1, \epsilon) = \sum_{u \in V(F_1)} \gamma(\ell(u), \epsilon),$$

$$\delta(\epsilon, F_2) = \sum_{v \in V(F_2)} \gamma(\epsilon, \ell(v)),$$

$$\delta(F_1, F_2) = \min \begin{cases} \min\limits_{u \in r(F_1)} \{\delta(F_1 - u, F_2) + \gamma(\ell(u), \epsilon)\}, \\ \min\limits_{v \in r(F_2)} \{\delta(F_1, F_2 - v) + \gamma(\epsilon, \ell(v))\}, \\ \min\limits_{(u,v) \in r(F_1) \times r(F_2)} \{ \\ \quad \delta(F_1 - T_1(u), F_2 - T_2(v)) \\ \quad +\delta(T_1(u) - u, T_2(v) - v) \\ \quad +\gamma(\ell(u), \ell(v))\}, \end{cases}$$

where $r(F_i)$ denotes the set of the roots of trees in F_i. However, in this case, the resulting dynamic programming algorithm does not work in polynomial time because an exponential number of subforests appear in the recursion. Though it is possible to obtain polynomial time algorithms based on this recursion in some restricted cases [16], [25], they do not cover wide classes of unordered trees.

LCST can also be defined for unordered trees based on an *unordered edit distance mapping* (which is also called as a mapping when there is no confusion). $M \subseteq V(T_1) \times V(T_2)$ is called a mapping if the following conditions are satisfied for any two pairs $(u_1, v_1), (u_2, v_2) \in M$:

(i) $u_1 = u_2$ iff $v_1 = v_2$,
(ii) u_1 is an ancestor of u_2 iff v_1 is an ancestor of v_2.

It is to be noted that these two conditions are the same as the first two conditions for an ordered edit distance mapping. Using this mapping, LCST is defined in the same way as in the case of ordered trees. It is known that the LCST problem is also MAX SNP-hard even for unordered trees of bounded degree [29].

Different from the edit distance problem, some approximation algorithms are known for the LCST problem for unordered trees. In particular, $2h$-approximation algorithm and $O(\log^2 n)$-approximation algorithm are known [16], where h is the height of a shorter input tree. Here, we briefly review these algorithms.

4.3 $2h$-approximation

The $2h$-approximation algorithm is quit simple and its pseudocode [16] is given below, where we assume w.l.o.g. that the root of T_1 matches to the root of T_2.

for all depth d of T_1 **do**
  Let I be the set of nodes of T_1 at depth d
  Let $F := T_2$ and $A := \{\}$
  repeat until F is empty **do**
    Pick any leaf v of F
    Delete v from F
    if $\ell(v) = \ell(u)$ for some $u \in I$ **then**

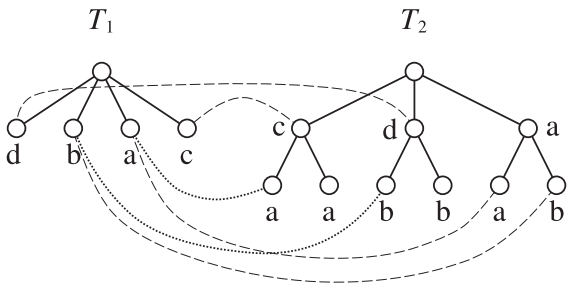

Fig. 7 Dotted curves and dashed curves correspond to approximate mapping and optimal mapping, respectively.

Add v to A
Delete u from I
Delete ancestors of v from F

For each depth d of T_1, this algorithm greedily and repeatedly matches a leaf of T_2 to a node of depth d in T_1. If a leaf v of T_2 is matched to some node, v and its ancestors are deleted. Figure 7 illustrates how the algorithm works for the case of $d = 1$.

Here, we briefly analyze the approximation ratio. If the height of T_1 is 1. it is seen that each node v added to a mapping can eliminate at most two nodes in I from the optimal mapping: u, and a node matched to some ancestor of v. Thus, we have the approximation ratio of 2, from which the approximation ratio of $2h$ follows for the general case.

In the above analysis, one match can eliminate two matches. If two matches eliminate at most three matches, we can have the approximation ratio of $1.5h$. We have developed such an algorithm by utilizing weighted bipartite matching [3].

4.4 $\log^2 n$-approximation

If both trees are linear chains (i.e., the outdegree of every non-leaf node is 1), the LCST problem can be reduced to the string edit distance problem and thus can be solved in $O(nm)$ time. Furthermore, we can find an optimal mapping between two forests in polynomial time using weighted bipartite matching if each forest consists of linear chains: assign the size of LCST between two linear chains as a weight to an edge corresponding to this pair of chains.

Using the heavy chain decomposition repeatedly, each tree can be decomposed into a set of $\log n$ forests of linear chains. By computing an optimal mapping between every pair of forests and taking the maximum one, we can have the approximation ratio of $\log^2 n$ [16]. This can be easily generalized to the case of multiple trees (t trees) and the approximation ratio of $t \log^t n$ is obtained, where an optimal weighted bipartite matching is replaced by an approximate weighted t-dimensional matching [16].

5. Tree Alignment

Since the tree edit distance problem requires $\Omega(n^3)$ time (under the class of decomposition algorithms) for ordered trees and is NP-hard for unordered trees, special cases and other distance measures have been studied. Since *tree alignment* is well-known among these variants [20], we briefly review it here.

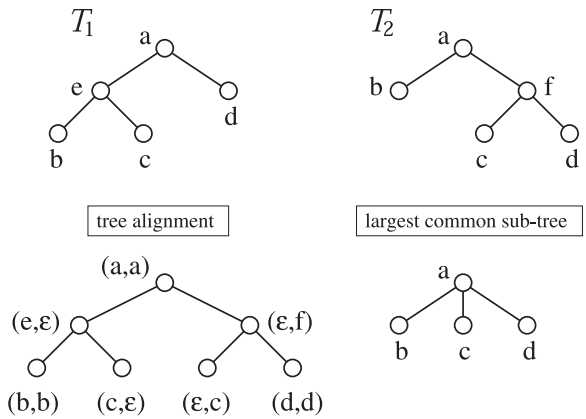

Fig. 8 Tree alignment versus largest common sub-tree.

An *alignment* A between T_1 and T_2 is obtained by inserting nodes labeled with ϵ, which corresponds to a gap symbol in string alignment, into T_1 and T_2 so that the resulting trees become isomorphic ignoring labels, and then overlaying the resulting trees. For a node v in A having label pair (x, y), the cost is defined as $\gamma(x, y)$. The cost of A is the sum of costs of all nodes. An *optimal tree alignment* between T_1 and T_2 is an alignment between T_1 and T_2 with the minimum cost. The cost of an optimal tree alignment is denoted by $\alpha(T_1, T_2)$.

Tree alignment is regarded as a restricted case of edit distance in which all insertions must be performed before any deletions. Thus, we have $\delta(T_1, T_2) \leq \alpha(T_1, T_2)$. Figure 8 gives an example explaining these facts. For T_1 and T_2 in this figure, an optimal tree alignment is obtained by inserting two nodes into T_1 and two nodes into T_2 and thus the resulting cost is 4 under the unit cost model. On the other hand, T_2 is obtained by deleting a node labeled 'e' from T_1 and then inserting a node labeled 'f' and thus the resulting cost is 2. From these, we can see $\delta(T_1, T_2) = 2 \leq 4 = \alpha(T_1, T_2)$. It is also seen from Fig. 8 that an optimal tree alignment is quite different from LCST. It is interesting to note that the minimum cost sequence of edit operations can be modified with preserving the cost so that all deletion operations are performed before any insertion operations.

An optimal tree alignment for ordered trees can be computed in $O(nm(d_1 + d_2)^2)$ time using a dynamic programming algorithm [20], where d_1 and d_2 are the maximum degrees of T_1 and T_2, respectively. Therefore, it is faster than the optimal decomposition algorithm for edit distance if trees are of bounded degree. Though the tree alignment problem is MAX SNP-hard for unordered trees, it can be solved in $O(nm)$ time if the maximum degrees of both trees are bounded by a constant. Therefore, especially for unordered trees, tree alignment might be used as an alternative to tree edit distance. However, it is known that $\alpha(T_1, T_2)$

does not satisfy the conditions of distance metric. Therefore, we should be careful when using tree alignment.

Another interesting variant of tree edit distance is *tree inclusion*. It asks whether T_1 is isomorphic to a sub-tree of T_2. That is, it asks whether T_1 is obtained only by deletion operations from T_2. Though the tree inclusion problem is NP-hard for unordered trees, it can be solved in $O(nm)$ time if the maximum degree of T_1 is bounded by a constant [22].

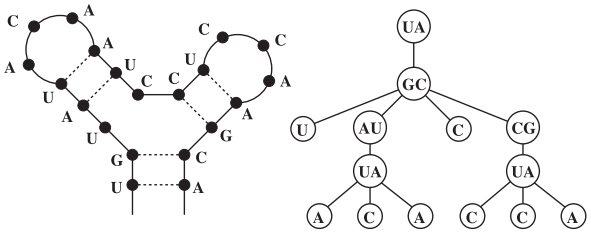

Fig. 9 RNA secondary structure and its tree representation.

6. Applications to Bioinformatics

Several kinds of data appearing in bioinformatics can be represented as rooted trees. For example, RNA secondary structures can be represented as rooted ordered trees. RNA (Ribonucleic acid) is a molecule consisting of a chain of four types of nucleotides (bases): A (adenine), C (cytosine), G (guanine), and U (uracil). In contrast to DNA, RNA usually occurs as a single-stranded molecule. In many cases, RNA molecules are folded into their own three-dimensional structures. However, it is difficult to determine, analyze or predict RNA three-dimensional structures. On the other hand, important features of an RNA three-dimensional structure are well-described by a set of base pairs that are connected by hydrogen bonds. This set of base pairs is called an *RNA secondary structure*, and many of these structures can be represented as rooted ordered trees. Figure 9 gives an artificial example of an RNA secondary structure and its tree representation. Once RNA secondary structures are represented as rooted ordered trees, we can apply algorithms for tree edit distance. However, it might be more useful not to directly use tree edit distance but to use variants that are specialized to RNA secondary structures. A number of such variants have been proposed and several systems have been developed [9], [21]

Another important example of tree structured data is a phylogenetic tree. A *phylogenetic tree* is a rooted unordered tree that represents a history of evolution of various biological species. Since topologies of phylogenetic trees depend on reconstruction algorithms, parameters used in the algorithms and data provided for the algorithms, it is important to compare various phylogenetic trees. However, tree edit distance or its variants are not usually used for comparison of phylogenetic trees because leaves in phylogenetic trees have special and important meanings (each leaf corresponds to one species). Thus, various algorithms that are specialized for comparison of phylogenetic trees have been developed [18].

6.1 Application to Glycan Search

Glycans, which are also known as carbohydrate sugar chains, are important biomolecules. Glycans are quite vital for the development and functioning of multicellular organisms, and are generally found on exterior surface of cells. Some glycans play an important role in cell-cell interactions, and some others play an important role in protein folding cooperating with Chaperone proteins.

The importance of glycans has been recognized in the field of bioinformatics since the beginning of the 21st century and then various studies have been done. Based on some earlier work, a new publically available database called KEGG Glycan has been recently constructed [17]. Along with the construction of the database, it was recognized that there was no tool for searching similar glycan structures. Therefore, a search tool named KCaM (KEGG Carbohydrate Matcher) was developed [7] along with algorithms for comparison of glycan structures [6]. In this subsection, we briefly review these algorithms.

It is known that most glycans have tree structures. In some cases, glycans can be regarded as rooted ordered trees. However, more flexible matchings can be done if glycans are regarded as rooted unordered trees (i.e., the possibility of missing similar structures is low if glycans are regarded as unordered trees).

Though there exist several variants, we explain the global glycan alignment algorithm here. Since the tree edit distance problem for unordered trees is NP-hard, it is quite difficult to develop practical algorithms for glycan alignment. Though an A* algorithm was developed for unordered tree edit distance [19], it takes a lot of time for large structures and does not seem to be suitable for use of similarity searching against many structures (e.g., 10,000 structures). Tree alignment might be used because unordered tree alignment can be computed in $O(nm)$ time for bounded degree trees. However, the constant factor depending on the degree bound and hidden in $O(nm)$ is large. Furthermore, similarity score should be computed instead of distance because use of score is much more common in bioinformatics [13]. Thus, we did not adopt edit distance or tree alignment and developed another scoring scheme.

We modified deletion and insertion operations in tree edit so that the resulting algorithm works in polynomial time. Since insertion is symmetric to deletion, we only explain the deletion operation. In tree edit, all children of the deleted node u become children of the parent of u. However, in glycan alignment, only one child can be a child of the parent and the other children are deleted along with their descendants (see Fig. 10). This restriction was introduced not from a biological viewpoint but from a practical constraint that the algorithm should work efficiently. The surviving child is chosen so that the total score, which is defined below, is maximized under this restriction.

As in tree edit, we can relate the score with a kind of

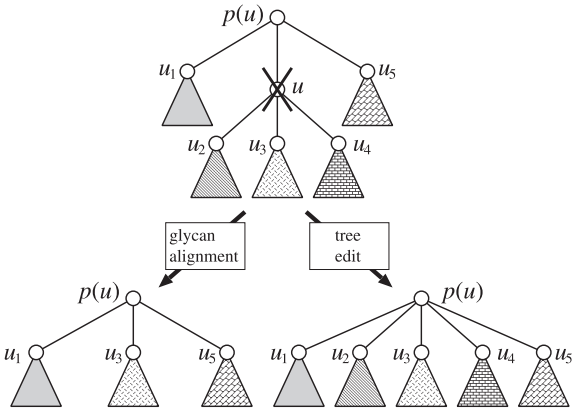

Fig. 10 Difference of deletion operations between glycan alignment and tree edit.

mapping or alignment. $A \subseteq V(T_1) \times V(T_2)$ is called a *glycan alignment* if the following conditions are satisfied for any pair $(u_1, v_1), (u_2, v_2) \in A$:

(i) $u_1 = u_2$ iff $v_1 = v_2$,
(ii) u_1 is an ancestor of u_2 iff v_1 is an ancestor of v_2,
(iii) if $u_1 \in V(T_1)$ (resp. $v_1 \in V(T_2)$) does not appear in A, at most one child of u_1 (resp. v_1) and its descendants can appear in A.

It should be noted that condition (iii) is not included in the tree edit mapping. This condition is introduced according to the modification of edit operations. We define the score of an alignment A by

$$\sum_{(u,v)\in A} \gamma(u,v) + \sum_{u\in V(T_1)-A_1} \gamma(u,\epsilon) + \sum_{v\in V(T_2)-A_2} \gamma(\epsilon,v),$$

where A_i denotes the set of nodes of T_i appearing in A.

The score of an optimal glycan alignment A is computed by a dynamic programming algorithm based on the following recursion.

$$\sigma(u,\epsilon) = \sum_{u_i\in T_1(u)} \gamma(u_i,\epsilon),$$

$$\sigma(\epsilon,v) = \sum_{v_i\in T_2(v)} \gamma(\epsilon,v_i),$$

$$\sigma(u,v) = \max\{\sigma_1(u,v), \sigma_2(u,v), \sigma_3(u,v)\},$$

$$\sigma_1(u,v) = \max_{u_i\in chd(u)} \left\{ \begin{array}{l} \sigma(u_i,v) + \gamma(u,\epsilon) \\ + \sum\limits_{u_j\in chd(u)-\{u_i\}} \sigma(u_j,\epsilon) \end{array} \right\},$$

$$\sigma_2(u,v) = \max_{v_i\in chd(v)} \left\{ \begin{array}{l} \sigma(u,v_i) + \gamma(\epsilon,v) \\ + \sum\limits_{v_j\in chd(v)-\{v_i\}} \sigma(\epsilon,v_j) \end{array} \right\},$$

$$\sigma_3(u,v) = \gamma(u,v) + \max_{M\in\mathcal{M}(u,v)} \left\{ \begin{array}{l} \sum\limits_{(u_i,v_j)\in M} \sigma(u_i,v_j) \\ + \sum\limits_{u_k\in chd(u)-M_1} \sigma(u_k,\epsilon) \\ + \sum\limits_{v_k\in chd(v)-M_2} \sigma(\epsilon,v_k) \end{array} \right\}.$$

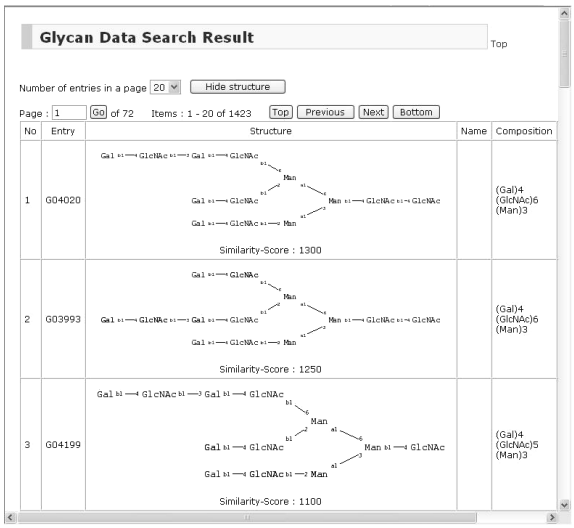

Fig. 11 Snapshot of KCaM.

In the above, $\mathcal{M}(u,v)$ denotes the set of all possible matchings between $chd(u)$ and $chd(v)$, and M_i denotes the set of nodes of T_i appearing in M. $\sigma(u,v)$ corresponds to the score between $T_1(u)$ and $T_2(v)$ and thus the score between T_1 and T_2 is given by $\sigma(r(T_1), r(T_2))$. The meaning of $\gamma(u,v)$ is a bit different from that in tree edit and is much more similar to those of *score matrices* which are widely used for comparison of biological sequences [13]. In this case, $\gamma(u,v)$ represents the similarity between nodes u and v, not the distance between u and v. Each of $\gamma(u,\epsilon)$ and $\gamma(\epsilon,v)$ denotes the penalty (usually, it takes negative value) of deleting a node.

How to define values of $\gamma(u,v)$ is not a trivial task. We defined these values via discussion with researchers having deep biological knowledge. Since the precise definition of $\gamma(u,v)$ includes some biological details [6], we do not give it in this paper. The meanings of three terms appearing in the third recursion are as follows: $\sigma_1(u,v)$ corresponds to deletion of u, $\sigma_2(u,v)$ corresponds to deletion of v, and $\sigma_3(u,v)$ corresponds to matching between u and v. In order to compute $\sigma_3(u,v)$, a maximum score matching between the children of u and the children of v must be computed. Since this matching can be computed in polynomial time, the overall algorithm works in polynomial time (though a simple exhaustive algorithm was employed for computing this matching in the current implementation for the purpose of practical efficiency).

The above mentioned algorithm and its variants were implemented in the KCaM web server (see also Fig. 11) [7], on which similarity search against more than 10,000 glycan structures can be done in several or several tens of seconds. This fact suggests that the developed algorithms are quite useful in practice.

7. Concluding Remarks

In this survey paper, we overviewed edit distance problems

Table 1 Results on tree edit distance and related problems.

Problem	Type	Time/ Approximation
tree edit	ordered	$\Theta(n^3)$ time [12]
		$O(n^{3/4})$ approx. in $O(n^2)$ time [4]
tree edit	unordered	MAX SNP-hard [29]
LCST	ordered	$\Theta(n^3)$ time [12]
LCST	unordered	MAX SNP-hard [29]
		$O(\log^2 n)$ approx. [16]
tree alignment	ordered and bounded degree	$O(n^2)$ time [20]
tree alignment	unordered	MAX SNP-hard [20]
tree alignment	unordered and bounded degree	$O(n^2)$ time [20]
tree inclusion	ordered	$O(n^2)$ time [22]
tree inclusion	unordered	NP-hard [22]
tree inclusion	unordered and bounded degree	$O(n^2)$ time [22]

The result in [4] holds only for bounded degree trees. The results in [29] hold even for bounded degree trees.

and algorithms for ordered trees and for unordered trees. The surveyed results are summarized in Table 1. We also reviewed their applications to bioinformatics problems. These applications suggest that tree edit is important not only from a theoretical viewpoint but also from a practical viewpoint. Though we have not much considered restricted versions of tree edit distance, various kinds of restricted classes of trees and/or edit operations have been studies (see [10], [24] for the details). Here we briefly discuss restrictions of degrees and heights since these restrictions were considered in this paper. The assumption of bounded degree is reasonable when we consider chemical compounds having tree structures and glycans because the maximum degrees of these structures are clearly bounded by some constants. The assumption of bounded height is reasonable when we consider XML documents because the heights of trees representing XML documents are low in many cases [3].

From a theoretical viewpoint, to my knowledge, several important open problems remain. For edit distance between ordered trees, an optimal $O(n^3)$ time decomposition algorithm was developed [12]. However, it might be possible to develop an $o(n^3)$ time algorithm that does not belong to the class of decomposition algorithms. Deciding the existence of such an algorithm is left as an important open problem. Another open problem is to improve the approximation ratio of [4] because it seems that this ratio is far from optimal. For edit distance between unordered trees, no approximation algorithm has been known for the general case [14]. Therefore, development of such an algorithm is left as an important open problem. For the largest common sub-tree problem, improvement of $O(\log^2 n)$ approximation ratio [16] remains as an open problem.

Further studies should also be done from a practical viewpoint. Though an A* algorithm was proposed for the unordered tree edit distance problem [19], it cannot be applied to comparison of large structures. Since it is based on a simple heuristic, there exists much room for development of much faster algorithms. Although trees cover several types of data in bioinformatics, comparison of other types of data (e.g., metabolic networks, protein-protein interaction networks, chemical compounds) is becoming important. Therefore, development of appropriate distance measures and efficient algorithms for such kinds of data is important. Of course, a lot of studies have been done on matching of graphs using various kinds of meta-heuristics [11]. However, existing methods are not yet satisfactory (especially for comparison of large biological networks) and thus further studies should be done. The methodologies developed in tree edit might be useful for such a development.

Acknowledgments

I would like to thank all collaborators in the works mentioned in this paper.

References

[1] T. Akutsu and M.M. Halldórsson, "On the approximation of largest common subtrees and largest common point sets," Theor. Comput. Sci., vol.233, pp.33–50, 2000.

[2] T. Akutsu, "A relation between edit distance for ordered trees and edit distance for Euler strings," Inf. Process. Lett., vol.100, pp.105–109, 2006.

[3] T. Akutsu, D. Fukagawa, and A. Takasu, "Improved approximation of the largest common subtree of two unordered trees of bounded height," Inf. Process. Lett., vol.109, pp.165–170, 2008.

[4] T. Akutsu, D. Fukagawa, and A. Takasu, "Approximating tree edit distance through string edit distance," Algorithmica, to appear.

[5] A. Amir, T. Hartman, O. Kapah, B.R. Shalom, and D. Tsur, "Generalized LCS," Theor. Comput. Sci., vol.409, pp.438–449, 2008.

[6] K.F. Aoki, A. Yamaguchi, Y. Okuno, T. Akutsu, N. Ueda, M. Kanehisa, and H. Mamitsuka, "Efficient tree-matching methods for accurate carbohydrate database queries," Genome Informatics, vol.14, pp.134–143, 2003.

[7] K.F. Aoki, A. Yamaguchi, N. Ueda, T. Akutsu, H. Mamitsuka, S. Goto, and M. Kanehisa, "KCaM (KEGG Carbohydrate Matcher): A software tool for analyzing the structure of carbohydrate sugar chains," Nucleic Acids Research, vol.32, pp.W267–W272, 2004.

[8] T. Aratsu, K. Hirata, and T. Kuboyama, "Approximating tree edit distance through string edit distance for binary tree codes," Proc. 35th Conf. Current Trends in Theory and Practice of Computer Science (SOFSEM 2009), pp.93–104, 2009.

[9] R. Backofen, S. Chen, D. Hermelin, G.M. Landau, M.A. Roytberg, O. Weimann, and K. Zhang, "Locality and gaps in RNA comparison," J. Computational Biology, vol.14, pp.1074–1087, 2007.

[10] P. Bille, "A survey on tree edit distance and related problem," Theor. Comput. Sci., vol.337, pp.217–239, 2005.

[11] D. Conte, P. Foggia, C. Sansone, and M. Vento, "Thirty years of graph matching in pattern recognition," Int. J. Pattern Recognit. Artif. Intell., vol.18, pp.265–298, 2004.

[12] 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 (ICALP 2007), pp.146–157, 2007.

[13] R. Durbin, S. Eddy, K. Krogh, and G. Mitchison, Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids, Cambridge University Press, Cambridge, UK, 1998.

[14] D. Fukagawa, T. Akutsu, and A. Takasu, "Constant factor approximation of edit distance of bounded height unordered trees," Proc. 16th String Processing and Information Retrieval Symposium, pp.7–17, 2009.

[15] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Co., New York, 1979.
[16] M.M. Halldórsson and K. Tanaka, "Approximation and special cases of common subtrees and editing distance," Proc. 7th International Symposium on Algorithms and Computation (ISAAC 96), pp.75–84, 1996.
[17] K. Hashimoto, S. Goto, S. Kawano, K.F. Aoki-Kinoshita, N. Ueda, M. Hamajima, T. Kawasaki, and M. Kanehisa, "KEGG as a glycome informatics resource," Glycobiology, vol.16, pp.63R–70R, 2006.
[18] V.T. Hoang and W-K. Sung, "Fixed parameter polynomial time algorithms for maximum agreement and compatible supertrees," Proc. 25th International Symposium on Theoretical Aspects of Computer Science (STACS 2008), pp.361–372, 2008.
[19] Y. Horesh, R. Mehr, and R. Unger, "Designing an A* algorithm for calculating edit distance between rooted-unordered trees," J. Computational Biology, vol.13, pp.1165–1176, 2006.
[20] T. Jiang, L. Wang, and K. Zhang, "Alignment of trees - an alternative to tree edit," Theor. Comput. Sci., vol.143, pp.137–148, 1995.
[21] M. Khaladkar, V. Bellofatto, J. T-L. Wang, B. Tian, and B.A. Shapiro, "RADAR: a web server for RNA data analysis and research," Nucleic Acids Research, vol.35, pp.W300–W304, 2007.
[22] P. Kilpeläinen and H. Mannila, "Ordered and unordered tree inclusion," SIAM J. Comput., vol.24, pp.340–356, 1995.
[23] P.N. Klein, "Computing the edit-distance between unrooted ordered trees," Proc. 6th European Symposium on Algorithms (ESA 98), pp.91–102, 1998.
[24] T. Kuboyama, Matching and Learning in Trees, Doctoral Thesis, University of Tokyo, 2007.
[25] D. Shasha, J.T-L. Wang, K. Zhang, and F.Y. Shih, "Exact and approximate algorithms for unordered tree matching," IEEE Trans. Syst., Man Cybern., vol.24, no.4, pp.668–678, 1994.
[26] K-C. Tai, "The tree-to-tree correction problem," J. ACM, vol.26, pp.422–433, 1979.
[27] V.V. Vazirani, Approximation Algorithms, Springer, Berlin, 2001.
[28] K. Zhang and D. Shasha, "Simple fast algorithms for the editing distance between trees and related problems," SIAM J. Comput., vol.18, pp.1245–1262, 1989.
[29] K. Zhang and T. Jiang, "Some MAX SNP-hard results concerning unordered labeled trees," Inf. Process. Lett., vol.49, pp.249–254, 1994.

Tatsuya Akutsu received his M.Eng degree in Aeronautics in 1986 and a Dr. Eng. degree in Information Engineering in 1989 both from University of Tokyo, Japan. From 1989 to 1994, he was with Mechanical Engineering Laboratory, Japan. He was an associate professor in Gunma University from 1994 to 1996 and in Human Genome Center, University of Tokyo from 1996 to 2001 respectively. He joined Bioinformatics Center, Institute for Chemical Research, Kyoto University, Japan as a professor in Oct. 2001. His research interests include bioinformatics and discrete algorithms.