An A* Algorithm for Computing Edit Distance between Rooted Labeled Unordered Trees[⋆]

Shoichi Higuchi[1], Tomohiro Kan[1], Yoshiyuki Yamamoto[1], and Kouichi Hirata[2]

[1] Graduate School of Computer Science and Systems Engineering
[2] Department of Artificial Intelligence
Kyushu Institute of Technology
Kawazu 680-4, Iizuka 820-8502, Japan
{syou_hig,kan,yamamoto,hirata}@dumbo.ai.kyutech.ac.jp

Abstract. In this paper, we design an A* algorithm for computing the edit distance between rooted labeled unordered trees. First, we introduce some *lower bounding functions* that provide the constant factor lower bounds on the edit distance. Then, by using the lower bounding functions as a heuristic function, we design the A* algorithm as the best-first search for the *edit distance search tree*. Finally, we give experimental results for the A* algorithm.

1 Introduction

Rooted labeled unordered trees (*trees*, for short) are rooted trees whose nodes are labeled and in which only ancestor relationship are significant. Such trees arise naturally in many fields such as glycan data or phylogenetic trees in bioinformatics, chemical compounds and their properties in chemistry, object representation and recognition in computer vision, and so on (*cf.*, [6]). For many such applications, it is necessary to compare tree by some meaningful distance measure.

The most famous distance measure between trees is the *edit distance* [1,7,8,11]. The edit distance between trees (that we sometimes call the *unordered tree edit distance*) is formulated as the minimum cost to transform a tree to another tree by applying *edit operations* of *substitutions*, *deletions* and *insertions* to trees. However, it is known that the problem of computing the unordered tree edit distance is intractable, that is, NP-hard [8,10] and MAX SNP-hard [3,9].

In order to compare two phylogenetic trees, Horesh *et al.* [4] have developed an A* *algorithm* for computing the edit distance between rooted *unlabeled* unordered trees, which is also an intractable problem. This A* algorithm uses three *lower bounding functions* that provide constant factor lower bounds on the unordered tree edit distance, including the *degree histogram L_1-distance* introduced by Kailing *et al.* [5]. Note that Kailing *et al.* [5] have introduced not only the

[⋆] This work is partially supported by Grand-in-Aid for Scientific Research 20500126, 21500145 and 22240010 from the Ministry of Education, Culture, Sports, Science and Technology, Japan.

degree histogram L_1-distance but also the *label histogram L_1-distance* as lower bounding functions.

Motivated by this A* algorithm [4], in this paper, we design an A* algorithm for computing the edit distance between rooted *labeled* unordered trees. First, we use not only three lower bounding functions introduced by Horesh *et al.* [4], that is, the difference of the number of nodes [4], the degree histogram L_1-distance [4,5] and the degree histogram L_∞-distance [4] but also additionally two lower bounding functions, that is, the label histogram L_1-distance [5] and the label histogram L_∞-distance. Next, we introduce the *edit distance search tree* to transform the problem of finding the shortest path in a connected graph for the standard A* algorithm to one of computing the unordered tree edit distance. Then, by setting the heuristic function to the maximum value of the above 5 lower bounding functions, we design the A* algorithm as the best-first search for the edit distance search tree with constructing just necessary branches.

Finally, we implement the A* algorithm and give experimental results for glycan data. Then, we compare the running time for computing the unordered tree edit distance by the A* algorithm with one by the exhaustive search algorithm designed by Shasha *et al.* [6] and the clique-based algorithm designed by Fukagawa *et al.* [2]. Furthermore, we evaluate the effect of 5 lower bounding functions in the execution of the A* algorithm.

2 Preliminaries

A *tree* is a connected graph without cycles. For a tree $T = (V, E)$, we denote V and E by $V(T)$ and $E(T)$, respectively. Also the *size* of T is $|V|$ and denoted by $|T|$. We sometimes denote $v \in V(T)$ by $v \in T$. A *rooted tree* is a tree with one node r chosen as its *root*. Here, we denote the root of a rooted tree T by $r(T)$. We denote the (complete) subtree of T rooted at $v \in T$ by $T(v)$.

For each node v in a rooted tree T with the root r, let $UP_T(v)$ be the unique path from v to r. The *parent* of $v(\neq r)$ is its adjacent node on $UP_T(v)$. The parent of the root r is undefined. We say that u is a *child* of v if v is the parent of u. Two nodes with the common parent are called *siblings*. A *leaf* is a node having no children.

Furthermore, the *depth* of v is the number of edges in the path from v to $r(T)$, that is, $|UP_T(v)| - 1$, and the *depth* of T is the maximum depth for every node in T, which we denote by $d(T)$. Also the *degree* of v is the number of the children of v, and the *degree* of T is the maximum degree for every node in T.

We say that a rooted tree is *ordered* if a left-to-right order among siblings is given; *Unordered* otherwise. Also we say that a tree is *labeled* over Σ if each node is assigned a symbol from a fixed finite alphabet Σ, where we denote the label of a node v by $l(v)$. We sometimes identify v with $l(v)$. In this paper, we call a rooted labeled unordered tree over Σ a *tree*, simply.

Definition 1 (Edit operations). Let T be a tree. Then, we call the following three operations *edit operations*. Also see Figure 1.

1. *Substitution*: Change the label of the node v in T (from l_1 to l_2).
2. *Deletion*: Delete a non-root node v in T (labeled by l_1) with a parent v' (labeled by l'), making the children of v become the children of v'. The children are inserted in the place of v as a subset of the children of v'.
3. *Insertion*: The complement of deletion. Insert a node v (labeled by l_2) as a child of v' (labeled by l') in T making v the parent of a subset of the children of v'.

For a special *blank* symbol $\varepsilon \notin \Sigma$, let $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$. Then, we represent each edit operation by $l_1 \mapsto l_2$, where $(l_1, l_2) \in (\Sigma_\varepsilon \times \Sigma_\varepsilon - \{(\varepsilon, \varepsilon)\})$. The operation is a substitution if $l_1 \neq \varepsilon$ and $l_2 \neq \varepsilon$, a deletion if $l_2 = \varepsilon$, and an insertion if $l_1 = \varepsilon$.

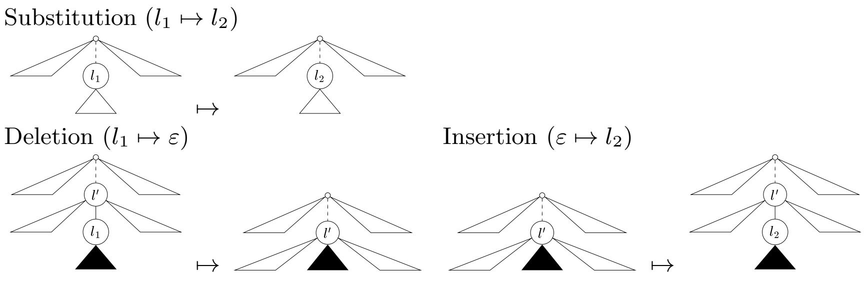
Substitution ($l_1 \mapsto l_2$)
l_1
$\mapsto$
l_2
Deletion ($l_1 \mapsto \varepsilon$)
l'
l_1
$\mapsto$
l'
Insertion ($\varepsilon \mapsto l_2$)
l'
$\mapsto$
l'
l_2

Fig. 1. Edit operations for trees

We define a *cost function* $\gamma : (\Sigma_\varepsilon \times \Sigma_\varepsilon - \{(\varepsilon, \varepsilon)\}) \mapsto \mathbf{R}$ on pairs of labels. We constrain a cost function γ to be a *metric*, that is, $\gamma(l_1, l_2) \geq 0$, $\gamma(l_1, l_1) = 0$, $\gamma(l_1, l_2) = \gamma(l_2, l_1)$ and $\gamma(l_1, l_3) \leq \gamma(l_1, l_2) + \gamma(l_2, l_3)$.

For a cost function γ, we define the *cost* of an edit operation by setting $\gamma(l_1 \mapsto l_2) = \gamma(l_1, l_2)$. The *cost* of a sequence $\mathbf{s} = s_1, \ldots, s_k$ of edit operations is given by $\gamma(S) = \sum_{i=1}^{k} \gamma(s_i)$.

Definition 2 (Edit distance). Let T and S be trees and γ a cost function. Then, the *edit distance* $\tau(T, S)$ between T and S under γ is defined as follow:

$$\tau(T, S) = \min \left\{ \gamma(\mathbf{s}) \,\middle|\, \begin{array}{l} \mathbf{s} \text{ is a sequence of edit operations} \\ \text{transforming from } T \text{ to } S \end{array} \right\}.$$

The edit distance is closely related to the following mapping [7].

Definition 3 (Mapping). For trees T and S, we say that the triple (M, T, S) is a *mapping* between T and S if $M \subseteq V(T) \times V(S)$ and every pair (v_1, w_1) and (v_2, w_2) in M satisfies the following conditions.

1. $v_1 = v_2$ iff $w_1 = w_2$ (one-to-one condition).
2. $v_1 \leq v_2$ iff $w_1 \leq w_2$ (ancestor condition).

We will use M instead of (M, T, S) when there is no confusion.

Let M be a mapping between T and S. Also let I_T (*resp.*, I_S) be the set of nodes in T (*resp.*, S) but not in M. Then, the *cost* $\gamma(M)$ of M is given as $\sum_{(v,w)\in M} \gamma(l(v), l(w)) + \sum_{v\in I_T} \gamma(l(v), \varepsilon) + \sum_{w\in I_S} \gamma(\varepsilon, l(w))$. Hence, it is known the following relationship between $\tau(T, S)$ and $\gamma(M)$ [7].

$$\tau(T, S) = \min\{\gamma(M) \mid M \text{ is a mapping between } T \text{ and } S\}.$$

3 A* Algorithm and Lower Bounding Functions

The A* *algorithm* is an algorithm to find the shortest path from the start node a to the goal node z in a connected graph by using an estimated length smaller than the actual length from a current node n to z.

For a current node n, suppose that $g(n)$ and $h(n)$ are the actual length of the shortest path from a to n and one from n to z, respectively. Then, for the length $f(n)$ of the shortest path from a to z through n, it holds that $f(n) = g(n)+h(n)$. However, since $f(n)$ is unknown in the procedure, we replace the actual length $f(n)$, $g(n)$ and $h(n)$ with the estimated length $f^*(n)$, $g^*(n)$ and $h^*(n)$ satisfying that $f^*(n) = g^*(n) + h^*(n)$.

Since we can find $g(n)$ in the procedure, we set $g^*(n)$ to $g(n)$. On the other hand, since we cannot find $h(n)$ in the procedure, we set $h^*(n)$, called a *heuristic function*, to a function that is guaranteed to be always smaller than $h(n)$.

In particular, in order to design the A* algorithm for computing $\tau(T, S)$, in this paper, we formulate such a heuristic function h^* based on the following *lower bounding functions* on τ. Here, throughout of this paper, we assume that a cost function is a *unit cost function* μ [11] satisfying that $\mu(l_1, l_2) = \mu(l_1, \varepsilon) = \mu(\varepsilon, l_2) = 1$ for $l_1, l_2 \in \Sigma$ and $l_1 \neq l_2$. We can extend μ to an arbitrary cost function easily.

Definition 4 (Lower bounding function). Let T be a tree. We call the histogram consisting of the degree of a node and its frequency in T the *degree histogram* of T. Also we call the histogram consisting of the label of a node and its frequency in T the *label histogram* of T.

Let T and S be trees. Then, we define $n(T, S)$ as $\big||T| - |S|\big|$, that is, the difference between the number of nodes in T and S. Also we define the *degree histogram L_1-distance* $d_1(T, S)$ and the *degree histogram L_∞-distance* $d_\infty(T, S)$ (*resp.*, the *label histogram L_1-distance* $l_1(T, S)$ and the *label histogram L_∞-distance* $l_\infty(T, S)$) as the L_1- and L_∞-distances between the degree (*resp.*, label) histograms of T and S.

Lemma 1 (Horesh *et al.* [4], Kailing *et al.* [5]). *For trees T and S, the following statements hold.*

1. $\tau(T, S) \geq n(T, S)$ [4].
2. $\tau(T, S) \geq d_1(T, S)/3$ [4,5] *and* $\tau(T, S) \geq d_\infty(T, S)$ [4].
3. $\tau(T, S) \geq l_1(T, S)/2$ [5] *and* $\tau(T, S) \geq l_\infty(T, S)$.

Proof. It is sufficient to show that $l_\infty(T,S) \leq \tau(T,S)$, that is, how many values of $l_\infty(T,S)$ change when an edit operation is applied. We denote the frequency of the label a in a tree T by $f(a,T)$.

When transforming from T to S by a substitution of b for a, it holds that $f(a,T)-1=f(a,S)$, $f(b,T)+1=f(b,S)$ and $f(c,T)=f(c,S)$ for every label c except a and b. Then, it holds that $l_\infty(T,S) \leq 1$. On the other hand, when transforming from T to S by a deletion of a, it holds that $f(a,T)-1=f(a,S)$ and $f(c,T)=f(c,S)$ for every label c except a. Then, it holds that $l_\infty(T,S) \leq 1$.

Hence, it holds that $l_\infty(T,S) \leq \tau(T,S)$. □

By Lemma 1, for nodes $t \in T$ and $s \in S$, we use a heuristic function $h^*(t,s)$ in the A* algorithm as the maximum value of the above 5 lower bounding functions with constant factors of $T(t)$ and $S(s)$, that is:

$$h^*(t,s) = \max \left\{ \begin{array}{l} n(T(t),S(s)), \\ d_1(T(t),S(s))/3,\ d_\infty(T(t),S(s)), \\ l_1(T(t),S(s))/2,\ l_\infty(T(t),S(s)) \end{array} \right\}. \quad (1)$$

Example 1. Consider the trees T and S in Figure 2. Then, the degree histograms and the label histograms of T and S are described as follows.

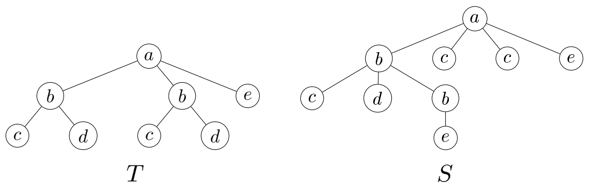

Fig. 2. Trees T and S in Example 1

degree	T	S
0	5	6
1	0	1
2	2	0
3	1	1
4	0	1

label	T	S
a	1	1
b	2	2
c	2	3
d	2	1
e	1	2

The values of the lower bounding functions for $r_1 = r(T)$ and $r_2 = r(S)$ are described as follows.

$$n(r_1,r_2)=1, d_1(r_1,r_2)=5, d_\infty(r_1,r_2)=2, l_1(r_1,r_2)=3, l_\infty(r_1,r_2)=1.$$

Hence, it holds that $h^*(r_1,r_2) = \max\{1, 5/3, 2, 3/2, 1\} = 2$.

4 A* Algorithm for Computing the Tree Edit Distance for Unordered Trees

In order to transform the problem of finding the shortest path to one of computing $\tau(T,S)$ in the A* algorithm, we introduce the *edit distance search tree* $ET(T,S)$ between T and S. Suppose that every node $t \in T$ and $s \in S$ is numbered by its breadth-first search index $bf_T(t)$ and $bf_S(s)$ starting from 0.

Definition 5 (Edit distance search tree). For trees T and S, an *edit distance search tree* $ET(T,S)$ of T and S is a tree such that the depth is $|T|-1$, the label of the root is 0 and every non-leaf node has $|S|$ children labeled by $\varepsilon, 1, \ldots, |S|-1$.

Furthermore, we say that a node v in $ET(T,S)$ is *valid* if the following set M_v of pairs of nodes in T and S forms a mapping between T and S.

$$M_v = \left\{ (t,s) \in T \times S \,\middle|\, \begin{array}{l} w \in UP_{ET(T,S)}(v) - \{\varepsilon\}, \\ d(w) = bf_T(t), l(w) = bf_S(s) \end{array} \right\}.$$

In this paper, we identify the edit distance search tree $ET(T,S)$ with one consisting of just valid nodes, and we also call the latter the edit distance search tree. Hence, the depth of $ET(T,S)$ is $|T|-1$ and the degree of $ET(T,S)$ is at most $|S|$. Every node $v \in ET(T,S)$ denotes the pair $(t,s) \in T \times S$, which is a component of a mapping, such that $d(v) = bf_T(t)$ and $l(v) = bf_S(s) (\neq \varepsilon)$.

Example 2. Consider the trees T and S in Figure 3 (left). Here, the number attached to nodes in T and the number of nodes in S denote the breadth-first search index of T and S, respectively.

Figure 3 (right) illustrates the edit distance search tree $ET(T,S)$ of T and S. For example, the path $\langle 0, 1, \varepsilon, 2 \rangle$ in $ET(T,S)$ consisting of the underlined number represents the mapping $\{(0,0),(1,1),(3,2)\}$ between T and S. In this path, the node labeled with 1 at depth 1 has just one child ε, because the mapping $\{(0,0),(1,1)\}$ cannot contain the pair $(2,2)$ satisfying the ancestor condition.

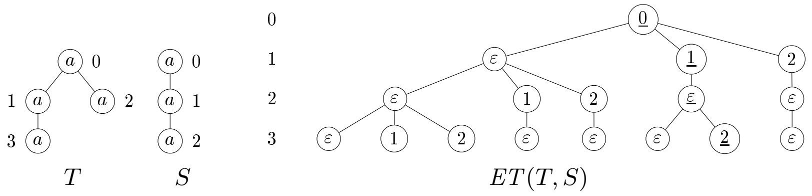

Fig. 3. Trees T and S (left) and the edit distance search tree $ET(T,S)$ (right) in Example 2

We explain how to compute the values of $g^*(t,s)$ and $h^*(t,s)$ in the A* algorithm. Let v be a node in $ET(T,S)$ such that $d(v) = bf_T(t)$ and $l(v) = bf_S(s)$ and v_p the parent of v in $ET(T,S)$ such that $d(v_p) = bf_T(t_p)$ and $l(v_p) = bf_S(s_p)$. Also we define the set $N_v(s) \subseteq S$ as follows.

$$N_v(s) = \left\{ s' \in S \;\middle|\; \begin{array}{l} 0 \le bf_S(s') \le bf_S(s), \text{ and} \\ M_v \cup \{(t', s')\} \text{ is not a mapping for every } t' \in T \end{array} \right\}.$$

Then, we compute $g^*(t, s)$ as follow.

$$g^*(t, s) = g^*(t_p, s_p) + |N_v(s)|. \tag{2}$$

On the other hand, consider $h^*(t, s)$. If $s \ne \varepsilon$, then we can compute $h^*(t, s)$ according to the equation (1). Otherwise, that is, in the case that $s = \varepsilon$, first search for the nearest ancestor $t' \in UP_T(t)$ to t in T such that $d(v') = bf_T(t')$ and $l(v') = bf_S(s') \ne \varepsilon$ for some $v' \in ET(T, S)$. For this t', let v be a node in $ET(T, S)$ such that $d(v) = t'$. Also suppose that $l(v) = s', d(v_p) = t'_p$ and $l(v_p) = s'_p$. Then, we define $c(t')$ as follows.

$$c(t') = \begin{cases} g^*(t'_p, s'_p) - g^*(t', s'), & \text{if } l(v) \ne \varepsilon, \\ g^*(t'_p, s'_p) - g^*(t', s') + 1, & \text{if } l(v) = \varepsilon. \end{cases}$$

Then, we compute $h^*(t, s)$ with the equation.

$$h^*(t, s) = \begin{cases} \text{the right hand side of the equation (1)}, & \text{if } s \ne \varepsilon, \\ h^*(t', s') - \displaystyle\sum_{t'' \in T(t')} c(t''), & \text{if } s = \varepsilon. \end{cases} \tag{3}$$

Hence, the A* algorithm computes $\tau(T, S)$ by finding the path from the root to the leaves with the minimum estimated value $f^*(t, s) = g^*(t, s) + h^*(t, s)$ in $ET(T, S)$ according to the equations (1), (2) and (3). Since the full construction of $ET(T, S)$ in the A* algorithm is too redundant, we design the A* algorithm as the best-first search for $ET(T, S)$ with constructing just necessary branches of $ET(T, S)$.

Finally, we summarize the A* algorithm for computing $\tau(T, S)$ as follows, where L is a list of triples. Assume here that every mapping contains the pair of the roots in T and S.

1. Add $((0, 0), g^*(0, 0), h^*(0, 0))$ to L and draw the node labeled by 0 in $ET(T, S)$.
2. Select $((t, s), g^*(t, s), h^*(t, s))$ in L such that $g^*(t, s) + h^*(t, s)$ is minimum.
3. If $bf_T(t) = |T| - 1$, then output $g^*(t, s) + h^*(t, s)$ and halt. Otherwise:
   (a) Select $v \in ET(T, S)$ such that $d(v) = t$ and $l(v) = s$.
   (b) Draw the node u such that $l(u) = \varepsilon$ in $ET(T, S)$ as the child of v, and add $((t + 1, \varepsilon), g^*(t + 1, \varepsilon), h^*(t + 1, \varepsilon))$ to L.
   (c) For $t' \in T$ such that $bf_T(t') = d(v) + 1$ and for every $s' \in S$, if s' satisfies that $M_v \cup \{(t', s')\}$ forms a mapping between T and S, then draw the node u in $ET(T, S)$ such that $bf_S(s') = l(u)$ as the child of v and add $((t', s'), g^*(t', s'), h^*(t', s'))$ by the equations of (1), (2) and (3) to L.
   (d) Go to the statement 2.

Example 3. We apply the A* algorithm to compute the edit distance between T and S in Figure 4. Here, the number attached to nodes in T and S denotes the breadth-first search index of T and S, respectively.

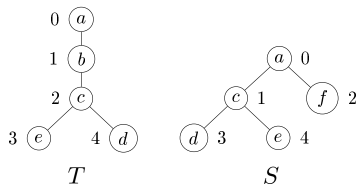

Fig. 4. Trees T and S in Example 3

Then, Figure 5 illustrates the running example of the A* algorithm for the trees T and S in Figure 4, with constructing the edit distance search tree $ET(T, S)$, from the root to leaves. Here, the number attached to a node n in $ET(T, S)$ denotes the value of $g^*(t, s)+h^*(t, s)$. Now we explain the run of the A* algorithm with Figure 5.

At the step (0), the A* algorithm adds the pair $(0, 0)$ to a mapping and constructs the children of 0 at depth 1 in $ET(T, S)$.

At the step (1), the A* algorithm selects the path $\langle 0, \varepsilon\rangle$ in $ET(T, S)$, because the value $1+1$ of the node ε is minimum in the values at the depth 1 in $ET(T, S)$. Then, the A* algorithm adds no pair to a mapping and constructs the children of ε at depth 2 in $ET(T, S)$.

At the step (2), the A* algorithm selects the path $\langle 0, \varepsilon, 1\rangle$ in $ET(T, S)$, because the value $2+0$ of the node 1 is minimum in the values at the depth 2 in $ET(T, S)$. Then, the A* algorithm adds the pair $(2, 1)$ to a mapping and constructs the children of 1 at depth 3 in $ET(T, S)$.

At the step (3), the A* algorithm selects the path $\langle 0, \varepsilon, 1, 4\rangle$ in $ET(T, S)$, because the value $2 + 0$ of the node 4 is minimum in the values at the depth 3 in $ET(T, S)$. Then, the A* algorithm adds the pair $(3, 4)$ to a mapping and constructs the children of 4 at depth 4 in $ET(T, S)$.

At the step (4), the A* algorithm selects the path $\langle 0, \varepsilon, 1, 4, 3\rangle$ in $ET(T, S)$, because the value $2 + 0$ of the node 3 is minimum in the values at the depth 4 in $ET(T, S)$. Then, the A* algorithm adds the pair $(4, 3)$ to a mapping.

Hence, the A* algorithm returns $2+0 = 2$ as $\tau(T, S)$, and its mapping between T and S is $\{(0, 0), (2, 1), (3, 4), (4, 3)\}$.

5 Experimental Results

We give experimental results by comparing the A* algorithm with the exhaustive search algorithm designed by Shasha *et al.* [6] and the clique-based algorithm designed by Fukagawa *et al.* [2]. Here, our computer environment is that OS is Microsoft Windows 7, CPU is Core i7 920 2.67GHz and RAM is 3GB.

We use the same dataset as [2], consisting of 352 glycan data including 137 leukemia and 14 erythrocyte. While we implement both the A* algorithm and

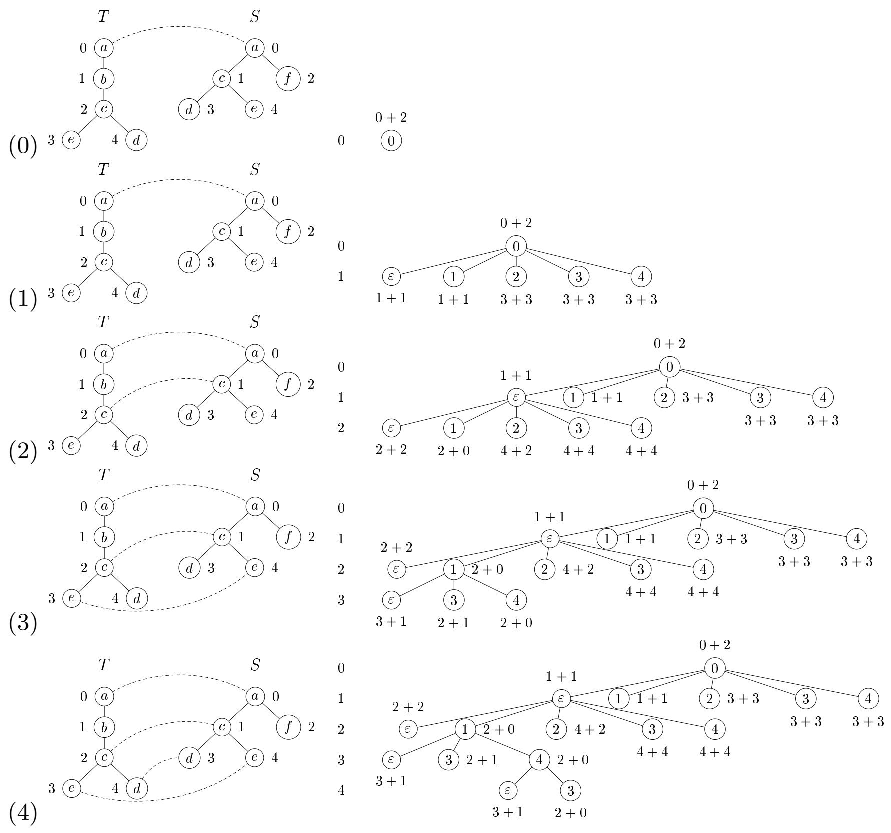

Fig. 5. The running example of the A* algorithm for the trees T and S in Figure4, with constructing the edit distance search tree $ET(T, S)$

the exhaustive search algorithm, we cite the result of the clique-based algorithm to the paper [2][1]. Then, we obtain the result described in Table 1.

Hence, the A* algorithm is much efficient than the exhaustive search algorithm [6] and so efficient as the clique-based algorithm [2] to compute τ. It is advantage for the A* algorithm to adopt the unit cost function (and then extend an arbitrary cost function), while the clique-based algorithm [2] depends on a special cost function such that $\gamma(l_1, l_1) = 2$ and $\gamma(l_1, l_2) = 1$ for $l_1 \neq l_2$.

Furthermore, we evaluate the effect of lower bounding functions in the A* algorithm. Table 2 shows the running time of the A* algorithm when excluding at most one lower bounding function.

Hence, for glycan data, $n(T, S)$ is the most effective lower bounding function. Also $d_\infty(T, S)$ and $l_\infty(T, S)$ are more effective than $d_1(T, S)/3$ and $l_1(T, S)/2$, respectively.

[1] The computer environment of the clique-based algorithm [2] is that OS is Microsoft Windows XP, CPU is Intel Core 2 Duo 2.8GHz and RAM is 3.48GB.

Table 1. The running time (sec.) to compute unordered tree edit distance

		all data	leukemia and erythrocyte
exhaustive search algorithm	[6]	2133.03	751.28
A* algorithm		161.56	21.36
clique-based algorithm	[2]	–	48.33

Table 2. The effect of lower bounding functions in the A* algorithm (sec.)

excluded lower bounding function	all data	leukemia and erythrocyte
none	161.56	21.36
$n(T,S)$	338.61	41.60
$d_1(T,S)/3$	169.12	21.28
$d_\infty(T,S)$	207.06	32.64
$l_1(T,S)/2$	176.92	21.92
$l_\infty(T,S)$	187.44	24.32

6 Conclusion

In this paper, we have designed and implemented the A* algorithm for computing an edit distance between rooted labeled unordered trees. Then, we have applied the A* algorithm to glycan data and obtained the result that the A* algorithm is much efficient than the exhaustive search algorithm [6] and so efficient as the clique-based algorithm [2]. Furthermore, we have evaluated that the difference of the nodes of subtrees is the most effective lower bounding function.

It is a future work to apply the A* algorithm to other data such as XML data and evaluate the effect of lower bounding functions, that is, analyze the relationship between the structures of trees and the effect of lower bounding functions. It is also a future work to improve the A* algorithm by introducing other lower bounding functions or to design other algorithm to compute an unordered edit distance more efficient than the A* algorithm.

Acknowledgment. The authors would like to thank Prof. Tetsuji Kuboyama at Gakushuin University for his fruitful suggestion to improve the experimental results. Also they would like to thank the anonymous reviewers of ALSIP'11 for their valuable comments to revise this paper.

References

1. Bille, P.: A survey on tree edit distance and related problems. Theoret. Comput. Sci. 337, 217–239 (2005)
2. Fukagawa, D., Tamura, T., Takasu, A., Tomita, E., Akutsu, T.: A clique-based method for the edit distance between unordered trees and its application to analysis of glycan structures. BMC Bioinformatics 12 (2011)
3. Hirata, K., Yamamoto, Y., Kuboyama, T.: Improved MAX SNP-Hard Results for Finding an Edit Distance between Unordered Trees. In: Giancarlo, R., Manzini, G. (eds.) CPM 2011. LNCS, vol. 6661, pp. 402–415. Springer, Heidelberg (2011)
4. Horesh, Y., Mehr, R., Unger, R.: Designing an A^* algorithm for calculating edit distance between rooted-unordered trees. J. Comput. Bio. 13, 1165–1176 (2006)
5. Kailing, K., Kriegel, H.-P., Schönauer, S., Seidl, T.: Efficient Similarity Search for Hierarchical Data in Large Databases. In: Bertino, E., Christodoulakis, S., Plexousakis, D., Christophides, V., Koubarakis, M., Böhm, K. (eds.) EDBT 2004. LNCS, vol. 2992, pp. 676–693. Springer, Heidelberg (2004)
6. Shasha, D., Wang, J.T.-L., Zhang, K., Shih, F.Y.: Exact and approximate algorithms for unordered tree matching. IEEE Trans. Sys. Man and Cybernet. 24, 668–678 (1994)
7. Tai, K.-C.: The tree-to-tree correction problem. J. ACM 26, 422–433 (1979)
8. Zhang, K., Shasha, D.: Tree pattern matching. In: Apostolico, A., Galil, Z. (eds.) Pattern Matching Algorithms, pp. 341–371 (1997)
9. Zhang, K., Jiang, T.: Some MAX SNP-hard results concerning unordered labeled trees. Inform. Process. Lett. 49, 249–254 (1994)
10. Zhang, K., Statman, R., Shasha, D.: On the editing distance between unordered labeled trees. Inform. Process. Lett. 42, 133–139 (1992)
11. Zhang, K., Shasha, D.: Simple fast algorithms for the editing distance between trees and related problems. SIAM J. Comput. 18, 1245–1262 (1989)