Vol.2011-BIO-26 No.3
2011/9/13

An Improved Clique-Based Method for Computing Edit Distance between Rooted Unordered Trees

Tomoya Mori [†1], Takeyuki Tamura [†1], Daiji Fukagawa [†2], Atsuhiro Takasu [†3], Etsuji Tomita [†4] and Tatsuya Akutsu[†1]

Tree structures are suitable for representing biological objects such as RNA secondary structures so that it is important in computational biology to compare tree structures. Though there are various metrics proposed for computing similarity between tree structured data, tree edit distance is one of the most widely used. However, it is known that the tree edit distance problem is NP-hard for unordered trees. Fukagawa *et al.* have recently proposed a clique-based method for computing the tree edit distance between unordered trees in which each instance of the tree edit distance problem is transformed into an instance of the maximum vertex weighted clique problem and then an existing clique algorithm is applied. In this article, we propose an improved clique-based method for computing the tree edit distance between rooted unordered trees. Different from the previous method, we combine a dynamic programming approach with clique-based approach. Furthermore, we introduce heuristic techniques, which do not violate the optimality of the solution. Applied to comparison of large glycan structures, our improved method is much faster than the previous method in most cases of comparison of large glycan structures.

1. Introduction

There exist various kinds of tree structured biological data such as RNA secondary structures[1)], glycan structures[2)], and vascular trees[3)]. Therefore, analysis of tree structured data is important and various techniques have been applied to analysis of these tree structured data.

†1 Bioinformatics Center, Institute for Chemical Research, Kyoto University, Kyoto 611-0011, Japan
†2 Faculty of Culture and Information Science, Doshisha University, Kyoto 610-0394, Japan
†3 National Institute of Informatics, Tokyo 101-8430, Japan
†4 University of Electro-Communications, Tokyo 182-8585, Japan

Though various metrics are proposed for computing similarity between trees, tree edit distance is one of the most widely used[4)]. In this measure, the similarity between two trees is measured by the minimum cost sequence of edit operations that transforms one tree into another tree where an edit operation is either a *deletion* of a node, an *insertion* of a node, or a *substitution* of a label of a node. For the tree edit distance problem for ordered trees, Tai developed an $O(n^6)$ time algorithm[5)], where n is the number of nodes in a larger input tree. After several improvements, Demaine *et al.* developed an $O(n^3)$ time algorithm and showed that this bound is optimal under some computation strategy[6)].

Tree edit distance between ordered trees is useful if the ordering among children has an important meanings. However, it is preferable to regard input trees as unordered trees in some applications[2),7)]. Unfortunately, Zhang *et al.* proved that the tree edit distance problem for unordered trees is NP-hard[8)]. In order to cope with this hardness, Akutsu *et al.* developed a fixed parameter algorithm which works in $O(2.62^k \cdot poly(n))$ time[9)], where k is the maximum allowed edit distance. Their algorithm might be useful for comparison of very similar trees (i.e, k is small). However, it is not useful for comparison of non-similar trees. Horesh *et al.* developed an A* algorithm[7)]. Though their algorithm works efficiently for moderate size trees, it can only handle unit cost cases (i.e., the cost of each edit operation is 1). Fukagawa *et al.* proposed a practical method for computing the tree edit distance between unordered trees[10)] using algorithms for computing the *maximum clique*[11)]. In this method, the tree edit distance problem is transformed into the maximum vertex weighted clique problem and an existing clique solver[12)] is applied. The method was applied to comparison and search of similar glycan structures and shown to be efficient for moderate size tree structures. However, it was not fast if large glycan or tree structures are given.

In this paper, we present an improved clique-based method for computing the tree edit distance between unordered trees. Different from the previous method[10)], the improved method is basically a dynamic programming algorithm that repeatedly solves instances of the maximum vertex weighted clique problem as sub-problems. Because of this improvement, sparser graphs are generated and thus maximum clique instances can be solved more efficiently in many cases though multiple instances must be solved in the improved method. Furthermore,

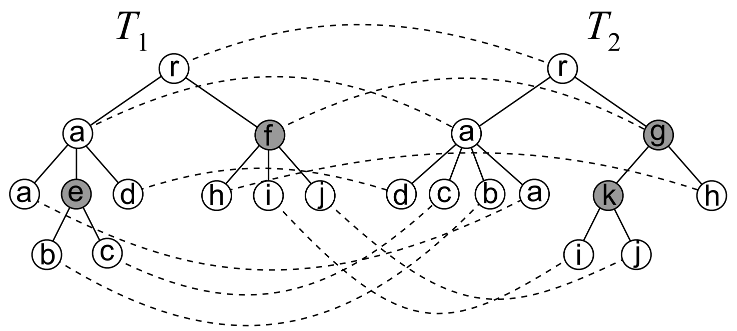

Fig. 1 Example of tree edit operations and edit distance mapping for unordered trees. T_2 is obtained from T_1 by deletion of node with label e, insertion of node with label k and substitution of node with label f, where the same label can appear multiple times in the same tree. The corresponding mapping M is shown by broken curves.

we introduce heuristic techniques which do not violate the optimality of the solution. We compare the improved method with our previous method using glycan data obtained from the KEGG database[13]. The result shows that the improved method is much faster than the previous method in most cases of comparison of large glycan structures.

2. Tree Edit Distance

We briefly review *tree edit distance* and *edit distance mapping* for rooted, labeled and unordered trees[4),8)]. Let T be a rooted unordered tree, which is not given a left-to-right order among siblings. We assume that each node v has a label $l(v)$ over an alphabet Σ. $r(T)$, $V(T)$ and $E(T)$ denote the root, the set of nodes, and the set of edges of T, respectively. For a node $v \in V(T)$, $des(v)$ and $T(v)$ denote the set of descendants of v (not including v) and the subtree induced by v and its descendants, respectively. In the following, n denotes the number of nodes in a larger tree (i.e. $n = \max\{|V(T_1)|, |V(T_2)|\}$).

An *edit operation on a tree* T is either a *deletion*, an *insertion*, or a *substitution*, each of which is defined by (see also **Fig. 1**):

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 into the set of the children of u.

Insertion: Inverse of delete. Insert a node v as a child of u in T , making v the parent of some of the children of u.

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

For each edit operation, the *cost* is defined as follows:

- $\gamma(a, b)$: cost of substituting a node with label a to label b,
- $\gamma(a, \epsilon)$: cost of deleting a node labeled with a,
- $\gamma(\epsilon, a)$: cost of inserting a node labeled with a.

The edit distance $dist(T_1, T_2)$ between two unordered trees T_1 and T_2 is the cost of the minimum cost sequence of edit operations that transforms T_1 into T_2, where we adopt the following standard assumption so that $dist(T_1, T_2)$ becomes a distance metric[4),8)]:

- $\gamma(a, b) \geq 0$ 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 $\Sigma' = \Sigma \cup \{\epsilon\}$.

There exists a close relationship between the edit distance and the *edit distance mapping* (or just *mapping*)[4),8)]. $M \subseteq V(T_1) \times V(T_2)$ 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) $u_1 \in des(u_2)$ iff $v_1 \in des(v_2)$.

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 equality holds[4),8)]:

$$dist(T_1, T_2) = \min_M \left\{ \sum_{u \in I_1} \gamma(l(v), \epsilon) + \sum_{v \in I_2} \gamma(\epsilon, l(v)) + \sum_{(u,v) \in M} \gamma(l(u), l(v)) \right\}. \quad (1)$$

Here we define a *score function* $f(u, v)$ for $(u, v) \in V(T_1) \times V(T_2)$ defined by

$$f(u, v) = \gamma(l(u), \epsilon) + \gamma(\epsilon, l(v)) - \gamma(l(u), l(v)). \quad (2)$$

Then, we can see that $f(u, v) = f(v, u) \geq 0$ holds. It should also be mentioned that under the unit cost model (i.e., $\gamma(a, b) = 1$ for all $a \neq b$), $f(v, v) = 2$ and $f(u, v) = 1$ hold for $l(u) \neq l(v)$. Let $score(M)$ be the score of a mapping M defined by

$$score(M) = \sum_{(u,v)\in M} f(u,v). \tag{3}$$

Let M_{OPT} be the mapping with the maximum score. Then, it is known that the following equality holds[9]:

$$dist(T_1,T_2) = \sum_{u\in V(T_1)} \gamma(l(u),\epsilon) + \sum_{v\in V(T_2)} \gamma(\epsilon,l(v)) - score(M_{OPT}) \tag{4}$$

assuming that the root of T_1 corresponds to the root of T_2 in M_{OPT}. It is to be noted that the first and second terms in the right hand side of the last equality are invariant with a mapping. Therefore, this equality means that the tree edit distance can be obtained by computing a mapping with the maximum score.

Though the edit distance problem for unordered trees is NP-hard, it can be solved (in exponential time) using a dynamic programming algorithm[4]. For a forest (i.e., a set of unordered trees) F, $roots(F)$ denotes a set of the roots of trees in F. $T-v$, $F-v$, and $F-T(v)$ denote the tree obtained by deleting v from T, the forest obtained by deleting v, and the forest obtained by deleting $T(v)$. We define $D(F_1,F_2)$ between two unordered forests F_1 and F_2 by the following dynamic programming procedure[9] [⋆1].

$$D(F_1,\epsilon) = \sum_{u\in V(F_1)} \gamma(l(u),\epsilon), \tag{5}$$

$$D(\epsilon,F_2) = \sum_{v\in V(F_2)} \gamma(\epsilon,l(v)), \tag{6}$$

$$D(F_1,F_2) = \min \begin{cases} \min_{u\in roots(F_1)} \{D(F_1-u,F_2)+\gamma(l(u),\epsilon)\}, \\ \min_{v\in roots(F_2)} \{D(F_1,F_2-v)+\gamma(\epsilon,l(v))\}, \\ \min_{(u,v)\in roots(F_1)\times roots(F_2)}\{ \\ \qquad D(F_1-T_1(u),F_2-T_2(v)) \\ \qquad +D(T_1(u)-u,T_2(v)-v) \\ \qquad +\gamma(l(u),l(v))\}. \end{cases} \tag{7}$$

⋆1 The roots need not correspond to each other in this procedure. However, we can let roots correspond to each other by setting deletion costs for the roots very large.

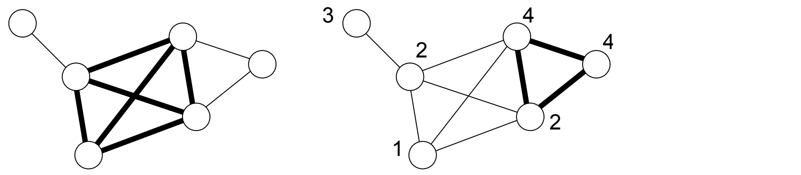

Fig. 2 Example of the maximum clique and the maximum vertex weighted clique. The size of the maximum clique of the left graph is four, while the weight of the maximum vertex weighted clique of the right graph is three.

Then, it is seen that $dist(T_1,T_2) = D(F_1,F_2)$ holds from Ref.4).

3. Method

3.1 Maximum Vertex Weighted Clique

Let $G(V,E)$ be an undirected graph. A subgraph $G'(V',E')$ of $G(V,E)$ is called a *clique* if it is a complete subgraph (i.e., $\{\{v_i,v_j\}|v_i,v_j\in V', v_i\neq v_j\} = E'$). The *maximum clique problem* is to find a clique with the maximum number of vertices in a given undirected graph $G(V,E)$. Though the maximum clique problem is NP-hard, several practical algorithms have been developed[11]. In this paper, we use a variant of the maximum clique problem called *the maximum vertex weighted clique problem*. In this variant, each vertex v has a weight $w(v)$ and the problem is to find a clique $G'(V',E')$ which maximizes $\sum_{v\in V'} w(v)$ (see also **Fig. 2**). Nakamura and Tomita developed an efficient algorithm called MWCQ for this variant[12]. MWCQ is a depth-first search algorithm for finding the maximum vertex weighted clique and it is based on the branch-and-bound method. We employ MWCQ as a solver for the maximum vertex weighted clique problem.

3.2 Previous Method

Before presenting our improved clique-based method, we briefly review the previous clique-based method[10] (see also **Fig. 3**). In this paper, we call the previous method CliqueEdit. CliqueEdit is based on a simple reduction from the tree edit distance problem for unordered trees to the maximum clique problem. Based on Eq.(4), for calculating the tree edit distance, it is enough to find a mapping M maximizing $\sum_{(u,v)\in M} f(u,v)$. In order to find such a mapping, an undirected graph $G(V,E)$ is constructed from two input trees T_1 and T_2 by

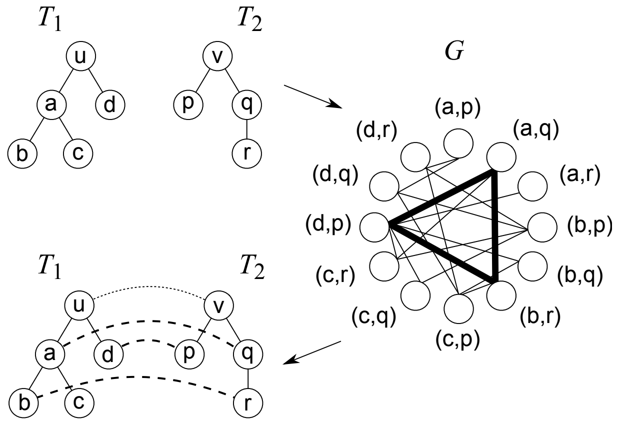

Fig. 3 Example of reduction from the edit distance problem for unordered trees to the maximum vertex weighted clique problem.

$$V = \{(u,v)|u \in V(T_1), u \neq r(T_1), v \in V(T_2), v \neq r(T_2)\}, \quad (8)$$

$$\begin{aligned} E = \{\{(u_1,v_1),(u_2,v_2)\}|&u_1 \neq u_2, v_1 \neq v_2, \\ &u_1 \in des(u_2) \text{ iff } v_1 \in des(v_2), \\ &u_2 \in des(u_1) \text{ iff } v_2 \in des(v_1)\}, \end{aligned} \quad (9)$$

where the first two conditions and the last two conditions in the definition of E correspond to conditions (i) and (ii) for the edit distance mapping, respectively. We can see that there is a one-to-one correspondence between the set of cliques and the set of mappings (i.e., (u,v) in a clique corresponds to (u,v) in a mapping M). By assigning a weight $w(x) = f(u,v)$ to each vertex $x = (u,v) \in V$, an optimal mapping M_{OPT} corresponds to a maximum vertex weighted clique. Therefore, the tree edit distance problem can be solved by computing a maximum vertex weighted clique.

3.3 Improved Method

In order to improve CliqueEdit, we combine a *dynamic programming* approach employed in Ref.9) with the clique-based approach. We call the resulting method DpCliqueEdit. Let $(u,v) \in V(T_1) \times V(T_2)$. We define $W[u,v]$ be the score of an optimal mapping between $T_1(u)$ and $T_2(v)$ where the root of $T_1(u)$ need not correspond to the root of $T_2(v)$. We compute $W[u,v]$ in a bottom up way (i.e., from leaves to roots) using dynamic programming. Suppose that $W[u',v']$ are already computed for all $(u',v') \in des(u) \times des(v)$. Then, we construct an undirected vertex weighted graph $G_{(u,v)}(V_{(u,v)}, E_{(u,v)})$ by

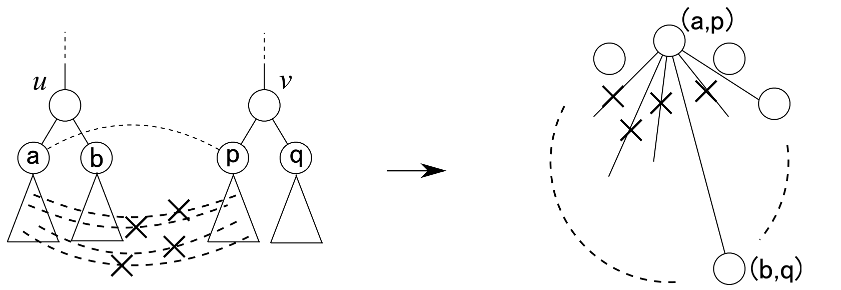

Fig. 4 Difference between the reductions in CliqueEdit and DpCliqueEdit. In computation of $W[a,p]$ in DpCliqueEdit, edges corresponding to mappings among descendants of a and q are not generated for $G_{(a,p)}(V_{(a,p)}, E_{(a,p)})$.

$$V_{(u,v)} = \{(u_1,v_1)|u_1 \in des(u), v_1 \in des(v)\}, \quad (10)$$

$$\begin{aligned} E_{(u,v)} = \{\{(u_1,v_1),(u_2,v_2)\}|&u_1 \neq u_2, v_1 \neq v_2, \\ &u_1 \notin des(u_2), u_2 \notin des(u_1), \\ &v_1 \notin des(v_2), v_2 \notin des(v_1)\}, \end{aligned} \quad (11)$$

$$w((u_1,v_1)) = W[u_1,v_1]. \quad (12)$$

Let W_{max} be the weight of the maximum vertex weight clique for $G(u,v)$. Then, we calculate $W[u,v]$ by [⋆1].

$$W[u,v] = \max \begin{cases} \max_{v' \in des(v)} W[u,v'] \\ \max_{u' \in des(u)} W[u',v] \\ W_{max} + f(u,v) \end{cases} \quad (13)$$

Different from the reduction in CliqueEdit, edges are not created in DpCliqueEdit if there is a descendant-ancestor relation between u_1 and u_2 (or between v_1 and v_2, see also **Fig. 4**). Therefore, it is expected that graphs constructed in DpCliqueEdit are much sparser than those in CliqueEdit though DpCliqueEdit must solve many clique instances.

3.4 Heuristics

In addition to the use of dynamic programming, we introduce some heuris-

⋆1 Slight modifications are required if u or v is a root.

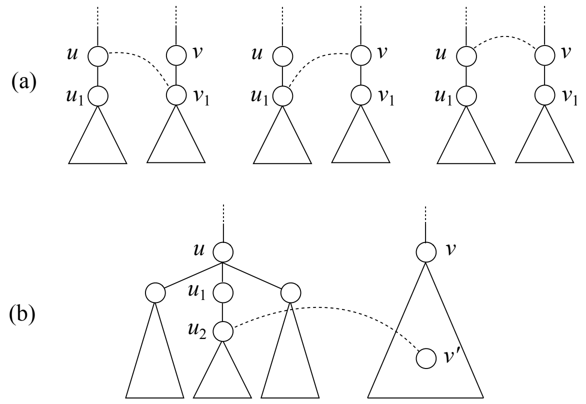

Fig. 5 (a) and (b) explain the heuristic technique (1) and (2), respectively.

tic techniques in order to reduce the computation time without violating the optimality of the solution.

An important observation is that $W[u_1, v_1] \geq W[u_2, v_1]$ always holds if u_2 is a descendant of u_1. Based on this observation, we introduce the following two heuristic techniques (see also **Fig. 5**).

(1) Each of u and v has only one child.
In this case, we need not construct $G_{(u,v)}$. Instead, we can compute $W[u, v]$ by $W[u, v] = \max\{W[u, v_1], W[u_1, v], W[u_1, v_1] + f(u, v)\}$, where u_1 and v_1 are the children of u and v, respectively.

(2) $u_2 \in des(u)$ (resp. $v_2 \in des(v)$) does not have a sibling.
In case, we need not generate a vertex (u_2, v') for any v' (resp. (u', v_2) for any u') in the construction of $G_{(u,v)}$ because a mapping between $T_1(u_2)$ and $T_2(v)$ can be included in a mapping between $T_1(u_1)$ and $T_2(v)$ where u_1 is the parent of u_2.

4. Result

We implemented DpCliqueEdit using C language and compared DpCliqueEdit with CliqueEdit. In both implementations, we employed MWCQ[12] as a solver for the maximum vertex weighted clique problem. We performed computational experiments using a PC with Intel Core2 Quad 3.00GHz CPU and 7.7GB memory. As tree structures, we used glycan structures obtained from KEGG/Glycan database[13]. For evaluation of the method, we used the standard weighting scheme (i.e., $f(v, v) = 2$ and $f(u, v) = 1$ for $l(u) \neq l(v)$) corresponding to the unit cost edit distance. In the computational experiments, we randomly selected 45 pairs of glycan structures with a specified range of the total number of nodes (i.e., the sum of the number of nodes in T_1 and T_2) and measured the average CPU time per pair. Each glycan structures we used for this computational experiment has from15 to 54 nodes. In this computational experiment, four pairs of glycan were excluded because both programs could not output a solution within two minutes. The result of this computational experiment is shown in **Table 1**.

From this table, it is seen that DpCliqueEdit is much faster than CliqueEdit for non-small glycan structures. Though CliqueEdit is faster than DpCliquedit for small glycan structures, comparison of large glycan structures is more crucial because it takes a large amount of time.

In another computational experiment, we transformed the maximum vertex weighted clique problem into the maximum clique problem, and employed MCQ[14] instead of MWCQ in order to find the maximum clique. This is because MCQ is faster than MWCQ[10]. When we transform the maximum vertex weighted clique problem into the maximum clique problem, we construct an unweighted graph from a weighted graph. Let $G(V, E)$ be a weighted graph such that $V = v_1, ..., v_n$ and $w(v_i) = w_i (i = 1, 2, ..., n)$. From $G(V, E)$, we construct an unweighted graph $\hat{G}(\hat{V}, \hat{E})$ by (see also **Fig. 6**)

Table 1 Comparison of CliqueEdit and DpCliqueEdit. Average CPU time (sec.) per glycan pair is shown for each case.

total number of nodes	CliqueEdit	DpCliqueEdit
30 ∼ 34	0.0050	0.0100
35 ∼ 39	0.0525	0.0450
40 ∼ 44	0.0280	0.0240
45 ∼ 49	0.1900	0.0643
50 ∼ 54	12.4000	1.8700
55 ∼ 59	1.6900	0.1340
60 ∼ 64	0.7300	0.2150
65 ∼ 69	28.6000	0.2580
70 ∼ 74	5.5200	0.5900
75 ∼ 79	4.2600	0.8200

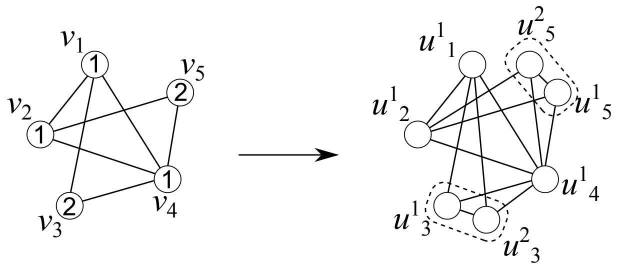

Fig. 6 Example of transformation of a vertex weighted graph into an unweighted graph.

$$\hat{V} = \bigcup_{i=1}^{n} \hat{V}_i \quad (\text{where } \hat{V}_i = \{u_i^j | j = 1, 2, ..., w_i\})$$
$$\hat{E} = \{\{u_i^j, u_k^l\} \mid \{v_i, v_k\} \in E \vee i = k\}.$$

Then, the weight of maximum vertex weighted clique of $G(V, E)$ correspond to the size of the maximum clique of $\hat{G}(\hat{V}, \hat{E})$.

However, DpCliqueEdit using MCQ was not so fast as that using MWCQ because larger and denser graphs are constructed in reduction from the maximum vertex weighted clique problem into the maximum clique problem.

5. Concluding Remarks

In this paper, we have presented an improved clique-based method for computing the tree edit distance between rooted unordered trees. The improved method is much faster than the previous method in most cases of comparison of large glycan structures. Though improved method is not faster for comparison of small glycan structures, it is not crucial because comparison of large glycan structures takes much longer CPU time than that of small glycan structures, and we can run both methods in parallel (using multi-core CPUs that are very common in recent PCs) and stop the other process if one process finishes.

Though the improved method is much faster than the previous method, there still exist cases for which it takes long CPU time. In particular, it takes very long CPU time if there exist many leaves. In such a case, constructed graphs would contain many vertices and edges and thus a clique algorithm does not work efficiently. How to cope with such difficult cases is left as future work.

References

1) Jiang, T., Lin, G., Ma, B. and Zhang., K.: A general edit distance between RNA structures, *Journal of Computational Biology*, Vol.9, pp.371–388 (2002).
2) Aoki, K.F., Yamaguchi, A., Ueda, N., Akutsu, T., Mamitsuka, H., Goto, S. and Kanehisa, M.: KCaM (KEGG Carbohydrate Matcher): a software tool for analyzing the structures of carbohydrate sugar chains, *Nucleic Asids Research*, Vol.32, pp. 267–272 (2004).
3) Yu, K.-C., Ritman, E.L. and Higgns, E.: System for the analysis and visualization of large 3D anatomical trees, *Computers in Biology and Medicine*, Vol.37, pp.1802–1830 (2007).
4) Bille, P.: A survey on tree edit distance and related problem, *Theoretical Computer Science*, Vol.337, pp.217–239 (2005).
5) Tai, K.-C.: The tree-to-tree correction problem, *Journal of ACM*, Vol.26, pp.422–433 (1979).
6) Demaine, E.D., Mozes, S., Rossman, B. and Weimann, O.: An optimal decomposition algorithm for tree edit distance, *ACM Transactions on Algorithms*, Vol.6, p.1 (2009).
7) Horesh, T., Mehr, R. and Unger, R.: Designing an A* algorithm for calculating edit distance between rooted-unordered trees, *Journal of Computational Biology*, Vol.13, pp.1165–1176 (2006).
8) Zhang, K., Statman, R. and Shasha, D.: On the editing distance between unordered labeled trees, *Infomation Processing Letters*, Vol.42, pp.133–139 (1992).
9) Akutsu, T., Fukagawa, D., Takasu, A. and Tamura, T.: Exact algorithms for computing tree edit distance between unordered trees, *Theoretical Computer Science*, Vol.421, pp.352–364 (2011).
10) Fukagawa, D., Tamura, T., Takasu, A., Tomita, E. and Akutsu, T.: A clique-based method for the edit distance between unordered tree and its application to analysis of glycan structures, *BMC Bioinformatics*, Vol.12(Suppl 1), S14 (2011).
11) Tomita, E., Akutsu, T. and Matsunaga, T.: Efficient algorithms for finding a maximum and maximal cliques: Effective tools for bioinformatics, *Biomedical Engineering, Trends, Researches and Technologies*, pp.625–640 (2011).
12) Nakamura, T. and Tomita, E.: Efficient algorithms for finding a maximum clique with maximum vertex weight, Technical report, the University of Electro-Communications. UEC-TR-CAS3-2005.
13) Kanehisa, M., Goto, S., Furumichi, M., Tanabe, M. and Hirakawa, M.: KEGG for representation and analysis of molecular networks involving diseases and drugs, *Nucleic Acids Research*, Vol.38, pp.D355–D360 (2010).
14) Tomita, E. and Seki, T.: An efficient branch-and-bound algorithm for finding a maximum clique, *4th International Conference on Discrete Mathematics and Theoretical Computer Science*, Vol.2731, pp.278–289 (2003).