Improved Methods for Computing Distances Between Unordered Trees Using Integer Programming

Eunpyeong Hong(✉), Yasuaki Kobayashi, and Akihiro Yamamoto

Kyoto University, Kyoto, Japan
ephong93@gmail.com, kobayashi@iip.ist.i.kyoto-u.ac.jp,
akihiro@i.kyoto-u.ac.jp

Abstract. Kondo et al. (DS 2014) proposed integer linear programming formulations for computing the tree edit distance and its variants between unordered rooted trees. They showed that the tree edit distance, segmental distance, and bottom-up segmental distance problems respectively have integer linear programming formulations with $O(nm)$ variables and $O(n^2m^2)$ constraints, where n and m are the number of nodes of two input trees. In this work, we propose new integer linear programming formulations for these three distances and the bottom-up distance by combining with dynamic programming. For computing the tree edit distance, we solve $O(nm)$ subproblems, each of which is formulated by an integer linear program with $O(nm)$ variables and $O(n+m)$ constraints. For the other three distances, each subproblem can be reduced to the maximum weight matching problem in a bipartite graph which is solvable in polynomial time. In order to compute the distances from the solutions of subproblems, we also give a unified integer linear formulation with $O(nm)$ variables and $O(n+m)$ constraints. We conducted a computational experiment to evaluate the performance of our methods. The experimental results show that our methods remarkably outperformed to the previous methods due to Kondo et al.

Keywords: Tree edit distance · Unorderd trees · Integer programming · Dynamic programming

1 Introduction

In machine learning applications, it is important to compare (dis)similarities between tree-structured data such as XML and RNA secondary structures. There are many measures of similarities between two trees. The tree edit distance [16] is one of the most widely used measures, which is defined as the minimum cost of edit operations to transform a tree into another. However, the tree edit distance may not be appropriate to use in some applications. In this context, many variants of the tree edit distance have been proposed (see [12], for example).

X. Gao et al. (Eds.): COCOA 2017, Part II, LNCS 10628, pp. 45–60, 2017.
https://doi.org/10.1007/978-3-319-71147-8_4

The distance to be considered in this paper are the tree edit distance, segmental distance [9], bottom-up segmental distance [9] and bottom-up distance [17].

It is known that most of distances between ordered rooted trees can be computed in polynomial time. For example, Tai [16] showed that the tree edit distance between ordered rooted trees can be computed in $O(n^3m^3)$ time, where n and m are the number of nodes of input trees, and Demaine et al. [4] improved the running time to $O(nm^2(1+\log \frac{n}{m}))$. If input trees are unordered, the problems of computing the above four distances are known to be not only NP-hard [20], but also MAX SNP-hard [9,17,19]. Akutsu et al. studied the tree edit distance problem between unordered trees from a theoretical algorithmic perspective. They gave an approximation algorithm and exact algorithms [1–3]. From the practical point of view, several researches for the unordered tree edit distance have been done so far. Horesh et al. [7] proposed an A* algorithm for unlabeled unordered trees and Higuchi et al. [6] extended it for labeled trees. Fukagawa et al. [5] proposed a method to reduce the edit distance problem into the maximum weight clique problem and used an algorithm due to [15] to solve it. They showed that the clique-based method is as fast as A*-based method. Mori et al. [14] improved it by applying a dynamic programming approach. They showed that their method is faster than the previous clique-based method. Kondo et al. [11] proposed a method to reduce an instance of the edit distance problem into an instance of integer linear programming (IP) problem with $O(nm)$ variables and $O(n^2m^2)$ constraints. However, their IP formulation has a large number of constraints and hence their method may not be applicable to moderate-sized instances. Although they showed that their method is faster than the clique-based method of Mori et al. [14] when input trees have large degree nodes, their IP-based method is not very efficient for the other case.

An advantage of IP-based method is that we can easily make an IP formulation representing variations of the edit distance by adding some further constraints. In fact, Kondo et al. showed IP formulations which represent segmental distance and bottom-up segmental distance by adding appropriate constraints. Another advantage of this method is that we can use state-of-the-art IP solvers (e.g. CPLEX, Gurobi), which can quickly solve many hard problems.

In this paper, we propose new methods to compute the edit distance, segmental distance, bottom-up segmental distance and bottom-up distance between unordered rooted trees. The improvement of computational efficiency is obtained by applying a dynamic programming approach due to [14]. However, it is not only sufficient to apply the dynamic programming but it is necessary to use a structural property of rooted trees. Their dynamic programming approach with this property allows us to drastically reduce the number of constraints in our IP formulations for the above distances. For the edit distance problem, our method has to solve $O(nm)$ subproblems each of which has only $O(n+m)$ constraints. For the other distances, each subproblem except the problem of combining the solutions of subproblems can be reduced to the maximum weighted matching problem in a bipartite graph, which can be solved in polynomial time using the Hungarian method [13].

The rest of the paper is organized as follows. We give notations and preliminary results in Sect. 2 and briefly explain the previous method in Sect. 3. In Sect. 4, we introduce our new methods. In order to evaluate our methods, we implemented previous and our methods and conducted experiment using Glycan dataset [10] and CSLOGS dataset [18]. The results of our experiments are shown in Sect. 5. Finally, we conclude our paper with some discussions.

2 Preliminaries

Let T be a rooted tree. The root of T is denoted by $r(T)$. In this paper, we may simply write T to represent the set of nodes of T. For $x, y \in T$, $x \leq y$ means that x is on the unique path between the root and y. If $x \leq y$ and $x \neq y$, we write $x < y$ and say that x is an *ancestor* of y and y is a *descendant* of x. It is easy to see that the relation $\leq$ is a partial order on T. The *parent* of x, denoted by $p(x)$, is the closest ancestor of x. A node $y > x$ is called a *child* of x if there is no z with $x < z < y$. The set of children of x is denoted by $C(x)$. We call the number of children of x the *degree* of x. A node x is called a *leaf* if it has no children. The set of all leaves of a tree T is denoted by $L(T)$. Nodes x and y are *siblings* if they have the same parent. A tree is called unordered tree if there is no order among siblings. Let Σ be a finite alphabet and $l_T : T \to \Sigma$ a labeling function. A tuple (T, l_T) is called a *labeled* tree. For $x \in T$, we use $T(x)$ to denote the subtree of T rooted at x. For notational convenience, we simply write $T - x$ to denote the subgraph of T obtained by removing a node x.

2.1 Tree Edit Distance

The tree edit distance between two trees is defined as the minimum cost of *edit operations* to transform a tree into another.

Definition 1 (Edit Operations). *Let T be a tree.* Edit operations *on T consist of the following three operations.*

Substitution. Replace the label of a node in T with a new label.
Deletion. Delete a non-root node t of T, making all children of t be the children of $p(t)$.
Insertion. Insert a new node t as a child of some node v in T, making some children of v be the children of t.

Let $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$, where ε is a blank symbol not in Σ. In order to describe costs on edit operations, we denote each of the edit operations by a pair in $\Sigma_\varepsilon \times \Sigma_\varepsilon \setminus \{(\varepsilon, \varepsilon)\}$. Substituting a node labeled with a by another node labeled with b is denoted by (a, b). Inserting a node labeled with b is denoted by (ε, b). Deleting a node labeled with a is denoted by (a, ε). Let $d : \Sigma_\varepsilon \times \Sigma_\varepsilon \setminus \{(\varepsilon, \varepsilon)\} \to \mathbb{R}^+$ be a cost function on edit operations. Assume, in this paper, that d is a metric. In the following, we simply write $d(x, y)$ for $(x, y) \in T_1 \times T_2$ to represent $d(l_1(x), l_2(y))$, where l_1 and l_2 are labeling functions on two trees T_1 and T_2, respectively.

Let $E = \langle e_1, e_2, \ldots, e_t \rangle$ be a sequence of edit operations, where $e_i = (a_i, b_i)$ for $a_i, b_i \in \Sigma_\varepsilon$. The cost of the sequence is defined as $cost(E) = \sum_{1 \leq i \leq t} d(e_i)$.

Definition 2 (Tree Edit Distance [16]). *Let T_1 and T_2 be trees and $\mathcal{E}(T_1, T_2)$ the set of all sequences of edit operations which transform T_1 into T_2. The* tree edit distance *between T_1 and T_2 is defined as $D_{Edit}(T_1, T_2) = \min_{E \in \mathcal{E}(T_1,T_2)} cost(E)$.*

A *mapping* between T_1 and T_2 is a subset of $T_1 \times T_2$. The set of nodes that belongs to a mapping M is denoted by $V(M)$. Tai [16] gave a combinatorial characterization of the tree edit distance by means of a mapping, which is called a *Tai mapping*.

Definition 3 (Tai Mapping [16]). *Let T_1 and T_2 be trees. A mapping M is called a* Tai mapping *if it satisfies the following constraints for every $(x, y), (x', y')$ in M:*

One-to-one correspondence: $x = x' \Leftrightarrow y = y'$,
Preserving ancestor-descendant relationship: $x < x' \Leftrightarrow y < y'$.

The cost of a Tai mapping M is defined as

$$cost(M) = \sum_{(x,y)\in M} d(x, y) + \sum_{x \in T_1 \setminus V(M)} d(x, \varepsilon) + \sum_{y \in T_2 \setminus V(M)} d(\varepsilon, y).$$

Let $\mathcal{M}^{\text{Tai}}(T_1, T_2)$ be the set of all Tai mappings between T_1 and T_2. Tai [16] showed the following theorem.

Theorem 1 ([16]). *For two trees T_1 and T_2, $D_{Edit}(T_1, T_2) = \min_{M \in \mathcal{M}^{Tai}(T_1,T_2)} cost(M)$.*

2.2 Variants of Edit Distance

The tree edit distance is one of the most widely used to measure a similarity between two trees. However, it may not be appropriate for some applications because one may need a distance on which some specific structure of trees is reflected. Many variants of the tree edit distance have been proposed in the literature [9,17]. We work on the following three variants, which are defined by mappings rather than edit operations.

Definition 4 (Segmental Mapping [9]). *Let T_1 and T_2 be trees. A Tai mapping M between T_1 and T_2 is called a* segmental mapping *if for any $(x, y), (x', y') \in M$ with $x < x'$ and $y < y'$, $(p(x'), p(y')) \in M$.*

Definition 5 (Bottom-up Segmental Mapping [9]). *Let T_1 and T_2 be trees. A segmental mapping M between T_1 and T_2 is called a* bottom-up segmental mapping *if for any $(x, y) \in M$, there is $(x', y') \in M$ such that x', y' are leaves with $x \leq x'$ and $y \leq y'$.*

Definition 6 (Bottom-up Mapping [17]). *Let T_1 and T_2 be trees. A Tai mapping M between T_1 and T_2 is called a* bottom-up mapping *if for any $(x, y) \in M$, the submapping obtained from M by restricting to $C(x) \times C(y)$ forms a bijection between $C(x)$ and $C(y)$.*

Let us note that the condition in Definition 6 can be restated in the following way: M is a bottom-up mapping if for any $(x, y) \in M$, the submapping obtained from M by restricting to $T_1(x) \times T_2(y)$ is an isomorphism mapping, ignoring the label information.

Definition 7 ([9,17]). *Let T_1 and T_2 trees. Denote the sets of all possible segmental mappings, bottom-up segmental mappings, and bottom-up mappings between T_1 and T_2 by $\mathcal{M}^{Sg}(T_1, T_2), \mathcal{M}^{BotSg}(T_1, T_2)$, and $\mathcal{M}^{Bot}(T_1, T_2)$, respectively. The* segmental distance, bottom-up segmental distance, *and* bottom-up distance *between T_1 and T_2, which are denoted by $D_{Sg}(T_1, T_2), D_{BotSg}(T_1, T_2)$, and $D_{Bot}(T_1, T_2)$ respectively, are defined as follows:*

$$
\begin{aligned}
D_{Sg}(T_1, T_2) &= \min_{M \in \mathcal{M}^{Sg}(T_1, T_2)} cost(M) \\
D_{BotSg}(T_1, T_2) &= \min_{M \in \mathcal{M}^{BotSg}(T_1, T_2)} cost(M) \\
D_{Bot}(T_1, T_2) &= \min_{M \in \mathcal{M}^{Bot}(T_1, T_2)} cost(M).
\end{aligned}
$$

3 Previous Method [11]

In the rest of this paper, fix input trees T_1 and T_2, and let $n = |T_1|$ and $m = |T_2|$. Kondo et al. [11] proposed an integer linear programming formulation for the tree edit distance. For the tree edit distance between T_1 and T_2, we introduce a binary variable $m_{x,y}$ for every $(x, y) \in T_1 \times T_2$ which takes value 1 if and only if $(x, y) \in \mathcal{M}^{\mathrm{Tai}}(T_1, T_2)$. Then, we can reformulate the cost of a Tai mapping M as:

$$
\begin{aligned}
cost(M) &= \sum_{(x,y) \in M} d(x, y) + \sum_{x \in T_1 \setminus V(M)} d(x, \varepsilon) + \sum_{y \in T_2 \setminus V(M)} d(\varepsilon, y) \\
&= \sum_{(x,y) \in T_1 \times T_2} d(x, y) m_{x,y} + \sum_{x \in T_1} d(x, \varepsilon) \left\{ 1 - \sum_{y \in T_2} m_{x,y} \right\} \\
&\quad + \sum_{y \in T_2} d(\varepsilon, y) \left\{ 1 - \sum_{x \in T_1} m_{x,y} \right\} \\
&= \sum_{(x,y) \in T_1 \times T_2} \{ d(x, y) - d(x, \varepsilon) - d(\varepsilon, y) \} m_{x,y} + \sum_{x \in T_1} d(x, \varepsilon) + \sum_{y \in T_2} d(\varepsilon, y).
\end{aligned}
$$

The two constraints of Tai mapping are directly formulated as the following inequalities:

$$
\begin{aligned}
\sum_{y \in T_2} m_{x,y} &\leq 1 \quad \text{for all } x \in T_1, \\
\sum_{x \in T_1} m_{x,y} &\leq 1 \quad \text{for all } y \in T_2, \\
m_{x,y} + m_{x',y'} &\leq 1 \quad \text{for all } (x, y), (x', y') \in T_1 \times T_2 \text{ s.t. } x < x' \not\Leftrightarrow y < y'.
\end{aligned}
$$

The first two constraints are equivalent to the one-to-one correspondence of Tai mapping: For any node $x \in T_1$ (resp. $y \in T_2$), at most one node of T_2 (resp. T_1) is allowed to be paired. The third constraint is equivalent to the ancestor-descendant preservation: For any two pairs which do not preserve the ancestor-descendant relationship, both of them cannot be included in M simultaneously. This formulation contains $O(nm)$ variables and $O(n^2m^2)$ constraints.

Kondo et al. also gave IP formulations for the segmental distance and bottom-up segmental distance. These distances can be formulated by imposing additional constraints on the formulation of the tree edit distance. In regard of the segmental mapping, the constraints of segmental mapping can be represented as follows:

$$m_{x,y} + m_{x',y'} \leq m_{p(x'),p(y')} + 1, \text{ for all } (x,y),(x',y') \in T_1 \times T_2 \text{ s.t. } x < x' \text{ and } y < y'.$$

The constraints of bottom-up segmental mapping can also be represented as follows:

$$m_{x,y} \leq \sum_{\substack{x' \in L(T_1(x)),\\ y' \in L(T_2(y))}} m_{x',y'}, \text{ for all } (x,y) \in T_1 \times T_2 \text{ s.t. } x \notin L(T_1) \text{ and } y \notin L(T_2).$$

The above two formulations also contain $O(nm)$ variables and $O(n^2m^2)$ constraints.

4 Improved Method

4.1 Improved Method for Tree Edit Distance

In this subsection, we propose a new IP formulation for the edit distance problem by combining a dynamic programming approach due to [14]. The dynamic programming computes a minimum cost Tai mapping $M_{x,y}$ between $T_1(x)$ and $T_2(y)$ with $(x,y) \in M_{x,y}$ for $(x,y) \in T_1 \times T_2$ in a bottom-up manner. Once we have the solutions for all pairs $(x,y) \in T_1 \times T_2$, we can construct a minimum cost Tai mapping between T_1 and T_2.

First, we modify the objective function

$$\text{minimize} \sum_{(x,y) \in T_1 \times T_2} \{d(x,y) - d(x,\varepsilon) - d(\varepsilon,y)\} m_{x,y} + \sum_{x \in T_1} d(x,\varepsilon) + \sum_{y \in T_2} d(\varepsilon,y)$$

to

$$\text{maximize} \sum_{(x,y) \in T_1 \times T_2} w_{x,y} m_{x,y},$$

where $w_{x,y} = d(x,\varepsilon) + d(\varepsilon,y) - d(x,y)$. This modification is valid since the second and third terms do not affect the minimization. Since the solution of our

subproblem for $T_1(x)$ and $T_2(y)$ must contain the root pair (x, y), the objective function on the input trees $T_1(x)$ and $T_2(y)$ can be represented as

$$\text{maximize} \sum_{(x',y') \in (T_1(x)-x) \times (T_2(y)-y)} w_{x',y'} m_{x',y'} + w_{x,y}. \tag{1}$$

We denote by $W_{x,y}$ the maximum value of (1). If at least one of x and y is a leaf, $W_{x,y} = w_{x,y}$. Thus, in the following, we assume that neither x nor y is a leaf. The idea for our dynamic programming is that $W_{x,y}$ can be recursively computed from the values $W_{x',y'}$ for $x < x'$ and $y < y'$. To be precise, let $\mathcal{M}^*(T_1(x), T_2(y))$ be the set of all Tai mappings M between $T_1(x)$ and $T_2(y)$ such that $x, y \notin V(M)$ and both $T_1 \cap V(M)$ and $T_2 \cap V(M)$ are antichains in $(T_1(x), \leq)$ and $(T_2(y), \leq)$, respectively. We call $M \in \mathcal{M}^*(T_1(x), T_2(y))$ an *incomparable mapping* between $T_1(x)$ and $T_2(y)$. For a Tai mapping M, let $w(M) = \sum_{(x,y)\in M} w_{x,y}$ and for an incomparable mapping M, let $W(M) = \sum_{(x,y)\in M} W_{x,y}$. The following lemma is a key ingredient of our formulation.

Lemma 1. $W_{x,y} = \max_{M \in \mathcal{M}^*(T_1(x), T_2(y))} W(M) + w_{x,y}$.

Proof. We first show that the left-hand side is at most the right-hand side. Let M be a Tai mapping between $T_1(x)$ and $T_2(y)$ with $(x, y) \in M$ and $w(M) = W_{x,y}$. Then, M can be uniquely decomposed into $\{(x, y)\}, M_{x_1,y_1}, M_{x_2,y_2}, \ldots, M_{x_k,y_k}$ such that for any $1 \leq i \leq k$, M_{x_i,y_i} is a Tai mapping between $T_1(x_i)$ and $T_2(y_i)$ with $(x_i, y_i) \in M_{x_i,y_i}$ and $\{(x_i, y_i) : 1 \leq i \leq k\} \in \mathcal{M}^*(T_1(x), T_2(y))$. Such a decomposition can be obtained by choosing minimal node pairs $(x_i, y_i) \in M \setminus \{(x, y)\}$ with respect to $\leq$: For any $(x', y') \in M$ either $x_i \leq x'$ and $y_i \leq y'$, or x_i and y_i are not comparable to x' and y', respectively. For each $1 \leq i \leq k$, we have $w(M_{x_i,y_i}) \leq W_{x_i,y_i}$. Therefore, $W_{x,y} = w(M) = \sum_{1\leq i\leq k} w(M_{x_i,y_i}) + w_{x,y} \leq \sum_{1\leq i\leq k} W_{x_i,y_i} + w_{x,y} \leq \max_{M^* \in \mathcal{M}^*(T_1(x),T_2(y))} W(M^*) + w_{x,y}$.

To show the converse, let M be an incomparable mapping between $T_1(x)$ and $T_2(y)$. For each $(x', y') \in M$, we let $M_{x',y'}$ be a Tai mapping between $T_1(x')$ and $T_2(y')$ such that $W_{x',y'} = w(M_{x',y'})$ and $(x', y') \in M_{x',y'}$. Since $T_1(x) \cap V(M)$ and $T_2(y) \cap V(M)$ are antichains, $\bigcup_{(x',y')\in M} M_{x',y'} \cup \{(x, y)\}$ is a Tai mapping between $T_1(x)$ and $T_2(y)$. Therefore, we have$W(M) + w_{x,y} \leq \sum_{(x',y')\in M} w(M_{x',y'}) + w_{x,y} \leq W_{x,y}$ and hence the lemma holds. □

By Lemma 1, our problem is to maximize

$$\sum_{(x',y')\in M} W_{x',y'} m_{x',y'} + w_{x,y}$$

subject to $M \in \mathcal{M}^*(T_1(x), T_2(y))$.

Mori et al. [14] reduced the problem of finding a maximum weight incomparable mapping to the maximum vertex weight clique problem, which corresponds to the maximum weight independent set problem on complement graphs. Their reduction can be interpreted as the following constraint:

$$m_{x',y'} + m_{x'',y''} \leq 1 \text{ for all } (x', y'), (x'', y'') \in T_1(x) \times T_2(y) \text{ s.t. } x' < x'' \text{ or } y' < y''.$$

However, this formulation contains $\Omega(n^2m^2)$ constraints.

In order to reduce the number of constraints, we will exploit a structure of rooted trees. For a node $x \in T$ and a leaf $l \in L(T(x))$, let P_{xl}^{T} be the unique path between x and l in T. Then, for any $M \in \mathcal{M}^*(T_1(x), T_2(y))$ and any $l \in L(T_1(x))$ (resp. $l \in L(T_2(y))$), at most one node of $P_{xl}^{T_1}$ (resp. $P_{yl}^{T_2}$) can be chosen in M, that is,

$$\sum_{x' \in P_{xl}^{T_1} - x} \sum_{y' \in T_2(y) - y} m_{x',y'} \leq 1 \quad \text{for all } l \in L(T_1(x)),$$
$$\sum_{y' \in P_{yl}^{T_2} - y} \sum_{x' \in T_1(x) - x} m_{x',y'} \leq 1 \quad \text{for all } l \in L(T_2(y)).$$

This is formalized by the following lemma.

Lemma 2. *Let $x \in T_1$ and $y \in T_2$. Then, $W_{x,y}$ can be computed by the following IP.*

$$\begin{aligned}
&\textit{maximize} && \sum_{x' \in T_1(x)-x, y' \in T_2(y)-y} W_{x',y'} m_{x',y'} + w_{x,y} && \\
&\textit{subject to} && \sum_{x' \in P_{xl}^{T_1} - x} \sum_{y' \in T_2(y)-y} m_{x',y'} \leq 1 && \textit{for all } l \in L(T_1(x)) \\
& && \sum_{y' \in P_{yl}^{T_2} - y} \sum_{x' \in T_1(x)-x} m_{x',y'} \leq 1 && \textit{for all } l \in L(T_2(y)) \\
& && m_{x',y'} \in \{0,1\} && \textit{for all } x' \in T_1(x) - x, y' \in T_2(y) - y.
\end{aligned}$$

Proof. By Lemma 1, it suffices to prove that $M = \{(x', y') : x' \in T_1(x), y' \in T_2(y), m_{x',y'} = 1\}$ is an incomparable mapping if and only if $m_{*,*}$ is a feasible solution.

Suppose first that $M \in M^*(T_1(x), T_2(y))$. Since $T_1(x) \cap V(M)$ forms an antichain in $(T_1, \leq)$, M has at most one node in $P_{xl}^{T_1}$ for each $l \in L(T_1(x))$. Therefore, binary variables $m_{x',y'}$ do not violate the first type constraints. A symmetric argument for $T_2(y) \cap V(M)$ implies that $m_{*,*}$ is a feasible solution for the IP.

Suppose, for contradiction, $m_{*,*}$ is a feasible solution and there are $(x', y'), (x'', y'')$ in M that violate the condition of incomparable mapping. There are two possibilities: (x', y') and (x'', y'') violate the one-to-one correspondence of Tai mapping or at least one of $x' < x''$ or $y' < y''$ holds. For the former case, assume without loss of generality that $x' = x''$ and $y' \neq y''$. In this case, the pairs contribute at least two to a constraint for each $l \in T_1(x')$, which contradict the feasibility of $m_{*,*}$. For the latter case, assume without loss of generality that $x' < x''$. In this case, there is a path $P_{xl}^{T_1} - x$ that contains both x' and x''. The pairs contribute at least two to a constraint for such $l \in L(T_1(x))$, which also contradict the feasibility of $m_{*,*}$. Therefore, the lemma holds. □

For $x \in T_1$ and $y \in T_2$, we can compute $W_{x,y}$ by using the formulation of Lemma 2. The remaining task is to compute $D_{Edit}(T_1, T_2)$ from the values $W_{x,y}$.

Theorem 2. *Let opt be the optimal value of the following IP. Then,* $D_{Edit}(T_1, T_2) = \sum_{x \in T_1} d(x, \varepsilon) + \sum_{y \in T_2} d(\varepsilon, y) - opt$.

$$
\begin{aligned}
&\textit{maximize} && \sum_{x \in T_1, y \in T_2} W_{x,y} m_{x,y} \\
&\textit{subject to} && \sum_{x \in P^{T_1}_{r(T_1)l}} \sum_{y \in T_2} m_{x,y} \leq 1 \ \textit{for all } l \in L(T_1) \\
& && \sum_{y \in P^{T_2}_{r(T_2)l}} \sum_{x \in T_1} m_{x,y} \leq 1 \ \textit{for all } l \in L(T_2) \\
& && m_{x,y} \in \{0,1\} \qquad \textit{for all } x \in T_1, y \in T_2.
\end{aligned}
$$

The proof of Theorem 2 is analogous to those of Lemmas 1 and 2. Our method has $O(nm)$ subproblems, each of which contains $O(nm)$ variables and only $O(|L(T_1) + |L(T_2)|)$ constraints.

4.2 Improved Methods for Variants of Edit Distance

We have seen that the tree edit distance can be computed by the following two steps: (1) for each $x \in T_1$ and $y \in T_2$, compute $W_{x,y}$, and (2) combine the solutions $W_{x,y}$ of subproblems to obtain the tree edit distance between T_1 and T_2 as in Theorem 2. In this subsection, we show that the segmental distance, bottom-up segmental distance, and bottom-up distance can be computed in the same manner.

Segmental Distance. Let x and y be nodes of two trees T_1 and T_2, respectively. We denote here by $W_{x,y}$ the maximum weight, that is the maximum value of (1), of segmental mappings $M_{x,y}$ between $T_1(x)$ and $T_2(y)$ with $(x, y) \in M_{x,y}$. If either x or y is a leaf, we have $W_{x,y} = w_{x,y}$. Thus, we suppose otherwise. Suppose $W_{x',y'}$ have already computed for each $(x', y') \in (T_1(x) \times T_2(y)) \setminus \{(x, y)\}$. Observe that for any segmental mapping $M_{x,y}$ with $(x, y) \in M_{x,y}$, if a child of x is in $V(M_{x,y})$, it must be paired with a child of y in $V(M_{x,y})$. Moreover, if a descendant x' of x that is not a child of x is in $V(M_{x,y})$, the child of x that is an ancestor of x' must be in $V(M_{x,y})$. These observations imply that $M_{x,y}$ can be constructed by a disjoint union of mappings $M_{x',y'}$ for $x' \in C(x)$ and $y' \in C(y)$, where $M_{x',y'}$ is a segmental mapping between $T_1(x')$ and $T_2(y')$ with $(x', y') \in M_{x',y'}$. Therefore, in order to compute $W_{x,y}$, we construct a bipartite graph $G_{x,y}$ as follows. For each $z \in C(x) \cup C(y)$, we create a vertex v_z and for each $x' \in C(x)$ and $y' \in C(y)$, add an edge between $v_{x'}$ and $v_{y'}$ whose weight equals $W_{x',y'}$. The maximum weight of a matching in $G_{x,y}$ is exactly $W_{x,y}$. It is well-known that a maximum weight bipartite matching can be solved in polynomial time using Hungarian method [13].

When $W_{x,y}$ is computed for each $x \in T_1$ and $y \in T_2$, we can compute the segmental distance between T_1 and T_2 by using the IP formulation described in Theorem 2.

Bottom-Up Segmental Distance. Since any bottom-up segmental mapping is a segmental mapping, the above observations also hold and each subproblem can be reduced to a maximum weight matching problem in a bipartite graph as well.

The only difference from the case of segmental distance is that for every node z in $V(M_{x,y})$, there is a leaf that is a descendant of z in $V(M_{x,y})$. To this end, we need to exclude the following two cases from our solution. If exactly one of x and y is a leaf, then $W_{x,y}$ must be zero since (x, y) violates the condition of bottom-up segmental mapping. The other case is that neither x nor y is a leaf and the solution of the maximum weight matching equals zero. This implies that an optimal mapping between $T_1(x)$ and $T_2(y)$ consists of a single pair (x, y), which also violates the condition of bottom-up segmental mapping. Therefore, we set $W_{x,y} = 0$ in this case.

Bottom-Up Distance. First, we propose a naive IP formulation for computing bottom-up distance. A straightforward implication from Definition 6 is that if $(x, y) \in M$, the mapping between $C(x)$ and $C(y)$ must be a bijection. A naive formulation can be obtained from that of Tai mapping by adding the following constraints:

$$m_{x,y} \leq \sum_{y' \in C(y)} m_{x',y'} \text{ for all } (x, y) \in T_1 \times T_2 \text{ and for all } x' \in C(x),$$
$$m_{x,y} \leq \sum_{x' \in C(x)} m_{x',y'} \text{ for all } (x, y) \in T_1 \times T_2 \text{ and for all } y' \in C(y).$$

This formulation contains $O(nm)$ variables and $O(n^2m^2)$ constraints.

Since bottom-up mapping is a subclass of bottom-up segmental mapping, we can apply the technique used for the bottom-up segmental distance as well. All we have to do is consider the case when two trees $T_1(x)$ and $T_2(y)$ are structurally isomorphic, i.e., they are isomorphic ignoring the labels. Thus, for $x \in T_1$ and $y \in T_2$, we set $W_{x,y} = 0$ if two subtrees $T_1(x)$ and $T_2(y)$ are not structurally isomorphic.

Our improved methods for the above three distances contain $O(nm)$ subproblems, each of which can be solved in polynomial time. For combining the solutions of these subproblems, we need to solve an integer program in Theorem 2. Such IPs also have $O(nm)$ variables and $O(n + m)$ constraints.

5 Experiments

To compare the experimental performance of our methods and the previous methods, we applied them to real tree-structured data. We used glycan data obtained from KEGG/Glycan database [10] and CSLOGS dataset [18] which consists of web log files. In our experiments, we adopt the *unit cost* for the cost function, which is defined as:

$$d(x, y) = \begin{cases} 0 & \text{if } l_1(x) = l_2(y) \\ 1 & \text{otherwise} \end{cases}.$$

We implemented the previous methods for computing edit distance (IP_Edit), segmental distance (IP_Sg), and bottom-up segmental distance (IP_BotSg) given

by Kondo et al. [11] and a naive method for computing bottom-up distance (IP_Bot) described in the previous section. We also implemented our methods for computing these four distances (DpIP_Edit, DpIP_Sg, DpIP_BotSg, and DpIP_Bot). In addition to the above implementations, we intended to compare our methods with the algorithm due to Mori et al. [14]. Their algorithm reduces the tree edit distance problem to the maximum weight clique problem and uses the maximum weight clique algorithm due to [15]. However, the purpose of our experiments is to compare formulations or reductions rather than the performance of specific IP or other solvers. Therefore, we used an ordinary IP formulation of the maximum weight clique problem instead of the algorithm of [15], which is denoted by IP_DpClique_E.

We implemented the methods mentioned above in Java 1.8 combined with IBM ILOG CPLEX 12.7. We have forced CPLEX to run in sequential mode, setting parameter `IloCplex.IntParam.Threads` to one. Every implementation of the presented methods is also single-threaded. The experiments were performed using a computer with 3.7 GHz Quad-Core Intel Xeon E5 and 32 GB RAM, under the Mac OS X.

5.1 Glycan Dataset

The results for edit distance with Glycan dataset are shown in Table 1. "# of nodes" in the table means the total number of nodes of two input trees. We randomly selected at most 100 input tree pairs from the Glycan dataset for each range of total number of nodes. Avg and t.o. stand for average execution time (in seconds) of successfully computed within 30 s and the number of instances timed out, respectively. "*" means that all instances in the range timed out. The table shows that DpIP_Edit is much faster than IP_Edit. IP_DpClique_E is slightly faster than IP_Edit. It is shown that DpIP_Edit also outperforms IP_DpClique_E. It implies that it is not sufficient to adopt a dynamic programming approach for

Table 1. Experimental results with Glycan for edit distance

# of nodes	# of instances	IP_Edit		DpIP_Edit		IP_DpClique_E	
		avg	t.o	avg	t.o	avg	t.o
50–54	100	2.393	0	0.308	0	0.994	0
55–59	100	4.661	0	0.417	0	1.576	0
60–64	88	11.661	6	0.576	0	2.894	0
65–69	36	17.774	4	0.669	0	3.433	0
70–74	100	13.209	7	0.654	0	11.799	7
75–79	29	20.771	9	0.823	0	11.411	7
80–84	9	18.705	8	1.094	0	14.941	6
85–89	5	0	5	1.330	0	21.838	3
90–94	4	0	4	1.442	0	0	4

Table 2. Experimental results with Glycan for segmental distance, bottom-up segmental distance, and bottom-up distance

# of nodes	# of instances	IP_Sg		DpIP_Sg		IP_BotSg		DpIP_BotSg		IP_Bot		DpIP_Bot	
		avg	t.o	avg	t.o	avg	t.o	avg	t.o	avg	t.o	avg	t.o
50–54	100	5.306	0	0.135	0	1.545	0	0.136	0	0.569	0	0.131	0
55–59	100	9.070	5	0.135	0	2.539	0	0.139	0	0.785	0	0.131	0
60–64	88	13.983	41	0.137	0	4.767	0	0.142	0	1.258	0	0.132	0
65–69	36	23.813	27	0.140	0	6.219	0	0.147	0	1.544	0	0.133	0
70–74	100	20.408	97	0.145	0	10.252	4	0.150	0	1.453	0	0.134	0
75–79	29	21.274	27	0.148	0	12.794	5	0.154	0	2.021	0	0.137	0
80–84	9	0	9	0.152	0	17.606	3	0.160	0	3.002	0	0.137	0
85–89	5	0	5	0.157	0	29.157	4	0.163	0	3.869	0	0.142	0
90–94	4	0	4	0.161	0	0	4	0.166	0	4.476	0	0.145	0

improving on the practical performance, and the revised IP formulation derived from the dynamic programming is of great importance for reducing the running time on the tree edit distance problem.

Table 2 shows the results for the variants of edit distance. For segmental distance and bottom-up segmental distance, the proposed methods (DpIP_Sg and DpIP_BotSg) finished computing within 1 s while the naive methods (IP_Sg and IP_BotSg) take longer than 30 s if the total size of input trees is large. For bottom-up distance, the naive method (IP_Bot) successfully computed within 30 s for all instances. However, our improved method (DpIP_Bot) is still much faster than the naive method.

Table 3. Experimental results with SUBLOG3 for edit distance

# of nodes	# of instances	IP_Edit		DpIP_Edit		IP_DpClique_E	
		avg	t.o	avg	t.o	avg	t.o
50–54	100	2.478	0	0.435	0	3.853	0
55–59	100	3.892	0	0.510	0	5.393	2
60–64	100	6.641	0	0.633	0	8.243	17
65–69	100	9.921	1	0.760	0	7.191	34
70–74	100	15.077	9	0.917	0	8.244	44
75–79	100	16.534	29	1.112	0	6.352	47
80–84	100	19.024	45	1.247	0	5.144	44
85–89	100	21.249	70	1.449	0	4.711	48
90–94	100	23.946	91	1.872	0	6.863	59
95–99	100	26.599	92	2.136	0	7.971	61

5.2 CSLOGS Dataset

We divided CSLOGS dataset into two subsets: SUBLOG3 and SUBLOG49. Every tree in SUBLOG3 (resp. SUBLOG49) is restricted to have the maximum degree at most 3 (resp. 49). We randomly selected at most 100 pairs from each dataset with a specified range of the total number of nodes.

The results of computation for SUBLOG3 are shown in Tables 3 and 4. Tables 5 and 6 shows the results for SUBLOG49. Compared to the results in SUBLOG3, the naive methods (IP_Edit, IP_Sg, IP_BotSg, and IP_Bot) in SUBLOG49 works faster. This property is what has been observed in the previous work by Kondo et al. In regard of IP_DpClique_E, it outperforms IP_Edit

Table 4. Experimental results with SUBLOG3 for segmental distance, bottom-up segmental distance and bottom-up distance

# of nodes	# of instances	IP_Sg		DpIP_Sg		IP_BotSg		DpIP_BotSg		IP_Bot		DpIP_Bot	
		avg	t.o	avg	t.o	avg	t.o	avg	t.o	avg	t.o	avg	t.o
50–54	100	5.978	0	0.136	0	1.970	0	0.140	0	0.568	0	0.131	0
55–59	100	10.208	7	0.136	0	2.922	0	0.141	0	0.764	0	0.132	0
60–64	100	13.791	31	0.141	0	5.245	0	0.145	0	1.076	0	0.134	0
65–69	100	18.372	57	0.144	0	6.562	1	0.148	0	1.390	0	0.135	0
70–74	100	20.195	75	0.146	0	8.513	15	0.151	0	1.856	0	0.137	0
75–79	100	22.485	87	0.149	0	11.003	10	0.154	0	2.372	0	0.138	0
80–84	100	22.865	91	0.150	0	12.489	18	0.157	0	3.031	0	0.139	0
85–89	100	26.028	94	0.154	0	14.864	25	0.160	0	3.746	0	0.140	0
90–94	100	26.866	98	0.158	0	17.244	48	0.167	0	4.861	0	0.144	0
95–99	100	0	100	0.160	0	18.644	57	0.170	0	5.808	0	0.147	0

Table 5. Experimental results with SUBLOG49 for edit distance

# of nodes	# of instances	IP_Edit		DpIP_Edit		IP_DpClique_E	
		avg	t.o	avg	t.o	avg	t.o
50–54	100	1.275	0	0.263	0	1.643	0
55–59	100	2.323	0	0.317	0	3.014	0
60–64	100	4.032	0	0.395	0	5.452	3
65–69	100	4.756	0	0.402	0	6.721	6
70–74	100	6.231	1	0.450	0	7.188	10
75–79	100	8.808	10	0.567	0	9.787	19
80–84	100	11.850	6	0.583	0	10.037	28
85–89	100	12.429	21	0.665	0	10.145	34
90–94	100	13.595	33	0.678	0	11.228	34
95–99	100	15.711	30	0.829	0	12.084	39

Table 6. Experimental results with SUBLOG49 for segmental distance, bottom-up segmental distance and bottom-up distance

# of nodes	# of instances	IP_Sg		DpIP_Sg		IP_BotSg		DpIP_BotSg		IP_Bot		DpIP_Bot	
		avg	t.o	avg	t.o	avg	t.o	avg	t.o	avg	t.o	avg	t.o
50–54	100	2.130	0	0.143	0	0.739	0	0.142	0	0.376	0	0.130	0
55–59	100	4.704	0	0.147	0	1.521	0	0.145	0	0.514	0	0.133	0
60–64	100	6.795	11	0.151	0	2.863	3	0.150	0	0.707	0	0.153	0
65–69	100	7.741	8	0.162	0	2.544	1	0.154	0	0.830	0	0.135	0
70–74	100	9.277	19	0.158	0	3.257	2	0.159	0	1.036	0	0.139	0
75–79	100	12.421	38	0.162	0	5.143	6	0.162	0	1.376	0	0.139	0
80–84	100	12.707	39	0.167	0	5.788	7	0.169	0	1.644	0	0.142	0
85–89	100	14.817	46	0.170	0	7.136	3	0.176	0	2.129	0	0.144	0
90–94	100	13.267	65	0.175	0	8.479	8	0.179	0	2.361	0	0.147	0
95–99	100	16.752	65	0.181	0	8.776	16	0.184	0	2.881	0	0.148	0

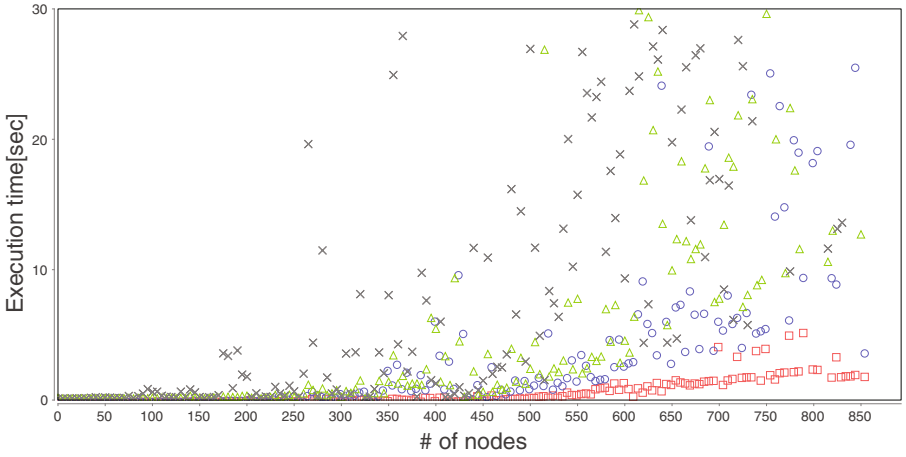

Fig. 1. The crosses, triangles, circles and squares represent the instances of the edit distance, segmental distance, bottom-up distance, and bottom-up segmental distance problem, respectively.

when the degrees of trees are small, though their performances are scarcely different with high-degree inputs.

We can observe that the proposed methods (DpIP_Edit, DpIP_Sg, DpIP_BotSg, and DpIP_Bot) remarkably outperformed the previous methods (IP_Edit, IP_Sg, IP_BotSg, and IP_Bot) as most of instances are computed within 2 s. In order to measure the scalability of the proposed methods, we used the wide range of dataset. We selected input tree pairs so that the number of total nodes ranges from around 0 to around 850. The results are shown in Fig. 1. For segmental distance and bottom-up segmental distance, the smallest instance

which exceeds our time limit of 30 s appears when the total number of nodes belongs to range 450–500 whereas it appears for the tree edit distance when the number of nodes belongs to range 150–200. For bottom-up distance, all instances selected in this experiments are solved within 7 s.

6 Conclusion and Discussion

We have proposed improved methods for computing the tree edit distance and its variants. While the naive IP formulation proposed by Kondo et al. [11] has $O(n^2m^2)$ constraints, our efficient IP formulation, though it has $O(nm)$ subproblems, only has $O(n + m)$ constraints. In case of segmental distance, bottom-up segmental distance and bottom-up distance, each subproblem, except for the problem combining the solutions of subproblems, can be reduced to the maximum weighted matching problem in a bipartite graph, which can be solved in polynomial time.

We performed some experiments using real tree-structured dataset. While the previous method only works for small-sized trees, our methods are still effective for large-sized trees. In particular, for segmental distance and bottom-up segmental distance, our methods are available for trees whose total size is up to 450, and for bottom-up distance, every instance is solved within 7 s.

An advantage of IP-based method is that we can easily give an IP formulation for another distance by adding some constraints to the IP formulation for edit distance. Therefore, extending our method to another important distance measure between unordered trees such as tree alignment distance [8] would be our future work. It would be interesting to develop practical algorithms for computing those distances without using general purpose solvers such as IP solvers or SAT solvers.

References

1. Akutsu, T., Fukagawa, D., Halldorsson, M.M., Takasu, A., Tanaka, K.: Approximation and parameterized algorithms for common subtrees and edit distance between unordered trees. Theor. Comput. Sci. **470**, 10–22 (2013)
2. Akutsu, T., Fukagawa, D., Takasu, A., Tamura, T.: Exact algorithms for computing the tree edit distance between unordered trees. Theor. Comput. Sci. **412**(4–5), 352–364 (2011)
3. Akutsu, T., Tamura, T., Fukagawa, D., Takasu, A.: Efficient exponential-time algorithms for edit distance between unordered trees. J. Discrete Algorithms **25**, 79–93 (2014)
4. Demaine, E.D., Mozes, S., Rossman, B., Weimann, O.: An optimal decomposition algorithm for tree edit distance. ACM Trans. Algorithms **6**(1), 1–19 (2009)
5. 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 Bioinform. **12**(Suppl 1), S13 (2011)

6. Higuchi, S., Kan, T., Yamamoto, Y., Hirata, K.: An A* algorithm for computing edit distance between rooted labeled unordered trees. In: Okumura, M., Bekki, D., Satoh, K. (eds.) JSAI-isAI 2011. LNCS (LNAI), vol. 7258, pp. 186–196. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32090-3_17
7. Horesh, Y., Mehr, R., Unger, R.: Designing an A* algorithm for calculating edit distance between rooted-unordered trees. J. Comput. Biol. **13**(6), 1165–1176 (2006)
8. Jiang, T., Wang, L., Zhang, K.: Alignment of trees — an alternative to tree edit. Theor. Comput. Sci. **143**(1), 137–148 (1995)
9. Kan, T., Higuchi, S., Hirata, K.: Segmental mapping and distance for rooted labeled ordered trees. In: Chao, K.-M., Hsu, T., Lee, D.-T. (eds.) ISAAC 2012. LNCS, vol. 7676, pp. 485–494. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35261-4_51
10. Kanehisa, M., Goto, S.: KEGG: Kyoto Encyclopedia of Genes and Genomes. Nucleic Acids Res. **28**(1), 27–30 (2000)
11. Kondo, S., Otaki, K., Ikeda, M., Yamamoto, A.: Fast computation of the tree edit distance between unordered trees using IP solvers. In: Džeroski, S., Panov, P., Kocev, D., Todorovski, L. (eds.) DS 2014. LNCS, vol. 8777, pp. 156–167. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11812-3_14
12. Kuboyama, T.: Matching and Learning in Trees. Ph.D. thesis, The University of Tokyo (2007)
13. Kuhn, H.W.: The Hungarian method for the assignment problem. Naval Res. Logistics Q. **2**(1–2), 83–97 (1955)
14. Mori, T., Tamura, T., Fukagawa, D., Takasu, A., Tomita, E., Akutsu, T.: A clique-based method using dynamic programming for computing edit distance between unordered trees. J. Computat. Biol. **19**(10), 1089–1104 (2012)
15. Nakamura, T., Tomita, E.: Efficient algorithms for finding a maximum clique with maximum vertex weight. Technical report, the University of Electro-Communications (2005). (in Japanese)
16. Tai, K.C.: The tree-to-tree correction problem. J. ACM **26**(3), 422–433 (1979)
17. Valiente, G.: An efficient bottom-up distance between trees. In: Proceedings Eighth Symposium on String Processing and Information Retrieval. IEEE (2001)
18. Zaki, M.: Efficiently mining frequent trees in a forest: algorithms and applications. IEEE Trans. Knowl. Data Eng. **17**(8), 1021–1035 (2005)
19. Zhang, K., Jiang, T.: Some MAX SNP-hard results concerning unordered labeled trees. Inf. Process. Lett. **49**(5), 249–254 (1994)
20. Zhang, K., Statman, R., Shasha, D.: On the editing distance between unordered labeled trees. Inf. Process. Lett. **42**(3), 133–139 (1992)