# Algorithms for Finding a Largest Common Subtree of Bounded Degree

Tatsuya Akutsu[1,a)] Takeyuki Tamura[1,b)] Avraham A. Melkman[2,c)] Atsuhiro Takasu[3,d)]

***Abstract:*** In this report, we consider the largest common subtree problem, which is to find a bijective mapping between subsets of nodes of two input rooted trees of maximum cardinality or weight that preserves labels and ancestry relationship. This problem is known to be NP-hard for unordered trees. In this technical report, we consider a restricted unordered case in which the maximum outdegree of a common subtree is bounded by a constant $D$. We present an $O(n^D)$ time algorithm where $n$ is the maximum size of two input trees, which improves a previous $O(n^{2D})$ time algorithm. We also present an $O((H^2 \cdot 2^{2H-1} \cdot D^{2H})^{D-1} poly(n))$ time algorithm, where $H$ is the maximum height of two input trees.

## 1. Introduction

Developing algorithms for finding a common structure between two or more given data sets is very important in computer science. For tree structured data, extensive studies have been done on finding a *largest common subtree* (LCST)[*1] based on a bijective mapping between subsets of nodes of the two input trees which preserves labels and ancestry relationship, a mapping which is intimately related to the *edit distance* problem for rooted trees [21]. The LCST and related problems have various applications in bioinformatics including comparison of glycans [5], vascular networks [19], and cell lineage data [10]. They also have applications in comparison and search of XML data [11] and documents processed by natural language processing [18]. In many applications, it is required or desirable to treat input trees as unordered trees rather than ordered trees because the ordering of children is not uniquely determined in many cases [5], [10], [11], [18], [19].

For the LCST and edit distance problems for ordered trees, Tai developed an $O(n^6)$ time algorithm [16], where $n$ is the maximum number of nodes in the input trees. After several improvements, Demaine et al. [6] developed an $O(n^3)$ time algorithm and showed that this bound is optimal under a certain model.

On the other hand, the LCST and edit distance problems for unordered trees are known to be NP-hard even for bounded degree input trees [21]. Furthermore, several MAX SNP-hardness results are known for both problems [1], [8], [20]. In order to cope with these hardness results, approximation algorithms [1], fixed-parameter algorithms [1], [2], [15], efficient exponential time algorithms [4], and branch and bound algorithms [10], [14] have been developed for LCST and/or edit distance problems.[*2] However, none of them is yet satisfactory for handling large scale data and thus further development is needed.

Another approach was recently proposed by Akutsu et al. [2]: utilization of a constraint on the maximum outdegree (i.e., the maximum number of children) of common subtrees. They developed an $O(n^{2D})$ time algorithm for computing an LCST of bounded outdegree $D$, where $D$ is a constant. They also developed an $O(n^2)$ time algorithm for the case of $D = 2$. Constraining the maximum outdegree of a common subtree is reasonable in several applications because the maximum outdegree is usually bounded by a small constant in such data as glycans [5], vascular networks [19] and parse trees, and thus the maximum outdegree of common trees should also be bounded by a small constant (otherwise, it would not be a common structure).

In this technical report, we present an improved $O(n^D)$ time algorithm.[*3] The improvement is achieved by reducing the number of combinations to be searched for among the children of each node in an LCST. All combinations are basically examined in the previous $O(n^{2D})$ algorithm, whereas this number is significantly reduced in the improved algorithm by making use of tables to avoid redundant calculations and the property that the parent in an LCST is uniquely determined if at least two of their children are determined from each input tree. Furthermore, we present a parameterized algorithm that works in $O((H^2 \cdot 2^{2H-1} \cdot D^{2H})^{D-1} poly(n))$ time, where $H$ is the maximum height of two input trees and the degree of $poly(n)$ does not de-

[1] Bioinformatics Center, Institute for Chemical Research, Kyoto University, Gokasho, Uji, Kyoto, 611-0011, Japan
[2] Ben-Gurion University of the Negev, Beer-Sheva, 84105, Israel
[3] National Institute of Informatics, Tokyo, 101-8430, Japan
[a)] takutsu@kuicr.kyoto-u.ac.jp
[b)] tamura@kuicr.kyoto-u.ac.jp
[c)] melkmans@cs.bgu.ac.il
[d)] takasu@nii.ac.jp

[*1] Although a LCST is not necessarily a subgraph of the input trees, the term is commonly used in this context.

[*2] LCST and edit distance problems are equivalent in optimization, but are different in approximation and on some parameters.

[*3] The algorithm was significantly simplified from that in a preliminary version [3].

pend on $D$ or $H$. Since LCST is known to be NP-hard even for trees of height at most two [1], this result is meaningful at least from a theoretical viewpoint.[*4]

## 2. Preliminaries

For a rooted unordered tree $T = (V, E)$, $V(T)$ denotes the set of nodes and $r(T)$ denotes the root of $T$. For a node $v \in V(T)$, $p(v)$ denotes the parent of $v$ ($p(v) = v$ if $v$ is the root), $chd(v)$ denotes the set of children of $v$, $deg(v)$ denotes the *outdegree* of $v$ (i.e., $deg(v) = |chd(v)|$), $\ell(v)$ denotes the label of $v$ where a label is given from a finite or infinite alphabet $\Sigma$, $des(v)$ and $anc(v)$ denote the sets of descendants and ancestors of $v$ where $v \notin des(v)$ and $v \notin anc(v)$, and $T(v)$ denotes the subtree of $T$ induced by $v$ and its descendants.

The LCST problem is defined via a bijective *mapping* between subsets of the nodes of two input trees $T_1$ and $T_2$ that preserves the ancestor-descendant relationship: if $u$ is mapped to $v$ and $u'$ to $v'$, then $u$ is an ancestor of $u'$ in $T_1$ iff $v$ is an ancestor of $v'$ in $T_2$. Let $f(u, v)$ denote the weight for a matched node pair $(u, v)$ by a mapping $M$. Then the LCST problem is to find a bijective mapping $M$ maximizing $W(M) = \sum_{(u,v)\in M} f(u, v)$ (see Fig. 1).

If we define $f(u, v)$ by $f(u, v) = del(u) + ins(v) - sub(u, v)$ where $del(u)$, $ins(v)$, and $sub(u, v)$ are the costs for deletion of a node $u$, insertion of a node $v$, and substitution of the label of $u$ by the label of $v$, respectively, it is known [21] that the edit distance (i.e., the minimum cost sequence of editing operations that transforms $T_1$ to $T_2$) is given by

$$\sum_{u\in V(T_1)} del(u) + \sum_{v\in V(T_2)} ins(v) - W(M).$$

If we define the weight function by $f(u, v) = 1$ if $\ell(u) = \ell(v)$, and $f(u, v) = 0$ otherwise, the LCST problem is to find a common subtree (based on a bijective mapping) with the maximum number of nodes. In this technical report, we consider a general weight function $f(u, v)$ and thus nodes with different labels can match each other. However, as mentioned in Section 1, we impose the constraint that the maximum outdegree of a common subtree is at most $D$, that is, the subtree of $T_1$ induced by the nodes appearing in $M$ must have maximum outdegree less than or equal to $D$. Therefore, the LCST problem with maximum outdegree $D$ is to find a mapping $M$ with the maximum weight satisfying this condition.

In Section 4, we put on a node $v \in V(T_1)$, or a node $w \in V(T_2)$, the constraint that it does not appear in any mapping giving a common subtree. Imposing this constraint is equivalent to setting $f(v, y) = -\infty$ for all nodes $y \in V(T_2)$, or $f(x, w) = -\infty$ for all nodes $x \in V(T_1)$. We call such a node *inactive*.

## 3. Previous Algorithms

In this section, we briefly review the previous algorithms for finding an LCST of bounded outdegree $D$ (see [2] for details) since our proposed $O(n^D)$ time algorithm is based on them.

[*4] In a preliminary version [3], we claimed that computation of LCST of bounded outdegree $D$ is $W[1]$-hard. However, there is a crucial error in the proof and it is still unclear whether the problem is $W[1]$-hard for trees of unbounded height.

Let $S(x, y)$ be the weight of an LCST of $T_1(x)$ and $T_2(y)$ of bounded outdegree $D$. Then, $S(x, y)$ can be computed by the following dynamic programming procedure (the initialization part is omitted):

$$S(x,y) = \max \begin{cases} \max_{h=0,\ldots,D} \left\{ \max_{x_1,\ldots,x_h\in des(x), y_1,\ldots,y_h\in des(y)} \left[ \left( \sum_{i=1}^{h} S(x_i, y_i) \right) + f(x, y) \right] \right\}, \\ \max_{y_1\in des(y)} S(x, y_1), \\ \max_{x_1\in des(x)} S(x_1, y), \end{cases} \quad (1)$$

where $x_i \notin des(x_j) \cup \{x_j\}$ and $y_i \notin des(y_j) \cup \{y_j\}$ must be satisfied for any $i \neq j$, and such tuples as $(x_1, \ldots, x_h)$ and $(y_1, \ldots, y_h)$ are called *consistent*. It is straightforward to see that this algorithm works in $O(n^{2D+2})$ time.

This algorithm was improved by using the least common ancestors (LCAs). Let $lca(z_1, z_2, \ldots, z_h)$ denote the LCA of $z_1, z_2, \ldots, z_h$. Then, all $S(x, y)$ can be computed by the following dynamic programming procedure, where it can be made to work in $O(n^{2D})$ time by modifying the innermost 'for' loop [2]:

**Procedure** *LcaBasedLCST*$(T_1, T_2, D)$
  **for** all $(x, y) \in V(T_1) \times V(T_2)$ **do** $S(x, y) \leftarrow f(x, y)$;
  **for** all $h \in \{1, \ldots, D\}$ **do**
    **for** all consistent tuples $(x_1, \ldots, x_h)$ **do**
      $x_a \leftarrow lca(x_1, \ldots, x_h)$;
      **for** all consistent tuples $(y_1, \ldots, y_h)$ **do**
        $y_a \leftarrow lca(y_1, \ldots, y_h)$;
        **for** all $(x, y)$ with $x \in anc(x_a) \cup \{x_a\}$
        and $y \in anc(y_a) \cup \{y_a\}$ **do**
          $S(x, y) \leftarrow \max\{S(x, y), S(x_1, y_1)$
          $+ \cdots + S(x_h, y_h) + f(x, y)\}$;

## 4. Improved Algorithm

### 4.1 Preliminaries

In this section, we present some preliminary considerations that will be useful in the development of an $O(n^D)$ time algorithm for computing an LCST of bounded outdegree $D$.

The following lemma allows attention to be restricted to binary input trees.

**Lemma 4.1.** *If an LCST of bounded outdegree $D$ can be computed in $O(n^{f(D)})$ time for $T_1$ and $T_2$ of bounded outdegree 2 where $D$ is a constant, then an LCST of bounded outdegree $D$ can be computed in $O(n^{f(D)})$ time for any $T_1$ and $T_2$.*

*Proof.* We modify each node $v$ of outdegree $d > 2$ by $d-1$ nodes $v_1, \ldots, v_{d-1}$ with outdegree 2 as shown in Fig. 2. Let $T_1'$ and $T_2'$ be the resulting trees. Then, the maximum outdegree of $T_1'$ and $T_2'$ is 2. We inactivate $v_2, \ldots, v_{d-1}$ (i.e., $v_2, \ldots, v_{d-1}$ cannot appear in a mapping). Then, it is straightforward to see that an LCST of bounded outdegree $D$ for $T_1$ and $T_2$ has the same weight as an LCST of bounded outdegree $D$ for $T_1'$ and $T_2'$ has.

Since the number of nodes in each $T_i'$ is at most $2n - 3 < 2n$, the total computation time is

$$O((2n)^{f(D)}) = O(2^{f(D)} \cdot n^{f(D)}) = O(n^{f(D)})$$

for any constant $D$. □

Accordingly, we assume below that the maximum outdegree of input trees is 2. Furthermore, we can assume w.l.o.g. (without loss of generality) that every internal node has outdegree 2.[*5] We also assume w.l.o.g. that every internal node of LCST has outdegree $D$. We can get such a tree with the same score as the optimal one by adding $D$ children to each internal node of $T_1$ and $T_2$ and letting $f(x, y) = 0$ for any of such children pairs $(x, y)$ and letting $f(x, y) = -\infty$ if exactly one of $x$ and $y$ is such a node (and then applying Lemma 4.1). The desired trees can be obtained by removing nodes corresponding to added nodes. Although this increases the size of input trees, it does not increase the degree of the polynomial in $n$.

In the development of the algorithm it will be convenient to employ the notion of a *consistent* set of descendants of a node.

**Definition 4.2.** *A set of $D$ descendants $\{y_1, \ldots, y_D\}$ of a node $y$ is consistent if none of the descendants is an ancestor of another descendant, and $y$ is the LCA of $\{y_1, \ldots, y_D\}$.*

In order to enumerate all possible sets of consistent descendants we will use binary trees with $D$ leaves. We call such a tree a *skeleton tree* and denote it $T^s$ (see also Fig. 3). For our purposes it will be sufficient to consider only consistent descendants of $y$ that are obtained as follows.

( 1 ) Let $m$ be any mapping of the internal nodes of $T^s$ to nodes of $T$ that maps the root of $T^s$ to $y$, and preserves ancestry relationships.

( 2 ) For each leaf $\ell_i$ of $T^s$, if $\ell_i$ is the left son of internal node $p$ set $y_i = lson(m(p))$, and otherwise set $y_i = rson(m(p))$,

where $lson(x)$ and $rson(x)$ denote the left and right children of $x$, respectively. The totality of sets of $D$ consistent descendants of $y$ obtained in this manner will be denoted $C_D(y)$. For example, $C_2(y)$ is the singleton containing the set $\{lson(y), rson(y)\}$. For simplicity we let $C_1(y)$ be the singleton containing the set $\{y\}$.

Noting that a given skeleton tree with $D$ leaves has only $D - 1$ internal nodes, and that the root of the skeleton tree has to be mapped to $y$, yields the following result.

**Lemma 4.3.** *For any node $y$ the number of sets of $D$ consistent descendants of $y$ obtained as above is $O(n^{D-2})$, i.e. $|C_D(y)| = O(n^{D-2})$.*

### 4.2 Main algorithm

We turn now to the description of the improved LCST algorithm. It computes, successively and bottom-up, the following generalization of $S(x, y)$.

**Definition 4.4.** *Let $k \geq 2$, and let $\Pi_k$ be the set of all permutations $\pi$ on $\{1, \ldots, k\}$. Given a node $x$ in $T_1$ and a set of nodes $Y = \{y_1, \ldots, y_k\} \in C_k(y)$, with $y \in T_2$, define*

$$F_k(x; Y) = \max_{\{x_1,\ldots,x_k\} \in C_k(x)} \max_{\pi \in \Pi_k} \sum_{i=1}^{k} S(x_i, y_{\pi(i)}).$$

For simplicity we set $F_1(x; \{y\}) = \max\{S(lson(x), y), S(rson(x), y)\}$.

The value we wish to compute, $S(x, y)$, can be obtained from $F_D$ by

$$S(x, y) = \max \begin{cases} \max\{S(lson(x), y), S(rson(x), y)\}, \\ \max\{S(x, lson(y)), S(x, rson(y))\}, \\ f(x, y) + \max_{Y \in C_D(y)} F_D(x; Y). \end{cases} \quad (2)$$

To compute $F_D$ we use for $k \geq 2$ the recursion formula

$$F_k(x; Y) = \max_{Z \subset Y, 1 \leq |Z| = j \leq k-1} \{F_j(lson(x); Z) + F_{k-j}(rson(x); Y \setminus Z)\}. \quad (3)$$

In particular, the base case $k = 2$ is given by

$$F_2(x; \{y_1, y_2\}) = \max\{\ S(lson(x), y_1) + S(rson(x), y_2), \\ S(lson(x), y_2) + S(rson(x), y_1)\}.$$

Next we analyze the time required by a bottom-up implementation of this dynamic programming algorithm. Observe first of all that once the necessary values of $F_j$ have been computed, equation (3) takes time that depends only on $k$, i.e. only on $D$ and not on $n$. Computing $F_k(x; Y)$ for all $x$ and all $Y$ with $|Y| \leq D - 1$ takes therefore in all $O(n^D)$ time (with the constant depending on $D$).

Consider now the total time taken up by the computations resulting from equation (2). For fixed $x$ and $y$ the third line examines $|C_D(Y)|$ values of $F_D$, each of which is computed in constant time, using equation (3), from the values of $F_{D-1}$. According to Lemma 4.3, $|C_D(y)| = O(n^{D-2})$. Hence the total time necessary for computing $S(x, y)$ for all $x$ and $y$ is $O(n^D)$.

In summary, the above considerations prove the following Theorem.

**Theorem 4.5.** *A largest common subtree of bounded outdegree $D$ can be computed in $O(n^D)$ time for fixed $D$.*

Readers may think that if the weight of a LCST does not change if the degree bound is changed from $D$ to $D + 1$, then it is an optimal LCST without degree bound. However, it is not difficult to construct a counterexample. Consider the example shown in Fig. 4. We consider a weight function defined by $f(a, a) = 2$, and $f(x, y) = 1$ for any other $(x, y)$. Then, the weight of an LCST is 4 for $D = 2$ and $D = 3$, but is 5 for $D = 4$.

We have so far considered subtrees based on bijective mappings (i.e., subtrees obtained by deletions and substitutions of nodes of arbitrary degree). We can also consider the problem of finding common homeomorphic subtrees (for which only nodes with outdegree at most 1 may be deleted) while imposing the same degree constraints. Although the original problem is known to be solvable in polynomial time [17], the imposition of the same degree constraints enables a speeding up of the running time as follows.

**Theorem 4.6.** *Given trees $T_1$ and $T_2$, on $n_1$ and $n_2$ nodes respectively, a largest common homeomorphic subtree of bounded outdegree $D$ can be computed in time $O(Dn_1n_2)$.*

*Proof.* Although the algorithms of [12] are phrased for unrooted trees they can be easily be adapted to take advantage of the fact that the trees are rooted, and that a bounded degree largest common homeomorphic subtree is required, as follows.

Let $S_D(x, y)$ be the weight of a largest common homeomorphic

[*5] This can be done by adding a dummy child $w$ (i.e., $w$ is inactive) to each node $u$ of outdegree 1.

subtree of bounded outdegree $D$ between $T_1$ and $T_2$. Denote by $C(x)$ the set of children of $x$ in its tree. The recursion for $S(x, y)$ is

$$\begin{aligned} S_D(x,y) = \max\{ & \max\{S_D(x,v) : v \in C(y)\}, \\ & \max\{S_D(u,y) : u \in C(x)\}, \\ & \max\{MWM_D(C(x),C(y)) + f(x,y)\} \\ & \}. \end{aligned}$$

Here $MWM_D(C(x), C(y))$ is the weight of the maximum weight matching of size $D$ between $C(x)$ and $C(y)$. The computation of this weight can be reduced to a min-cost max-flow problem on the flow network with vertices $s_1, s_2, t$ in addition to $C(x)$ and $C(y)$, and the following edges: $(s_1, s_2)$ with capacity $D$ and cost 0, $(s_2, u), u \in C(x)$ and $(v, t), v \in C(y)$ all with capacity 1 and cost 0, and $(u, v)$ with capacity 1 and cost $-S_D(u, v)$ for all $u \in C(x), v \in C(y)$. The number of edges of the network is $d_x d_y + 1 + d_x + d_y$ where $d_x$ is the outdegree of $x$, so that the time for constructing this network is $O(d_x d_y)$. By adapting the arguments of [12] it can be shown that the min-cost flow of size $D$ can be found by repeatedly augmenting the flow by 1 unit along a min-cost path $D$ times, and that the time required is $O(Dd_x d_y)$.

Thus the time taken by a dynamic programming implementation of the recursion is

$$O(\sum_{x \in T_1, y \in T_2} Dd_x d_y) = O(Dn_1 n_2).$$

□

## 5. Parameterized Algorithm

In this section, we present a parameterized algorithm for LCST that works in $O((H^2 \cdot 2^{2H-1} \cdot D^{2H})^{D-1} poly(n))$ time, where $H$ is the maximum height of two input trees and the degree of $poly(n)$ does not depend on $D$ or $H$.

It is to be noted that if the size of an alphabet $\Sigma$ is also considered as a parameter, there exists an almost trivial parameterized algorithm as below.

( 1 ) Enumerate all possible trees under constraints on $D, H, \Sigma$
( 2 ) For each tree, check whether it is a subtree of both input trees using tree inclusion.

It is known that tree inclusion (i.e., deciding whether $T_2$ is obtained from $T_1$ using only insertion operations) for unordered trees can be solved in $O(2^{2D} poly(n))$ time [9]. For our parameterized algorithm the tree inclusion has to be modified so as to take into account the cost of substitutions and insertions. As shown in [13], the modified tree inclusion algorithm also runs in $O(2^{2D} poly(n))$ time. Since the number of possible trees does not depend on $n$, the above algorithm works in $O(f(D, H, |\Sigma|) poly(n))$ time.

Hereafter, we assume that $\Sigma$ is not fixed.

### 5.1 Maximum Weight Bipartite Matching with $d$-Edges

Let $G(U \cup V; E)$ be a bipartite graph in which each edge $e = (u, v)$ has weight $w(e) = w(u, v)$. For a set of edges $M \subseteq E$, we define $w(M) = \sum_{e \in M} w(e)$. We compute a matching $M \subseteq E$ with maximum $w(M)$ under the condition that $|M| = d$, where $d$ is a given constant.

Although this problem can be solved in polynomial time with respect to $|E|$ and $d$, we present a fixed parameter version as an introduction to the fixed-parameter algorithm for LCST. Let $M_{OPT}$ denotes an optimal solution.

The following observations are important (see also Fig. 5).

**Proposition 5.1.** *Let $e = (u, v) \in E$ be an edge with highest weight. Then, either $e$ is an edge in an $M_{OPT}$ or both $u$ and $v$ are endpoints of edges in an $M_{OPT}$.*

**Proposition 5.2.** *Let $e = (u, v) \in E$ be an edge with highest weight. Suppose that $e \notin M_{OPT}$. Let $(u, v_1), \ldots, (u, v_d)$ be edges in $E$ connecting to $u$ with 2nd to $(d+1)$-th highest weights (recall that the highest one is $(u, v)$). Then, one of these edges appears in some optimal solution.*

*Proof.* Assume that $(u, v_0)$ appears in $M_{OPT}$ and any of $(u, v_i)$ $(i \in \{1, \ldots, d\})$ does not appear in $M_{OPT}$. Since $M_{OPT}$ consists of $d$ edges, there is at least one $v_i$ $(i \in \{1, \ldots, d\})$ which does not appear in $M_{OPT}$. By removing $(u, v_0)$ from $M_{OPT}$ and adding $(u, v_i)$ to $M_{OPT}$, we could increase or keep the weight of the matching (see also Fig. 5), which contradicts to the assumption. □

These propositions lead to the following algorithm, where $NE((u, v)) = \{(u', v') | (u', v') \in E \text{ and } (u' = u \text{ or } v' = v)\}$. It is straight-forward to check that this algorithm works in $O((d+1)^d poly(|E|))$ time.

Procedure $FptBipartite(M, E, d)$
  Let $e = (u, v)$ be an edge in $E$ with the highest weight;
  **if** $d = 1$ **then return** $w(M \cup \{(u, v)\})$;
  $w \leftarrow FptBipartite(M \cup \{e\}, E - NE(e), d - 1)$;
  **for all** $v_i \in \{v_1, \ldots, v_d\}$ **do**
    $w_1 \leftarrow FPTbipartite(M \cup \{(u, v_i)\}, E - NE((u, v_i)), d - 1)$;
    **if** $w < w_1$ **then** $w \leftarrow w_1$;
  **return** $w$.

### 5.2 FPT-Algorithm

In the following, for a node $u$ in $T$, we let $Anc(u) = anc(u) \cup \{u\}$ and $Des(u) = des(u) \cup \{u\}$, and $h(u)$ and $d(u)$ denote the height and depth of $u$, respectively ($h(u) = 0$ if $u$ is a leaf, and $d(u) = 0$ if $u$ is the root). For a tree $T$, $h(T)$ denotes the height of $T$ (i.e., $h(T) = \max_{u \in V(T)} h(u)$). Let $S^0(x, y)$ denote the weight of an LCST of $T_1(x)$ and $T_2(y)$ under the condition that $x$ is mapped to $y$. In the following, the score means $S^0(x, y)$.

The basic strategy of the FTP algorithm for LCST, described below, is the same as that of the basic one described by equation (1) in Section 3. However, instead of examining all possible $x_1, \ldots . x_d \in des(x)$ and $y_1, \ldots . y_d \in des(y)$, the only candidate $d$-tuples that are examined are ones generated by a procedure remindful of Section 5.1, but rather more complex and involving some new ideas.

Procedure $FptBdhLCST(T_1, T_2, D)$
  **for all** leaf pairs $(x, y) \in L(T_1) \times L(T_2)$ **do**
    $S^0(x, y) \leftarrow f(x, y)$;
  **for all** other pairs $(x, y) \in V(T_1) \times V(T_2)$ **do**

(in a bottom up manner)
$S^0(x, y) \leftarrow f(x, y) + \max_{d=0,\ldots,D}\{$
$\max_{((x_1,y_1),\ldots,(x_d,y_d))\in CandTuples(T_1(x),T_2(y),d)}[\sum_{i=1}^{d} S^0(x_i, y_i)]\}$;
**return** $\max_{x,y} S^0(x, y)$.

The time complexity of the procedure depends on the size of the set of candidate tuples generated by $CandTuples(T_1(x), T_2(y), d)$. If the size of this set is bounded by $f(D, H)$ and the time required for generation of this set is $O(f(D, H)poly(n))$ where $H$ is the maximum height of two input trees and $n = \max(|V(T_1)|, |V(T_2)|)$, the total time complexity is also $O(f(D, H)poly(n))$. Furthermore, if the set of candidate tuples always contains at least one tuple consisting of children of $(x, y)$ in some LCST then it is clear that the procedure is correct.

Next, we describe how to generate a set of candidate $d$-tuples. Let $V_{=b}(T)$ and $T_{\leq b}$ denote the set of nodes at depth $b$ in $T$ and the subtree of $T$ induced by the nodes of depth at most $b$, respectively. Basically, a set of candidate tuples is generated by applying the approach employed in $FptBipartite(M, E, d)$ to $V_{=b_1}(T_1)$ and $V_{=b_2}(T_2)$ for all pairs of $b_1 \in \{1, 2, \ldots, h(T_1)\}$ and $b_2 \in \{1, 2, \ldots, h(T_2)\}$. However, since many nodes may become unavailable once an ancestor or descendant node appears in an optimal $d$-tuple, we need to keep many more node pairs.

Suppose that we are trying to find a $d$-tuple $\langle(x_1, y_1), \ldots, (x_d, y_d)\rangle$ that maximizes $\sum S^0(x_i, y_i)$, the core part of $FptBdhLCST(T_1, T_2, D)$. We call such a $d$-tuple an *optimal d-tuple* (for $(x, y)$ and $d$). Following is the procedure to generate a set of candidate $d$-tuples, where we use $T^1$ and $T^2$ for the trees from which the candidates are selected, in order to distinguish them from the original input trees $T_1$ and $T_2$ ($T^1$ and $T^2$ are subtrees of $T_1$ and $T_2$, respectively).

Procedure $CandTuples(T^1, T^2, d)$
  **if** $|V(T^1)| = 1$ or $|V(T^2)| = 1$ **then return** {};
  $(x, y) \leftarrow \mathrm{argmax}_{(x,y)\in(V(T^1)-\{r(T^1)\})\times(V(T^2)-\{r(T^2)\})} S^0(x, y)$;
  **if** $d = 1$ **then return** $\{(x, y)\}$;
  $Q \leftarrow \{\}$;
  **for** $b_1 = 1$ **to** $h(T^1)$ **do**
    **for** $b_2 = 1$ **to** $h(T^2)$ **do**
      $R \leftarrow CandPairs(T^1_{\leq b_1}, T^2_{\leq b_2})$;
      **for all** $(x, y) \in R$ **do**
        $P \leftarrow CandTuples(T^1 \ominus x, T^2 \ominus y, d - 1)$;
        $Q \leftarrow (\{(x, y)\} \times P) \cup Q$;
  **return** $Q$.

Let us briefly explain the procedure (see also Fig. 6). Consider the case of $|V(T^1)| > 1$, $|V(T^2)| > 1$, and $d > 1$, the other cases being straightforward. For all depth pairs $(b_1, b_2)$, we generate a set $R$ of pairs each of which is a candidate for participation in an optimal $d$-tuple. For each candidate pair in $R$, we recursively search for candidates for the remaining $d - 1$ pairs. Note that if $x$ appears in a pair in a $d$-tuple, none of its ancestors or descendants can appear in the remaining $d - 1$ pairs. Therefore, denoting by $T \ominus u$ the tree obtained by deleting $Anc(u) \cup des(u) - \{r(T)\}$, the recursive call is made on $T^1 \ominus x$ and $T^2 \ominus y$. For each depth pair $(b_1, b_2)$, $R$ is analogous to the set $\{(u, v), (u, v_1), (u, v_2), \ldots, (u, v_d)\}$ in $FptBipartite(M, E, d)$, although it contains many more pairs.

In order to describe $CandPairs(T^1, T^2)$, we define some terms (see Fig. 7). For a tree $T$, let $\hat{V}(T)$ be a set of nodes each of which has a descendant whose depth in $T$ is $h(T)$. Let $u \in \hat{V}(T)$ be a node at depth $b = h(T) - h$ in $T$. The set of leaves of depth $h$ in $T(u)$ will be called a *level-h block* (headed by $u$), and will be denoted $B(u)$. In particular, a leaf of depth $h$ is a level-0 block, where we identify a leaf with the set consisting of only this leaf. We identify a set of pairs between $B(r(T^1))$ and $B(r(T^2))$ with a set of edges between $B(r(T^1))$ and $B(r(T^2))$. For a node pair $(x', y') \in V(T^1) \times V(T^2)$ and a set of edges $P$, $deg(B(x'), B(y'), P)$ denotes the number of edges between $B(x')$ and $B(y')$ in $P$. If $P$ is clear from the context, we omit $P$ and simply write $deg(B(x'), B(y'))$. We impose the constraint that $deg(B(x'), B(y'), P) \leq g(h(x'), h(y'))d^{h(x')+h(y')}$ for any node pair $(x', y') \in \hat{V}(T^1) \times \hat{V}(T^2)$, where $g(i, j)$ is given by

$$g(i, j) = \begin{cases} 1 & \text{if } i = 0 \text{ or } j = 0, \\ g(i, j-1) + g(i-1, j) & \text{otherwise.} \end{cases}$$

Thus $g(i, j) \leq 2^{i+j-1}$ for $i + j \geq 1$.

The procedure $CandPairs(T^1, T^2)$, whose pseudocode follows below, greadily adds edges (i.e., pairs) between $B(r(T^1))$ and $B(r(T^2))$ (i.e., between leaves of depth $h(T^1)$ in $T^1$ and leaves of depth $h(T^2)$ in $T^2$), in descending order of scores under some degree constraints (see Fig. 8).

Procedure $CandPairs(T^1, T^2)$
  $P \leftarrow \{\}$;
  **for all** pairs $(x, y) \in B(r(T^1)) \times B(r(T^2))$
  in descending order of $S^0(x, y)$ **do**
    **for all** pairs $(x', y') \in \hat{V}(T^1) \times \hat{V}(T^2)$ **do**
      **if** $deg(B(x'), B(y'), P \cup \{(x, y)\})$
      $> g(h(x'), h(y'))d^{h(x')+h(y')}$ **then**
        **skip to** next $(x, y)$;
    $P \leftarrow P \cup \{(x, y)\}$;
  **return** $P$.

### 5.3 Analysis

We begin with analysis of the time complexity.

**Proposition 5.3.** *$FptBdhLCST(T_1, T_2, D)$ works in $O((H^2 \cdot 2^{2H-1} \cdot D^{2H})^{D-1} poly(n))$ time.*

Proof is omitted in this version.

Next, we present a key lemma showing that $CandPairs(T^1, T^2)$ does not miss a required pair. In the following, $M_{OPT}$ means an optimal $d$-tuple for the original trees (i.e., $T_1(x), T_2(x)$ in $FptBdhLCST(T_1, T_2, D)$), where a tuple can be regarded as a set of pairs of nodes.

**Lemma 5.4.** *If there exists a pair $(x_0, y_0) \in M_{OPT}$ such that $(x_0, y_0) \in B(r(T^1)) \times B(r(T^2))$, then $P$ outputted by $CandPairs(T^1, T^2)$ must contain $(x_0, y_0)$ or $(x'_0, y'_0) \in M'_{OPT}$, where $M'_{OPT}$ is another (or the same) optimal d-tuple (for the original trees).*

Proof is omitted in this version.

**Theorem 5.5.** *LCST of bounded outdegree D can be computed in* $O((H^2 \cdot 2^{2H-1} \cdot D^{2H})^{D-1} poly(n))$ *time, where H is the maximum height of two input trees.*

*Proof.* We show by mathematical induction that $Q$ outputted by $CandTuples$ $(T^1, T^2, d)$ contains at least one optimal $d$-tuple for $(r(T^1), r(T^2))$ if it exists.

When $d = 1$, an optimal pair $(x, y)$ is clearly contained in $Q$ (because $Q = \{(x, y)\}$).

Suppose $d > 2$. Let $(x, y)$ be a pair in an optimal $d$-tuple $M_{opt}$, where $d(x) = b_1$ and $d(y) = b_2$. Lemma 5.4 states that $P$ outputted by $CandPairs$ $(T^1_{\leq b_1}, T^2_{\leq b_2})$ contains $(x', y')$ such that $(x', y') \in M_{OPT}$ or $(x', y') \in M'_{OPT}$, where $M'_{OPT}$ is another optimal $d$-tuple, $d(x') = b_1$, and $d(y') = b_2$. Then, the remaining $d - 1$ pairs must be found in the following recursive calls by the assumption of mathematical induction.

Therefore, $Q$ outputted by $CandTuples(T^1, T^2, d)$ contains an optimal $d$-tuple. Since an optimal $d$-tuple is not missed for any $d = 0, 1, \ldots, D$, $FptBdhLCST(T_1, T_2, D)$ works correctly. By combining with Prop. 5.3, we have the theorem. □

It is left as an open problem to decide whether the bounded degree LCST problem without the height constraint is fixed-parameter tractable or $W[1]$-hard.

**Acknowledgment**

We would like to thank Yefim Dinitz for helpful comments. This work was partially supported by the Collaborative Research Programs of National Institute of Informatics. T.A. and T.T. were partially supported by JSPS, Japan: Grant-in-Aid 22650045 and Grant-in-Aid 23700017, respectively.

**References**

[1] T. Akutsu, D. Fukagawa, M.M. Halldórsson, A. Takasu, K. Tanaka, Approximation and parameterized algorithms for common subtrees and edit distance between unordered trees, Theoret. Comput. Sci. 470 (2013) 10–22.

[2] T. Akutsu, D. Fukagawa, A. Takasu, T. Tamura, Exact algorithms for computing tree edit distance between unordered trees. Theoret. Comput. Sci. 421 (2011) 352–364.

[3] T. Akutsu, T. Tamura, A.A. Melkman, A. Takasu, On the complexity of finding a largest common subtree of bounded degree, in: Proc. 19th International Symposium on Fundamentals of Computation Theory, in: LNCS, vol. 8070, Springer, 2013, pp. 4–15.

[4] T. Akutsu, T. Tamura, D. Fukagawa, A. Takasu, Efficient exponential time algorithms for edit distance between unordered trees, in: Proc. 23rd Annual Symposium on Combinatorial Pattern Matching, in: LNCS, vol. 7354, Springer, 2012, pp. 360–372.

[5] K.F. Aoki, A. Yamaguchi, N. Ueda, T. Akutsu, H. Mamitsuka, S. Goto, M. Kanehisa, KCaM (KEGG Carbohydrate Matcher): A software tool for analyzing the structures of carbohydrate sugar chains, Nucl. Acids Res. 32 (2004) W267–W272.

[6] E.D. Demaine, S. Mozes, R. Rossman, O. Weimann, An optimal decomposition algorithm for tree edit distance, ACM Tran. Algorithms 6 (2009) 1.

[7] J. Flum, M. Grohe, Parameterized Complexity Theory. Springer, 2006.

[8] K. Hirata, Y. Yamamoto, T. Kuboyama, Improved MAX SNP-hard results for finding an edit distance between unordered trees, in: Proc. 22nd Annual Symposium on Combinatorial Pattern Matching, in: LNCS, vol. 6661, Springer, 2011, pp. 402–415.

[9] P. Kilpeläinen, H. Mannila, Ordered and unordered tree inclusion. SIAM J. Comput. 24, 340–356 (1995)

[10] Y. Horesh, R. Mehr, R. Unger, Designing an A* algorithm for calculating edit distance between rooted-unordered trees, J. Comput. Biol. 6 (2006) 1165–1176.

[11] D. Milano, M. Scannapieco, T. Catarci, Structure-aware XML object identification, Data Eng. Bulletin 29 (2006) 67–74.

[12] N. Milo, S. Zakov, E. Katzenelson, E. Bachmat, Y. Dinitz, M. Ziv-Ukelson, Unrooted unordered homeomorphic subtree alignment of RNA trees, J. Alg. in Mol. Biol. 8 (2013) 13.

[13] T. Mori, J. Hwang, T. Tamura, A. Takasu, T. Akutsu, Fast similar subtree search using weighted tree inclusion, In preparation.

[14] T. Mori, T. Tamura, D. Fukagawa, A. Takasu, E. Tomita, T. Akutsu, A clique-based method using dynamic programming for computing edit distance between unordered trees, J. Comput. Biol. 19 (2012) 1089–1104.

[15] D. Shasha, J.T.-L. Wang, K. Zhang, F.Y. Shih, Exact and approximate algorithms for unordered tree matching, IEEE Trans. Syst., Man, and Cyber. 24 (1994) 668–678.

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

[17] G. Valiente, Algorithms on Trees and Graphs. Springer, 2002.

[18] K. Wang, Z. Ming, T.-S. Chua, A syntactic tree matching approach to finding similar questions in community-based QA services, in: Proc. Int. ACM SIGIR Conf. Research and Development in Information Retrieval, ACM Press, 2009, pp. 187–194.

[19] K.-C.Yu, E.L. Ritman, W.E. Higgins, System for the analysis and visualization of large 3D anatomical trees, Computers in Biology and Medicine 27 (2007) 1802–1830.

[20] K. Zhang, T. Jiang, Some MAX SNP-hard results concerning unordered labeled trees, Inform. Proc. Lett. 49 (1994) 249–254.

[21] K. Zhang, R. Statman, D. Shasha, On the editing distance between unordered labeled trees, Inform. Proc. Lett. 42 (1992) 133–139.

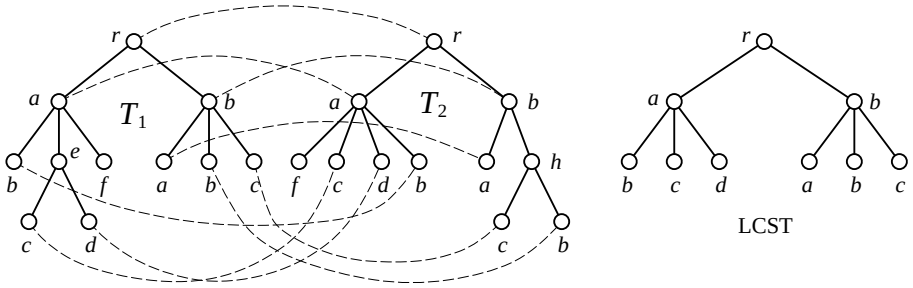


**Fig. 1** Example of an LCST under the weight function giving the maximum number of nodes (i.e., $f(u,v) = 1$ if $\ell(u) = \ell(v)$, and $f(u,v) = 0$ otherwise) and the outdegree constraint of $D = 3$. The corresponding mapping $M$ is shown by dashed curves. If $D = 4$, a node labeled $f$ can be added as a child of the left child of the root of LCST.

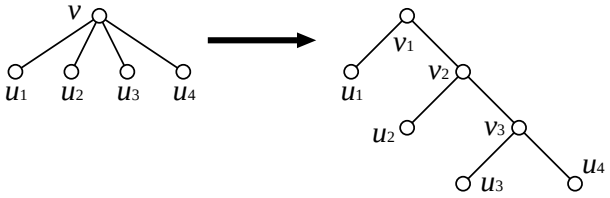

**Fig. 2** Transformation of a high outdegree node into nodes with outdegree 2.

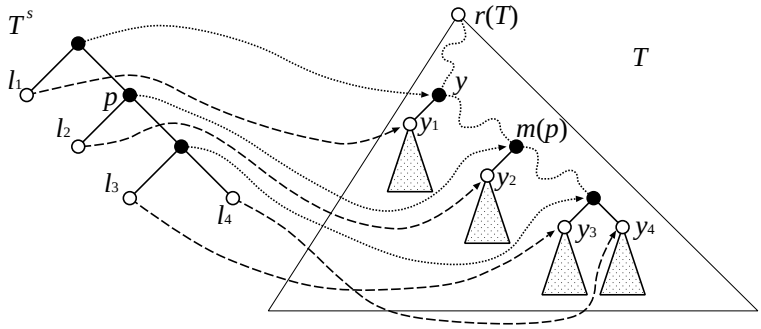


**Fig. 3** Example of a skelton tree $T^s$ and a mapping $m$. Dotted arrows represent $m(p)$, and dashed arrows represent the resulting mapping between leaves of $T^s$ and $\{y_1, \ldots, y_D\}$, where $D = 4$ in this example.

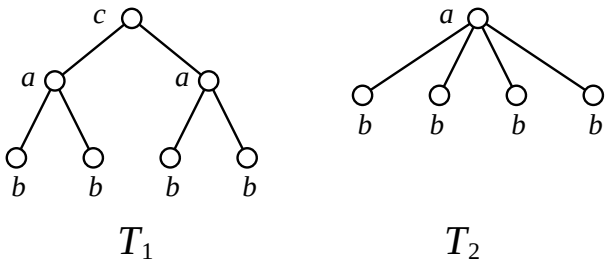


**Fig. 4** Example for LCST under different degree bounds.

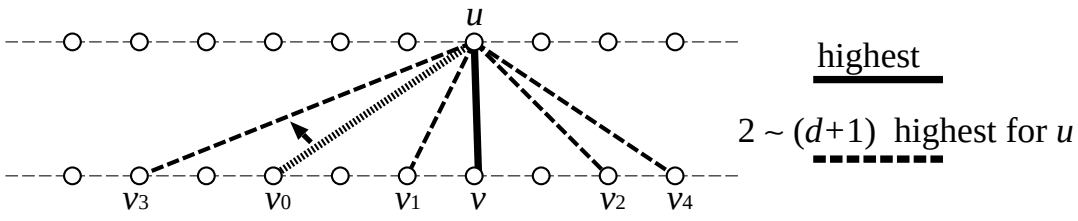


**Fig. 5** Illustration of the proof of Prop. 5.2, where $d = 4$ in this example.

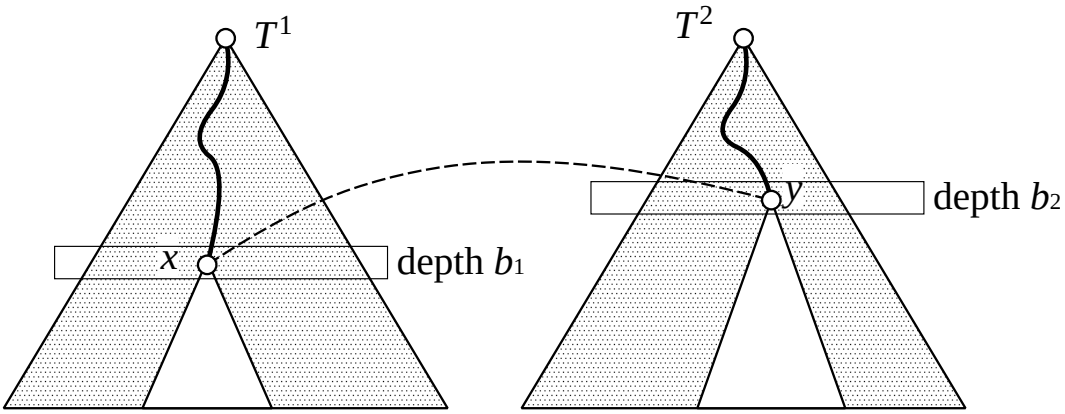


**Fig. 6** Illustration of $CandTuples(T^1, T^2, d)$. $R$ is chosen from pairs between depth $b_1$ nodes in $T^1$ and depth $b_2$ nodes in $T^2$, where all depth pairs are examined. For each $(x, y) \in R$, the remaining $d - 1$ pairs are searched between gray regions.

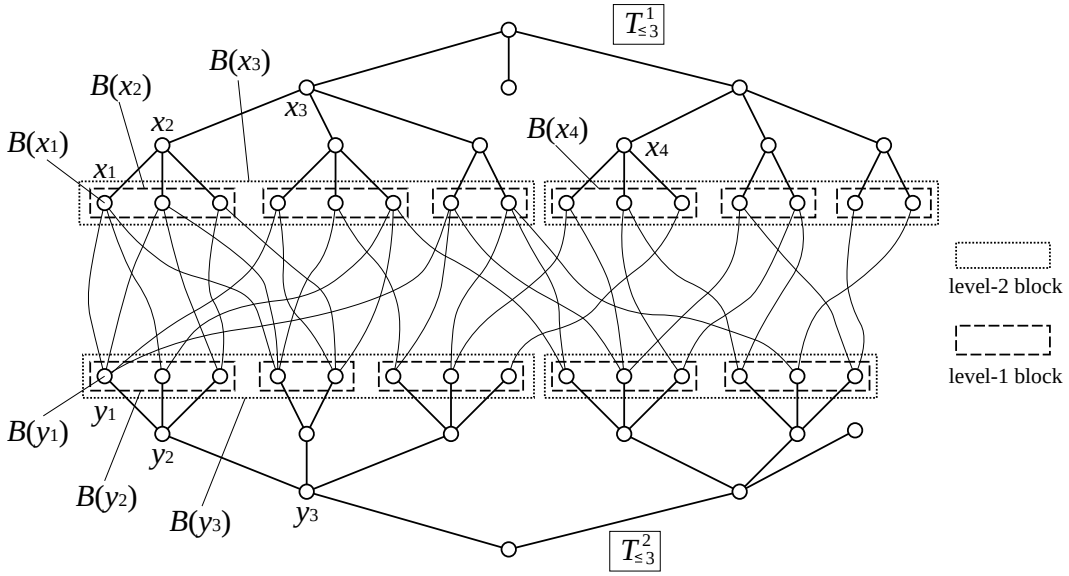


**Fig. 7** Illustration for degree constraints in $CandPairs(T^1_{\leq 3}, T^2_{\leq 3})$. In this example, $deg(B(x_1), B(y_2), P) = 2$, $deg(B(x_1), B(y_3), P) = 3$, $deg(B(x_2), B(y_1), P) = 2$, $deg(B(x_2), B(y_2), P) = 5$, $deg(B(x_2), B(y_3), P) = 8$, $deg(B(x_3), B(y_3), P) = 17$, $deg(B(x_4), B(y_2), P) = 0$, and $deg(B(x_4), B(y_3), P) = 2$, where curved edges denote those in the current $P$. Note that this figure does not illustrate the procedure itself.

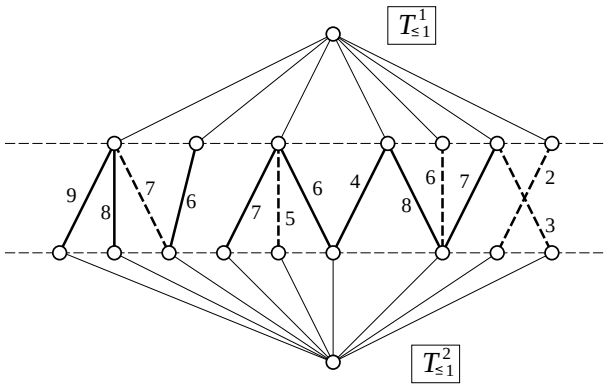


**Fig. 8** Example of $CandPairs(T^1_{\leq 1}, T^2_{\leq 1})$ for $d = 2$. The score is attached to each pair (i.e., each edge) where edges with score 0 are omitted. Bold edges are included in $P$ but dashed edges are not included in $P$. Since $2d^2 = 8$, 8 edges are selected.