On the Complexity of Finding a Largest Common Subtree of Bounded Degree

Tatsuya Akutsu[a,*], Takeyuki Tamura[a], Avraham A. Melkman[b], Atsuhiro Takasu[c]

[a] *Bioinformatics Center, Institute for Chemical Research, Kyoto University, Uji, Kyoto 611-0011, Japan.*
[b] *Ben-Gurion University of the Negev, Beer-Sheva, 84105, Israel.*
[c] *National Institute of Informatics, Tokyo 101-8430, Japan.*

Abstract

The largest common subtree problem 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. The problem is known to be NP-hard for unordered trees. In this paper, 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.

Keywords: Tree edit distance, Unordered trees, Dynamic programming, Parameterized complexity

1. Introduction

In computer science trees are one of the fundamental data structures, and extraction of a common structure between two or more given data sets is a fundamental problem. In order to find a common structure between two trees, extensive studies have been done on finding a *largest common*

*Corresponding author. Tel.: +81-774-38-3015 Fax.:+81-774-38-3022
Email addresses: takutsu@kuicr.kyoto-u.ac.jp (Tatsuya Akutsu), tamura@kuicr.kyoto-u.ac.jp (Takeyuki Tamura), melkman@cs.bgu.ac.il (Avraham A. Melkman), takasu@nii.ac.jp (Atsuhiro Takasu)

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 [23]. The LCST and related problems have various applications in bioinformatics including comparison of glycans [5], vascular networks [21], and cell lineage data [12]. They also have applications in comparison and search of XML data [13] and documents processed by natural language processing [20]. 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, 12, 13, 20, 21].

For the ordered case of the LCST and edit distance problems, $O(n^6)$ time algorithm was developed by Tai [18], where n is the maximum number of nodes in the input trees. After several improvements, Demaine et al. [8] developed an $O(n^3)$ time algorithm and showed that this bound is optimal under a certain model.

However, for unordered case of the LCST and edit distance problems, they are known to be NP-hard even for bounded degree input trees [23]. Moreover, several MAX SNP-hardness results are known for both problems [1, 10, 22]. In order to cope with these hardness results, approximation algorithms [1], fixed-parameter algorithms [1, 2, 17], efficient exponential time algorithms [4], and branch and bound algorithms [12, 16] 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.

Recently another approach was proposed by Akutsu et al. [2], in which the maximum outdegree (i.e., the maximum number of children) of common subtrees is fixed. 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 [21] and parse trees, and thus the maximum outdegree of common trees should also be bounded

[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.

by a small constant (otherwise, it would not be a common structure). For example, phylogenetic trees are usually binary trees and it is often required to find a binary agreement tree [6] although specialized algorithms have been developed for comparison of phylogenetic trees because labels of leaves play an important role.

In this paper, 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 descendants of a node x in the first input tree and the descendants of a node y in the second input tree, when nodes are mapped to the same node in an LCST. Whereas the previous $O(n^{2D})$ algorithm basically examined all D-tuples consisting of D pairs of descendants of x and y, the improved algorithm examines significantly fewer combinations by making use of tables to avoid redundant calculations and by making use of the property that the parent in an LCST is uniquely determined if at least two of its children are determined from each input tree.

Furthermore, we present a parameterized algorithm that runs 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. Since the LCST problem 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] It is to be noted that this algorithm is not comparable to other fixed-parameter algorithms [1, 2, 17] because our algorithm depends on the maximum height of input trees and a constraint on the maximum outdegree of an LCST, whereas the algorithm in [2] depends on the edit distance between two input trees and the algorithms in [1, 17] depend on the number of branching nodes. Furthermore, the ideas introduced in this paper are very different from those in [1, 2, 17]. In particular, the fixed-parameter algorithm in [2] is based on repeated elimination of identical subtrees and is useful only for comparison of similar input trees, whereas the fixed-parameter algorithm in this paper is based on examination of a limited number of node pairs and is (theoretically) useful only for comparison of low height trees.

[3]The algorithm was significantly simplified from the one in a preliminary version [3].

[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.

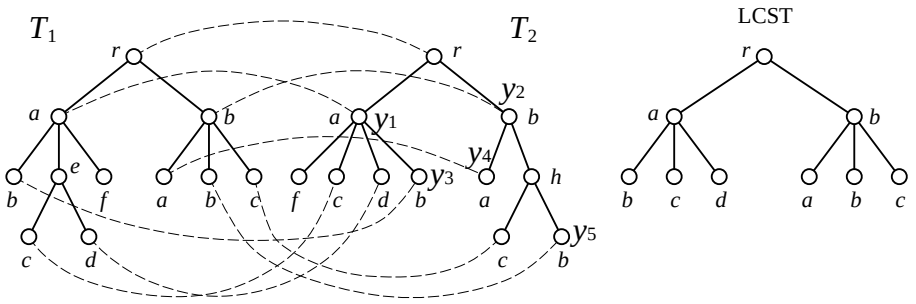

Figure 1: Example of an LCST with the weight function giving the maximum number of nodes (i.e., $f(u,v) = 1$ if $\ell(u) = \ell(v)$, and $f(u,v) = 0$ otherwise), and outdegree constraint of $D = 3$. The corresponding mapping M is shown by dashed curves. If $D = 4$, a node labeled f can be added as a child of the left child of the root of LCST. The nodes labeled y_1 through y_5, are used to illustrate consistent sets of descendants in Section 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 Σ, $des(v)$ 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. For a set S of edges or node pairs, if an edge $\{u, v\}$ or a pair (u, v) is a member of S, we say that u (resp., v) *appears in* S.

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 [23] 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 paper, 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 will use the device of imposing 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. Such a node will be called *inactive*. 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 $x \in V(T_1)$.

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.

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*, see Definition 4.2. It is straightforward to see that this algorithm works in $O(n^{2D+2})$ time.

This algorithm was improved by using 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, which can be made to run 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. □

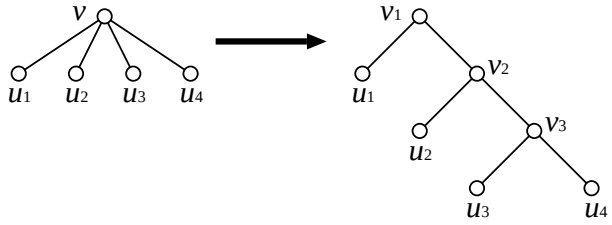

Figure 2: Transformation of a high outdegree node into nodes with outdegree 2.

Denote by T' the binary tree obtained from the general tree T using the modification described in Lemma 4.1. It is not difficult to see that a tree S with bounded outdegree D is a subtree of T if and only if S is a subtree of T'. Note in particular that here D may be greater than 2. Consequently there is no loss of generality in assuming that the maximum outdegree of input trees is 2, and we will do so in this section. Furthermore, we can assume without loss of generality (w.l.o.g.) 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 S of D descendants $\{y_1, \ldots, y_D\}$ of a node y is consistent if none of the descendants in S is an ancestor of another descendant in S, and y is the LCA of $\{y_1, \ldots, y_D\}$.*

For example, in Fig. 1, $\{y_1, y_2\}$, $\{y_1, y_4, y_5\}$, $\{y_2, y_3\}$ are consistent sets of descendants of $r(T_2)$, whereas $\{y_1, y_3\}$, $\{y_1, y_2, y_4\}$, $\{y_1, y_2, y_3\}$ are inconsistent sets of descendants of $r(T_2)$. In order to enumerate all possible sets of

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

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 ℓ_i of T^s, if ℓ_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.

Denote by $C_D(y)$ the set comprising all tuples of D consistent descendants of y obtained in this manner. 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 $\{y\}$.

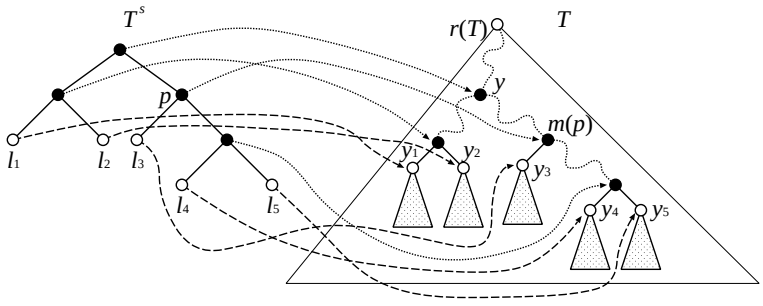

Figure 3: Example of a skeleton tree T^s and a mapping m. Dotted arrows represent $m(p)$, and dashed arrows represent the resulting mapping between leaves of T^s and $\{y_1, \ldots, y_D\}$, where $D = 5$ in this example.

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 is qualitatively simpler than the preliminary version given in [3], although it retains the same order of magnitude of running time, and the description of the algorithm was also greatly simplified. The algorithm computes, successively and bottom-up, the following generalization of $S(x, y)$.

Definition 4.4. *Let $k \geq 2$, and let Π_k be the set of all permutations π 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

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

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

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

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

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

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.*

Suppose a tree T is a LCST for outdegree $D = d$ as well as for outdegree $D = d + 1$. Is T then an LCST for any outdegree $D \geq d$? The answer is negative, as demonstrated by the example of the trees in Fig. 4. 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 it is 5 for $D = 4$.

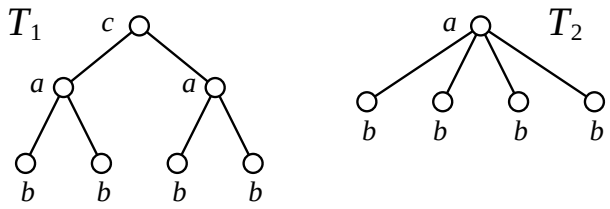

Figure 4: Example for LCST under different degree bounds.

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 [19], 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 [14] are phrased for unrooted trees they can be easily 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 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

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

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 [14] 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 $O(D d_x d_y)$.

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

$$O(\sum_{x \in T_1, y \in T_2} D d_x d_y) = O(D n_1 n_2).$$

□

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 Σ 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, Σ
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 [11]. 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 [15], 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 Σ 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, as shown in [7] and also shown algorithmically as part of the proof of Theorem 4.6, we present a fixed parameter version as an introduction to the fixed-parameter algorithm for LCST. Let M_{OPT} denote an optimal solution.

The following observations provide the basis for the algorithm.

Proposition 5.1. *Let $e = (u, v) \in E$ be an edge with highest weight. Then there is an optimal matching in which both u and v participate as endpoints of edges, possibly of the same edge $e = (u, v)$.*

Proof. Suppose there is an optimal matching M_{OPT} in which u does not appear. If v does not appear in M_{OPT} either, then let M' be a matching obtained by replacing an arbitrary edge in M_{OPT} with e. Otherwise, let $e' = (u', v)$ be an edge in M_{OPT}, and let M' be a matching obtained by replacing e' with e. In both cases the matching M' contains e and has weight no less than that of M_{OPT}.

The case that v does not appear in M_{OPT} is handled similarly. □

Proposition 5.2. *Let $e = (u, v_1) \in E$ be an edge with highest weight and let $(u, v_i), i = 1, \ldots, deg(u)$ be the edges out of u sorted according to weight (the order between equal-weight edges is immaterial). Then there is an optimal matching in which one of the edges (u, v_i) with $i \leq min(deg(u), d)$ participates.*

PROOF. In case $deg(u) \leq d$ the statement follows from Proposition 5.1.

Assuming $deg(u) > d$, let M_{OPT} be an optimal matching in which both u and v_1 appear, as ensured by Proposition 5.1. Suppose that the edge (u, v_i) participating in M_{OPT} has $i > d$. Since M_{OPT} consists of d edges and v_1 appears in it, there is at least one v_j with $1 < j \leq d$ that does not appear in M_{OPT}, see also Fig. 5. Upon replacing (u, v_i) by (u, v_j), a matching is obtained whose weight is no less than that of M_{OPT}, proving the proposition. □

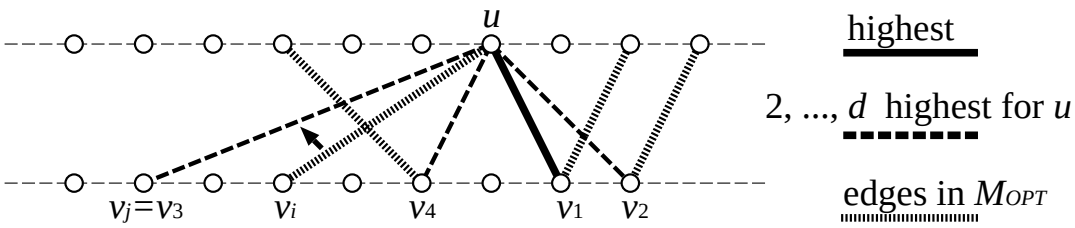

Figure 5: Illustration of the proof of Prop. 5.2, where $d = 4$ in this example.

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 the algorithm runs in $O(d^d poly(|E|))$ time.

Procedure $FptBipartite(M, E, d)$
  Let $e = (u, v_1)$ be an edge in E with the highest weight;
  if $d = 1$ **then return** $w(M \cup \{(u, v_1)\})$;
  sort the edges $(u, v_i), i = 1, \ldots, deg(u)$ according to weight;
  $d' \leftarrow \min(deg(u), d)$;
  return $\max_{1 \leq i \leq d'} \{FptBipartite(M \cup \{(u, v_i)\}, E - NE((u, v_i)), d - 1)\}$.

5.2. FPT-Algorithm

In the following, for a node u in T, let $Anc(u) = anc(u) \cup \{u\}$ and $Des(u) = des(u) \cup \{u\}$, and let $h(u)$ 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 similar to the one underlying the recursion equation (1) in Section 3. There, given a pair of nodes (x, y), all possible tuples of $d \leq D$ descendant pairs $(x_1, y_1), \ldots, (x_d, y_d)$ are examined. The resulting algorithm runs in $O(n^{2D+2})$ time because $O(n^{2D})$ tuples are examined for each pair (x, y). In contrast, the FPT algorithm examines, for given a pair of nodes (x, y), a number of tuples that does not depend on n.

In order to limit the number of tuples, we use the idea embodied in $FptBipartite(M, E, d)$. In that algorithm, in order to find a set of d edges (i.e., d pairs) with the maximum total weight, d edges are examined in each recursive call, each of which in turn invokes d recursive calls. Since the recursive depth is limited to d where the depth of the first recursive call is regarded as 1, the total number of recursive calls is $O(d^d)$. It is to be noted that all edges in an optimal solution are selected from distinct recursive depths. It should also be noted that the removal of the edges in $NE(\ldots)$ from further recursive calls ensures that those calls consider only edges that do not conflict with previously selected edges.

The FPT algorithm for LCST uses a similar stragegy of examining only sets of d ($d \leq D$) descendant pairs for each node pair (x, y). One difference from the bipartite matching case is that the descendants of the roots of input trees are layered (according to their depths). Since the ancestor-descendant relationship plays an important role in trees, the algorithm examines candidate pairs separately for all possible combinations of depths of the two input trees (see Fig. 6). Furthermore, if (x_1, y_1) is selected as a candidate descendant pair of (x, y), no descendant or ancestor of x_1 or y_1 can appear in the remaining set of descendant pairs. Therefore, the analogue of $NE(\cdots)$ includes also all pairs in which one of the vertices is an ancestor or a descendant. Furthermore, for the same reason the algorithm needs to examine a much larger number of candidate pairs (compared to d pairs) in each recursive call, where this number depends not only on D but also on the heights of input trees (but not on n).

The main routine of the FPT algorithm for LCST is $FptBdhLCST(T_1, T_2, D)$; it corresponds to the dynamic programming algorithm based on equation (1). The set of tuples of d descendant pairs that it examines, a much smaller set than all possible tuples of d pairs, is generated by $CandTuples(T^1, T^2, d)$. The latter routine uses recursive calls in a manner reminiscent of $FptBipartite(M, E, d)$. However, instead of examining d pairs at each recursive call, $CandTuples(T^1, T^2, d)$ examines a much larger number of pairs, generated by $CandPairs(T^1, T^2)$.

Since $CandPairs(T^1, T^2)$ is rather involved, we describe these three procedures in a top-down manner.

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.

Here $T \ominus u$ denotes the tree obtained by deleting $Anc(u) \cup des(u) - \{r(T)\}$.

Let us briefly explain the procedure (see also Fig. 6). Consider the case of $|V(T^1)| > 1$, $|V(T^2)| > 1$, and $d > 1$. 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. For each depth pair (b_1, b_2), R is analogous to the set $\{(u, v_1), (u, v_2), \ldots, (u, v_d)\}$ in $FptBipartite(M, E, d)$, although it contains many more pairs.

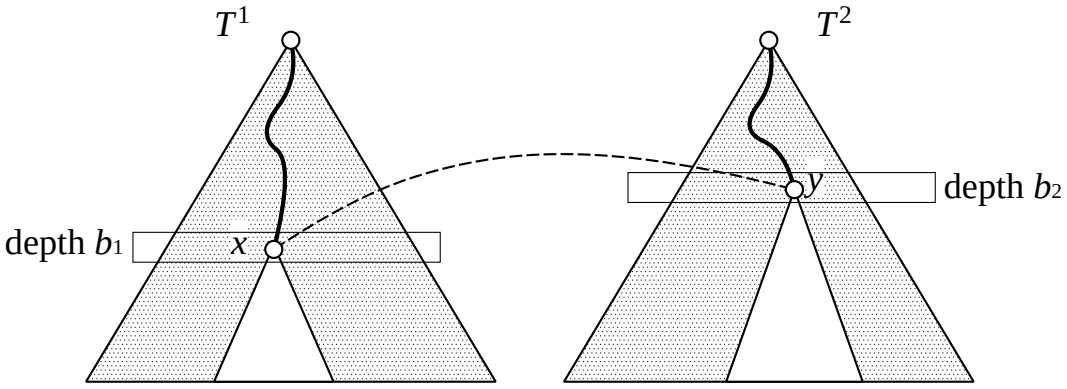

Figure 6: Illustration of $CandTuples(T^1, T^2, d)$. R is chosen from pairs between depth b_1 nodes in T^1 and depth b_2 nodes in T^2, where all depth pairs are examined. For each $(x, y) \in R$, the remaining $d-1$ pairs are searched between gray regions.

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 $i + j \geq 1$.

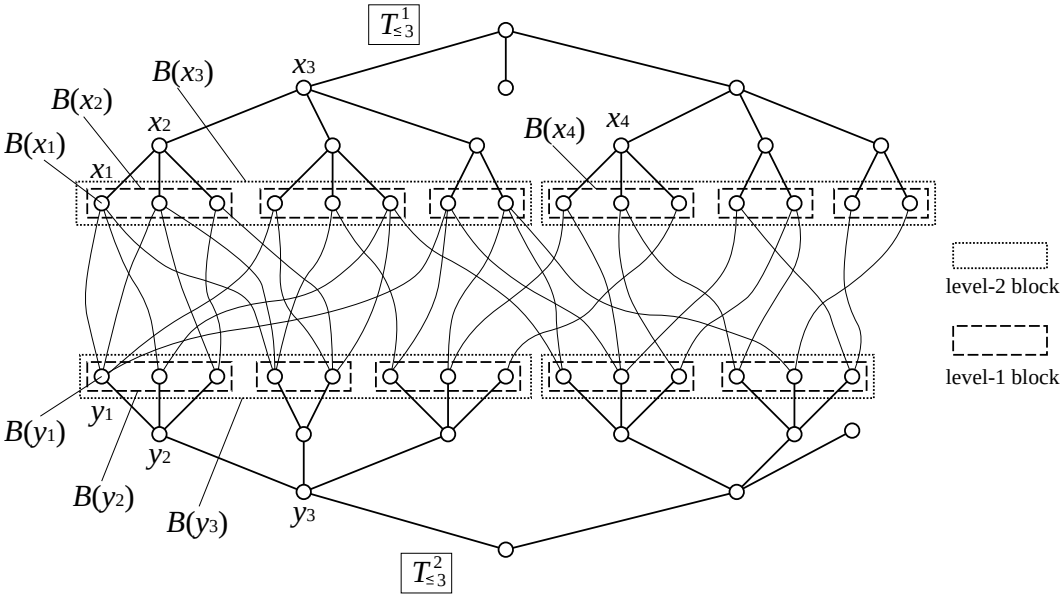

Figure 7: Illustration for degree constraints in $CandPairs(T^1_{\leq 3}, T^2_{\leq 3})$. In this example, $deg(B(x_1), B(y_2), P) = 2$, $deg(B(x_1), B(y_3), P) = 3$, $deg(B(x_2), B(y_1), P) = 2$, $deg(B(x_2), B(y_2), P) = 5$, $deg(B(x_2), B(y_3), P) = 8$, $deg(B(x_3), B(y_3), P) = 17$, $deg(B(x_4), B(y_2), P) = 0$, 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.

The procedure $CandPairs(T^1, T^2)$, whose pseudocode follows below, greedily adds edges 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.

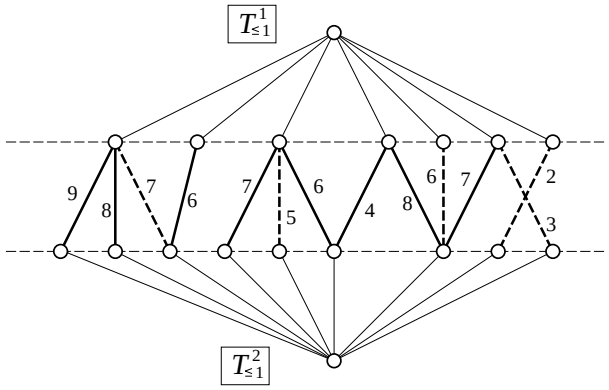

Figure 8: Example of $CandPairs(T^1_{\leq 1}, T^2_{\leq 1})$ for $d = 2$. The score is attached to each pair (i.e., each edge) where edges with score 0 are omitted. Bold edges are included in P but dashed edges are not included in P. Since $2d^2 = 8$, 8 edges are selected.

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. Because of the degree constraint, the number of pairs generated by $CandPairs(T^1_{\leq b_1}, T^2_{\leq b_2})$ is bounded by $2^{2H-1} \cdot D^{2H}$. Since $CandPairs(T^1_{\leq b_1}, T^2_{\leq b_2})$

is called at most H^2 times per each recursion level of $CandTuples(T^1, T^2, d)$, the total number of pairs generated at each recursion level ($d > 1$) is bounded by $H^2 \cdot 2^{2H-1} \cdot D^{2H}$.

Next, we analyze the size of the set of tuples given by $CandTuples(T^1, T^2, d)$. At the first call, we examine $H^2 \cdot 2^{2H-1} \cdot D^{2H}$ pairs for each of which recursive call is invoked. In the case of $d = 1$, only one pair is returned. Therefore, the total number of tuples is bounded by $(H^2 \cdot 2^{2H-1} \cdot D^{2H})^{D-1}$.

Finally, we analyze the main procedure. In this procedure, $CandTuples$ (T^1, T^2, d) is called $O(Dn^2)$ times. Therefore, the total number of tuples $((x_1, y_1), \ldots, (x_d, y_d))$ examined in this procedure is bounded by $O(D(H^2 \cdot 2^{2H-1} \cdot D^{2H})^{D-1} n^2)$. Since polynomial time is clearly enough per tuple in all procedures and $D \leq n$ trivially holds, the total time complexity is $O((H^2 \cdot 2^{2H-1} \cdot D^{2H})^{D-1} poly(n))$. □

Next, we present a key lemma showing that $CandPairs(T^1, T^2)$ does not miss a required pair. The basic idea of the proof is similar to that of Proposition 5.2 and is summarized as: (i) if we select sufficiently many high scoring pairs, one pair must be contained in some optimal solution, (ii) it is enough to examine at most d edges connecting to each node at the bottom level.[6] However, due to the ancestor-descendant relationship, we need to consider many more pairs. For example, consider height 1 tree T_1 and height 2 tree T_2 in Fig. 9. Suppose that x_0 appears in an optimal d-tuple for $(r(T_1), r(T_2))$, where $d = 2$ in this example. Suppose also that (x_0, y_1), (x_0, y_2), (x_0, y_3), and (x_0, y_4) are 4 highest-scoring pairs between x_0 and the bottom nodes in T_2, with $f(x_0, y_1) > f(x_0, y_2) > f(x_0, y_3) > f(x_0, y_4)$. In this case, examining $d = 2$ highest-scoring pairs connecting to x_0 is not enough because (x_0, y_1) or (x_0, y_2) may not be contained in an optimal 2-tuple if $f(x_1, y_5) > f(x_0, y_1)$ holds. Examining $d^2 = 4$ pairs, however, ensures that the desired pair will be found because an optimal tuple consists of 2 pairs. Of course, there is a case that (x_0, y_6) is contained in an optimal 2-tuple. However, such a case is treated separately because y_6 is not a bottom node. In the proof of the lemma, we generalize this discussion to an arbitrary pair of heights. In the following, M_{OPT} means an optimal d-tuple for the original trees (i.e., $T_1(x), T_2(y)$ in $FptBdhLCST(T_1, T_2, D)$), where a tuple can be regarded as a set of pairs of nodes.

[6] It does not mean that we need to examine d edges for every node.

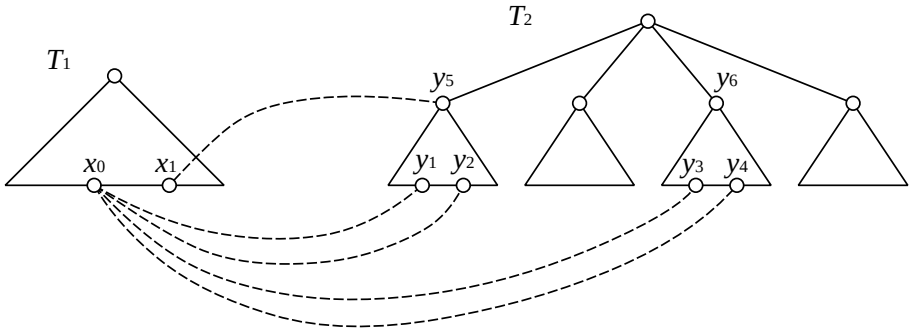

Figure 9: Illustration of the basic idea in the proof of Lemma 5.4. In this example, we need to consider d^2 pairs between x_0 in T_1 and the bottom nodes in T_2, where $d = 2$.

Lemma 5.4. *If there exists a pair $(x_0, y_0) \in B(r(T^1)) \times B(r(T^2))$ that appears in an optimal d-tuple, then the set P outputted by $CandPairs(T^1, T^2)$ contains at least one pair (x, y) that appears in some optimal d-tuple.*

PROOF. Let $x^0 = r(T^1)$ and $y^0 = r(T^2)$. We prove the lemma by contradiction. Suppose that $(x_0, y_0) \in M_{OPT}$ appears in $B(r(T^1)) \times B(r(T^2))$, but P does not contain any pair in any optimal d-tuple. It is to be noted that M_{OPT} can contain a pair including a node outside T^1 and T^2 (i.e., M_{OPT} can contain a pair including a node in $T_1(x)$ and $T_2(y)$). We assume in the following that $d \geq 2$ because only then is $CandPairs(T^1, T^2)$ invoked.

We begin with the case of $h(T^1) = h(T^2) = 1$, then prove the lemma for the case of $h(T^2) = h(T^2) = 2$, and finally extend the proof to the general case.

[Case of Height 1 Trees]

First we note that P can not contain any pair in M_{OPT} because of the assumption.

Recall that $deg(B(x), B(y))$ denotes the number of edges between $B(x)$ and $B(y)$ in P. The constraint $deg(B(x'), B(y')) \leq g(h(x'), h(y'))d^{h(x')+h(y')}$, limits the cases to be considered to the following:

(a) $deg(B(x_0), B(y^0)) = d$,

(a') $deg(B(x^0), B(y_0)) = d$,

(b) $deg(B(x_0), B(y^0)) < d$, $deg(B(x_0), B(y^0)) < d$, and $deg(B(x^0), B(y^0)) = 2d^2$,

(c) $deg(B(x_0), B(y^0)) < d$, $deg(B(x_0), B(y^0)) < d$, and $deg(B(x^0), B(y^0)) < 2d^2$,

because $x_0 \in B(x')$ (resp., $y_0 \in B(y')$) holds if and only if $x' = x_0$ or $x' = x^0$ (resp., $y' = y_0$ or $y' = y^0$). Fig. 10 illustrates cases (a), (a'), and (b).

In the following we assume that at each step none of the preceding conditions is satisfied.

(a) $deg(B(x_0), B(y^0)) = d$ holds.
The score could be increased (resp., kept) by replacing (x_0, y_0) with $(x_0, y_k) \in P$ such that y_k does not appear in a pair in M_{OPT}, which contradicts the assumption that M_{OPT} is an optimal d-tuple (resp., P does not contain any pair in any optimal d-tuple). Such a pair must exist because $|M_{OPT}| \leq d$ and $(x_0, y_0) \in M_{OPT}$.

(a') is handled similarly to (a).

(b) $|P| = g(1,1)d^2 = 2d^2$ (i.e., $deg(B(x^0), B(y^0)) = 2d^2$).
We bound the number of edges of P connecting to either endpoint of any edge in M_{OPT}. Since case (a) was not applicable we can assume that $deg(B(x), B(y^0)) < d$ and $deg(B(x^0), B(y)) < d$ hold for any pair $(x, y) \in M_{OPT}$ such that $x \in B(x^0)$ and $y \in B(y^0)$. Then, the required bound is given by

$$d \cdot \max(2(d-1), d) \ < \ 2d^2,$$

which means that there exists at least one edge $(x_h, y_k) \in P$ such that neither endpoint appears in M_{OPT}. Here we note that $S^0(x_0, y_0) \leq S^0(x_h, y_k)$ holds because otherwise (x_0, y_0) would have been added to P before (x_h, y_k) under the assumption that condition (a) does not hold. The score could be increased (resp., kept) by replacing (x_0, y_0) with $(x_h, y_k) \in P$, which contradicts the assumption that M_{OPT} is an optimal d-tuple.

(c) The remaining case.
From the assumption, $|P| < 2d^2$. Therefore, we could have added (x_0, y_0) or another pair with higher or the same score to P, which contradicts the assumption.

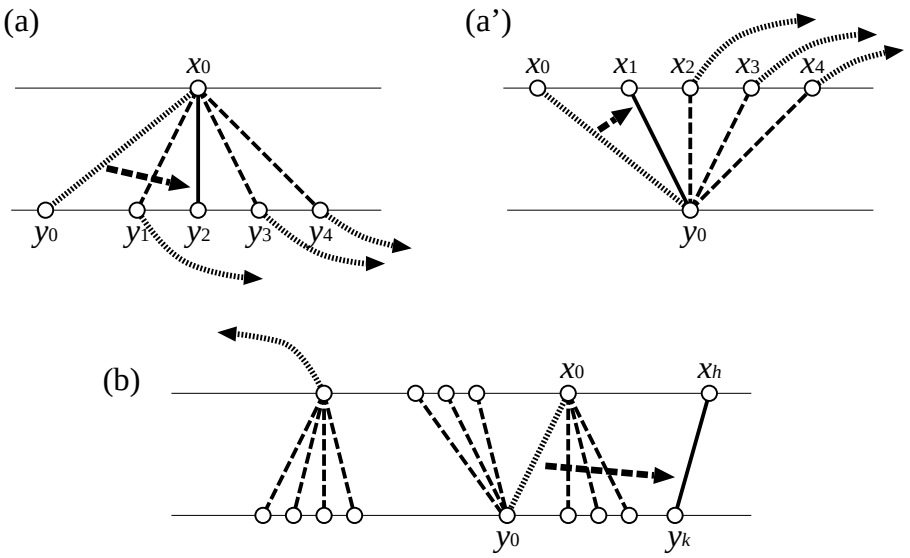

Figure 10: Cases (a), (a'), and (b) in the proof of Lemma 5.4 for height 1 trees, where $d = 4$. Dotted arrows/lines denote pairs in M_{OPT}, where dotted arrows mean connections to nodes outside T^1 and T^2. Dashed/bold lines denote pairs in P.

[**Case of Height 2 Trees**]

We say that block $B(u)$ is *inactivated* if u appears in M_{OPT}. Each node in block $B(u)$ is also *inactivated* in such a case. 'Inactivate' means that the node (or, any node in the block) cannot appear in M_{OPT} except u (see Fig. 11). We can assume that d pairs with $B(x^0) \times B(y^0)$ appear in M_{OPT}. If a descendant node of u appears in M_{OPT}, u is also *inactivated*. If u is not inactivated, u is called *active*. A pair (u, v) is also called inactivated if either u or v is inactivated.

Here we focus on level-2 blocks (i.e., height 2 trees). Let M_{OPT} be an optimal d-tuple. Let $B(x^0)$ and $B(y^0)$ be level-2 blocks. Let $(x_0, y_0) \in B(x^0) \times B(y^0)$ be a pair in M_{OPT} that is not in P (see Fig. 12).

From the constraint of $deg(B(x'), B(y')) \leq g(h(x'), h(y'))d^{h(x')+h(y')}$ and a fact that $x_0 \in B(x')$ and $y_0 \in B(y')$ hold if and only if

$$\begin{aligned}(x', y') \quad \in \quad & \{(x_0, p(y_0)), (p(x_0), y_0), (x_0, p^2(y_0)), (p^2(x_0), y_0), (p(x_0), p(y_0)), \\ & (p(x_0), p^2(y_0)), (p^2(x_0), p(y_0)), (p^2(x_0), p^2(y_0))\}\end{aligned}$$

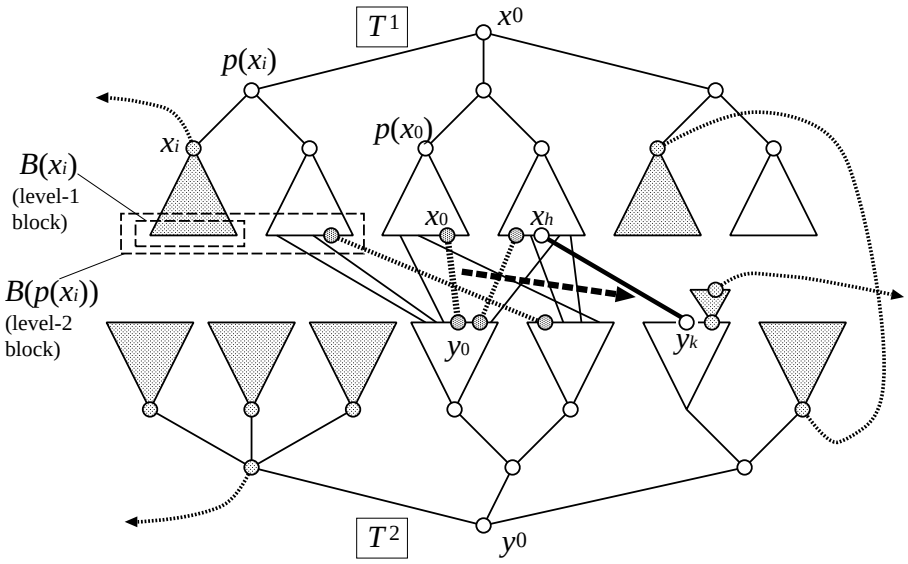

Figure 11: Gray regions and nodes represent inactivated blocks and nodes, where dotted arrows/lines denote pairs in M_{OPT}.

holds,[7] we only need to consider the following cases where we omit the symmetric cases (i.e., $(p(x_0), y_0)$, $(p^2(x_0), y_0)$, and $(p^2(x_0), p(y_0))$). It is to be noted if $x' = x_0$, we need not consider the cases that both x_h and y_k are active because $B(x_0) = \{x_0\}$ and x_0 is inactive by the assumption.

(a) $deg(B(x_0), B(p(y_0))) = d$.
The score could be increased or kept by replacing (x_0, y_0) with some other $(x_0, y_k) \in P$ such that y_k is active.

(b) $deg(B(x_0), B(p^2(y_0)) = g(0,2)d^2 = d^2$.
Since $deg(B(x_0), B(p(y_i))) \leq d$ holds for any $y_i \in B(y^0)$ and $deg(B(x_0), B(p(y_0))) < d$ holds, there exists an edge $(x_0, y_k) \in P$ such that y_k is active (note that both $B(p(x_0))$ and $B(p(y_0))$ are active except for the nodes in M_{OPT}, and at most $d-1$ other level-1 blocks ($B(p(y_k))$s) are inactivated by M_{OPT}). The score could be increased or kept by replacing (x_0, y_0) with (x_0, y_k).

[7] $p^2(x)$ denotes $p(p(x))$. We need not consider the case of $(x', y') = (x_0, y_0)$ because of the assumption.

(c) $deg(B(p(x_0)), B(p(y_0))) = g(1,1)d^2 = 2d^2$.
This case corresponds to case (b) for height 1 trees.

(d) There exists a pair $(x_h, y_k) \in B(p(x_0)) \times B(p(y_0))$ appearing in P such that both x_h and y_k are active.
The score could be increased or kept by replacing (x_0, y_0) with (x_h, y_k) because condition (a) nor (b) does not hold.

(e) $deg(B(p(x_0)), B(p^2(y_0))) = g(1,2)d^3$.
Note that $d-1$ pairs in M_{OPT} (excluding (x_0, y_0)) inactivate at most $(d-1)(g(0,2)d^2 + g(1,1)d^2)$ pairs. On the other hand, (x_0, y_0) inactivates at most $g(0,2)d^2 + g(1,0)d$ pairs. From

$$(d-1)(g(0,2)d^2 + g(1,1)d^2) \ + \ (g(0,2)d^2 + g(1,0)d) \ < \ g(1,2)d^3,$$

we can see that there exists a pair $(x_h, y_k) \in P$ such that x_h is active, $x_h \in B(p(x_0))$, y_k is active (accordingly $B(p(y_k))$ is active), and $y_k \in B(p^2(y_0)) - B(p(y_0))$. Since $deg(B(p(x_0)), B(p(y_0))) < 2d^2$, $S^0(x_0, y_0) \leq S^0(x_h, y_k)$ holds and thus the score could be increased or kept by replacing (x_0, y_0) with (x_h, y_k) (otherwise, (x_0, y_0) would have been added to P in place of (x_h, y_k)).

(f) There exists a pair $(x_h, y_k) \in B(p(x_0)) \times B(p^2(y_0))$ in P such that both x_h and y_k are active.
The score could be increased or kept by replacing (x_0, y_0) with (x_h, y_k).

(g) $deg(B(p^2(x_0)), B(p^2(y_0))) = g(2,2)d^4$.
Note that $d-1$ pairs in M_{OPT} (excluding (x_0, y_0)) inactivate at most $(d-1)(g(1,2)+g(2,1))d^3$ pairs because one endpoint of a pair in M_{OPT} inactivates at most $g(1,2)d^3 = g(2,1)d^3$ pairs. For example, (x_0, y_0) inactivates at most $g(0,2)d^2 + g(2,0)d^2$ pairs. From

$$2(d-1)g(1,2)d^3 \ + \ 2g(0,2)d^2 \ < \ g(2,2)d^4$$

we can see that there exists a pair $(x_h, y_k) \in P$ such that both x_h and y_k are active. Since $deg(B(p(x_0)), B(p^2(y_0))) < 4d^3$, the score could be increased or kept by replacing (x_0, y_0) with (x_h, y_k) (otherwise, (x_0, y_0) would have been added to P in place of (x_h, y_k)).

(h) $deg(B(p^2(x_0)), B(p^2(y_0))) < g(2,2)d^4$.
From the assumption, $|P| < g(2,2)d^4$. Therefore, we could have added (x_0, y_0) or another pair with higher or the same score to P.

[**General Case**]

Finally, we generalize the proof for arbitrary heights. Let $I = h(T^1)$ and $J = h(T^2)$. We show by means of the following procedure that a contradiction always occurs (see Fig. 13). For simplicity, we assume w.l.o.g. that $I \leq J$ and omit symmetric cases.

(1) Let $i = 0$. For $j = 1$ to J, repeat the following: if $deg(B(x_0), B(p^j(y_0))) = d^j$ holds, the score could be increased or kept by replacing (x_0, y_0) with some other pair $(x_0, y_k) \in P$ such that $y_k \in B(p^j(y_0)) - B(p^{j-1}(y_0))$ is active.

(2) For $i = 1$ to I, repeat step (3).

(3) For $j = i$ to J, repeat steps (4) and (5).

(4) If $deg(B(p^i(x_0)), B(p^j(y_0))) = g(i,j)d^{i+j}$, the score could be increased or kept by replacing (x_0, y_0) with some other pair $(x_h, y_k) \in P$ such that both $x_h \in B(p^i(x_0)) - B(p^{i-1}(x_0))$ and $y_k \in B(p^j(y_0)) - B(p^{j-1}(y_0))$ are active because

$$(d-1)(g(i-1,j)+g(i,j-1))d^{i+j-1} + (g(0,j)d^j + g(i,0)d^i) \ < \ g(i,j)d^{i+j}$$

holds.

(5) If $deg(B(p^i(x_0)), B(p^j(y_0))) < g(i,j)d^{i+j}$ and there exists a pair $(x_h, y_k) \in P$ such that both $x_h \in B(p^i(x_0)) - B(p^{i-1}(x_0))$ and $y_k \in B(p^j(y_0)) - B(p^{j-1}(y_0))$ are active, the score could be increased or kept by replacing (x_0, y_0) with (x_h, y_k).

(6) $deg(B(p^I(x_0)), B(p^J(y_0))) < g(I,J)d^{I+J}$ must hold. Therefore, we could have added (x_0, y_0) or another pair with higher or the same score to P.

The correctness of this procedure (i.e., it always finds a contradiction) can be shown by repeatedly applying the discussions in the proof for the height 2 trees (i.e., the level-2 case). □

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 $d \geq 2$. 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 26240034 and Grant-in-Aid 25730005, 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] R. Cole, M. Farach-Colton, R. Hariharan, T. M. Przytycka, M. Thorup, An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees, SIAM J. Comput. 30 (2000) 1385–1404.

[7] M. Dell'Amico and S. Martello, The k-cardinality assignment problem, Discrete Appl. Math. 76 (1997) 103-121.

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

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

[10] 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.

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

[12] 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.

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

[14] 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.

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

[16] 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.

[17] 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.

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

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

[20] 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.

[21] 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.

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

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

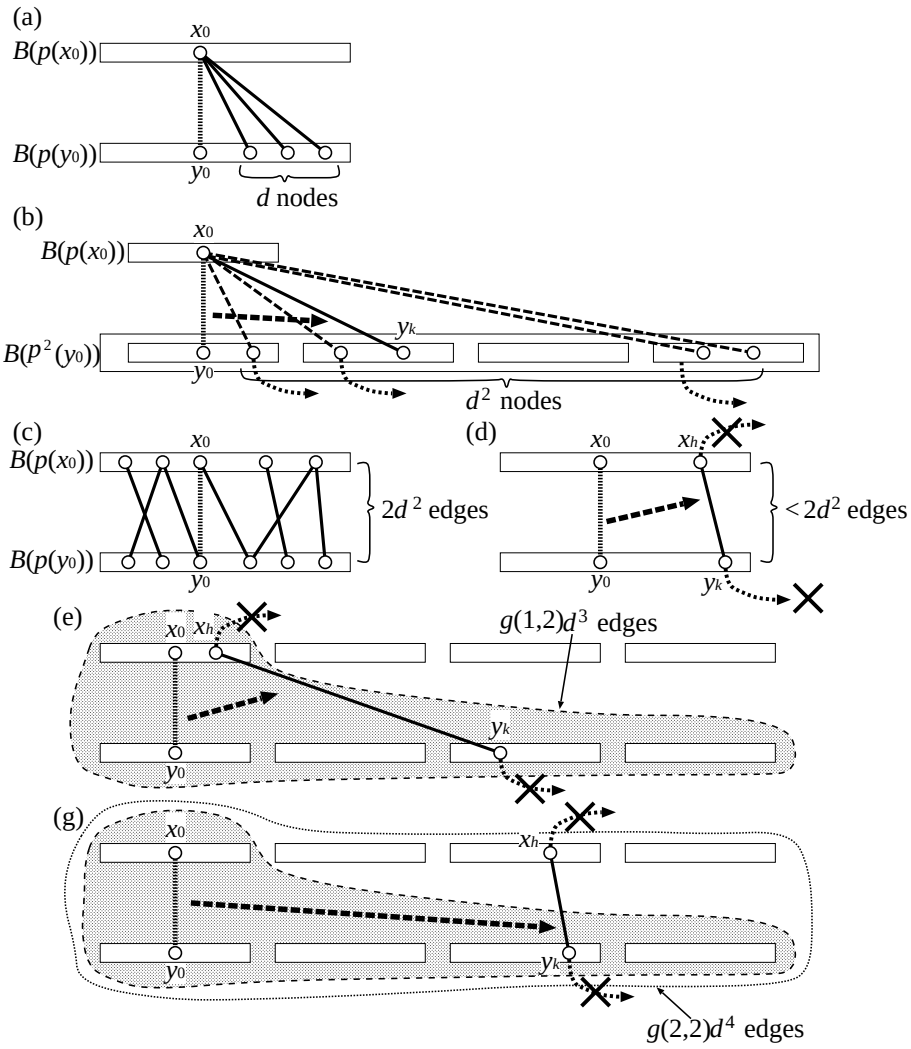

Figure 12: Illustration of cases in the proof of the correctness for level-2 blocks, where cases (f) and (h) are omitted here.

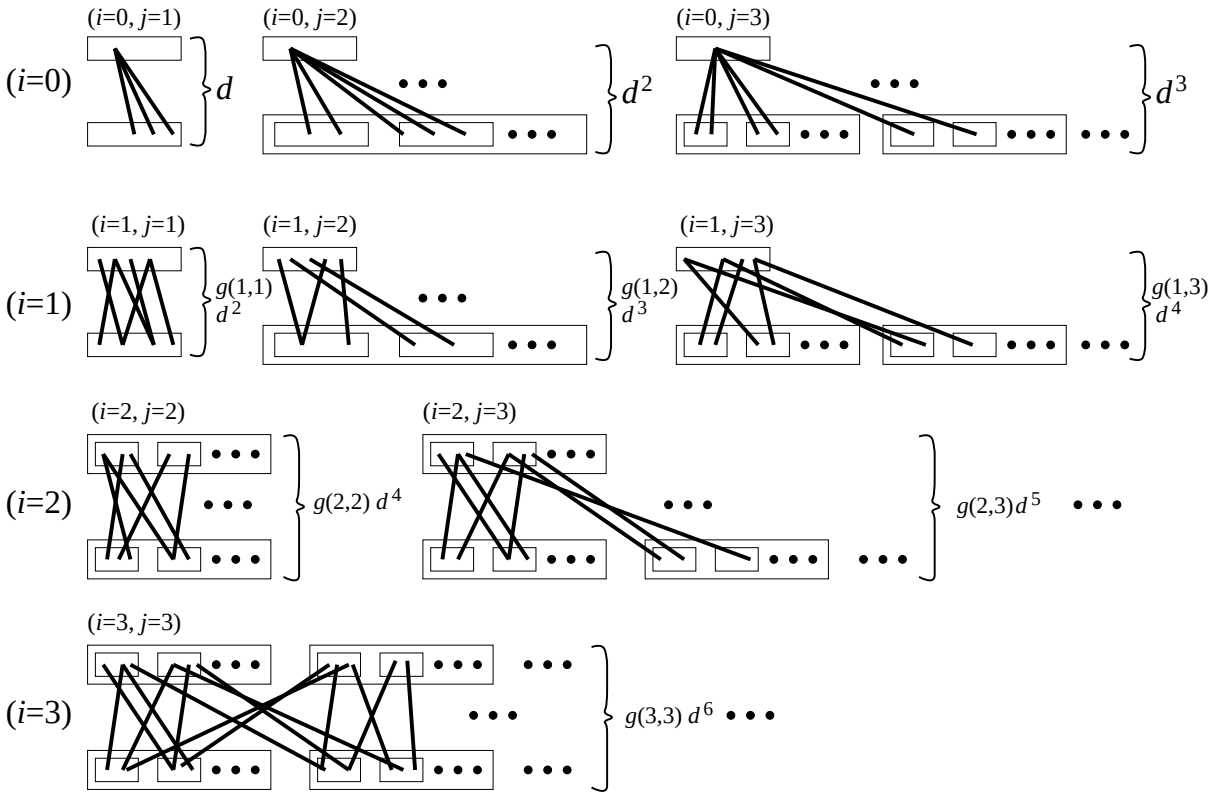

Figure 13: Illustration of cases in the proof of the correctness for the general case.