New and Improved Algorithms for Unordered Tree Inclusion

Tatsuya Akutsu[1]

Bioinformatics Center, Institute for Chemical Research, Kyoto University
Kyoto 611-0011, Japan

Jesper Jansson

Department of Computing, The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong, China

Ruiming Li

Bioinformatics Center, Institute for Chemical Research, Kyoto University
Kyoto 611-0011, Japan

Atsuhiro Takasu

National Institute of Informatics
Chiyoda-ku, Tokyo, 101-8430, Japan

Takeyuki Tamura[2]

Bioinformatics Center, Institute for Chemical Research, Kyoto University
Kyoto 611-0011, Japan

Abstract

The *tree inclusion problem* is, given two node-labeled trees P and T (the "pattern tree" and the "text tree"), to locate every minimal subtree in T (if any) that can be obtained by applying a sequence of node insertion operations to P. Although the *ordered* tree inclusion problem is solvable in polynomial time, the *unordered* tree inclusion problem is NP-hard. The currently fastest algorithm for the latter is from 1995 and runs in $O(poly(m, n) \cdot 2^{2d}) = O^*(2^{2d})$ time, where m and n are the sizes of the pattern and text trees, respectively, and d is the maximum outdegree of the pattern tree. Here, we develop a new algorithm that improves the exponent $2d$ to d by considering a particular type of ancestor-descendant relationships and applying dynamic programming, thus reducing the time complexity to $O^*(2^d)$. We then study restricted variants of the unordered tree inclusion problem where the number of occurrences of different node labels and/or the input trees' heights are bounded. We show that although the problem remains NP-hard in many such cases, it can be solved in polynomial time for $c = 2$ and in $O^*(1.8^d)$ time for $c = 3$ if the leaves of P are distinctly labeled and each label occurs at most c times in T. We also present a randomized $O^*(1.883^d)$-time algorithm for the case that the heights of P and T are one and two, respectively.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases parameterized algorithms, tree inclusion, unordered trees, dynamic programming

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2018.27

[1] JSPS KAKENHI #18H04113
[2] JSPS KAKENHI #25730005

29th International Symposium on Algorithms and Computation (ISAAC 2018).
Editors: Wen-Lian Hsu, Der-Tsai Lee, and Chung-Shou Liao; Article No. 27; pp. 27:1–27:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Tree pattern matching and measuring the similarity of trees are classic problem areas in theoretical computer science. One intuitive and extensively studied measure of the similarity between two rooted, node-labeled trees T_1 and T_2 is the *tree edit distance*, defined as the length of a shortest sequence of node insertion, node deletion, and node relabeling operations that transforms T_1 into T_2 [7]. When T_1 and T_2 are *ordered* trees, the tree edit distance can be computed in polynomial time. The first algorithm to achieve this bound ran in $O(n^6)$ time [20], where n is the total number of nodes in T_1 and T_2, and it was gradually improved upon until Demaine et al. [12] presented an $O(n^3)$-time algorithm thirty years later which was proved to be worst-case optimal under a conjecture that there is no truly subcubic time algorithm for the all pairs shortest paths problem [9]. On the other hand, the tree edit distance problem is NP-hard for *unordered* trees [25]. It is MAX SNP-hard even for binary trees in the unordered case [24], which implies that it is unlikely to admit a polynomial-time approximation scheme. Akutsu et al. [3, 5] have developed efficient exponential-time algorithms for this problem variant. As for parameterized algorithms, Shasha et al. [19] developed an $O(4^{\ell_1+\ell_2} \min(\ell_1, \ell_2)mn)$-time algorithm for the problem, where ℓ_1 and ℓ_2 are the numbers of leaves in T_1 and T_2, respectively. Using another parameter k, an $O^*(2.62^k)$-time algorithm was developed for the unit-cost edit operation model [4], where k is the edit distance and $O^*(f(\cdots))$ means $O(f(\cdots)poly(m, n))$. See [7] for other related results.

An important special case of the tree edit distance problem known as the *tree inclusion problem* is obtained when only node insertion operations are allowed. This problem has applications to structured text databases and natural language processing [8, 14, 21]. Here, we assume the following formulation of the problem: given a “text tree” T and a “pattern tree” P, locate every minimal subtree in T (if any) that can be obtained by applying a sequence of node insertion operations to P. (Equivalently, one may define the tree inclusion problem so that only node deletion operations on T are allowed.) For unordered trees, Kilpeläinen and Mannila [14] proved the problem to be NP-hard in general but solvable in polynomial time when the degree (outdegree) of the pattern tree is bounded from above by a constant. More precisely, the running time of their algorithm is $O(d \cdot 2^{2d} \cdot mn)$ time, where $m = |P|$, $n = |T|$, and d is the maximum degree of P. Bille and Gørtz [8] gave a fast algorithm for the case of ordered trees, and Valiente [21] developed a polynomial-time algorithm for a constrained version of the unordered case. Also note that the special case of the tree inclusion problem where node insertion operations are only allowed to insert new leaves corresponds to a subtree isomorphism problem, which can be solved in polynomial time for unordered trees [17].

1.1 Practical applications

Due to the rapid advance of AI technology, matching methods for knowledge base become more important. As a fundamental technique for searching knowledge base, researchers in database community have been studying the subtree similarity search. For example, Cohen and Or proposed a subtree similarity search algorithm for various distance functions [11], while Chang et al. proposed a top-k tree matching algorithm [10]. In the Natural Language Processing (NLP) field, researchers are incorporating the deep learning techniques into NLP problems and developing parsing/dependency trees processing algorithms [16]. Bibliographic matching is one of the most popular applications of real-world matching problems [15]. In most cases, single article has at most two or three versions, and it is very rare that single article includes the same name co-authors. Therefore, it may be reasonable to assume that the leaves of P are distinctly labeled and each label occurs at most c times in T.

Table 1 The computational complexity of some special cases of the unordered tree inclusion problem, where the last one is a randomized one. For any tree T, $h(T)$ denotes the height of T and $OCC(T)$ the maximum number of times that any leaf label occurs in T. As indicated in the table, either all nodes or only the leaves are labeled (the former is harder since it generalizes the latter).

Restriction	Labels on	Complexity	Reference
$h(T) = 2$, $h(P) = 1$, $OCC(T) = 3$, $OCC(P) = 1$	all nodes	NP-hard	Corollary 8
$h(T) = 2$, $h(P) = 2$, $OCC(T) = 3$, $OCC(P) = 1$	leaves	NP-hard	Theorem 9
$OCC(T) = 2$, $OCC(P) = 1$	all nodes	P	Theorem 11
$OCC(T) = 3$, $OCC(P) = 1$	all nodes	$O^*(1.8^d)$ time	Theorem 12
$h(T) = 2$, $h(P) = 1$	all nodes	$O^*(1.883^d)$ time	Theorem 14

The *extended tree inclusion problem* was proposed in [18], which is an optimization problem designed to make the unordered tree inclusion problem more useful for practical tree pattern matching applications, e.g., involving glycan data from the KEGG database [13], weblogs data [23], and bibliographical data from ACM, DBLP, and Google Scholar [15]. This problem asks for an optimal connected subgraph of T (if any) that can be obtained by performing node insertion operations as well as node relabeling operations to P while allowing non-uniform costs to be assigned to the different node operations; it was shown in [18] that the unrooted version can be solved in $O^*(2^{2d})$ time and a further extension of the problem that also allows at most k node deletion operations can be solved in $O^*((ed)^k k^{1/2} 2^{2(dk+d-k)})$ time where e is the base of the natural logarithm.

1.2 New results

We improve the exponential contribution to the time complexity of the fastest known algorithm for the unordered tree inclusion problem (Kilpeläinen and Mannila's algorithm from 1995 [14]) from 2^{2d} to 2^d, where d is the maximum degree of the pattern tree, so that the time complexity becomes $O(d2^d mn^2) = O^*(2^d)$. This improved bound is achieved by introducing a simple but quite useful idea of *minimal inclusion* and a different way of dynamic programming. Next, we study the problem's computational complexity for several restricted cases (see Table 1 for a summary) and give a polynomial-time algorithm for when the leaves in P are distinctly labeled and every label appears at most twice in T. Then, we derive an $O^*(1.8^d)$-time algorithm for the NP-hard case where the leaves in P are distinctly labeled and each label appears at most three times in T. Both are obtained by effectively utilizing a polynomial-time algorithm for 2-SAT. Finally, we derive a randomized $O^*(1.883^d)$ time algorithm for the case where the heights of P and T are one and two, respectively. It is obtained by a simple but non-trivial combination of the $O^*(2^d)$ time algorithm, an $O^*(1.234^m)$ time algorithm for SAT with m clauses [22], and color-coding [6]. Because of the page limit, some proofs are omitted in this version.

2 Preliminaries

From here on, all trees are rooted, unordered, and node-labeled. Let T be a tree. A *node insertion operation* on T is an operation that creates a new node v having any label and then: (i) attaches v as a child of some node u currently in T and makes v become the parent of a (possibly empty) subset of the children of u; or (ii) makes the current root of T become a child of v and lets v become the new root. For any two trees T_1 and T_2, we say that T_1 *is included in* T_2 if there exists a sequence of node insertion operations such that applying the sequence to T_1 yields T_2 (i.e., T_1 is obtained by node deletions from T_2).

For a tree T, $r(T)$, $h(T)$, and $V(T)$ denote its root, height, and the set of nodes in T, respectively. A *mapping* between two trees T_1 and T_2 is a subset $M \subseteq V(T_1) \times V(T_2)$ such that for every $(u_1, v_1), (u_2, v_2) \in M$, it holds that: (i) $u_1 = u_2$ if and only if $v_1 = v_2$; and (ii) u_1 is an ancestor of u_2 if and only if v_1 is an ancestor of v_2. T_1 is included in T_2 if and only if there is a mapping M between T_1 and T_2 such that $|M| = |V(T_1)|$ and u and v have the same node label for every $(u, v) \in M$ [20]. Such a mapping is called an *inclusion mapping*.

In the *tree inclusion problem*, the input consists of two trees P and T (also referred to as the "pattern tree" and the "text tree"), and the objective is to locate every minimal subtree of T that includes P. Define $m = |V(P)|$ and $n = |V(T)|$, and d denote the maximum degree of P. For any node v, let $\ell(v)$ and $Chd(v)$ denote its label and the set of its children. Also let $Anc(v)$ and $Des(v)$ denote the sets of strict ancestors and strict descendants of v, respectively, i.e., where v itself is excluded from these sets. For a node v in a tree T, $T(v)$ is the subtree of T induced by $Des(v) \cup \{v\}$. We write $P(u) \subset T(v)$ if $P(u)$ is included in $T(v)$ under the condition that u is mapped to v. For two trees T_1 and T_2, $T_1 \sim T_2$ denotes that T_1 is isomorphic to T_2 (with label information). The following concept plays a key role in our algorithm.

▶ **Definition 1.** We say that $T(v)$ minimally includes $P(u)$ (denoted as $P(u) \prec T(v)$) if $P(u) \subset T(v)$ holds and there is no $v' \in Des(v)$ such that $P(u) \subset T(v')$.

▶ **Proposition 2.** *Let $Chd(u) = \{u_1, \ldots, u_d\}$. $P(u) \subset T(v)$ holds if and only if the following conditions are satisfied.*

(1) $\ell(u) = \ell(v)$.

(2) *v has a set of descendants $D(v) = \{v_1, \ldots, v_d\}$ such that $v_i \notin Des(v_j)$ for all $i \neq j$.*

(3) *There exists a bijection ϕ from $Chd(u)$ to $D(v)$ such that $P(u_i) \prec T(\phi(u_i))$ holds for all $u_i \in Chd(u)$.*

Proof. Conditions (1) and (2) are obvious. To prove (3), suppose there exists a bijection ϕ' from $Chd(u)$ to $D(v)$ such that $P(u_j) \subset T(\phi'(u_j))$ holds for all $u_j \in Chd(u)$ and $P(u_i) \prec T(\phi(u_i))$ does not hold for some $u_i \in Chd(u)$. Then, there must exist $v' \in Des(\phi'(u_i))$ such that $P(u_i) \prec T(v')$ holds. Let ϕ'' be the bijection obtained by replacing a mapping from u_i to $\phi'(u_i)$ with that from u_i to v'. Clearly, ϕ'' gives a part of an inclusion mapping. Repeatedly applying this procedure, we can obtain a bijection satisfying all conditions. ◀

Note that the conditions of this proposition mainly state that all children of u must be mapped to descendants of v that do not have ancestor-descendant relationships. Since P is included in T if and only if there exists $v \in V(T)$ such that $P \prec T(v)$, we focus on how to decide if $P(u) \prec T(v)$ assuming that whether $P(u_j) \prec T(v_i)$ holds is known for all (u_j, v_i) with $u_j \in Des(u) \cup \{u\}$, $v_i \in Des(v) \cup \{v\}$, and $(u_j, v_i) \neq (u, v)$.

▶ **Proposition 3.** *Suppose that $P(u) \prec T(v)$ can be decided in $O(f(d, m, n))$ time assuming that whether $P(u_j) \prec T(v_i)$ holds is known for all descendant pairs (u_j, v_i). Then the unordered tree inclusion problem can be solved in $O(f(d, m, n)mn)$ time by using a bottom-up dynamic programming procedure.*

3 An $O(d2^d mn^2)$-time algorithm

The crucial parts of the algorithm in [14] are the definition of $S(v)$ and its computation (see [14] for the details since our algorithms are significantly different from theirs). For each fixed u in P, $S(v)$ is defined by

$$S(v) \quad = \quad \{U \subseteq Chd(u) |\ P(U) \subset T(v)\},$$

where $P(U)$ is the forest induced by nodes in U and their descendants and $P(U) \subset T(v)$ means that forest $P(U)$ is included in $T(v)$ (i.e., $T(v)$ can be obtained from $P(U)$ by node insertion operations). Clearly, the size of $S(v)$ is no greater than 2^d. Note that in this paper, we use S or $S(v)$ only to denote a set, not to denote a subtree. In the algorithm of [14], the following operation is performed from left to right for the children $v_1, \ldots, v_l$ of v:

$$S \quad := \quad \{U \cup R | U \in S, R \in S(v_i)\},$$

beginning from $S = \emptyset$, and $S(v)$ is determined based on the resulting S. However, this update operation on S causes an $O(d2^{2d})$ factor because it examines $O(2^d) \times O(2^d)$ set pairs. Therefore, in order to avoid this kind of operation, we need a new approach for computing $S(v)$, as explained below.

Given an unordered tree T, we fix any left-to-right ordering of its nodes (the ordering does not affect the correctness). Then, for any two nodes $v_i, v_j \in V(T)$ that do not have any ancestor-descendant relationship, either "v_i is left of v_j" or "v_i is right of v_j" is uniquely determined. We denote "v_i is left of v_j" by $v_i \lhd v_j$.

We focus on deciding if $P(u) \prec T(v)$ holds for fixed (u, v) because this part is crucial to reduce the exponential factor (we analyze the whole time complexity in Theorem 7). Assume w.l.o.g. (without loss of generality) that $Chd(u) = \{u_1, \ldots, u_d\}$ (i.e., u has d children). For simplicity, we assume until the end of this section that $P(u_i) \sim P(u_j)$ does not hold for any $u_i \neq u_j \in Chd(u)$. For any $v_i \in V(T(v))$, define $M(v_i)$ by $M(v_i) = \{u_j \in Chd(u) | P(u_j) \prec T(v_i)\}$. For example, $M(v_0) = \emptyset$, $M(v_2) = \{u_C\}$, and $M(v_3) = \{u_D, u_E\}$ in Figure 1. For any $v_i \in V(T(v))$, $LF(v, v_i)$ denotes the set of nodes in $V(T(v))$ each of which is left of v_i (see Figure 1 for an example). Then, we define $S(v, v_i)$ by

$$\begin{aligned} S(v, v_i) \quad = \quad & \{U \subseteq Chd(u) | P(U) \subset T(LF(v, v_i))\} \\ & \cup \{U \subseteq Chd(u) | (U = U' \cup \{u_j\}) \land (P(U') \subset T(LF(v, v_i))) \land (u_j \in M(v_i))\} \end{aligned}$$

where $T(LF(v, v_i))$ is the forest induced by nodes in $LF(v, v_i)$ and their descendants. Note that $P(\emptyset) \subset T(\ldots)$ always holds. The definition of $S(v, v_i)$ leads to a dynamic programming procedure for its computation. We explain $S(v, v_i)$ and related concepts using an example in Figure 1. Suppose that we have the relations of $P(u_A) \prec T(v_1), P(u_B) \prec T(v_1), P(u_C) \prec T(v_2)$, $P(u_D) \prec T(v_3), P(u_E) \prec T(v_3), P(u_D) \prec T(v_4), P(u_F) \prec T(v_4)$. Then, the following holds: $S(v, v_0) = \{\ \emptyset\ \}$, $S(v, v_1) = \{\ \emptyset,\ \{u_A\},\ \{u_B\}\ \}$, $S(v, v_2) = \{\ \emptyset,\ \{u_C\}\ \}$, $S(v, v_3) = \{\ \emptyset,\ \{u_D\},\ \{u_E\}\ \}$, $S(v, v_4) = \{\ \emptyset,\ \{u_D\},\ \{u_E\},\ \{v_F\},\ \{u_D, u_E\},\ \{u_D, u_F\},\ \{u_E, u_F\}\ \}$.

▶ **Proposition 4.** $S(v) = \cup_{v_i \in Des(v)} S(v, v_i)$.

Proof. Let $U \in S(v)$ and $d_U = |U|$. Let ϕ be an injection from U to $Des(v)$ giving an inclusion mapping for $P(U) \subset T(v)$. Let $\{v'_1, \ldots, v'_{d_U}\} = \{\phi(u_j) | u_j \in U\}$, where $v'_1 \lhd v'_2 \lhd \cdots \lhd v'_{d_U}$. Then, $v'_i \in LF(v, v'_{i+1})$ and $v'_i \in LF(v, v'_{d_U})$ hold for all $i = 1, \ldots, d_U - 1$. Furthermore, $P(u_j) \prec T(v'_i)$ holds for $v'_i = \phi(u_j)$. Therefore, $U \in S(v, v'_{d_U})$.

It is straightforward to see that $S(v, v_i)$ does not contain any element not in $S(v)$. ◀

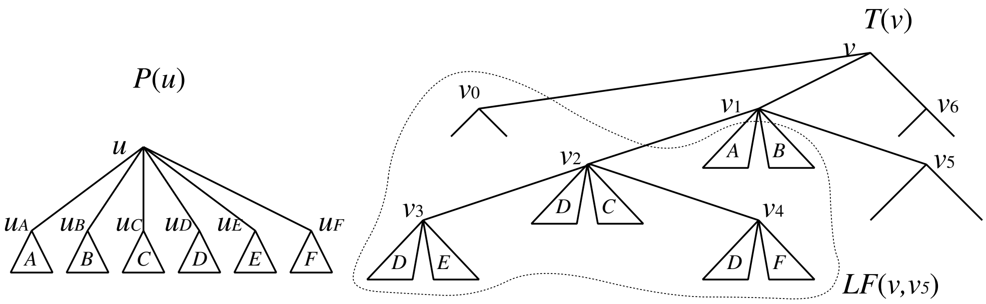

Figure 1 Example for explaining the key idea. A triangle X attached to v_i means that $P(u_X) \subset T(v_i)$ holds. Note that triangle D appears at v_2, v_3, and v_4. However, $P(u_D) \prec T(v_2)$ does not hold since it does not satisfy the minimality condition. Therefore, v_2 is never selected for matching to u_D in **AlgInc1**: if we need to match u_D to v_2, we can instead use a matching between u_D and v_3.

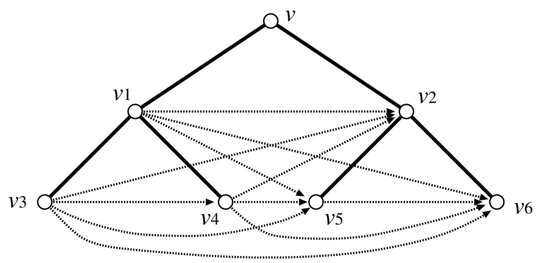

Figure 2 Example of a DAG $G(V, E)$ constructed from $T(v)$, where $v \notin V$, E is shown by dashed arrows, and $T(v)$ is shown by bold lines.

We construct a DAG (directed acyclic graph) $G(V, E)$ from $T(v)$ (see also Figure 2). V is defined by $V = V(T(v)) - \{v\}$, and E is defined by $E = \{(v_i, v_j)| \ v_i \lhd v_j, \}$. Then, we traverse $G(V, E)$ so that node v_i is visited only after all of its predecessors are visited. Let $Pred(v_i)$ denote the set of the predecessors of v_i (i.e., $Pred(v_i)$ is the set of nodes left of v_i). Recall that $M(v_i) = \{u_j \in Chd(u)| \ P(u_j) \prec T(v_i)\}$.

Then, we compute $S(v, v_i)$ by the following procedure, which is referred to as **AlgInc1**.

(1) $S_0(v_i) \leftarrow \bigcup_{v_j \in Pred(v_i)} S(v, v_j)$.

(2) $S(v, v_i) \leftarrow S_0(v_i) \cup \{S \cup \{u_h\}| \ u_h \in M(v_i), S \in S_0(v_i)\}$.

If $Pred(v_i) = \emptyset$, we let $S(v, v_i) \leftarrow \{\emptyset\} \cup \{\{u_h\}| \ u_h \in M(v_i)\}$. Finally, we let $S(v) \leftarrow \bigcup_{v_i \in Des(v)} S(v, v_i)$. Then, $P(u)$ is included in $T(v)$ with u corresponding to v iff u and v have the same label and $Chd(u) \in S(v)$.

▶ **Lemma 5.** ***AlgInc1*** *correctly computes $S(v, v_j)$ for all $v_j \in Des(v)$ in $O(d2^d n^2)$ time.*

Proof. Since it is straightforward to prove the correctness, we analyze the time complexity. The sizes of $S(v)$, $S(v, v_{i_j})$s, and $S_0(v_i)$s are $O(d2^d)$, and computation of each of such sets can be done in $O(d2^d n)$ time. Since the number of $S(v, v_{i_j})$s and $S_0(v_i)$s (per v) are $O(n)$, the total computation time is $O(d2^d n^2)$. ◀

If there exist $u_i, u_j \in Chd(u)$ such that $P(u_i) \sim P(u_j)$, we treat each element in $S(v)$, $S(v, v_{i_j})$s, and $S_0(v_i)$s as a multiset where any u_i and u_j such that $P(u_i) \sim P(u_j)$ are identified and the multiplicity of u_i is bounded by the number of $P(u_j)$s isomorphic to $P(u_i)$. Then, since $|Chd(u)| \leq d$ for all u in P, the size of each multiset is at most d and the number of different multisets is not greater than 2^d. Therefore, the same time complexity result holds. This discussion can also be applied to the following sections. Note that by treating these u_i and u_j separately, we need not change the algorithm. However, use of multi-sets plays an important role in Section 7.

AlgInc1 does a lot of redundant computations. In order to compute $S_0(v_i)$, we do not need to consider all v_{i_j}s that are left of v_i. Instead, we construct a tree $T'(v)$ from a given $T(v)$ by the following rule: for each pair of consecutive siblings (v_i, v_j) in $T(v)$, add a new sibling (leaf) $v_{(i,j)}$ between v_i and v_j. Newly added nodes are called *virtual nodes*. We construct a DAG $G'(V', E')$ on $V' = V(T'(v))$ by: $(v_i, v_j) \in E'$ iff one of the following holds

- v_j is a virtual node, and v_i is in the rightmost path of $T'(v_{j_1})$, where $v_j = v_{(j_1,j_2)}$.
- v_i is a virtual node, and v_j is in the leftmost path of $T'(v_{i_2})$, where $v_i = v_{(i_1,i_2)}$.

Then, we can use the same technique as **AlgInc1**, except that $G(V, E)$ is replaced by $G'(V', E')$. We denote the resulting algorithm by **AlgInc2**.

▶ **Lemma 6.** ***AlgInc2*** *correctly computes $S(v, v_j)$ for all $v_j \in Des(v)$ in $O(d2^d n)$ time.*

Since checking the minimality can be done in $O(m)$ time per (u, v), it is seen from Proposition 3 that the total time complexity is $O(d2^d mn^2)$. Since the size of each $S(v, v_i)$ is $O(d2^d)$ and we need to maintain information about $P(u) \prec T(v)$ and $P(u) \subset T(v)$ for all (u, v), the total space is $O(d2^d n + mn)$,

▶ **Theorem 7.** *Unordered tree inclusion can be solved in $O(d2^d mn^2)$ time using $O(d2^d n+mn)$ space.*

If we analyze the time complexity carefully, we can see that it is $O(d2^d h(T)mn)$ because each v_i is involved in computation of $P(u) \prec T(v)$ only for $v \in Anc(v_i)$. This result is better than that of [14] if d is not small (precisely, $d > c \log(h(T))$ for some constant c).

4 NP-hardness of unordered tree inclusion for pattern trees with unique leaf labels

For any node-labeled tree T, let $L(T)$ be the set of all leaf labels in T. For any $c \in L(T)$, let $OCC(T, c)$ be the number of times that c occurs in T, and define $OCC(T) = \max_{c \in L(T)} OCC(T, c)$.

The decision version of the tree inclusion problem is to determine whether T can be obtained from P by applying node insertion operations. Kilpeläinen and Mannila [14] proved that the decision version of unordered tree inclusion is NP-complete by reducing from Satisfiability. In their reduction, the clauses in a given instance of Satisfiability are represented by node labels in the constructed trees; in particular, for every clause C, each literal in C introduces one node in T whose node label represents C. By using 3-SAT instead of Satisfiability in their reduction, we immediately have:

▶ **Corollary 8.** *The decision version of the unordered tree inclusion problem is NP-complete even if restricted to instances where $h(T) = 2$, $h(P) = 1$, $OCC(T) = 3$, and $OCC(P) = 1$.*

In Kilpeläinen and Mannila's reduction, the labels assigned to the internal nodes of T are significant. Here, we consider the computational complexity of the special case of the problem where all internal nodes in P and T have the same label, or equivalently, where only the leaves are labeled. Then, we have the following.

▶ **Theorem 9.** *The decision version of the unordered tree inclusion problem is NP-complete even if restricted to instances where $h(T) = 2$, $h(P) = 2$, $OCC(T) = 3$, $OCC(P) = 1$, and all internal nodes have the same label.*

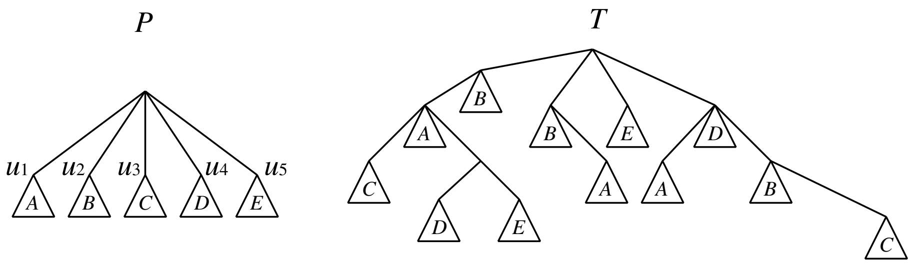

Figure 3 For these trees, $Occ(u_1, M) = Occ(u_2, M) = 3$, $Occ(u_3, M) = Occ(u_4, M) = Occ(u_5, M) = 2$, $d_2 = 3$, $d_3 = 2$, and $OCC(P, T) = 3$.

5 A polynomial-time algorithm for case of $OCC(P, T) = 2$

In this and the following sections, for the simplicity, we consider the decision version of unordered tree inclusion. However, by repeatedly applying each procedure $O(n)$ times, we can solve the locating problem version and thus the theorems hold as they are.

In this section, we require that each leaf of P has a unique label and that it appears at no more than k leaves in T. We denote this number k by $OCC(P, T)$ (see Figure 3). Note that the case of $OCC(P) = 1$ and $OCC(T) = k$ is included in the case of $OCC(P, T) = k$. From the unique leaf label assumption, we have the following observation.

▶ **Proposition 10.** *Suppose that $P(u)$ has a leaf labeled with b. If $P(u) \subset T(v)$, then v is an ancestor of a leaf (or leaf itself) with label b.*

We say that v_j is a minimal node for u_i if $P(u_i) \prec T(v_j)$ holds. It follows from this proposition that the number of minimal nodes is at most k for each u_i if $OCC(P, T) = k$.

When $k = 2$, we can have a chain of choices of the subtrees of P in T. This suggests that 2-SAT is useful. Indeed, by using a polynomial-time reduction to 2-SAT, we have:

▶ **Theorem 11.** *Unordered tree inclusion can be solved in polynomial time if $OCC(P, T) = 2$.*

6 An $O^*(1.8^d)$-time algorithm for case of $OCC(P, T) = 3$

In this section, we present an $O^*(1.8^d)$-time algorithm for the case of $OCC(P, T) = 3$, where d is the maximum degree of P, $m = |V(P)|$, and $n = |V(T)|$. Note that this case remains NP-hard from Theorem 9.

The basic strategy is use of dynamic programming: decide whether $P(u) \subset T(v)$ in a bottom-up way. Suppose that u has a set of children $U = \{u_1, \dots, u_d\}$. Since we use dynamic programming, we can assume that $P(u_i) \prec T(v_j)$ is known for all u_i and for all $v_j \in V(T(v)) - \{v\}$. We define $\mathcal{M}(u, v)$ by $\mathcal{M}(u, v) = \{(u_i, v_j) |\ P(u_i) \prec T(v_j) \ \wedge \ v_j \in V(T(v))\}$.

The crucial task of the dynamic programming procedure is to find an injective mapping ψ from $\{u_1, \dots, u_d\}$ to $V(T(v)) - \{v\}$ such that $P(u_i) \prec T(\psi(u_i))$ holds for all u_i $(i = 1, \dots, d)$ and there is no ancestor/descendant relationship between any $\psi(u_i)$ and $\psi(u_j)$ $(u_i \neq u_j)$. If this task can be performed in $O(f(d, m, n))$ time, from Proposition 3, the total complexity will be $O^*(f(d, m, n))$. We assume w.l.o.g. that ψ is given as a set of mapping pairs. For each $v_j \in V(T(v))$ and each $M \subseteq \mathcal{M}(u, v)$, we define $AncDes(v_j, T, M)$ by

$$AncDes(v_j, T, M) \quad = \quad \{(u_k, v_h) |\ (u_k, v_h) \in M \ \wedge \ v_h \in (\{v_j\} \cup Anc(v_j, T) \cup Des(v_j, T))\},$$

where $Anc(v_j, T)$ (resp., $Des(v_j, T)$) denotes the set of ancestors (resp., descendants) of v_j in T where $v_j \notin Anc(v_j, T)$ (resp., $v_j \notin Des(v_j, T)$).

Here, we define $Occ(u_i, M)$ by $Occ(u_i, M) = |\{j \mid (u_i, v_j) \in M\}|$, where $M = \mathcal{M}(u, v)$. Let d_3 (resp., d_2) be the number of u_is such that $Occ(u_i, M) = 3$ (resp., $Occ(u_i, M) = 2$) (see also Figure 3). We assume w.l.o.g. that $d_2 + d_3 = d$ because $Occ(u_i, M) = 1$ means that $\psi(u_i)$ is uniquely determined and thus we can ignore u_is with $Occ(u_i, M) = 1$. From Theorem 11, we can see the following if there are no two pairs $(u_{i_1}, v_{j_1}), (u_{i_2}, v_{j_2}) \in M$ such that $Occ(u_{i_1}, M) = 3$, $Occ(u_{i_2}, M) = 3$, and $(u_{i_2}, v_{j_2}) \in AncDes(v_{j_1}, T(v), M)$.

- The problem can be solved in $O^*(2^{d_3})$ time:
  For each u_i such that $Occ(u_i, M) = 3$ (i.e., $(u_i, v_{j_1}), (u_i, v_{j_2}), (u_i, v_{j_3}) \in M$), we choose $\psi(u_i) = v_{j_1}$ (i.e., $(u_i, v_{j_1}) \in \psi$) or not. Thus, there exist 2^{d_3} possibilities. After all the choices, there is no u_i such that $Occ(u_i, M) = 3$ and Theorem 11 can be applied.
- The problem can also be solved in $O^*(2^{d_2})$ time:
  For each u_i with $Occ(u_i, M) = 2$ (i.e., $(u_i, v_{j_1}), (u_i, v_{j_2}) \in M$), we must choose $\psi(u_i) = v_{j_1}$ or $\psi(u_i) = v_{j_2}$. Thus, there are 2^{d_2} possibilities. After all choices, each $(u_i, v_j) \in M$ with $Occ(u_i, M) = 2$ is removed, and thus there is no pairs $(u_{i_1}, v_{j_1}), (u_{i_2}, v_{j_2}) \in M$ such that $(u_{i_2}, v_{j_2}) \in AncDes(v_{j_1}, T(v), M)$ from the 'if' condition. Therefore, the problem is reduced to bipartite matching, which can be solved in polynomial time.

It means the problem can be solved in $O^*(\min(2^{d_3}, 2^{d_2}))$ time. We denote the condition (i.e., 'if' part of the above) and this algorithm by **(##)** and **ALG-##**, respectively, Therefore, the crucial point is how to (recursively) remove pairs such that $Occ(u_{i_1}, M) = 3$, $Occ(u_{i_2}, M) = 3$, and $(u_{i_2}, v_{j_2}) \in AncDes(v_{j_1}, T(v), M)$.

For a mapping ψ, we let $\psi \cup NULL = NULL$, where $NULL$ means that there is no valid mapping. The following is a pseudocode of the algorithm for finding a mapping ψ, where it is invoked as $FindMapping(\{u_1, \ldots, u_d\}, M)$ with $M = \mathcal{M}(u, v)$.

Procedure $FindMapping(U, M)$
  if condition **(##)** is satisfied **then**
    return mapping by **ALG-(##)**; **(#1)**
  Choose arbitrary $(u_{i_1}, v_{j_1}), (u_{i_2}, v_{j_2}) \in M$ such that $Occ(u_{i_1}, M) = 3$, $Occ(u_{i_2}, M) = 3$,
    and $(u_{i_2}, v_{j_2}) \in AncDes(v_{j_1}, T(v), M)$; **(#2)**
  $M' \leftarrow M - \{(u_{i_1}, v_{j_1})\}$; **(#3)**
  $\psi \leftarrow FindMapping(U, M')$;
  if $\psi \neq NULL$ **return** ψ;
  $M' \leftarrow M - AncDes(v_{j_1}, T(v), M)$; **(#4)**
  return $\{(u_{i_1}, v_{j_1})\} \cup FindMapping(U - \{u_{i_1}\}, M')$.

▶ **Theorem 12.** *Unordered tree inclusion can be solved in $O^*(1.8^d)$ time if $OCC(P, T) = 3$.*

7 A randomized algorithm for case of $h(P) = 1$ and $h(T) = 2$

In this section, we consider the case of $h(P) = 1$ and $h(T) = 2$, which is denoted by **IncH2** and remains NP-hard from Corollary 8. We assume w.l.o.g. that the roots of P and T have the same unique label and thus they must match in any inclusion mapping.

Let $U = \{u_1, \ldots, u_d\}$ be the set of children of $r(P)$. Let $v_1, \ldots, v_g$ be the children of $r(T)$, and let $v_{i,1}, \ldots, v_{i,n_i}$ be the children of each v_i.

First, we assume that $\ell(u_i) \neq \ell(u_j)$ holds for all $i \neq j$, where $\ell(v)$ denotes the label of v. This special case is denoted by **IncH2U**. Recall that **IncH2U** remains NP-hard.

IncH2U can be solved by a reduction to CNF SAT, which is different from the one in Section 5 and is considered as a reverse reduction of the one used for proving NP-hardness of unordered tree inclusion [14]. For each u_i, we define X_i^{POS} and X_i^{NEG} by

$$X_i^{POS} = \{x_j |\ \ell(u_i) = \ell(v_j)\}, \quad X_i^{NEG} = \{x_j |\ (\exists v_{j,k} \in Chd(v_j))(\ell(u_i) = \ell(v_{j,k}))\}.$$

For each u_i, we construct a clause C_i by $C_i = \left(\bigvee_{x_j \in X_i^{POS}} x_j \right) \vee \left(\bigvee_{x_j \in X_i^{NEG}} \overline{x_j} \right)$. Then, the resulting SAT instance is $\{C_1, \ldots, C_d\}$. Intuitively, $x_j = 1$ corresponds to a case that u_i is mapped to v_j, where $\ell(u_i) = \ell(v_j)$. Of course, multiple v_js may correspond to u_i. However, it is enough to consider an arbitrary one.

Then, it is straightforward to see that P is included in T iff $\{C_1, \ldots, C_d\}$ is satisfiable. Using Yamamoto's algorithm for SAT with d clauses [22], we have:

▶ **Proposition 13.** ***IncH2U*** *can be solved in* $O^*(1.234^d)$ *time.*

Next, we consider **IncH2**. We combine two algorithms: (A1) random sampling-based algorithm, and (A2) modified version of the $O(d2^d mn^2)$ time algorithm in Section 3.

For (A1), we employ a technique used in *color-coding* [6]. Let d_0 be the number of u_is having unique labels. Let $d_1 \leq d_2 \leq \cdots \leq d_h$ be the multiplicities of other labels in U. Note that $d_0 + d_1 + \cdots + d_h = d$ holds. Let $d - d_0 = \alpha d$.

For each label a_i with $d_i > 1$ (i.e., $i > 0$), we change the labels of nodes with label a_i in P to $a_i^1, a_i^2, \ldots, a_i^{d_i}$ in an arbitrary way. For each node v in T having label a_i, we assign a_i^j $(j = 1, \ldots, d_i)$ to v uniformly at random, and then apply the SAT-based algorithm for **IncH2U**. Let M be the set of pairs for an inclusion mapping from P to T. If all nodes of T appearing in M have different labels, a valid inclusion mapping can be obtained. This success probability is given by

$$\frac{d_1!}{d_1^{d_1}} \cdot \frac{d_2!}{d_2^{d_2}} \cdots \frac{d_h!}{d_h^{d_h}} \geq \frac{(\alpha d)!}{(\alpha d)^{(\alpha d)}}.$$

Note that this inequality is proved by repeatedly applying $\frac{d_1!}{d_1^{d_1}} \cdot \frac{d_2!}{d_2^{d_2}} \geq \frac{(d_1+d_2)!}{(d_1+d_2)^{d_1+d_2}}$, which is seen from $\frac{(d_1+d_2)^{d_1+d_2}}{d_1^{d_1} d_2^{d_2}} \geq \binom{d_1+d_2}{d_1} = \frac{(d_1+d_2)!}{d_1! d_2!}$. Since $\frac{k!}{k^k} \geq e^{-k}$ holds for sufficiently large k, the success probability is at least $e^{-\alpha d}$. Therefore, if we repeat the random sampling procedure $e^{\alpha d}$ times, the failure probability is at most $(1 - e^{-\alpha d})^{e^{\alpha d}} \leq e^{-1} < \frac{1}{2}$.

If we repeat the procedure $k(\log n)e^{\alpha d}$ times where k is any positive constant (i.e., the total time complexity is $O^*(1.234^d \cdot e^{\alpha d})$), the failure probability is at most $\frac{1}{n^k}$.

For (A2), we modify the $O(d2^d mn^2)$ time algorithm as follows. Recall that if there exist labels with multiplicity more than one, $S(v, v_i)$ is a multi-set. In order to represent a multi-set, we memorize the multiplicity of each label. Then, the number of distinct multi-sets is given by

$$N(d_0, \ldots, d_h) = 2^{d_0} \cdot \prod_{l=1}^{h} (d_l + 1).$$

Since $d_i + 1 \leq 3^{\lceil d_i/2 \rceil}$ holds for any $d_i \geq 2$, this number is bounded as

$$N(d_0, \ldots, d_h) \leq 2^{d_0} \cdot 3^{\lceil (d-d_0)/2 \rceil}.$$

Then, the time complexity of (A2) is $O^*(2^{(1-\alpha)d} \cdot 3^{(\alpha/2)d})$.

Since we can use the minimum of the time complexities of (A1) and (A2), the resulting time complexity is given by

$$\max_{\alpha} \min(O^*(1.234^d \cdot e^{\alpha d}), O^*(2^{(1-\alpha)d} \cdot 3^{(\alpha/2)d})).$$

By numerical calculation, this is $O^*(1.883^d)$.

▶ **Theorem 14.** ***IncH2*** *can be solved in randomized* $O^*(1.883^d)$ *time with probability at least* $1 - \frac{1}{n^k}$*, where* k *is any positive constant.*

It seems that the above algorithm can be de-randomized by using the k-perfect hash family as in [6]. However, since the construction of a k-perfect hash family has a high complexity, the resulting algorithm might have a time complexity much worse than $O^*(2^d)$.

8 Concluding remarks

We have improved the exponential factor of Kilpeläinen and Mannila's [14] well-known algorithm from 1995 for unordered tree inclusion from 2^{2d} to 2^d. Observe that the 2^d factor may not be optimal. Indeed, we have presented a randomized $O^*(1.883^d)$-time algorithm for the case of $h(P) = 1$ and $h(T) = 2$. However, we could not obtain an $O^*((2-\epsilon)^d)$-time algorithm for any constant $\epsilon > 0$ even for the case of $h(P) = h(T) = 2$. Development of an $O^*((2-\epsilon)^d)$-time algorithm for unordered tree inclusion, or showing an $\Omega(2^d)$ lower bound using recent techniques for proving lower bounds on various matching problems [1, 2, 9], is left as an open problem.

References

1. Amir Abboud, Arturs Backurs, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Or Zamir. Subtree isomorphism revisited. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1256–1271. SIAM, 2018.
2. Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming - Part 1*, pages 39–51. Springer, 2014.
3. Tatsuya Akutsu, Daiji Fukagawa, Magnús M. Halldórsson, Atsuhiro Takasu, and Keisuke Tanaka. Approximation and parameterized algorithms for common subtrees and edit distance between unordered trees. *Theoretical Computer Science*, 470:10–22, 2013.
4. Tatsuya Akutsu, Daiji Fukagawa, Atsuhiro Takasu, and Takeyuki Tamura. Exact algorithms for computing the tree edit distance between unordered trees. *Theoretical Computer Science*, 412(4-5):352–364, 2011.
5. Tatsuya Akutsu, Takeyuki Tamura, Daiji Fukagawa, and Atsuhiro Takasu. Efficient exponential-time algorithms for edit distance between unordered trees. *Journal of Discrete Algorithms*, 25:79–93, 2014.
6. Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995.
7. Philip Bille. A survey on tree edit distance and related problems. *Theoretical Computer Science*, 337(1):217–239, 2005.
8. Philip Bille and Inge Li Gørtz. The tree inclusion problem: In linear space and faster. *ACM Transactions on Algorithms (TALG)*, 7(3):38, 2011.
9. Karl Bringmann, Pawel Gawrychowski, Shay Mozes, and Oren Weimann. Tree edit distance cannot be computed in strongly subcubic time (unless APSP can). In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1190–1206. SIAM, 2018.

10 Lijun Chang, Xuemin Lin, Wenjie Zhang, Jeffrey Xu Yu, Ying Zhang, and Lu Qin. Optimal enumeration: Efficient top-k tree matching. *Proceedings of the VLDB Endowment*, 8(5):533–544, 2015.

11 Sara Cohen and Nerya Or. A general algorithm for subtree similarity-search. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 928–939. IEEE, 2014.

12 Erik D. Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann. An optimal decomposition algorithm for tree edit distance. *ACM Transactions on Algorithms (TALG)*, 6(1):2, 2009.

13 Minoru Kanehisa, Susumu Goto, Yoko Sato, Masayuki Kawashima, Miho Furumichi, and Mao Tanabe. Data, information, knowledge and principle: back to metabolism in KEGG. *Nucleic Acids Research*, 42(D1):D199–D205, 2013.

14 Pekka Kilpeläinen and Heikki Mannila. Ordered and unordered tree inclusion. *SIAM Journal on Computing*, 24(2):340–356, 1995.

15 Hanna Köpcke, Andreas Thor, and Erhard Rahm. Evaluation of entity resolution approaches on real-world match problems. *Proceedings of the VLDB Endowment*, 3(1-2):484–493, 2010.

16 Jiwei Li, Thang Luong, Dan Jurafsky, and Eduard H. Hovy. When Are Tree Structures Necessary for Deep Learning of Representations? In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 2304–2314, 2015. URL: http://aclweb.org/anthology/D/D15/D15-1278.pdf.

17 Jiří Matoušek and Robin Thomas. On the complexity of finding iso-and other morphisms for partial k-trees. *Discrete Mathematics*, 108(1-3):343–364, 1992.

18 Tomoya Mori, Atsuhiro Takasu, Jesper Jansson, Jaewook Hwang, Takeyuki Tamura, and Tatsuya Akutsu. Similar subtree search using extended tree inclusion. *IEEE Transactions on Knowledge and Data Engineering*, 27(12):3360–3373, 2015.

19 Dennis Shasha, Jason T. L. Wang, Kaizhong Zhang, and Frank Y. Shih. Exact and approximate algorithms for unordered tree matching. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(4):668–678, 1994.

20 Kuo-Chung Tai. The tree-to-tree correction problem. *Journal of the ACM (JACM)*, 26(3):422–433, 1979.

21 Gabriel Valiente. Constrained tree inclusion. *Journal of Discrete Algorithms*, 3(2):431–447, 2005.

22 Masaki Yamamoto. An improved $O^*(1.234^m)$-time deterministic algorithm for SAT. In *Proceedings of the 16th International Symposium on Algorithms and Computation*, pages 644–653. Springer, 2005.

23 Mohammed Javeed Zaki. Efficiently mining frequent trees in a forest: Algorithms and applications. *IEEE Transactions on Knowledge and Data Engineering*, 17(8):1021–1035, 2005.

24 Kaizhong Zhang and Tao Jiang. Some MAX SNP-hard results concerning unordered labeled trees. *Information Processing Letters*, 49(5):249–254, 1994.

25 Kaizhong Zhang, Rick Statman, and Dennis Shasha. On the editing distance between unordered labeled trees. *Information Processing Letters*, 42(3):133–139, 1992.