Algorithms for Finding a Most Similar Subforest

Jesper Jansson[1,⋆] and Zeshan Peng[2]

[1] Department of Computer Science and Communication Engineering, Kyushu University, 6-10-1 Hakozaki, Higashi-ku, Fukuoka 812-8581, Japan
[2] Department of Computer Science, The University of Hong Kong, Pokfulam Road, Hong Kong
jj@tcslab.csce.kyushu-u.ac.jp, zspeng@cs.hku.hk

Abstract. Given an ordered labeled forest F ("the target forest") and an ordered labeled forest G ("the pattern forest"), the *most similar subforest problem* is to find a subforest F' of F such that the distance between F' and G is minimum over all possible F'. This problem generalizes several well-studied problems which have important applications in locating patterns in hierarchical structures such as RNA molecules' secondary structures and XML documents. In this paper, we present efficient algorithms for the most similar subforest problem with forest edit distance for three types of subforests: simple substructures, sibling substructures, and closed subforests.

1 Introduction

An *ordered labeled tree* is a rooted tree in which the left-to-right ordering among nodes is fixed and each node is labeled by a symbol from a given alphabet. An *ordered labeled forest* is a sequence of ordered labeled trees. Ordered labeled trees and forests are useful data structures for hierarchical data representation; for example, XML documents are essentially ordered labeled trees [2] and RNA molecules' secondary structures without pseudoknots can be represented by ordered labeled forests [4, 7, 11] (see Fig. 1(a)–(c) for an example). Below, we refer to ordered labeled trees and ordered labeled forests as *trees* and *forests*, respectively.

In this paper, we study the following problem which we call *the most similar subforest problem*: Given a forest F ("the target forest") and a forest G ("the pattern forest"), find a subforest of F which is the most similar to G. There are many ways to define "subforest" and "most similar"; here, we consider three alternative definitions of "subforest" and show how to solve all the resulting problems efficiently when the *forest edit distance* [13, 16] is used to measure similarity. Our techniques combine and extend the techniques of [4] and [16].

The most similar subforest problem generalizes several other problems. For example, in the well-studied *forest inclusion problem*, the objective is to determine whether a given forest G can be obtained from another given forest F by only deleting nodes from F, and if so, finding the smallest subforest of F in

⋆ Supported by JSPS (Japan Society for the Promotion of Science).

M. Lewenstein and G. Valiente (Eds.): CPM 2006, LNCS 4009, pp. 377–388, 2006.

which G is included (this problem and a constrained variant have been studied in, e.g., [15, 8]). However, in case G is *not* included in F, one might still need to find a subforest F' of F such that G is very similar to F', or to measure how far from being included in F the pattern forest G is. This is precisely "the most similar subforest problem". As another example, consider the *string pattern problem*: given two strings S and T, find a most similar (using edit distance) substring of S to T. This problem has many applications to Stringology [3] and Bioinformatics [12]. Since a string can be represented by a tree in which all non-leaf nodes have exactly one child, the string pattern problem is just a special case of the most similar subforest problem. [1]

2 Preliminaries

Throughout this paper, we use the following notation and definitions.

Let F be any given forest. Denote the number of nodes in F by $|F|$, and define $\deg(F)$ (the *degree of* F) as the maximum number of children over all nodes in F, and $dp(F)$ (the *depth of* F) as the number of edges on the longest path from a root node in F to a leaf of F. The set of leaves in F is referred to as $L(F)$. For any node $i \in F$, define $p(i)$ as *the parent of* i, and denote the label of i by $label(i)$. Any $i_1, i_2 \in F$ are *siblings* if they have the same parent; if $i_1 \neq i_2$ also holds then i_1 and i_2 are *proper siblings*. To simplify the presentation, we assume that the roots of the trees in F share an imaginary parent node, denoted by $p(F)$, which is considered to belong to F and which is labeled by a special symbol '$\diamond$'. Define the *key nodes of* F as the set $K(F) = \{p(F) \cup i \mid i \in F \text{ has a left proper sibling}\}$. Clearly, it holds that $|K(F)| \leq |L(F)|$ (Lemma 6 in [16]).

Assume without loss of generality that the nodes of F are *numbered* according to the order in which they are visited by a left-to-right postorder traversal of F. Then, for any $i_1, i_2 \in F$, define $i_1 : i_2$ as the set of nodes whose numbers are greater than or equal to i_1 and less than or equal to i_2. For any siblings i_1 and i_2 with $i_1 \leq i_2$, define $i_1 \cdot\cdot\, i_2$ as the set of nodes consisting of i_1, i_2, and every node which is both a right sibling of i_1 and a left sibling of i_2 (if $i_1 > i_2$, define $i_1 \cdot\cdot\, i_2 = \emptyset$). Finally, for any $i \in F$, refer to the *leftmost* and *rightmost* siblings of i by $b(i)$ and $e(i)$, respectively, and define $m(i)$ as the smallest numbered node in the subtree rooted at i (note that by the left-to-right postordering of the nodes, $m(i)$ will always be the leftmost leaf in this subtree).

Forest edit distance [13, 16]**:** Define the following three *edit operations* on F:

–*Relabel:* Change the label of any node in F.
–*Delete:* Delete any node i from F by making all of i's children (if any) become children of $p(i)$ and then removing node i and the edge between i and $p(i)$.
– *Insert:* Insert a new node with any label into F (the inverse operation of delete).

[1] In fact, the running time of our algorithm for finding a most similar simple substructure in Section 3.1 with parameters $|L(F)| = 1$ and $|L(G)| = 1$ matches the fastest known algorithm for the string pattern problem.

See, e.g., [1, 4, 5, 13, 14, 16] for examples of these operations. Next, define an *edit mapping* M between two forests F and G as a set of pairs (i, j), where $i \in F$ and $j \in G$, such that for any two pairs $(i_1, j_1), (i_2, j_2) \in M$, the following properties are satisfied: **(1)** $i_1 = i_2$ if and only if $j_1 = j_2$; **(2)** i_1 is an ancestor of i_2 if and only if j_1 is an ancestor of j_2; and **(3)** $i_1 < i_2$ if and only if $j_1 < j_2$. For any $(i, j) \in M$, we say that node i is *linked with* node j in M. Let M be an edit mapping between F and G. Define its *left-linked set* as $M_F = \{i \mid (i, j) \in M\}$ and its *left-unlinked set* as $R_F = F \setminus M_F$, and define its *right-linked set* M_G and *right-unlinked set* R_G analogously. An edit mapping M between F and G uniquely determines a sequence of delete and relabel operations on F and G such that the resulting forests F' and G' are identical. More precisely, every $i \in R_F$ means "delete node i from F", every $j \in R_G$ means "delete node j from G", and every $(i, j) \in M$ with $label(i) \neq label(j)$ means "relabel i with the label of j".

From here on, we assume that the nodes in the input forests F are G are labeled by a fixed alphabet Σ where $\diamond \notin \Sigma$ (recall that the symbol '$\diamond$' is already in use). Moreover, we assume that '$-$' is a special blank symbol not in Σ and that we are given a fixed *distance function* $\gamma : (\Sigma \cup \{\diamond, -\}) \times (\Sigma \cup \{\diamond, -\}) \to \Re$, where $\Re$ is the set of real numbers and where for any $a, b \in \Sigma$, it holds that $\gamma(a, a) \leq 0$, $\gamma(a, b) \geq 0$ if $a \neq b$, $\gamma(a, -) \geq 0$, $\gamma(-, b) \geq 0$, and $\gamma(-, -) > 0$. We also assume that $\gamma(a, \diamond) = 0$, $\gamma(\diamond, b) = 0$, $\gamma(\diamond, \diamond) = 0$, $\gamma(\diamond, -) \geq 0$, $\gamma(-, \diamond) \geq 0$. For any $i \in F$ and $j \in G$, define $f(i) = label(i)$ and $g(j) = label(j)$. Then, for any $i \in F$ and $j \in G$, the *distance* between i and j is defined as $\gamma(i, j) = \gamma(f(i), g(j))$. Finally, define the *cost* of an edit mapping M as:

$$\delta(M) = \sum_{(i,j) \in M} \gamma(f(i), g(j)) + \sum_{i \in R_F} \gamma(f(i), -) + \sum_{j \in R_G} \gamma(-, g(j)).$$

An *optimal edit mapping* between two forests F and G is an edit mapping with the minimum cost: $\min\{\delta(M)\}$ over all possible M. This cost is called *the forest edit distance* between F and G, and is denoted by $\delta(F, G)$.

Subforest definitions: Let F be a forest. We define the following types of subforests of F. For any node i in F, the *subtree of F rooted at i* is the subtree consisting of i and all descendants of i, and is denoted by $F[i]$. For any siblings i_1, i_2, the set of subtrees rooted at $i_1 \cdots i_2$ forms a *closed subforest of F* (see also [4]). A *simple substructure of F* is any connected subgraph of F, and a *sibling substructure of F* is a set of disjoint simple substructures of F whose roots are siblings (not necessarily consecutive) in F. Finally, given any subset S of the nodes in F, the *restricted subforest* $F\|_S$ is defined as the forest obtained from F by deleting all nodes not in S. To illustrate these definitions, consider the forest F in Fig. 1(c) and the subforests of F shown in Fig. 1(d). Here, $F_1 = F\|_S$ is a restricted subforest for $S = \{3, 5, 6, 9, 13, 22, 31\}$, F_2 is a simple substructure of F, F_3 is a closed subforest of F, and F_4 is a sibling substructure of F.

We say that a *most similar subforest* (using any one of the above definitions for "subforest") of a forest F to a forest G is a subforest F' of F that minimizes the forest edit distance $\delta(F', G)$ over all possible F'. The *most similar subforest*

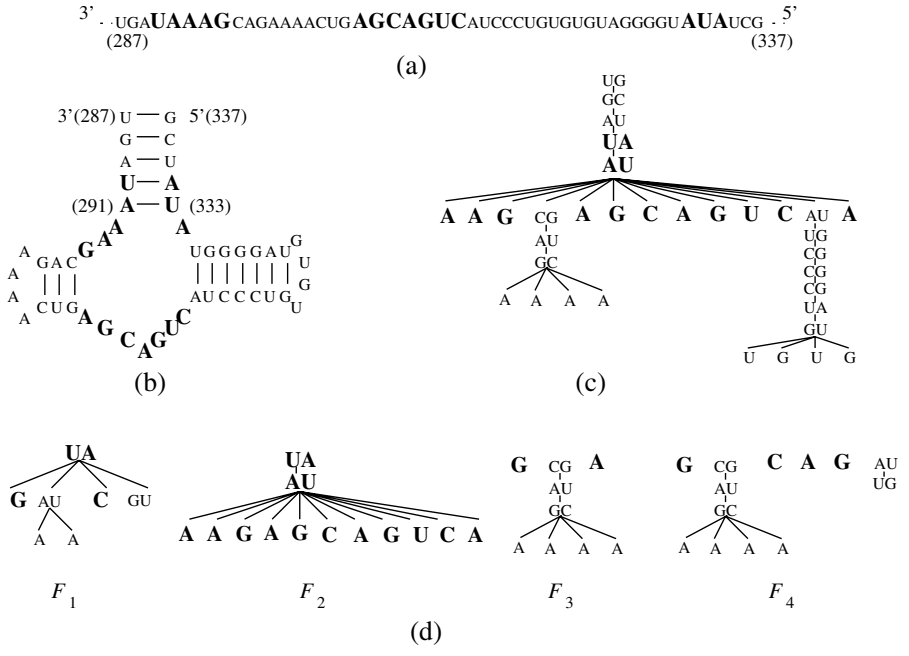

Fig. 1. (a) A segment of the primary structure of the cherry small circular viroid-like RNA molecule (accession number Y12833, GI:2347024) [10], (b) its secondary structure, (c) a forest representation F of the secondary structure (see, e.g., [4, 7, 11] for details), and (d) various types of subforests of F. The so-called *Hammerhead motif* [10], which corresponds to the pattern specified by subforest F_2, is marked in bold.

problem is: given two forests F and G, find a most similar subforest of F to G (again, using any one of the above definitions for "subforest").

Our contributions: In this paper, we show how to solve the most similar subforest problem efficiently, where "subforest" means "simple substructure", "sibling substructure", or "closed subforest". The time and space complexities of our algorithms are summarized in the next table.

Finding a most similar:	Complexity	Section
Simple substructure	$O(\|F\|\cdot\|G\|\cdot\min\{\|L(F)\|, dp(F)\}\cdot\min\{\|L(G)\|, dp(G)\})$ time, $O(\|F\|\cdot\|G\|)$ space	3.1
Sibling substructure	$O(\|F\|\cdot\|G\|\cdot\min\{\|L(F)\|, dp(F)\}\cdot\min\{\|L(G)\|, dp(G)\})$ time, $O(\|F\|\cdot\|G\|)$ space	3.2
Closed subforest	$O(\|F\|\cdot\|G\|\cdot\|L(F)\|\cdot\min\{\|L(G)\|, dp(G)\})$ time, $O(\|F\|\cdot\|G\| + \|L(F)\|\cdot dp(F)\cdot\|G\| + \|F\|\cdot\|L(G)\|\cdot dp(G))$ space	3.3

Related results: Tai [13] gave the first algorithm for computing the forest edit distance between two given forests F and G. Zhang and Shasha [16] gave a more

efficient algorithm for this problem running in $O(|F| \cdot |G| \cdot \min\{|L(F)|, dp(F)\} \cdot \min\{|L(G)|, dp(G)\})$ time and $O(|F| \cdot |G|)$ space. (Actually, these papers assumed F and G to be *trees*, but it is simple to extend their methods to forests.) More recently, Klein [9], Chen [1], and Touzet [14] have developed algorithms that are faster for certain kinds of inputs. Zhang and Shasha's algorithm [16] computes $\delta(F[i], G[j])$ for all $i \in F$, $j \in G$, and can therefore find a subtree rooted at a node in F which is the most similar to G, i.e., a most similar rooted subtree, but none of the algorithms from [1,9,13,14,16] can be used directly to efficiently find, e.g., a most similar simple substructure of F to G (the number of simple substructures of F may be exponential in $|F|$, so it is not practical to try them all separately).

An alternative measure of the similarity between two forests is the *forest alignment distance* (see [7] for a formal definition). Although the edit distance and alignment distance are equivalent for *strings*, they are not equivalent for trees and forests [7]. The algorithm of Jiang *et al.* [7] for computing the forest alignment distance runs in $O(|F| \cdot |G| \cdot (\deg(F) + \deg(G))^2)$ time, and its running time was improved for similar inputs in [5]. Note that the algorithm of Jiang *et al.* computes an optimal *global* alignment between F and G, meaning that all nodes of F and G contribute to the cost of the final solution. Recently, Höchsmann *et al.* [4] gave an algorithm for computing an optimal *local* alignment between F and G which finds a closed subforest F' of F and a closed subforest G' of G having the minimum forest alignment distance; a more efficient algorithm for this problem (running in $O(|F| \cdot |G| \cdot (\deg(F) + \deg(G))^2)$ time and $O(|F| \cdot |G| \cdot (\deg(F) + \deg(G)))$ space) along with some extensions to other types of subforests were given in [6]. Höchsmann *et al.* [4] also considered the problem of finding a closed subforest F' of F which minimizes the alignment distance to G (i.e., the analogue of our "most similar closed subforest problem" but using alignment distance instead of edit distance), which they called the *small-in-large closed subforest similarity problem*, and showed how to solve it in $O(|F| \cdot |G| \cdot \deg(F) \cdot \deg(G) \cdot (\deg(F) + \deg(G)))$ time and $O(|F| \cdot |G| \cdot \deg(F) \cdot \deg(G))$ space.

3 Algorithms for the Most Similar Subforest Problem

In this section, we present efficient algorithms for a finding a most similar subforest (simple substructure, sibling substructure, and closed subforest, respectively) of F to G. As a preprocessing step to all our algorithms below, we calculate and store $K(F), K(G)$, $L(F)$, and $L(G)$ according to their postorders in auxiliary arrays in linear time. Moreover, $m(i)$ for all $i \in F$, $i \in G$ are also precomputed.

3.1 An Algorithm for Finding a Most Similar Simple Substructure

We first introduce some additional terminology. Let F be a forest. Define a new edit operation called the *cut operation* on F as follows: for any node i in F, *cutting node* i means removing the entire subtree $F[i]$ (along with the parent edge of i if i is not a root node) from F at cost 0. Note that the cut operation

differs from the previously defined delete operation since it removes *all* the nodes in a subtree of F and is for free. For any two nodes u and v in F with $u \neq v$, we say that u and v are *consistent* if u is not a descendant of v and v is not a descendant of u. A set C of nodes from F is *consistent* if every pair of nodes in C is consistent. Denote the set of all consistent sets of nodes in F by $\mathcal{C}(F)$, and for any $C \in \mathcal{C}(F)$, let $F \ominus C$ be the forest obtained from F by cutting all nodes in C.

Suppose F' is a simple substructure of F rooted at a node i. By definition, F' is a connected subgraph of $F[i]$, which means that F' can be obtained from $F[i]$ by cutting all nodes in some (possibly empty) consistent set. We have:

Lemma 1. *Let i be a node in a forest F. F' is a simple substructure of F rooted at i if and only if $F' = F[i] \ominus C$ for some $C \in \mathcal{C}(F[i])$.*

To locate a most similar simple substructure of F to G, we look for a most similar simple substructure of $F[i]$ to G among all $i \in F$. By Lemma 1, this is equivalent to finding a $C \in \mathcal{C}(F[i])$ such that $\delta(F[i] \ominus C, G)$ is minimized since the cut operations do not contribute to the total cost of an edit mapping between $F[i]$ and G. (Observe that we are only allowed to cut nodes in F, and not in G by the problem definition.) For any given forests F' and G', define $\Psi(F', G') = \min_{C \in \mathcal{C}(F')}\{\delta(F' \ominus C, G')\}$. Then the goal of our algorithm is to compute $\min_{i \in F} \Psi(F[i], G)$. Below, we extend the techniques of [16] to derive some useful recurrences for computing certain values of Ψ.

First of all, it is easy to show that:

Lemma 2. $\Psi(\emptyset, \emptyset) = 0; \quad \Psi(F, \emptyset) = 0; \quad \Psi(\emptyset, G) = \sum_{j \in G} \gamma(-, g(j))$.

Proof. The first case is obvious since there is no cost for the empty mapping between two empty forests. For the second case, suppose $F = \langle T_1, \ldots, T_t \rangle$. We know that $F \ominus \{r(T_1), \ldots, r(T_t)\}) = \emptyset$, where $r(T)$ for any tree T refers to the root of T, so $\Psi(F, \emptyset) = \delta(\emptyset, \emptyset) = 0$. In the third case, we cannot cut any nodes, so $\Psi(\emptyset, G) = \sum_{j \in G} \gamma(-, g(j))$. □

Next, because of the left-to-right postordering of the nodes, we have $F[i] = F\|_{m(i):i}$ and $G[j] = G\|_{m(j):j}$. To compute $\Psi(F[i], G[j])$, we compute $\Psi(F\|_{m(i):x}, G\|_{m(j):y})$ for all $x \in \{m(i), \ldots, i\}$ and $y \in \{m(j), \ldots, j\}$. Intuitively, when considering nodes x and y, if the subtree $F[x]$ is very dissimilar to $G[y]$ then it will be better to cut x (i.e., remove the entire subtree $F[x]$ at once at no additional cost), in which case $F\|_{m(i):x}$ becomes just $F\|_{m(i):m(x)-1}$. On the other hand, if $F[x]$ is similar to $G[y]$ then x and y should be linked, or one of x and y should be deleted, and then the remaining parts of $F[x]$ and $G[y]$ linked.

Lemma 3. *For any $i \in F$, $j \in G$, $x \in \{m(i), \ldots, i\}$, and $y \in \{m(j), \ldots, j\}$,*

$$\Psi(F\|_{m(i):x}, G\|_{m(j):y}) = \min \begin{cases} \Psi(F\|_{m(i):m(x)-1}, G\|_{m(j):y}); \\ \Psi(F\|_{m(i):x-1}, G\|_{m(j):y}) + \gamma(f(x), -); \\ \Psi(F\|_{m(i):x}, G\|_{m(j):y-1}) + \gamma(-, g(y)); \\ \Psi(F\|_{m(i):m(x)-1}, G\|_{m(j):m(y)-1}) + \\ \qquad \Psi(F\|_{m(x):x-1}, G\|_{m(y):y-1}) + \gamma(f(x), g(y)). \end{cases}$$

Proof. Let $C \in \mathcal{C}(F\|_{m(i):x})$ be a consistent set that minimizes $\delta(F\|_{m(i):x} \ominus C, G\|_{m(j):y})$, i.e., such that $\delta(F\|_{m(i):x} \ominus C, G\|_{m(j):y}) = \Psi(F\|_{m(i):x}, G\|_{m(j):y})$, and let M be an optimal edit mapping between $F\|_{m(i):x} \ominus C$ and $G\|_{m(j):y}$. Consider nodes x and y and the set C:

- $x \in C$: In this case, x is cut and so the whole subtree $F[x]$ is removed at no cost. We get $\Psi(F\|_{m(i):x}, G\|_{m(j):y}) = \Psi(F\|_{m(i):m(x)-1}, G\|_{m(j):y})$.
- $x \notin C$: In this case, x is either deleted or linked with a node in $G\|_{m(j):y}$, and analogously for node y. There are three possible subcases:
  - $x \notin M_F$: Then x is deleted from $F\|_{m(i):x}$ in the optimal solution given by M, so $\Psi(F\|_{m(i):x}, G\|_{m(j):y}) = \Psi(F\|_{m(i):x-1}, G\|_{m(j):y}) + \gamma(f(x), -)$.
  - $y \notin M_G$: Then y is deleted from $G\|_{m(j):y}$ in the optimal solution given by M, so $\Psi(F\|_{m(i):x}, G\|_{m(j):y}) = \Psi(F\|_{m(i):x}, G\|_{m(j):y-1}) + \gamma(-, g(y))$.
  - $x \in M_F$ and $y \in M_G$: Then nodes x and y are linked in the optimal solution given by M. We get $\Psi(F\|_{m(i):x}, G\|_{m(j):y}) = \Psi(F\|_{m(i):m(x)-1}, G\|_{m(j):m(y)-1}) + \Psi(F\|_{m(x):x-1}, G\|_{m(y):y-1}) + \gamma(f(x), g(y))$. □

To simplify the implementation of the algorithm described below, rewrite the recurrence relation in Lemma 3 as follows so that $\Psi(F\|_{m(i):x}, G\|_{m(j):y})$ can be computed without needing to access the value of $\Psi(F\|_{m(x):x-1}, G\|_{m(y):y-1})$.

Lemma 4. *For any $i \in F$, $j \in G$, $x \in \{m(i), \ldots, i\}$, and $y \in \{m(j), \ldots, j\}$,*
1. if $m(i) = m(x)$ and $m(j) = m(y)$ then:

$$\Psi(F\|_{m(i):x}, G\|_{m(j):y}) = \min \begin{cases} \Psi(F\|_{m(i):m(x)-1}, G\|_{m(j):y}); \\ \Psi(F\|_{m(i):x-1}, G\|_{m(j):y}) + \gamma(f(x), -); \\ \Psi(F\|_{m(i):x}, G\|_{m(j):y-1}) + \gamma(-, g(y)); \\ \Psi(F\|_{m(i):x-1}, G\|_{m(j):y-1}) + \gamma(f(x), g(y)). \end{cases}$$

2. else:

$$\Psi(F\|_{m(i):x}, G\|_{m(j):y}) = \min \begin{cases} \Psi(F\|_{m(i):m(x)-1}, G\|_{m(j):y}); \\ \Psi(F\|_{m(i):x-1}, G\|_{m(j):y}) + \gamma(f(x), -); \\ \Psi(F\|_{m(i):x}, G\|_{m(j):y-1}) + \gamma(-, g(y)); \\ \Psi(F\|_{m(i):m(x)-1}, G\|_{m(j):m(y)-1}) + \Psi(F[x], G[y]). \end{cases}$$

Proof. We prove this lemma by showing that the new recurrences are equivalent to the one in Lemma 3. Note that in both cases, only the fourth term inside the min-bracket differs from Lemma 3.

1. Since $m(i) = m(x)$ and $m(j) = m(y)$, we have $\{m(i), \ldots, m(x) - 1\} = \emptyset$ and $\{m(j), \ldots, m(y) - 1\} = \emptyset$, and hence $\Psi(F\|_{m(i):m(x)-1}, G\|_{m(j):m(y)-1}) = 0$.
2. The definition of Ψ implies that $\Psi(F\|_{m(i):x}, G\|_{m(j):y}) \leq \Psi(F\|_{m(i):m(x)-1}, G\|_{m(j):m(y)-1}) + \Psi(F[x], G[y])$. Thus, inserting the right-hand side of this inequality into the min-expression in Lemma 3 does not affect its value, i.e., $\Psi(F\|_{m(i):x}, G\|_{m(j):y}) = \min\{\ldots, \Psi(F\|_{m(i):m(x)-1}, G\|_{m(j):m(y)-1}) + \Psi(F[x], G[y])\}$ where $\ldots$ denotes the four terms in Lemma 3. Now, by case 1 above, $\Psi(F[x], G[y]) \leq \Psi(F\|_{m(x):x-1}, G\|_{m(y):y-1}) + \gamma(f(x), g(y))$, so the fourth term in the new min-expression is redundant and can be deleted. □

Main loop:
Input: A target forest F and a pattern forest G.
1: $\Psi(\emptyset, \emptyset) := 0$.
2: **for** $i_1 := 1, \ldots, |K(F)|\}$ **do**
3: **for** $j_1 := 1, \ldots, |K(G)|$ **do**
4: $i := K(F)[i_1]$; $j := K(G)[j_1]$; Call `Compute_Psi`(i, j).
5: **return** $\min_{i \in F} \Psi(F[i], G)$.

Procedure `Compute_Psi`(i, j)**:**
1: **for** $x := m(i), \ldots, i$ **do** $\Psi(F\|_{m(i):x}, \emptyset) := 0$.
2: **for** $y := m(j), \ldots, j$ **do** $\Psi(\emptyset, G\|_{m(j):y}) := \Psi(\emptyset, G\|_{m(j):y-1}) + \gamma(-, g(y))$.
3: **for** $x := m(i), \ldots, i$ **do**
4: **for** $y := m(j), \ldots, j$ **do**
5: Calculate $\Psi(F\|_{m(i):x}, G\|_{m(j):y})$ according to Lemma 4.

Algorithm 1. Algorithm for finding a most similar simple substructure of F to G

For each $i \in F$, define $A(i)$ (*the nearest key node ancestor of* i) as follows. If $i \in K(F)$ then let $A(i) = i$; otherwise, let $A(i)$ be the nearest ancestor of i which belongs to $K(F)$. Define $A(j)$ for any node $j \in G$ analogously. We have:

Lemma 5. *For any* $i \in F$, $m(i) = m(A(i))$. *For any* $j \in G$, $m(j) = m(A(j))$.

We now describe the main algorithm (Algorithm 1) of this subsection. It calculates the minimum cost of an edit mapping between a simple substructure of F and G. The main loop considers all pairs of indices $i \in K(F)$ and $j \in K(G)$ in bottom-up order, and for each such pair of indices (i, j), it calls a procedure named `Compute_Psi` to obtain $\Psi(F\|_{m(i):x}, G\|_{m(j):y})$ for all $x \in \{m(i), \ldots, i\}$ and $y \in \{m(j), \ldots, j\}$ based on Lemmas 2 and 4. Finally, Algorithm 1 returns $\min_{i \in F} \Psi(F[i], G)$. The next theorem proves the correctness of this approach.

Theorem 1. *Algorithm 1 correctly computes the cost of an optimal solution.*

Proof. The correctness of all computed values follows from Lemmas 2 and 4. Let x be a node in F such that the cost of an optimal solution is given by $\Psi(F[x], G)$. We need to prove that the algorithm is guaranteed to compute $\Psi(F[x], G)$ even if $x \notin K(F)$. Observe that $\Psi(F[x], G) = \Psi(F\|_{m(x):x}, G) = \Psi(F\|_{m(A(x)):x}, G)$ by Lemma 5, and that $x \in \{m(A(x)), \ldots, A(x)\}$ because $m(A(x)) = m(x) \leq x$ and $x \leq A(x)$. Since $A(x) \in K(F)$ by the definition of $A(x)$ and since $p(G) \in K(G)$, the algorithm will always compute $\Psi(F[x], G)$. □

Theorem 2. *Algorithm 1 can be implemented to run in* $O(|F| \cdot |G| \cdot \min\{|L(F)|, dp(F)\} \cdot \min\{|L(G)|, dp(G)\})$ *time and* $O(|F| \cdot |G|)$ *space.*

Proof. Store $\Psi(F[i], G[j])$ for every $i \in F$ and $j \in G$ as soon as it is computed in a table M_1 of size $|F| \cdot |G|$. Also, allocate $(|F|+1) \cdot (|G|+1)$ additional space M_2 for `Compute_Psi` to temporarily store the computed values of $\Psi(F\|_{m(i):x}, G\|_{m(j):y})$ for all $x \in \{m(i) - 1, \ldots, i\}$ and $y \in \{m(j) - 1, \ldots, j\}$ for its current (i, j).

(M_2 is reused by successive calls to `Compute_Psi`.) In total, the space complexity is $O(|F| \cdot |G|)$.

To analyze the running time of this implementation, we first show that Step 5 in `Compute_Psi` (evaluating the expression in Lemma 4) for any $\Psi(F\|_{m(i):x}, G\|_{m(j):y})$ always takes $O(1)$ time. Whenever Step 5 is performed, the values of $\Psi(F\|_{m(i):m(x)-1}, G\|_{m(j):y})$, $\Psi(F\|_{m(i):x-1}, G\|_{m(j):y})$, $\Psi(F\|_{m(i):x}, G\|_{m(j):y-1})$, $\Psi(F\|_{m(i):x-1}, G\|_{m(j):y-1})$, and $\Psi(F\|_{m(i):m(x)-1}, G\|_{m(j):m(y)-1})$ are already stored in M_2. Therefore, we can directly evaluate the expression in $O(1)$ time if $m(i) = m(x)$ and $m(j) = m(y)$. If $m(i) < m(x)$ or $m(j) < m(y)$, we also need $\Psi(F[x], G[y])$ in $O(1)$ time. This value has already been computed and stored in M_1 because $m(i) < m(x)$ implies $m(i) < m(A(x))$ by Lemma 5 which in turn implies $A(x) < i$, and $m(j) < m(y)$ similarly implies $A(y) < j$; since at least one of these two conditions is true, the algorithm will have called `Compute_Psi`$(A(x), A(y))$ previously and hence already have computed $\Psi(F[x], G[y])$. Thus, Step 5 in `Compute_Psi` takes $O(1)$ time, which means that the algorithm's total running time is $O(\sum_{i \in K(F)} \sum_{j \in K(G)} |F[i]| \cdot |G[j]|)$. By Lemma 7 in [16], this sum can be rewritten as $O(|F| \cdot |G| \cdot \min\{|L(F)|, dp(F)\} \cdot \min\{|L(G)|, dp(G)\})$. The theorem follows. □

We remark that Algorithm 1 computes the *cost* of an optimal solution. Standard traceback techniques can be applied to also return a corresponding optimal edit mapping within the same asymptotic running time and space bounds.

3.2 An Algorithm for Finding a Most Similar Sibling Substructure

An algorithm for finding a most similar sibling substructure of F to G is given here. It is based on Algorithm 1 since finding a most similar sibling substructure is closely related to finding a most similar simple substructure, as shown next.

Suppose that $F' = \langle T_1, \ldots, T_s \rangle$ is a most similar sibling substructure of F to G, where $\{T_1, \ldots, T_s\}$ are simple substructures of F with roots $\{i_1, \ldots, i_s\}$, and where F' is non-empty. Then $\{i_1, \ldots, i_s\}$ are siblings in F. Let $S(i_1)$ be the set consisting of i_1 and all siblings of i_1 in F, and define $C = S(i_1) \setminus \{i_1, \ldots, i_s\}$. Note that C is a consistent set, i.e., $C \in \mathcal{C}(F)$, using the notation from Section 3.1. Consider the closed subforest $F\|_{m(b(i_1)):e(i_1)}$. It is clear that F' is also a most similar sibling substructure of $F\|_{m(b(i_1)):e(i_1)}$ to G. (This claim comes from cutting all nodes belonging to C from $F\|_{m(b(i_1)):e(i_1)}$ at zero cost.) So $\delta(F', G) = \Psi(F\|_{m(b(i_1)):e(i_1)}, G)$, and the problem turns into finding the minimum of $\Psi(F\|_{m(b(i_1)):e(i_1)}, G)$ over all $i_1 \in F$. By the left-to-right postordering of the nodes, this is equivalent to computing $\min_{i \in F} \Psi(F\|_{m(i):i-1}, G)$. [2]

We modify the implementation of Algorithm 1 given in the proof of Theorem 2 as follows. Allocate $O(|F|)$ extra space M_3 to also store the values of $\Psi(F\|_{m(i):i-1}, G)$ for all $i \in F$ as they are computed. Then, change Step 5 of the main loop to return the value $\min_{i \in F} \Psi(F\|_{m(i):i-1}, G)$ (found by checking M_3) instead. Clearly, the asymptotic time and space complexities are the same as for

[2] In contrast, recall from Section 3.1 that finding a most similar *simple* substructure is equivalent to computing $\min_{i \in F} \Psi(F\|_{m(i):i}, G)$.

Algorithm 1. To prove the correctness of the modified algorithm, we show that all values of $\Psi(F\|_{m(i):i-1}, G)$ for $i \in F$ are indeed computed. Let i be any node in F. Recall that $A(i)$ is the nearest key node ancestor of i. Then $m(i) = m(A(i))$ by Lemma 5, and $i \leq A(i)$ and $A(i) \in K(F)$. According to the proof of Theorem 2, for any $k \in K(F)$, the algorithm will compute $\Psi(F\|_{m(k):x}, G)$ for all $x \in \{m(k), \ldots, k\}$. Now, select $k = A(i)$ and $x = i - 1$. We obtain:

Theorem 3. *Given a target forest F and a pattern forest G, we can find a most similar sibling substructure of F to G over all sibling substructures of F in $O(|F|\cdot|G|\cdot\min\{|L(F)|, dp(F)\}\cdot\min\{|L(G)|, dp(G)\})$ time and $O(|F|\cdot|G|)$ space.*

3.3 An Algorithm for Finding a Most Similar Closed Subforest

We now provide an algorithm for finding a most similar closed subforest of F to G. The algorithm of [16] as well as Algorithm 1 from Section 3.1 are not suitable for this variant of the problem because if i_1 and i_2 are siblings in F with $i_1 < i_2$, $i_1 \in K(F)$ then the value of $\delta(F\|_{m(i_1):i_2}, G)$ is not calculated, whereas $F\|_{m(i_1):i_2} = F[i_1 \cdots i_2]$ might in fact be a most similar closed subforest of F to G. Therefore, we develop a different technique in this subsection. The proofs of Lemma 6, Lemma 7, and Theorem 4 below are similar to those of Lemma 2, Lemmas 3–4, and Theorems 1– 2, respectively, and have been omitted due to space constraints.

For any forest F, any leaf $l \in L(F)$, and any node $x \in F$, write $l \preceq x$ if l is a descendant of x in F, and $l \not\preceq x$ otherwise. Since $m(x)$ is precomputed, we can immediately test if $l \preceq x$ simply by checking if $m(x) \leq l \leq x$ is true. The next lemmas state how to efficiently calculate $\delta(F\|_{l:x}, G\|_{m(j):y})$ where $l \in L(F)$, $x \in \{l, \ldots, |F|\}$, $j \in K(G)$, and $y \in \{m(j), \ldots, j\}$.

Lemma 6. $\delta(\emptyset, \emptyset) = 0$; $\quad \delta(F, \emptyset) = \sum_{i \in F} \gamma(f(i), -)$; $\quad \delta(\emptyset, G) = \sum_{j \in G} \gamma(-, g(j))$.

Lemma 7. *For any $l \in L(F)$, $j \in K(G)$, $x \in \{l, \ldots, |F|\}$, $y \in \{m(j), \ldots, j\}$, $\delta(F\|_{l:x}, G\|_{m(j):y})$ is equal to the minimum of the following three values:*

$$
\begin{cases}
\delta(F\|_{l:x-1}, G\|_{m(j):y}) + \gamma(f(x), -); \\
\delta(F\|_{l:x}, G\|_{m(j):y-1}) + \gamma(-, g(y)); \\
\begin{cases}
\delta(F\|_{l:x-1}, G\|_{m(j):y-1}) + \gamma(f(x), g(y)), & \text{if } l \preceq x \text{ and } m(j) \preceq y; \\
\delta(\emptyset, G\|_{m(j):m(y)-1}) + \delta(F\|_{l:x}, G[y]), & \text{if } l \preceq x \text{ and } m(j) \not\preceq y; \\
\delta(F\|_{l:m(x)-1}, \emptyset) + \delta(F[x], G\|_{m(j):y}), & \text{if } l \not\preceq x \text{ and } m(j) \preceq y; \\
\delta(F\|_{l:m(x)-1}, G\|_{m(j):m(y)-1}) + \delta(F[x], G[y]), & \text{if } l \not\preceq x \text{ and } m(j) \not\preceq y.
\end{cases}
\end{cases}
$$

Now we are ready to describe Algorithm 2 for finding a most similar closed subforest of F to G. Its overall structure resembles that of Algorithm 1. For each leaf $l \in L(F)$ and node $j \in K(G)$, it calls a procedure named `Compute_Delta` which uses Lemma 7 to calculate $\delta(F\|_{l:x}, G\|_{m(j):y})$ for all $x \in \{l, \ldots, |F|\}$ and $y \in \{m(j), \ldots, j\}$. To enable each evaluation of Lemma 7 to be performed in $O(1)$ time, the algorithm temporarily stores the computed values of $\delta(F\|_{l:x}, G\|_{m(j):y})$ for all $x \in \{l, \ldots, |F|\}$ and $y \in \{m(j), \ldots, j\}$ until the next call to `Compute_Delta` using $O(|F| \cdot |G|)$ space; on the other hand, all computed values of the form

Main loop:
Input: A target forest F and a pattern forest G.
1: $\delta(\emptyset, \emptyset) := 0$.
2: **for** $l_1 := |L(F)|, \ldots, 1$ **do**
3: **for** $j_1 := 1, \ldots, |K(G)|$ **do**
4: $l := L(F)[l_1]$; $j := K(G)[j_1]$; Call `Compute_Delta`(l, j).
5: **return** $\min\{\delta(F\|_{m(i_1):i_2}, G) \mid i_1, i_2$ are siblings in $F\}$.

Procedure `Compute_Delta`(l, j):
1: **for** $x := l, \ldots, |F|$ **do** $\delta(F\|_{l:x}, \emptyset) := \delta(F\|_{l:x-1}, \emptyset) + \gamma(f(x), -)$.
2: **for** $y := m(j), \ldots, j$ **do** $\delta(\emptyset, G\|_{m(j):y}) := \delta(\emptyset, G\|_{m(j):y-1}) + \gamma(-, g(y))$.
3: **for** $x := l, \ldots, |F|$ **do**
4: **for** $y := m(j), \ldots, j$ **do**
5: Calculate $\delta(F\|_{l:x}, G\|_{m(j):y})$ according to Lemma 7.

Algorithm 2. Algorithm for finding a most similar closed subforest of F to G

$\delta(F[x], G[y])$, $\delta(F[x], G\|_{m(j):y})$ with $m(j) \preceq y$, and $\delta(F\|_{l:x}, G[y])$ with $l \preceq x$ are stored throughout the entire execution of the algorithm using an additional $O(|F|\cdot|G| + |F|\cdot|K(G)|\cdot dp(G) + |L(F)|\cdot dp(F)\cdot|G|)$ space. Finally, the algorithm returns $\min\{\delta(F\|_{m(i_1):i_2}, G) \mid i_1$ and i_2 are siblings in $F\}$.

Theorem 4. *Given a target forest F and a pattern forest G, we can find a most similar closed subforest of F to G over all closed subforests of F in $O(|F|\cdot|G|\cdot|L(F)|\cdot\min\{|L(G)|, dp(G)\})$ time and $O(|F|\cdot|G| + |L(F)|\cdot dp(F)\cdot|G| + |F|\cdot|L(G)|\cdot dp(G))$ space.*

4 Concluding Remarks

It is straightforward to generalize our algorithms to find a subforest F' of F and a subforest G' of G that are the most similar for any combination of the types of subforests considered above. For example, if both F' and G' should be simple substructures then we can modify Algorithm 1 to allow nodes in G to be cut too.

An open question is: Is it possible to extend the algorithms in this paper to other types of subforests? For example, one might consider *gapped subforests* (introduced in [6]), where a gapped subforest of F is obtained by removing from any closed subforest F' of F a set C of closed subforests such that no two closed subforests in C have the same parent in F'.

References

1. W. Chen. New algorithm for ordered tree-to-tree correction problem. *Journal of Algorithms*, 40(2):135–158, 2001.
2. G. Cobéna, S. Abiteboul, and A. Marian. Detecting changes in XML documents. In *Proceedings of the 18th IEEE International Conference on Data Engineering* (ICDE 2002), pages 41–52, 2002.

3. M. Crochemore and W. Rytter. *Text algorithms*. Oxford University Press, 1994.
4. M. Höchsmann, T. Töller, R. Giegerich, and S. Kurtz. Local similarity in RNA secondary structures. In *Proceedings of the IEEE Computational Systems Bioinformatics Conference* (CSB 2003), pages 159–168, 2003.
5. J. Jansson and A. Lingas. A fast algorithm for optimal alignment between similar ordered trees. *Fundamenta Informaticae*, 56(1–2):105–120, 2003.
6. J. Jansson, T. H. Ngo, and W.-K. Sung. Local gapped subforest alignment and its application in finding RNA structural motifs. In *Proceedings of the 15th International Symposium on Algorithms and Computation* (ISAAC 2004), pages 569–580, 2004.
7. T. Jiang, L. Wang, and K. Zhang. Alignment of trees - an alternative to tree edit. *Theoretical Computer Science*, 143:137–148, 1995.
8. P. Kilpeläinen and H. Mannila. Ordered and unordered tree inclusion. *SIAM Journal on Computing*, 24(2):340–356, 1995.
9. P. N. Klein. Computing the edit-distance between unrooted ordered trees. In *Proceedings of the 6th European Symposium on Algorithms* (ESA 1998), pages 91–102, 1998.
10. Motifs database. `http://subviral.med.uottawa.ca/cgi-bin/motifs.cgi`.
11. B. A. Shapiro and K. Zhang. Comparing multiple RNA secondary structures using tree comparisons. *Computer Applications in the Biosciences*, 6(4):309–318, 1990.
12. T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.
13. K.-C. Tai. The tree-to-tree correction problem. *Journal of the ACM*, 26(3):422–433, 1979.
14. H. Touzet. A linear time edit distance algorithm for similar ordered trees. In *Proceedings of the 16th Symposium on Combinatorial Pattern Matching* (CPM 2005), pages 334–345, 2005.
15. G. Valiente. Constrained tree inclusion. In *Proceedings of the 14th Symposium on Combinatorial Pattern Matching* (CPM 2003), pages 361–371, 2003.
16. K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):1245–1262, 1989.