August 2014

Efficient algorithms for local forest similarity and forest pattern matching

Fang Han
The University of Western Ontario

Supervisor
Kaizhong Zhang
The University of Western Ontario

Graduate Program in Computer Science

A thesis submitted in partial fulfillment of the requirements for the degree in Master of Science

EFFICIENT ALGORITHMS FOR LOCAL FOREST SIMILARITY AND FOREST PATTERN MATCHING

(Thesis format: Monograph)

by

Fang Han

Graduate Program in Computer Science

A thesis submitted in partial fulfillment
of the requirements for the degree of
Masters of Science

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

Abstract

Ordered labelled trees are trees where each node has a label and the left-to-right order among siblings is significant. Ordered labelled forests are sequences of ordered labelled trees. Ordered labelled trees and forests are useful structures for hierarchical data representation.

Given two ordered labelled forests F and G, the local forest similarity is to compute two sub-forests F' and G' of F and G respectively such that they are the most similar over all the possible F' and G'. Given a target forest F and a pattern forest G, the forest pattern matching problem is to compute a sub-forest F' of F which is the most similar to G over all the possible F'.

This thesis presents novel efficient algorithms for the local forest similarity problem and forest pattern matching problem for sub-forest. An application of the algorithms is that it can be used to locate the structural regions in RNA secondary structures which is the important data in RNA secondary structure comparison and function investigation. RNA is a chain of nucleotide, mathematically it is a string over a four letter alphabet; in computational molecular biology, ordered labelled trees are used to represent RNA secondary structures.

Keywords: trees , sub-trees , forests, sub-forests, forest edit similarity, forest removing similarity, local forest similarity, forest pattern matching, RNA secondary structure comparison

Acknowlegements

I would like to express sincere thanks to my supervisor Dr. Kaizhong Zhang who offered me the opportunity to work on this interesting project. His great enthusiasm and unusual patience trained me both in critical research attitude and strict scientific method; without his meticulous supervision and sound guidance, I could not make this research through to its completion. Also importantly, I'm grateful to his financial support; otherwise, it might have been impossible to pursue my studies.

I wish to acknowledge Zhewei Liang, who made valuable suggestions and contributed precious time. My great appreciation goes to his critical opinions to help me work out the thesis proposal.

I sincerely appreciate my girlfriend, Qin Dong, for her encouragements and understanding. I owe many thanks to my colleagues for their numerous encouragements and help whenever various difficulties seemed overwhelming. I would also like to express my sincere thanks to the people and staff around for their friendship to make my stay pleasant in the department.

Last but not least, I am deeply indebted to my parents for their love and understanding; you always deserve more than I could give you.

Contents

List of Figures

Abbreviation

FPM	Forest Pattern Matching
LFS	Local Forest Similarity
FES	Forest Edit Similarity
FRS	Forest Removing Similarity
LSS	Local Sequence Similarity
LCSS	Longest Common Sub-Sequence

Chapter 1

Introduction

1.1 Motivation and research objective

Ordered labelled trees are rooted trees where each node has a label and the left-to-right order among siblings is significant. An ordered labelled forest is a sequence of ordered labelled trees. Ordered labelled trees and forests are used to represent hierarchically structured information in many areas such as image processing, pattern recognition, natural language and structured document management [1, 21, 6, 18]. It is quite often a necessity to measure the similarity between two or more such trees and forests to determine the similarity between them and identify parts of the trees and forests that are similar.

Throughout this thesis, we refer to ordered labelled trees and ordered labelled forests as trees and forests respectively. Since trees and forests are useful object representations, the need for trees and forests comparison frequently arises. The RNA secondary structure comparison problem is among which a typical example. RNA is one of the major macromolecules essential for all known forms of life. In computational biology, RNA secondary structure can be represented as a tree or a forest. It is a challenging task to develop the efficient algorithms in order to reveal RNA secondary structure similarity and difference. The problem of RNA secondary structures comparison has attracted much attention in recent research.

The primary structure of a RNA molecule is a sequence of nucleotides and it folds back onto itself into a shape that is topologically a forest [1, 21, 6, 18], which we call the secondary structure. Figure 1.1 shows an example of the RNA GI:2347024 structure: (a) is a segment of the RNA sequence, (b) is its secondary structure and (c) is the corresponding tree representation. All the bonded pairs are treated as units and they are bold in (a), (b) and (c). Algorithms for the edit distance between two forests (trees) [17, 7] can be used to measure the global similarity of forests (trees).

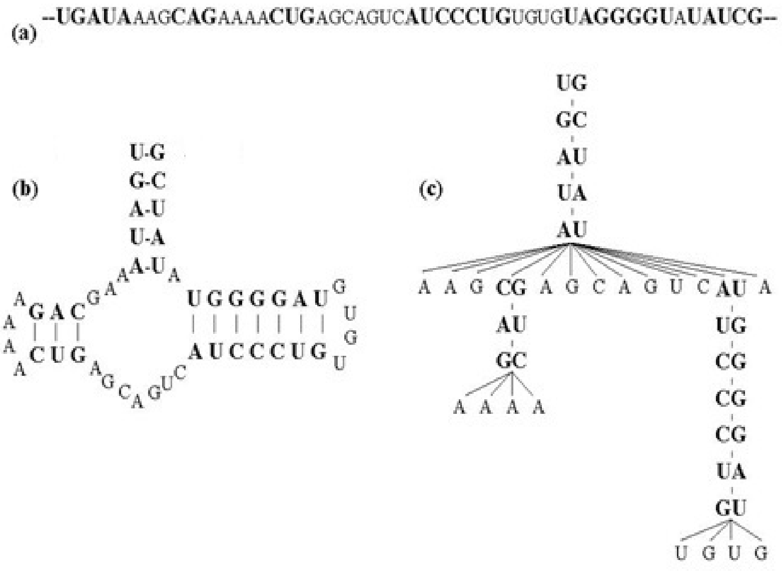


Figure 1.1: RNA structure and tree representation.[24]
(a) A segment of the RNA GI: 2347024 primary structure [9], (b) its secondary structure, (c) its tree representation.

1.2 Preliminaries and notation

We first present the definitions and notations used throughout the thesis.

Let T be a rooted ordered labelled tree of n nodes. Each node is labelled by a symbol from a given alphabet. Given any two nodes, they are either in an ancestor-descendant relationship, or in a left-right relationship. All the nodes in T are numbered with a left-to-right post-order numbering from 1 to n. The left most leaf will be numbered with one and the root will be numbered with n. Let F be an ordered labelled forest. It is defined as a sequence of ordered trees. Note that all the trees and forests mentioned in this thesis are ordered labelled trees and ordered labelled forests. A sub-tree of F is any connected sub-graph of F. Since each tree in F is a rooted tree, a sub-tree has a root. A sub-forest of F is a sub-sequence of sub-trees of F. The order of the roots of the sub-trees in F is consistent with the order of nodes in F. A complete sub-tree of F is a sub-tree consisting of a root and all of its descendants in F. A complete sub-forest of F is a sub-sequence of complete sub-trees of F. A sibling sub-forest is a sequence of sub-trees such that their roots are siblings. A closed sub-forest is a sequence of sub-trees such that their roots are consecutive siblings. Let $f[i]$ denotes the ith node in F. We also use $f[i]$ to represent the label of the ith node if there is no confusion. Let $l(i)$ denote the post-order number of the leftmost leaf descendant of $f[i]$. $F[i]$ denotes the complete sub-tree rooted at $f[i]$. $F[i..j]$ will generally be an ordered sub-forest of F in the post-order numbering, induced by the nodes numbered from i to j inclusively. D_F and L_F denote the depth and the number of leaves of F respectively. Let $|F|$ be the number of nodes in F. Let $subf(F)$ be the set of sub-forests of F. Let $subcf(F)$ be the set of complete sub-forests of F and $subcf(F, node\_set)$ be the set of complete sub-forests of F such that nodes in $node\_set$ are not in any of the sub-forests. $subcf(F)$ is actually equal to $subcf(F, \emptyset)$. Let $F \setminus f$ represent the sub-forest resulting from the deletion of sub-forest f from F.

As illustrated, Figure 1.2 shows the forest F in Figure 1.1(c) with these notations.

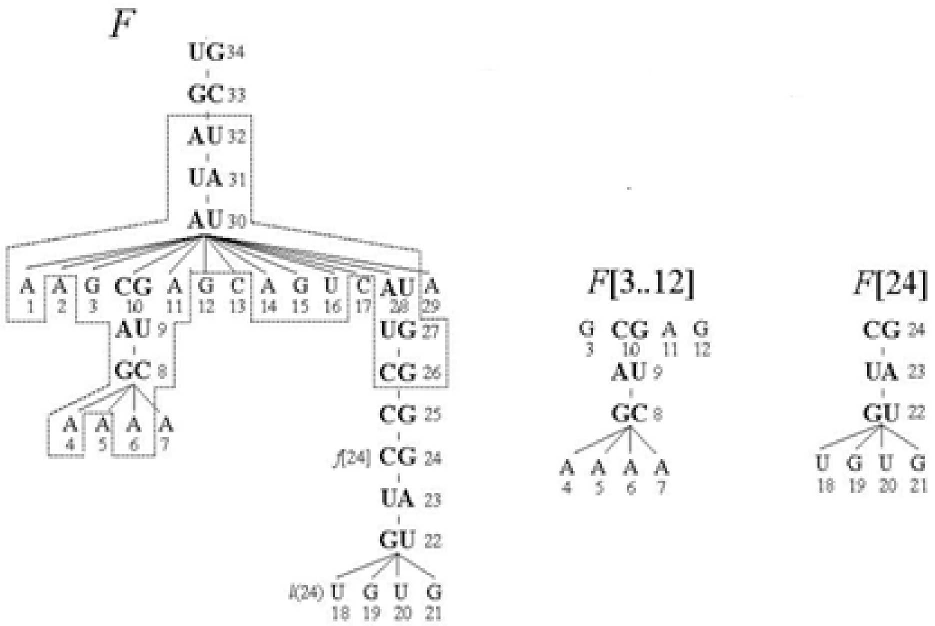


Figure 1.2: Forest F with notations. [10]
Examples of the forest F in Figure 1.1(c) and its sub-forests with notations.

1.3 Previous work

Given two trees F and G, the global tree edit distance problem is to compute the edit distance between them. Tai [3] presented an algorithm using $O(|F| \cdot |G| \cdot {D_F}^2 \cdot {D_G}^2)$ time. Later, Zhang and Shasha [7] developed an algorithm which improved the running time to $O(|F| \cdot |G| \cdot \min\{D_F, L_F\} \cdot \min\{D_G, L_G\})$. A better algorithm from Demaine *et al.* runs in $O(|F| \cdot |G|^2 \cdot (1 + \log \frac{|F|}{|G|}))$ time [17]. Both distance metric and similarity metric can be applied to these algorithms. These algorithms can also be used to solve forest distance problem.

Given a target forest F and a pattern forest G, the forest pattern matching(FPM) problem is to determine a sub-forest F' of F which is the most similar (having the maximum similarity score) to G over all the possible F'. Jansson and Peng [20] gave an algorithm for the special case of FPM for closed complete sub-forests that runs in

$O(|F| \cdot |G| \cdot L_F \cdot \min\{D_G, L_G\})$ time and $O(|F| \cdot |G| + L_F \cdot L_F \cdot |G| + |F| \cdot L_F L_G \cdot L_G D_G)$ space. Later, Zhang and Zhu [8] designed a more efficient algorithm which runs in $O(|F| \cdot |G| \cdot \min\{D_F, L_F\} \cdot \min\{D_G, L_G\})$ time and $O(|F| \cdot |G|)$ space.

Given two ordered labelled forests F and G, the local forest similarity problem is to determine a sub-forest F' and a G' of F and G respectively such that they are the most similar over all the possible F' and G'. Two special cases of this problem, when the sub-trees are a sibling sub-forest and closed sub-forest respectively, have been studied which are local forest similarity for sibling sub-forest and closed sub-forest. Liang [10] presented two efficient algorithms that run in $O(|F| \cdot |G| \cdot \min\{D_F, L_F\} \cdot \min\{D_G, L_G\})$ time and $O(|F| \cdot |G|)$ space. Peng [24] gave an algorithm for LFS on closed complete sub-forests using distance metrics that runs in $O(|F| \cdot |G| \cdot L_F \cdot D_G)$ time and $O(|F| \cdot |G|)$ space.

Both the distance metric [17, 7] and similarity metric [15, 10] can be used for the global similarity problem and pattern matching problem between two trees or two forests. However the distance metric is not suitable for the local similarity problem. For distance metrics, the goal is to compute the minimum score; so it is quite a possible scenario that the local similarity may end up with two identical leaves, one from each tree, matching and producing an optimal distance score zero. Therefore, the similarity metric defined in [15] is used for the local forest similarity algorithms to be investigated here.

1.4 Thesis organization

The organization of the thesis is as follows:

Chapter 2 introduces forest edit similarity (FES) which serves as the basis of the algorithms developed; the related essential concepts and definitions are also covered.

Chapter 3 presents the algorithm for the local forest similarity (LFS) problem for sub-forests which can be used to compute local similar regions between two forests. We first introduce the forest removing similarity (FRS) for sub-forest algorithm which computes the similarity scores between two sub-trees. Then, we introduce our algorithm to solve the LFS for sub-forest problem which is based on FRS for the sub-forest algorithm.

Chapter 4 presents the algorithm for forest pattern matching (FPM) problem for sub-forest. We first introduce the forest removing similarity (FRS) for forest pattern matching which can be used to compute the tree pattern matching scores between two sub-trees. Then, we introduce our algorithm to solve the LFS for sub-forest problem which is based on the FRS for forest pattern matching algorithm.

Chapter 5 depicts the implementation of our algorithms for the LFS problem and FPM problem. We also present some implementation by using RNA examples.

Chapter 6 contains conclusions and suggestions for future research.

Chapter 2

Tree Edit Distance and Similarity

Computing the distance or similarity between trees or forests under various measures has been accepted in the comparison of two forests or trees. Note that all the trees mentioned in this thesis are ordered labelled trees. Forest edit similarity uses both the similarity metric and forest edit mapping [10]; it serves as the basis to solve the forest removing similarity (FRS) problem, local forest similarity (LFS) and forest pattern matching (FPM).

2.1 Edit operations

Edit distance is based on edit operations on tree nodes. For an ordered labelled forest F, as Figure 2.1 shows, there are three edit operations:

(a) Change: to change one node label into another in F.

(b) Delete: to delete a non-root node l_2 from T with parent l_1, making the children of l_2 become the children of l_1. The children are inserted in the place of l_2 as a subsequence in the left-to-right order of the children of l_1. To delete a root node l_2 from F, the children of l_2 become roots of rooted trees.

(c) Insert: on the complement of deletion, inserting a node l_2 as a child of l_1 in F making l_2 the parent of a consecutive subsequence of the children of l_1.

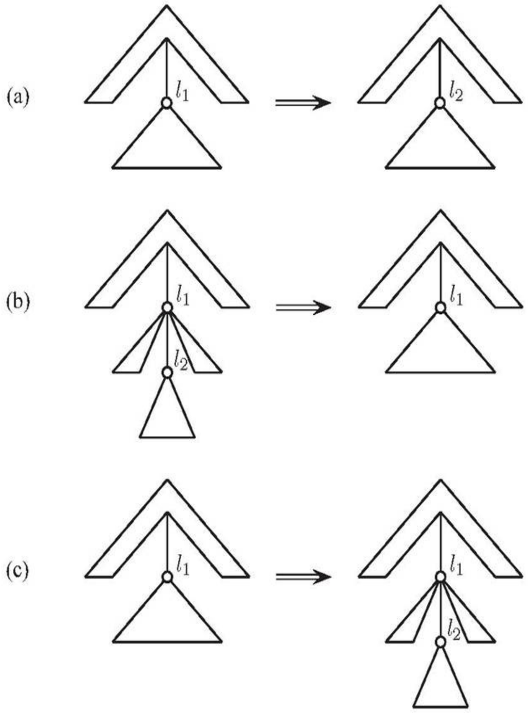


Figure 2.1: Forest edit operations [9].

We use Λ to present an empty label. An edit operation is a pair $(a, b) \neq (\Lambda, \Lambda)$ [3], where a is either Λ or a label of a node in forest F and b is either Λ or a label of a node in forest G; a change operation if $a \neq \Lambda$ and $b \neq \Lambda$; a delete operation if $b \neq \Lambda$; and an insert operation if $a \neq \Lambda$. Let $\sum'$ be a label alphabet and $\sum = \sum' \cup \Lambda$. Each edit operation is associated with a cost. We use a distance metric on $\sum$.

In mathematics, a distance metric on a set X is a function that a real-valued non-negative function d(a, b) on the Cartesian product X × X which satisfies the following conditions:

(1) d(a, b) $\geq$ 0, d(a, b) = 0 only if a = b;

(2) d(a, b) = d(b, a);

(3) d(a, b) $\leq$ d(b, c)+ d(a, c);

2.2 Edit distance

In this section, we introduce edit distance metric on trees and forests based on the distance metric on labels introduced in Section 2.1.

Edit distance is a way of quantifying how dissimilar two forests are by counting the minimum cost of a sequence of edit operations required to transform one into the other. If the costs of edit operations equal to 1, the distance between two ordered trees is considered to be the number of operations such as insert, delete and modify to transform one tree to another. If we want to know how to use a minimum cost to transform one tree to another, both cost and the sequence of operations have to be computed.

The tree edit distance problem is to compute the edit distance and a sequence of edit operations turning T_1 to T_2. The cost is the sum of the costs of the operations; an optimal edit distance between T_1 and T_2 is a sequence of operations between T_1 and T_2 of minimum cost.

2.3 Edit mappings and edit similarity

An edit mapping (or just a mapping) between T_1 and T_2 is a representation of the edit operations to transform T_1 to T_2, which is used in many of the algorithms for the tree edit distance problem. An edit mapping is a way to transform one tree to another. Given two forests F and G, let $f[i]$ and $g[j]$ be the nodes and corresponding labels in F and G, respectively. Note that both F and G have numbering. We use '$-$' in place of 'Λ' to represent an empty label.

Formally, define the M to be a mapping from F to G, where M is any set of pairs of integers (i, j) satisfying:

(1) $1 \leq \mathrm{i} \leq |\mathrm{F}|, 1 \leq \mathrm{j} \leq |\mathrm{G}|$;

(2) for any pair (i_1,j_1) and (i_2,j_2) in M,

(a) $i_1 = i_2$ if and only if $j_1 = j_2$ (one-to-one condition);

(b) f[i_1] is an ancestor of f[i_2] if and only if g[j_1] is an ancestor of g[j_2] (ancestor condition);

(c) f[i_1] is to the left of f[i_2] if and only if g[j_1] is to the left of g[j_2] (sibling condition).

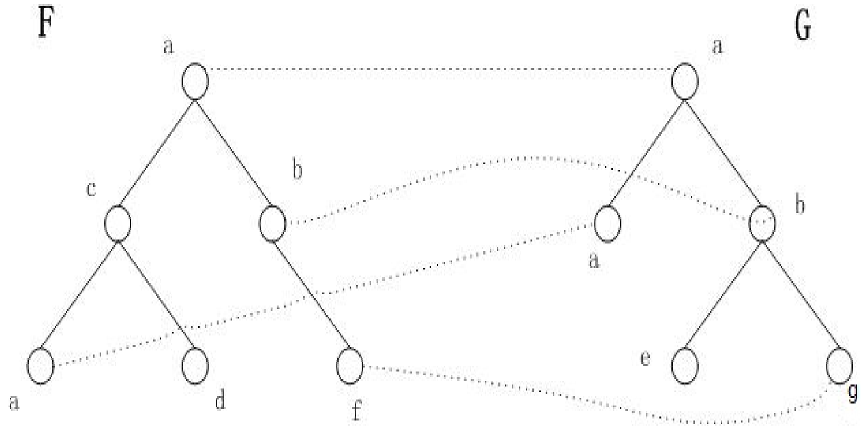


Figure 2.2: A forest edit mapping from F to G.

As shown in Figure 2.2, a mapping from F to G which consists of (1,1), (4,3), (5,4), (6,5). If a pair of nodes is in the mapping, they are connected by a dot line. If a node is not in the mapping(not touched by any line), it will be deleted or inserted. Later in this thesis, if a pair of nodes is in the mapping, we say they are touched by a line. Let I and J be the sets of nodes in F and G, respectively, not touched by any line in M. Then the distance D of an edit mapping M is given by:

$$\text{D(M)}= \sum_{(i,j)\in M} \text{d(f[i],g[j])} + \sum_{i'\in I} \text{d(f[i'],}-\text{)}+ \sum_{j'\in J} \text{d(}-\text{,g[j'])}, \text{ where}$$

$$\text{I}=\{ \text{ i} \mid \text{there is no j that (i,j)} \in \text{ M } \}$$

$$\text{J}=\{ \text{ j} \mid \text{there is no i that (i,j)} \in \text{ M } \}$$

From all possible mappings, we define minimum cost mapping between F and G as follows:

$$\sigma(F,\, G) = \min \{\ \mathrm{D(M)} \mid \mathrm{M\ is\ a\ mapping\ between\ F\ and\ G}\ \}.$$

As mentioned above, we used a distance metric in the mapping. A similarity metric also can be used in mapping. In this thesis, we use an edit similarity to replace the edit distance and we use similarity scores instead of distance costs. The similarity metric definition is described as: given a set A, a real-valued function s(a, b) on the Cartesian product A × A is a similarity score if, for any a, b, c $\in$ A, it satisfies the following conditions:

(1) s(a, a) $\geq$ 0;

(2) s(a, b)= s(b, a);

(3) s(a, a) $\geq$ s(a, b);

(4) s(a, b)+ s(b, c) $\leq$ s(b, b)+ s(a, c);

(5) s(a, a)= s(b, b)= s(a, b) if and only if a = b. [15]

Let the M to be a mapping from F to G. Let I and J be the sets of nodes in F and G, respectively, not touched by any line in M. Then the similarity S of an edit mapping M is given by:

$$\mathrm{S(M)}= \sum_{(i,j)\in M} \mathrm{s(f[i],g[j])} + \sum_{i'\in I}\mathrm{s(f[i'],-)}+ \sum_{j'\in J}\mathrm{s(-,g[j'])},\ \text{where}$$
$$\mathrm{I}=\{\ \mathrm{i} \mid \text{there is no j that (i,j)} \in \mathrm{M}\ \}$$
$$\mathrm{J}=\{\ \mathrm{j} \mid \text{there is no i that (i,j)} \in \mathrm{M}\ \}$$

The similarity metric can be extended to edit mapping from an alphabet. We use ϕ(F, G) to represent the edit similarity between two forests throughout this thesis. The definition of S making ϕ(F, G) a similarity metric between forests, is called a forest edit similarity. To compute the forest edit similarity means to compute the maximum score mapping which is denoted by ϕ(F, G).

$$\phi(\mathrm{F,\ G}) = \max \{\ \mathrm{S(M)} \mid \mathrm{M\ is\ a\ mapping\ between\ F\ and\ G}\ \}.$$

Based on the five properties listed above, s(a, a)≥0 and $\phi(F, F)$≥0. We usually set s(-,-)=0 because there would be neither bonus nor penalty when mapping two nodes with empty labels and set ϕ(∅, ∅)=0. So that s(a, -)≤0 and ϕ(F, ∅)≤0.[10]

2.4 Edit distance and edit similarity

The ordered edit distance metric was introduced by Tai [3]. Given two forests F and G, Tai presented an algorithm for the forest edit distance problem using $O(|F| \cdot |G| \cdot {D_F}^2 \cdot {D_G}^2)$ time and $O(|F| \cdot |G|)$ space by using pre-order numbering. Zhang and Shasha then presented a much simpler algorithm [7] which improved the running time to $O(|F| \cdot |G| \cdot \min\{D_F, L_F\} \cdot \min\{D_G, L_G\})$ by using post order numbering. In this thesis, we use edit similarity with some modifications of edit distance.

Lemma 2.1 *Let F, G be defined as above, l(i) is the leftmost descendent node of F(i)*

(1) $\phi(\emptyset, \emptyset) = 0$

(2) $\forall i \in F, \forall i' \in \{l(i), .., i\}$,
$\phi(F[l(i)..i'], \emptyset) = \phi(F[l(i)..i' - 1], \emptyset) + s(f[i'], -)$.

(3) $\forall j \in G, \forall j' \in \{l(j), .., j\}$,
$\phi(\emptyset, G[l(j)..j']) = \phi(\emptyset, G[l(j)..j' - 1]) + s(-, g[j'])$.

Proof This is directly from Lemma 3 in [7]. □

We consider mapping from $F[l(i)..i']$ to $G[l(j)..j']$ and that mapping must respect ancestor and sibling orders. If it includes a line from $f[i']$ to $g[j']$ and (i', j') is in the mapping, the mapping will consist of

(1) a mapping from the complete sub-tree $F[i']$ to the complete sub-tree $G[j']$.

(2) a mapping from the forest of F with the complete sub-tree $F[i']$ removed to the forest of G with the complete sub-tree $G[j']$ removed.

So we can have Lemma 2.2.

Lemma 2.2 *[7] $\forall(i,j)\in(F,G)$, $\forall i'\in\{l(i),\dots,i\}$ and $\forall j'\in\{l(j),\dots,j\}$,*

$$\phi(F[l(i)..i'],G[l(j)..j'])=\max\begin{cases}\phi(F[l(i)..i'-1],G[l(j)..j'])+s(f[i'],-),\\ \phi(F[l(i)..i'],G[l(j)..j'-1])+s(-,g[j']),\\ \phi(F[l(i)..l(i')-1],G[l(j)..l(j')-1])\\ \quad+\phi(F[l(i')..i'-1],G[l(j')..j'-1])\\ \quad+s(f[i'],g[j']).\end{cases}$$

Proof This is directly from Lemma 4 in [7]. □

To simplify the calculation of $\phi(F[l(i)..i'],G[l(j)..j'])$, consider two cases of $\phi(F[l(i)..i'],G[l(j)..j'])$

(1)$F[l(i)..i']$ and $G[l(j)..j']$ are both trees.

(2)$F[l(i)..i']$ or $G[l(j)..j']$ is a forest.

So that let i, j, F and G be defined as above, $l(i)\le i'\le i$ and $l(j)\le j'\le j$

$\forall(i,j)\in(F,G)$, $\forall i'\in\{l(i),\dots,i\}$ and $\forall j'\in\{l(j),\dots,j\}$.

If $l(i')=l(i)$ and $l(j')=l(j)$,

$$\phi(F[l(i)..i'],G[l(j)..j'])=\max\begin{cases}\phi(F[l(i)..i'-1],G[l(j)..j'])+s(f[i'],-),\\ \phi(F[l(i)..i'],G[l(j)..j'-1])+s(-,g[j']),\\ \phi(F[l(i)..i'-1],G[l(j)..j'-1])+s(f[i'],g[j']).\end{cases}$$

If $l(i')\neq l(i)$ and $l(j')\neq l(j)$,

$$\phi(F[l(i)..i'],G[l(j)..j'])=\max\begin{cases}\phi(F[l(i)..i'-1],G[l(j)..j'])+s(f[i'],-),\\ \phi(F[l(i)..i'],G[l(j)..j'-1])+s(-,g[j']),\\ \phi(F[l(i)..l(i')-1],G[l(j)..l(j')-1])+\phi(F[i'],G[j']).\end{cases}$$

This is directly from Lemma 5 in [7].

Based on Lemma 2.1 and Lemma 2.2, we have Algorithm 1 and Algorithms 3 shown below

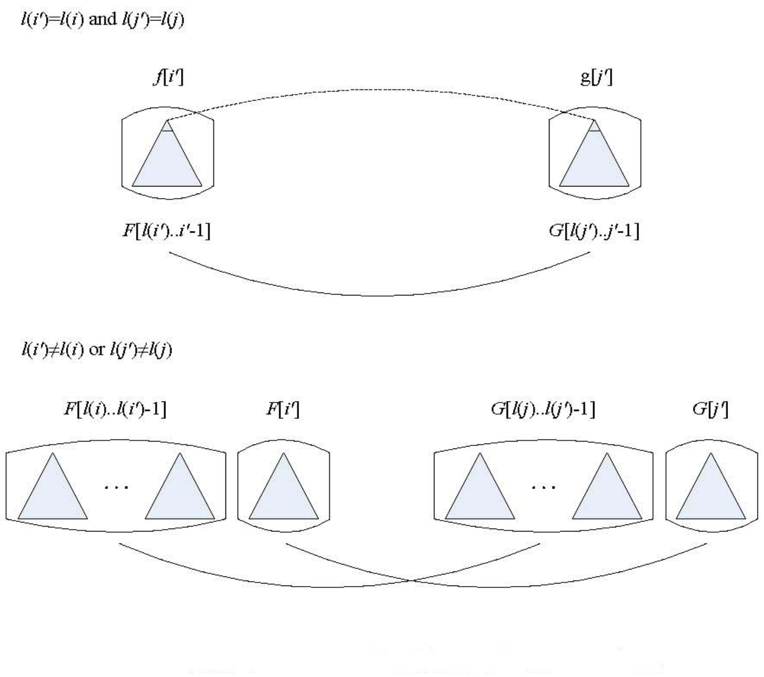


Figure 2.3: Lemma 2.2[7]

Algorithm 1: $TS(F, G)$

```
Input: Forest $F$ and $G$.
Output: $\phi(F[i], G[j])$, where $1 \leq i \leq |F|$ and $1 \leq j \leq |G|$.
for $i := 1$ to $|F|$ do
    for $j := 1$ to $|G|$ do
        $TreeSimilarity(i, j)$
    end
end
```

Zhang-Shasha's algorithm also used the concept of key root [23]. Define k is a key root node and the key root node set K of forest F as follows:

$$K(F) = \{k | \text{there exists no } k' > k \text{ such that } l(k) = l(k')\}.$$

That is, if k is in K then either k is the root of F or k has a left sibling. Intuitively, this set will be the roots of all the sub-trees of tree F that need separate computations. In array K, the order of the elements is in increasing order. With the introduction of the key roots [7] concept, we only need to compute $TreeSimilarity(i, j)$ with $i \in K(F)$ and $j \in K(G)$, instead of all i and j.

Algorithm 2: *TreeSimilarity*(i, j)

```
$\phi(\emptyset, \emptyset) = 0$
for $i' := l(i)$ to $i$ do
    $\phi(F[l(i)..i'], \emptyset) = \phi(F[l(i)..i'-1], \emptyset) + s(f[i'], -)$
end
for $j' := l(j)$ to $j$ do
    $\phi(\emptyset, G[l(j)..j']) = \phi(\emptyset, G[l(j)..j'-1]) + s(-, g[j'])$
end
for $i' := l(i)$ to $i$ do
    for $j' := l(j)$ to $j$ do
        if $l(i') = l(i)$ and $l(j') = l(j)$ then
            $\phi(F[l(i)..i'], G[l(j)..j']) = \max \begin{cases} \phi(F[l(i)..i'-1], G[l(j)..j']) + s(f[i'], -), \\ \phi(F[l(i)..i'], G[l(j)..j'-1]) + s(-, g[j']), \\ \phi(F[l(i)..i'-1], G[l(j)..j'-1]) + s(f[i'], g[j']) \end{cases}$
            $\phi(F[i'], G[j']) = \phi(F[l(i)..i'], G[l(j)..j'])$
        else
            $\phi(F[l(i)..i'], G[l(j)..j']) = \max \begin{cases} \phi(F[l(i)..i'-1], G[l(j)..j']) + s(f[i'], -), \\ \phi(F[l(i)..i'], G[l(j)..j'-1]) + s(-, g[j']), \\ \phi(F[l(i)..l(i')-1], G[l(j)..l(j')-1]) + \phi(F[i'], G[j']) \end{cases}$
        end
    end
end
```

Algorithm 3: $SZS(F, G)$

```
Input: Forest $F$ and $G$.
Output: $\phi(F[i], G[j])$, where $1 \leq i \leq |F|$ and $1 \leq j \leq |G|$.
compute $K(F)$ and $K(G)$ and sort them in increasing order into arrays $K_1$ and $K_2$, respectively
for $i' := 1$ to $|K_1|$ do
    for $j' := 1$ to $|K_2|$ do
        $i := K_1[i']$
        $j := K_2[j']$
        $TreeSimilarity(i, j)$
    end
end
```

Chapter 3

Local Forest Similarity For Sub-forests

Local forest similarity (LFS) aims at computing locally similar regions in two forests. In this Chapter, we consider the LFS problem for forests which are sequences of sub-trees. The roots of the sub-trees are nodes of the forests. Given two ordered labelled forests F and G, the local forest similarity problem is to determine sub-forests F' and G' of F and G respectively such that they are the most similar over the all possible F' and G'.

The algorithm of LFS is to compute the maximum score of the following formula,

$$\begin{gathered} LFS(F,G){=}\max_{F',G'}\phi(F',G') \\ F' \in subf(F) \\ G' \in subf(G). \end{gathered}$$

3.1 The algorithm for local sequence similarity and longest common sub-sequence

Sequences are a special case of trees/forests. A sequence can be considered as a linear tree or a forest so that all the trees in it have only one node. For sequence, the LFS problem becomes a local sequence similarity problem when we consider it as a linear tree and weighted longest common sub-sequence problem when we consider it as a forest of singleton trees.

Local sequence similarity (LSS) problem [2] is to determine two sub-sequences from two given sequences such that they are most similar. Figure 3.1 shows the local sequence alignment.

The problem is defined below.

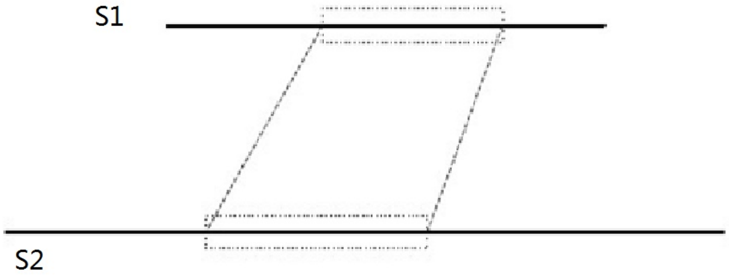


Figure 3.1: Local sequence similarity alignment.

Given two sequences $S_1[1..m]$, $S_2[1..n]$, SMS computes the maximum score using the following the formula

$$SMS(S_1, S_2){=}\max\{\phi(S_1[i..j], S_2[k..l])|1 \leq i \leq j \leq m, 1 \leq k \leq l \leq n\}$$

Dynamic programming was used to solve this problem by Smith and Waterman [2]. They also give another interpretation of the LSS problem. The $SMS(i, j)$ is the maximum similarity score of two segments ending at $S_1[i], S_2[j]$

$$SMS(i, j){=}\max_{i',j'}\{0; \phi(S_1[i'..i], S_2[j'..j])|1 \leq i' \leq i, 1 \leq j' \leq j\}$$

The $SMS(S_1, S_2)$ is defined as follow:

$$SMS(S_1, S_2){=}\max_{i,j}\{SMS(i, j)\}$$

Algorithm 4: Local sequence similarity

```
Input: $S_1$, $S_2$, All the $s(i,-)$ ,$s(-,j)$, $s(i,j)$ where $i, j \geq 0$
Output: $\max\phi(S_1[i'..i], S_2[l'..l])$, where $1 \leq i' \leq i \leq |S_1|, 1 \leq j' \leq j \leq |S_2|$.
for $i = 0$ to $|S_1|$ do
    $\phi[i, 0] = 0$
end
for $j = 0$ to $|S_2|$ do
    $\phi[0, j] = 0$
end
for $i = 1$ to $|S_1|$ do
    for $j = 1$ to $|S_2|$ do
        $\phi(i,j) = \max \begin{cases} 0, \\ \phi(i-1, j-1) + s(i,j), \\ \phi(i-1, j) + s(i,-), \\ \phi(i, j-1) + s(-,j). \end{cases}$
    end
end
```

As Algorithm 4 shows, consider the optimal solution, there are two segments ending at (i, j). In the dynamic program matrix, initial deletions of S_2 are set at $\phi[0, j] = 0$. Initial insertions of S_1 are set at $\phi[i, 0] = 0$. Think of the all possible segments ending at(i, j), if $\phi(i-1, j-1) + s(i, j), \phi(i-1, j) + s(i, -), \phi(i, j-1) + s(-, j)$ ends with all negative values, then there is no positive scoring segment ending at (i, j) and the value is 0 since the optimal solution is two empty segments. If there is one or more positive scoring segments ending at (i, j), (i, j) is in the alignment. We pick the largest positive score as an optimal path.

The longest common subsequence (LCS) problem is to compute the longest subsequence common between two sequences. Figure 3.2 shows the longest common sub-sequence. Given two sequences $S_1[1..m]$ and $S_2[1..n]$, Algorithm 5 computes the maximum length of the LCS by using dynamic programming. From Algorithm 5, to compute the longest common sub-sequences to $C[i, j]$, compare the elements $S_1[i]$ and $S_2[j]$. If they are equal, then the sequence $C[i-1, j-1]$ is extended by that element. If they are not equal, then the longer of the two sequences, $C[i-1, j]$ and $C[i, j-1]$, will be retained.

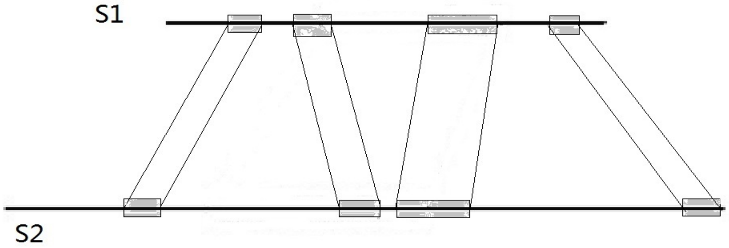


Figure 3.2: Longest common sub-sequence

Algorithm 5: Longest common sub-sequence

```
Input: $S_1$, $S_2$, $C[|S_1|][|S_2|] = 0$
Output: LCS
for $i = 0$ to $|S_1|$ do
    $C[i, \emptyset] = 0$
end
for $j = 0$ to $|S_2|$ do
    $C[\emptyset, j] = 0$
end
for $i = 1$ to $|S_1|$ do
    for $j = 1$ to $|S_2|$ do
        if $S_1[i]{=}S_2[j]$, then
            $C[i, j] = C[i-1, j-1] + 1$
        end
        if $S_1[i]{\neq}S_2[j]$, then
            $C[i, j] = \max\begin{cases} C[i-1, j] \\ C[i, j-1] \end{cases}$
        end
    end
end
```

3.2 Forest removing similarity for sub-forests

From the definition, we want to compute sub-forests F' and G'. An ordered labelled forest is a sequence of ordered labelled trees. The unit of the sequence is a sub-tree. In the computation, we actually need to determine a sequence of nodes, $i_1, i_2, ...i_m$ from F and G respectively such that they are not of ancestor-descendant relationship.

Lemma 3.1 *Given the forests F and G, let $i_1, i_2, ...i_m$ be sequence of nodes of F. let $j_1, j_2, ...j_m$ be a sequence of nodes of G, then*

$$\begin{gathered} LFS(F,G) {=} \max_{i_1,...i_m,j_1...j_m,m} \sum_{k=1}^{m} \max_{f,g} \phi(F[i_k] \setminus f, G[j_k] \setminus g) \\ f \in subcf(F[i_k], i_k), g \in subcf(G[j_k], j_k) \\ 1 \leq m \leq \min(|F|, |G|) \\ 1 \leq i_k \leq |F|,\ 1 \leq j_k \leq |G| \end{gathered}$$

Proof Suppose F' and G' are the optimal solution such that they can make maximum $LFS(F,G)$.

$$\begin{gathered} LFS(F,G) {=} \max_{F',G'} \phi(F',G') \\ F' \in subf(F) \\ G' \in subf(G) \end{gathered}$$

Both sub-forests F' and G' are sequence of sub-trees. Each sub-tree has a root. $\phi(F',G')$ is a mapping from F' to G'. Assume all the roots are in the mapping and roots of F' map with roots of G'. If not, there is one or more roots not in the mapping, then they could be deleted. Situations like this will cost score penalty which is non-positive. So removing of these nodes will not make the score worse. F' and G' are ordered labelled forests. The root of the first sub-tree in F' must map with the root of the first sub-tree in G', and the rest of the roots map with each other according to the order. Consider all roots are in the mapping and map according to the order, the number of sub-trees of F' should be equal with the number of sub-trees of G'. Consider a pair nodes i_k and j_k from $\phi(F',G')$ mapping with each other, the score is $\phi(F[i_k] \setminus f, G[j_k] \setminus g)$. $\phi(F',G')$ should be the sum of each pair of sub-tree's maximum edit similarity score. Normally, $\max \phi(F',G')$ will be larger than 0. If not, the best mapping is empty tree.

Suppose there are m sub-trees in F' and G' respectively.

$$\begin{gathered} \phi(F',G') {=} \sum_{n=1}^{m} \max_{f,g} \phi(F[i_k] \setminus f, G[j_k] \setminus g) \\ f \in subcf(F[i_k], i_k), g \in subcf(G[j_k], j_k) \end{gathered}$$

So that

$$LFS(F,G) = \max_{i_1,\dots i_m, j_1 \dots j_m, m} \sum_{k=1}^{m} \max_{f,g} \phi(F[i_k] \setminus f, G[j_k] \setminus g)$$
$$f \in subcf(F[i_k], i_k), g \in subcf(G[j_k], j_k)$$
$$1 \leq \mathrm{m} \leq \min(|F|, |G|)$$
$$1 \leq i_k \leq |F|,\ 1 \leq j_k \leq |G|$$

□

From Lemma 3.1, in order to compute $LFS(F,G)$, two steps are necessary. First step, compute all the $\phi(F[i_k] \setminus f, G[j_k] \setminus g)$. Second step, select several(m) $\phi(F[i_k] \setminus f, G[j_k] \setminus g)$ such that the sum of the scores are optimal. For the first step, to calculate $\phi(F[i_k] \setminus f, G[j_k] \setminus g)$, we extend Smith and Waterman's LSS algorithm from sequences to trees. In their LSS algorithm, they removed prefix of the sub-sequence. In our algorithm, complete sub-trees can be removed without penalty. In this section, we propose forest removing similarity(FRS) for sub-forests algorithm to calculate the maximum similarity scores of all $F[i]$ and $G[j]$ when some sub-forests can be removed from both of $F[i]$ and $G[j]$.

The problem is defined as follows:

Given two forests F and G, FRS for sub-forests

$$\Phi_{rr}(F[i], G[j]) = \max_{f,g} \phi(F[i] \setminus f, G[j] \setminus g)$$
$$f \in subcf(F[i], i)$$
$$g \in subcf(G[j], j)$$

The subscript "rr" represents that sub-forests or children nodes of both F and G can be removed.

We use the formula from [7]

For $\Phi_{rr}(F[l(i)..i'], G[l(j)..j'])$, where $l(i) \leq i' \leq i$ and $l(j) \leq j' \leq j$.

(a) $\Phi_{rr}(\emptyset, \emptyset) = 0$.

(b) $\Phi_{rr}(F[l(i)..i'], \emptyset) = 0$.

(c) $\Phi_{rr}(\emptyset, G[l(j)..j']) = 0$.

$$\text{(d) } \Phi_{rr}(F[l(i)..i'], G[l(j)..j']) = \max \begin{cases} \Phi_{rr}(F[l(i)..l(i')-1], G[l(j)..j']), \\ \Phi_{rr}(F[l(i)..i'], G[l(j)..l(j')-1]), \\ \Phi_{rr}(F[l(i)..i'-1], G[l(j)..j']) + s(f[i'], -), \\ \Phi_{rr}(F[l(i)..i'], G[l(j)..j'-1]) + s(-, g[j']), \\ \Phi_{rr}(F[l(i)..l(i')-1], G[l(j)..l(j')-1]) \\ \qquad + \Phi_{rr}(F[l(i'), i'-1], G[l(j'), j'-1]) + s(f[i'], g[j']). \end{cases}$$

For case(a), edit operation is not required. For case(b), $F[l(i)..i']$ can be removed so that no edit operation required. For case(c), $G[l(j)..j']$ can be removed so that no edit operation required .

For case(d), five cases should be considered

For the $\Phi_{rr}(F[l(i)..i'], G[l(j)..j'])$, five cases should be considered

Case 1: whether or not the sub-tree $F[l(i')..i']$ is removed.

Case 2: whether or not the sub-tree $G[l(j')..j']$ is removed.

If it is not one of the cases above, consider the optimal mapping M between $F[l(i)..i']$and $G[l(j)..j']$ after we perform an optimal removal of sub-trees of $F[l(i)..i']$ and $G[l(j)..j']$.

The mapping can be extended to$f[i']$, and $g[j']$ in three ways

Case 3: $f[i']$ is not touched by a line in M, then $(i', -) \in M$.

Case 4: $g[j']$ is not touched by a line in M, then $(-, j') \in M$.

Case 5: $f[i']$ and $g[j']$ are both touched by lines in M, then $(i', j') \in M$.

This is directly from[7].

To simplify the calculation of $\phi(F[l(i)..i'], G[l(j)..j'])$, consider two cases of $\phi(F[l(i)..i'], G[l(j)..j'])$

(1) $F[l(i)..i']$ and $G[l(j)..j']$ are both trees.

(2) $F[l(i)..i']$ or $G[l(j)..j']$ is a forest.

Let i, j, F and G be defined as above, $l(i) \leq i' \leq i$ and $l(j) \leq j' \leq j$ $\forall(i,j) \in (F,G)$, $\forall i' \in \{l(i), \ldots, i\}$ and $\forall j' \in \{l(j), \ldots, j\}$.

If $l(i') = l(i)$ and $l(j') = l(j)$,

$$\Phi_{rr}(F[l(i)..i'], G[l(j)..j']) = \max \begin{cases} 0 \\ \Phi_{rr}(F[l(i)..i'-1], G[l(j)..j']) + s(f[i'], -), \\ \Phi_{rr}(F[l(i)..i'], G[l(j)..j'-1]) + s(-, g[j']), \\ \Phi_{rr}(F[l(i)..i'-1], G[l(j)..j'-1]) + s(f[i'], g[j']). \end{cases}$$

If $l(i') \neq l(i)$ and $l(j') \neq l(j)$,

$$\Phi_{rr}(F[l(i)..i'], G[l(j)..j']) = \max \begin{cases} \Phi_{rr}(F[l(i)..l(i')-1], G[l(j)..j']), \\ \Phi_{rr}(F[l(i)..i'], G[l(j)..l(j')-1]), \\ \Phi_{rr}(F[l(i)..i'-1], G[l(j)..j']) + s(f[i'], -), \\ \Phi_{rr}(F[l(i)..i'], G[l(j)..j'-1]) + s(-, g[j']), \\ \Phi_{rr}(F[l(i)..l(i')-1], G[l(j)..l(j')-1]) + \Phi_{rr}(F[i'], G[j']). \end{cases}$$

This algorithm can be implemented to run in $O(|F| \cdot |G| \cdot \min\{D_F, L_F\} \cdot \min\{D_G, L_G\})$ time and $O(|F| \cdot |G|)$ space by using the Zhang-Shasha algorithm.

Zhang-Shasha-RemovingSimilarity is an efficient method to solve tree removing similarity problem. $\Phi_{rr}(F[i'], G[j'])$ is the key data of LFS problem and is needed in next section.

Algorithm 6: *RemovingSimilarity*(F, G)

```
Input: Forest $F$ and $G$.
Output: $\Phi_{rr}(F[i], G[j])$, where $1 \leq i \leq |F|$ and $1 \leq j \leq |G|$.
compute $K(F)$ and $K(G)$ and sort them in increasing order into arrays $K_1$ and $K_2$ respectively.
for $i' := 1$ to $|K_1|$ do
    for $j' := 1$ to $|K_2|$ do
        $i := K_1[i']$
        $j := K_2[j']$
        $TreeRemovingSimi(i, j)$
    end
end
```

Algorithm 7: *TreeRemovingSimi*(i, j)

```
$\Phi_{rr}(\emptyset, \emptyset) = 0$
for $i' := l(i)$ to $i$ do
    $\Phi_{rr}(F[l(i)..i'], \emptyset) = 0$
end
for $j' := l(j)$ to $j$ do
    $\Phi_{rr}(\emptyset, G[l(j)..j']) = 0$
end
for $i' := l(i)$ to $i$ do
    for $j' := l(j)$ to $j$ do
        if $l(i') = l(i)$ and $l(j') = l(j)$ then
            $\Phi_{rr}(F[l(i)..i'], G[l(j)..j']) = \max\begin{cases} 0, \\ \Phi_{rr}(F[l(i)..i'-1], G[l(j)..j']) + s(f[i'], -), \\ \Phi_{rr}(F[l(i)..i'], G[l(j)..j'-1]) + s(-, g[j']), \\ \Phi_{rr}(F[l(i)..i'-1], G[l(j)..j'-1]) + s(f[i'], g[j']) \end{cases}$
            $\Phi_{rr}(F[i'], G[j']) = \Phi_{rr}(F[l(i)..i'], G[l(j)..j'])$
        else
            $\Phi_{rr}(F[l(i)..i'], G[l(j)..j']) = \max\begin{cases} \Phi_{rr}(F[l(i)..l(i')-1], G[l(j)..j']), \\ \Phi_{rr}(F[l(i)..i'], G[l(j)..l(j')-1]), \\ \Phi_{rr}(F[l(i)..i'-1], G[l(j)..j']) + s(f[i'], -), \\ \Phi_{rr}(F[l(i)..i'], G[l(j)..j'-1]) + s(-, g[j']), \\ \Phi_{rr}(F[l(i)..l(i')-1], G[l(j)..l(j')-1]) \\ \quad + \Phi_{rr}(F[i'], G[j']) \end{cases}$
        end
    end
end
```

3.3 Our algorithms for local forest similarity

In this section, we modify the weighted longest common sub-sequence algorithm and use it on trees to calculate $LFS(F, G)$, the second step mentioned in Lemma 3.1. A matrix DP[$|F|$][$|G|$] is designed to calculate $LFS(F, G)$ by using dynamic programming.

Lemma 3.2 *Define i is ith node in F, j is jth node in G,*
$DP(i, j) = LFS(F[1...i], G[1...j]), DP[i, \emptyset] = 0, DP[\emptyset, j] = 0$ *then*

$$DP(i, j) = \max \begin{cases} DP(i-1, j) \\ DP(i, j-1) \\ DP(l(i)-1, l(j)-1) + \Phi_{rr}(F[i], G[j]) \end{cases}$$

Proof Consider the optimal solution mapping of F[1...i] and G[1...j], suppose $i_1, ...i_m$, $j_1...j_m$ are the optimal sub-tree's roots of optimal solution mapping. Compute $DP(i, j)$, to determine whether or not node i is i_m and whether or not node j is j_m. Case 1, node i is not i_m. The maximum similarity score is from $F[1...i-1], G[1...j]$. Case 2, node j is not j_m. The maximum similarity score is from $F[1...i], G[1...j-1]$. Case 3, node i is i_m and node j is j_m. Both $f[i]$ and $g[j]$ are in the mapping so that they are the last pair till (i,j). From definitions above, $F[i]$ and $G[j]$ should be mapping with each other. The optimal edit similarity mapping score is $\Phi_{rr}(F[i], G[j])$ so that $DP(i, j) = DP(l(i)-1, l(j)-1) + \Phi_{rr}(F[i], G[j])$.

□

Figure 3.3 shows three cases of computation for local forest similarity of $F[1...i], G[1...j]$. Based on Lemma 3.2, we design Algorithm 8 as follows

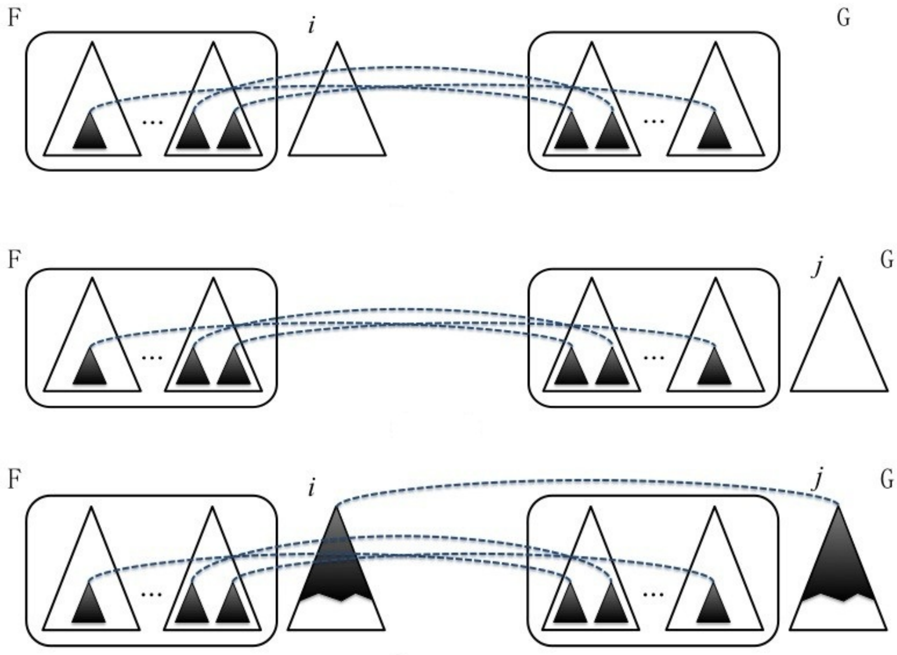


Figure 3.3: Local forest similarity operations

Algorithm 8: Local forest similarity for sub-forest

```
Input: forest $F$ and $G$ $\Phi_{rr}(F[i], G[j])$ where $1 \leq i \leq |F|$ and $1 \leq j \leq |G|$
Output: DP[$|F|$][$|G|$]
for $i = 0$ to $|F|$ do
    $DP[i, \emptyset] = 0$
end
for $j = 0$ to $|G|$ do
    $DP[\emptyset, j] = 0$
end
for $i = 1$ to $|F|$ do
    for $j = 1$ to $|G|$ do
        $DP(i,j) = \max \begin{cases} DP(i-1,j) \\ DP(i,j-1) \\ DP(l(i)-1, l(j)-1) + \Phi_{rr}(F[i], G[j]) \end{cases}$
    end
end
```

The local forest similarity for sub-forest algorithm shows how similar F and G are. The value of DP[$|F|$][$|G|$] is the maximum similarity score. The detail of sub-forests is needed and the mapping can be calculated by Algorithm 9.

TRStraceback is a procedure used in Algorithm 9. The main idea of *TRStraceback* algorithm is to calculate $\Phi_{rr}(F[i], G[j])$ by using *TreeRemovingSimilarity* algorithm and using a matrix which is large enough ($|F| * |G|$) for storing intermediate results. Then, use the matrix to determine the optimal result. When tracebacked to i and j, the algorithm determines whether or not they are involved. If so , *TRStraceback* algorithm calculates how $F[i]$ and $G[j]$ are mapped with each other by calculating the sequence of mapping operations (insertion\deletion\mapping) so that the optimal solution can be achieved.

Algorithm 9: Traceback

```
Input: Matrix DP[$|F|$][$|G|$]
Output: max $\phi(F \setminus f, G \setminus g)$
i=|F|
j=|G|
while $i \neq 0$ and $j \neq 0$ do
    if $DP[i][j] == DP[i][j-1]$ then
        $j--$;
        $Continue$;
    end
    if $DP[i][j] == DP[i-1][j]$ then
        $i--$;
        $Continue$;
    end
    if $DP[i][j] == DP[l[i]-1][l[j]-1] + \Phi_{rr}(F[i], G[j])$ then
        if $\Phi_{rr}(F[i], G[j]) > 0$ then
            $TRStraceback(i, j)$;
        end
        $i = l[i] - 1$;
        $j = l[j] - 1$;
        $Continue$;
    end
end
```

In this Chapter, we have introduced our algorithm for forest removing similarity problem. The implementation and experimental results are described in Chapter 5.

Chapter 4

Forest Pattern Matching For Sub-forests

Forest pattern matching (FPM) aims at computing regions of forest F which are most similar to another forest G. Generally, F is much larger than G. In this chapter, we consider the FPM problem. Given two ordered labelled forests F and G, the forest pattern matching for sub-forest problem is to determine sub-forest F' of F such that it is the most similar to G over all the possible F' .

The goal of FPM for sub-forests is to compute the maximum score of the following formula,

$$\begin{gathered} FPM(F,G)=\max_{F'} \phi(F',G) \\ F' \in subf(F) \end{gathered}$$

4.1 The algorithm for sequence pattern matching

It is known from section 3.1 that a sequence is a special case of a tree/forest. In this section, we introduce the pattern matching problem on sequences.

Sequence pattern matching (SPM) is to compute the best fit of a "short" sequence into a "longer" sequence. The SPM algorithm computes where the short pattern approximately appears in the longer sequence. Consider the problem of fitting sequence

$A = a_1...a_n$ into $B = b_1...b_m$ where n is much smaller than m. The problem is to compute

$$SPM(A,\, B)= \max\{\phi(A, b_k b_{k+1}...b_{l-1}b_l) : 1 \leq k \leq l \leq m\}$$

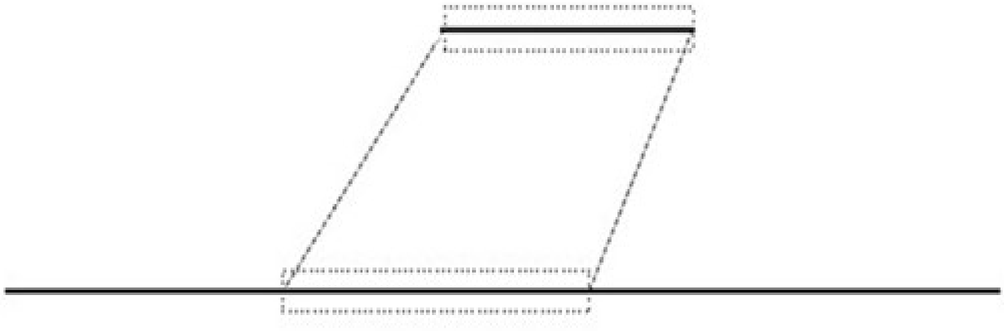


Figure 4.1: Sequence pattern matching

From Figure 4.1, we take another approach. Note that deletions of the beginning and the end of B are without penalty. Define

$$SPM(i,j) = \max_{k,j}\{\phi(a_1a_2...a_{i-1}a_i, b_kb_{k+1}...b_{j-1}b_j) : 1 \leq k \leq j \leq m, i = n\}$$

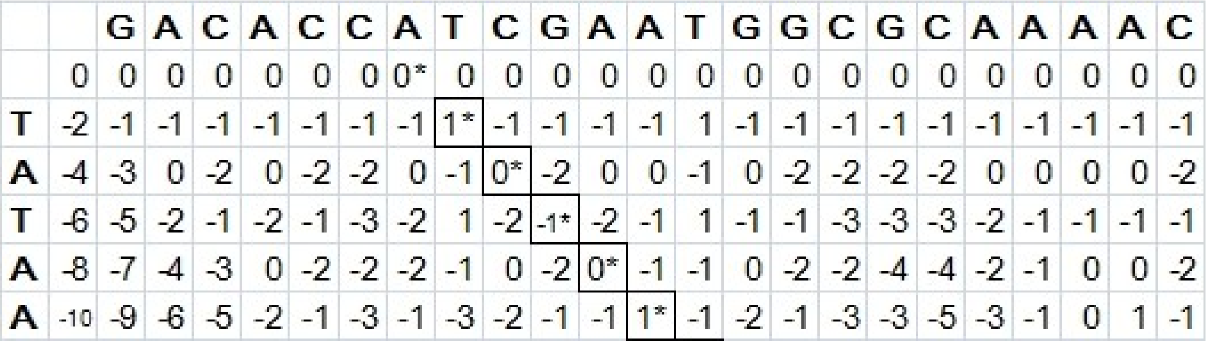
Deletions of the beginning of B without penalty can be encoded when a dynamic programming matrix is set up. The score of the initial deletion of B ,$b_1b_2...b_j$, is set at $\phi[0,j] = 0$. Each letter of A must be accounted for so that $\phi[i,0]$=$-i * s(i,-)$. After initial matching, deletion at the end of B as well as A must be accounted for. Choose the best score of $SPM(i,j)$ from $\phi(i-1,j-1)+s(i,j), \phi(i-1,j)+s(i,-)$ and $\phi(i,j-1)+s(-,j)$ [11].

Two sequences are given as examples, and set $s(a,a) = 1, s(a,b) = -1, s(a,-) = s(-,a) = -2$. Figure 4.2 shows the result of sequence pattern matching compued by Algorithm 10.

A=TATAAT

B=GACACCATCGAATGGCGCAAAACCTT

```
Algorithm 10: Sequence Pattern Matching
Input: sequence $A$ and $B$, All the $s(i,-)$ ,$s(-,j)$, $s(i,j)$ where $i,j \geq 0$
Output: $\max\{\phi(a_1a_2...a_{i-1}a_i, b_kb_{k+1}...b_{l-1}b_l)\}$
$\phi[\emptyset,\emptyset] = 0$
for $i = 1$ to $|A|$ do
    $\phi[i,\emptyset] = \phi[i-1,\emptyset] + s(i,-)$
end
for $j = 1$ to $|B|$ do
    $\phi[\emptyset,j] = 0$
end
for $i = 1$ to $|S_1|$ do
    for $j = 1$ to $|S_2|$ do
        $\phi(i,j) = \max \begin{cases} \phi(i-1,j-1) + s(i,j), \\ \phi(i-1,j) + s(i,-), \\ \phi(i,j-1) + s(-,j). \end{cases}$
    end
end
```



		G	A	C	A	C	C	A	T	C	G	A	A	T	G	G	C	G	C	A	A	A	A	C
	0	0	0	0	0	0	0	0*	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T	-2	-1	-1	-1	-1	-1	-1	-1	1*	-1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
A	-4	-3	0	-2	0	-2	-2	0	-1	0*	-2	0	0	-1	0	-2	-2	-2	-2	0	0	0	0	-2
T	-6	-5	-2	-1	-2	-1	-3	-2	1	-2	-1*	-2	-1	1	-1	-1	-3	-3	-3	-2	-1	-1	-1	-1
A	-8	-7	-4	-3	0	-2	-2	-2	-1	0	-2	0*	-1	-1	0	-2	-2	-4	-4	-2	-1	0	0	-2
A	-10	-9	-6	-5	-2	-1	-3	-1	-3	-2	-1	-1	1*	-1	-2	-1	-3	-3	-5	-3	-1	0	1	-1

Figure 4.2: Result of sequence pattern matching

4.2 Forest pattern matching for sub-forests

Given two forests F and G where F is larger than G, we use two steps to compute FPM. Step 1, we compute the the forest pattern matching for each pair of sub-trees between F and G. Step 2, we extend sequence pattern matching algorithm from sequences to forests.

In sequence pattern matching, the deletion of the short sequence will cost a penalty. In forest pattern matching, given two forests F and G where F is larger than G, the deletion of any node in G will cost a penalty. The prefix of the larger sequence's sub-sequence can be removed in SPM. In our algorithm, complete sub-trees can be removed from F without penalty. Define FPMSS as follows

$$FPMSS(F[i_k], G[j_k]) = \max_f \phi(F[i_k] \setminus f, G[j_k])$$
$$f \in subcf(F[i_k]).$$

In this section, we propose the forest pattern matching (FPM) for sub-forests algorithm to calculate $FPMSS(F[i], G[j])$ of all $F[i]$ and $G[j]$ when some sub-forests can be removed from $F[i]$.

The problem is defined as follows:

Given two forests F and G, F is larger than G, FPM for sub-forests

$$\Phi_R(F[i], G[j]) = \max_f \phi(F[i] \setminus f, G[j])$$
$$f \in subcf(F[i])$$

The subscript "R" represents that only sub-forests or child nodes of F can be removed. To simplify the calculation of $\Phi_R(F[i], G[j])$, consider two cases of $\phi(F[l(i)..i'], G[l(j)..j'])$

(1) $F[l(i)..i']$ and $G[l(j)..j']$ are both trees.

(2) $F[l(i)..i']$ or $G[l(j)..j']$ is a forest.

Let i,j F and G be defined as above, $l(i) \leq i' \leq i$ and $l(j) \leq j' \leq j$
$\forall (i,j) \in (F, G)$, $\forall i' \in \{l(i), \ldots, i\}$ and $\forall j' \in \{l(j), \ldots, j\}$.

If $l(i') = l(i)$ and $l(j') = l(j)$,

$$\Phi_R(F[l(i)..i'], G[l(j)..j']) = \max \begin{cases} \Phi_R(F[l(i)..i'-1], G[l(j)..j']) + s(f[i'], -), \\ \Phi_R(F[l(i)..i'], G[l(j)..j'-1]) + s(-, g[j']), \\ \Phi_R(F[l(i)..i'-1], G[l(j)..j'-1]) + s(f[i'], g[j']). \end{cases}$$

If $l(i') \neq l(i)$ and $l(j') \neq l(j)$,

$$\Phi_R(F[l(i)..i'], G[l(j)..j']) = \max \begin{cases} \Phi_R(F[l(i)..l(i')-1], G[l(j)..j']) \\ \Phi_R(F[l(i)..i'-1], G[l(j)..j']) + s(f[i'], -), \\ \Phi_R(F[l(i)..i'], G[l(j)..j'-1]) + s(-, g[j']), \\ \Phi_R(F[l(i)..l(i')-1], G[l(j)..l(j')-1]) + \Phi_{Rr}(F[i'], G[j']). \end{cases}$$

In the local similarity calculations, prefix of the sub-sequence can be deleted without penalty. If the two sub-forests or sub-sequences are not quite similar (negative), their scores are set to 0. But in pattern matching, it can be negative because the deletion of the smaller forest costs penalty.

This algorithm can be implemented to run in $O(|F| \cdot |G| \cdot \min\{D_F, L_F\} \cdot \min\{D_G, L_G\})$ time and $O(|F| \cdot |G|)$ space by using Zhang-Shasha algorithm.

```
Algorithm 11: $FPMSS(F, G)$
  Input: Forest $F$ and $G$.
  Output: $\Phi_R(F[i], G[j])$, where $1 \leq i \leq |F|$ and $1 \leq j \leq |G|$.
  compute $K(F)$ and $K(G)$ and sort them in increasing order into arrays $K_1$ and
  $K_2$ respectively.
  for $i' := 1$ to $|K_1|$ do
      for $j' := 1$ to $|K_2|$ do
          $i := K_1[i']$
          $j := K_2[j']$
          $FPMSSProcedue(i, j)$
      end
  end
```

Algorithm 12: $FPMSSProcedue(i, j)$

```
$\Phi_R(\emptyset, \emptyset) = 0$
for $i' := l(i)$ to $i$ do
    $\Phi_R(F[l(i)..i'], \emptyset) = 0$
end
for $j' := l(j)$ to $j$ do
    $\Phi_R(\emptyset, G[l(j)..j']) = G[l(j)..j' - 1]) + s(-, g[j'])$
end
for $i' := l(i)$ to $i$ do
    for $j' := l(j)$ to $j$ do
        if $l(i') = l(i)$ and $l(j') = l(j)$ then
            $\Phi_R(F[l(i)..i'], G[l(j)..j']) =$
                $\max \begin{cases} \Phi_R(F[l(i)..i' - 1], G[l(j)..j']) + s(f[i'], -), \\ \Phi_R(F[l(i)..i'], G[l(j)..j' - 1]) + s(-, g[j']), \\ \Phi_R(F[l(i)..i' - 1], G[l(j)..j' - 1]) + s(f[i'], g[j']) \end{cases}$
            $\Phi_R(F[i'], G[j']) = \Phi_R(F[l(i)..i'], G[l(j)..j'])$
        else
            $\Phi_R(F[l(i)..i'], G[l(j)..j']) =$
                $\max \begin{cases} \Phi_R(F[l(i)..l(i') - 1], G[l(j)..j']), \\ \Phi_R(F[l(i)..i' - 1], G[l(j)..j']) + s(f[i'], -), \\ \Phi_R(F[l(i)..i'], G[l(j)..j' - 1]) + s(-, g[j']), \\ \Phi_R(F[l(i)..l(i') - 1], G[l(j)..l(j') - 1]) \\ \qquad + \Phi_R(F[i'], G[j']) \end{cases}$
        end
    end
end
```

4.3 Our algorithms for forest pattern matching

In this section, we design an algorithm to compute the FPM. A matrix DP[$|F|$][$|G|$] is used to calculate $FPM(F, G)$ by using dynamic programming.

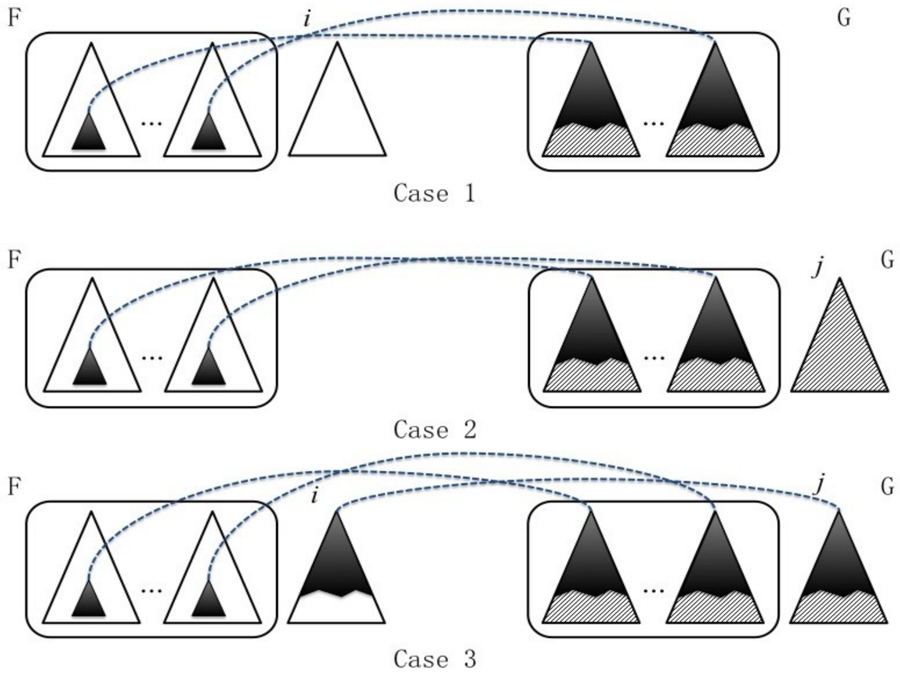
Lemma 4.1 *Define i is ith node in F, j is jth node in G,*
$DP(i, j) = FPM(F[1...i], G[1...j]), DP[i, \emptyset] = 0, DP[\emptyset, j] = DP[\emptyset, j-1]) + s(-, g[j])$, *then*

$$DP(i,j) = \max\begin{cases} DP(i-1, j) \\ DP(i, j-1) + s(-, g[j]) \\ DP(l(i)-1, l(j)-1) + \Phi_R(F[i], G[j]) \end{cases}$$

Proof Consider the optimal solution mapping of F[1...i] and G[1...j], suppose $j_1 ... j_m$ are optimal solutions mapping sub-tree's roots in G and they have no parent node in the mapping. $i_1, ... i_m$ are corresponding nodes in F. Compute $DP(i, j)$, to determine whether or not node i is i_m and whether or not node j is j_m. Case 1, node i is not i_m. The maximum pattern matching score is from $F[1...i-1], G[1...j]$. Case 2, node j is not j_m. The maximum pattern matching score is from $F[1...i], G[1...j-1]$ plus the penalty of deleting j. Case 3, node i is i_m and node j is j_m. Both $f[i]$ and $g[j]$ are in the mapping so that they are the last pair till (i,j). From definitions above, $F[i]$ and $G[j]$ should be mapped with each other. The optimal pattern matching score is $\Phi_R(F[i], G[j])$ so that $DP(i, j) = DP(l(i) - 1, l(j) - 1) + \Phi_R(F[i], G[j])$

□

Figure 4.3 shows three cases of computation for forest pattern matching of $F[1...i], G[1...j]$. Based on Lemma 4.1, we design Algorithm 13 as follow:



Case 1

Case 2

Case 3

Figure 4.3: Forest pattern matching operations

The traceback method is the same as for LFS problem. In this Chapter, we introduce our algorithm for forest pattern matching problem. The implementation and experimental results are in Chapter 5.

```
Algorithm 13: Forest pattern matching
  Input: forest $F$ and $G$ $\Phi_{Rr}(F[i], G[j])$ where $1 \leq i \leq |F|$ and $1 \leq j \leq |G|$
  Output: DP$[|F|][|G|]$
for $i = 0$ to $|F|$ do
    $DP[i, \emptyset] = 0$
end
for $j = 0$ to $|G|$ do
    $DP[\emptyset, j] = DP[\emptyset, j-1]) + s(-, g[j])$
end
for $i = 1$ to $|F|$ do
    for $j = 1$ to $|G|$ do
        $DP(i,j) = \max \begin{cases} DP(i-1, j) \\ DP(i, j-1) + s(-, g[j']) \\ DP(l(i)-1, l(j)-1) + \Phi_{Rr}(F[i], G[j]) \end{cases}$
    end
end
```

Chapter 5

Implementation and Experimental Results

In this Chapter, we present the implementation of the algorithms of LFS and FPM for sub-forests and experimental results.

5.1 Implementation

A C++ program is written for the algorithms. First, user needs to choose either LFS or FPM for the calculation. Second, user has to type in the names of score matrix file and forest data file. The program will begin the calculation automatically if the two files can be found successfully. At last, user has to type in a file name. All the results will be written in the file. Figure 5.1 shows the flowchart of the program.

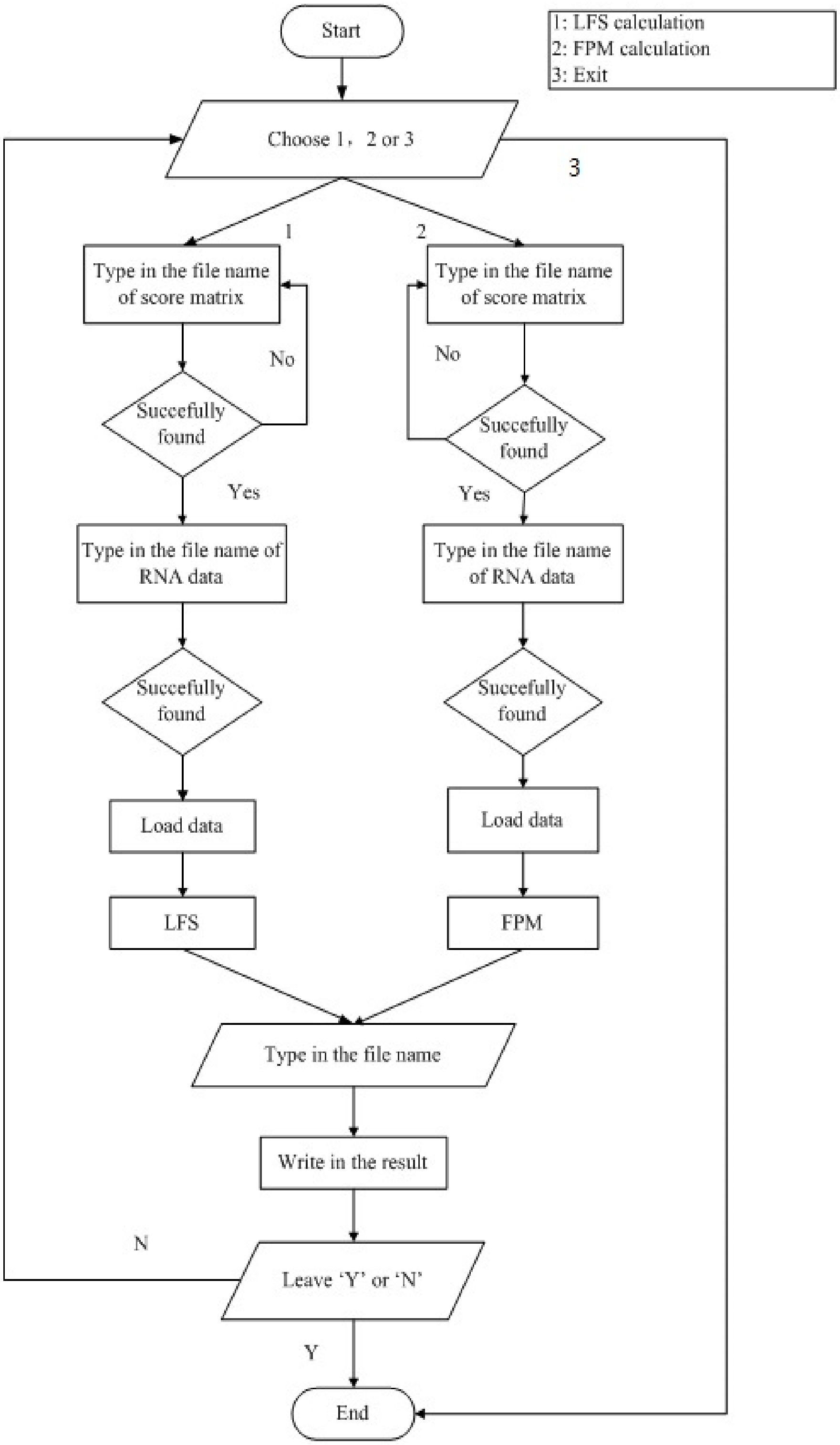


Figure 5.1: Flowchart of program.

5.2 Experimental results and discussion

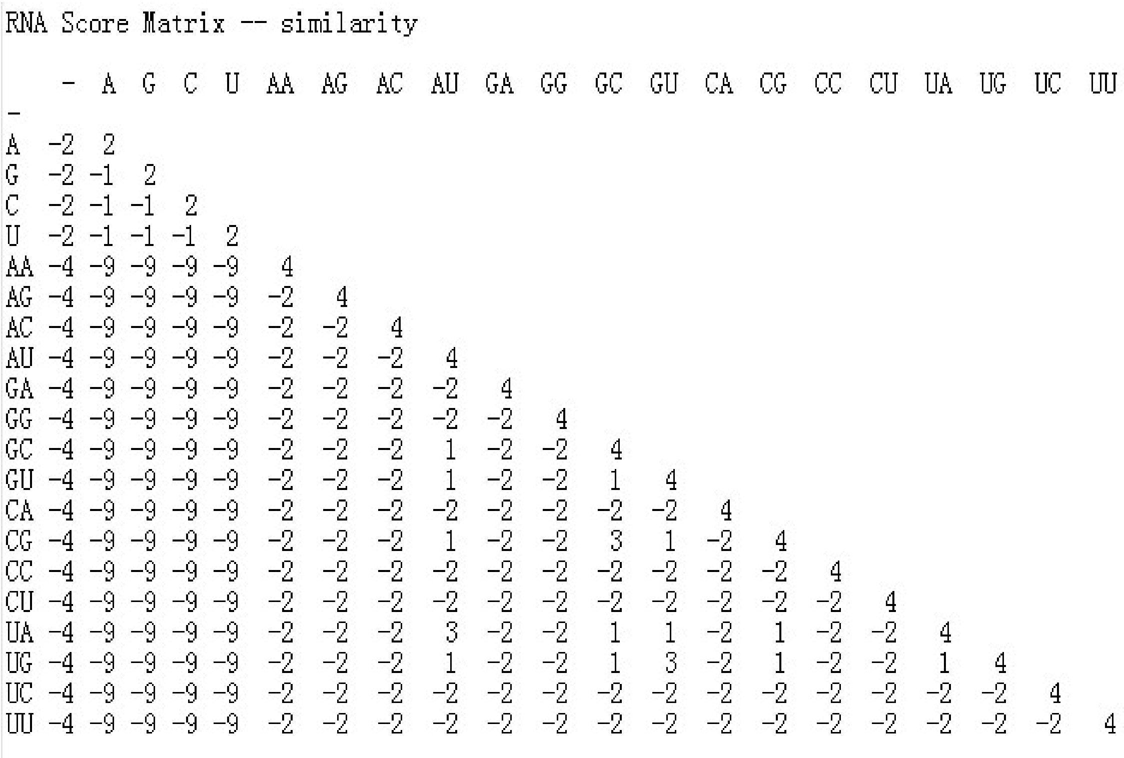
Five experiments are designed to test the algorithm. Experiment 1, 2 and 5 are for LFS and experiment 3 and 4 are for FPM.

Pseudoknot-free RNA can be represented as forests. Our forest data are taken from RNA cupriavidus metallidurans and streptomyces bikiniensis. Cupriavidus metallidurans and streptomyces bikiniensis, the RNase P RNA structures of bacteria, are pseudoknot-free. Cupriavidus metallidurans are renamed from ralstonia metallidurans and previously known as ralstonia eutropha and alcaligenes eutrophus. The images are taken from the website http://www.mbio.ncsu.edu/RNaseP/.

A forest data file has two sections. The first section represents the forest which is also the primary structure of forest. The second section contains all the information of forest's secondary structure such as the start bases, end bases, stem size etc.. A '<' sign alone on a line separates the two sections.

The score matrix file is significant such that results can be different by using different score matrix for the same forests. In our experiments, we use the same score matrix to test our algorithm. As shown in figure 5.2, the scores are set that match is 2, mismatch is -1, insertion and deletion is -2 for each single base. For a base pair, match is 4, mismatch is -2, insertion and deletion is -4. The penalty of matching a single base to a base pair is -9 since it is not reasonable. A high penalty score setting up can avoid this situation during the computation. Note that the scores are 3 when AU matches UA, CG matches GC and GU matches UG since these kinds of matches are treated as the good cases in RNA computing although the base pairs are not exactly the same. When the other reasonable base pairs match with each other, we set the score to 1 because they are common situations in regard of RNA secondary structures. ' - ' represents insertion and deletion and '   ' represents the base is removed.

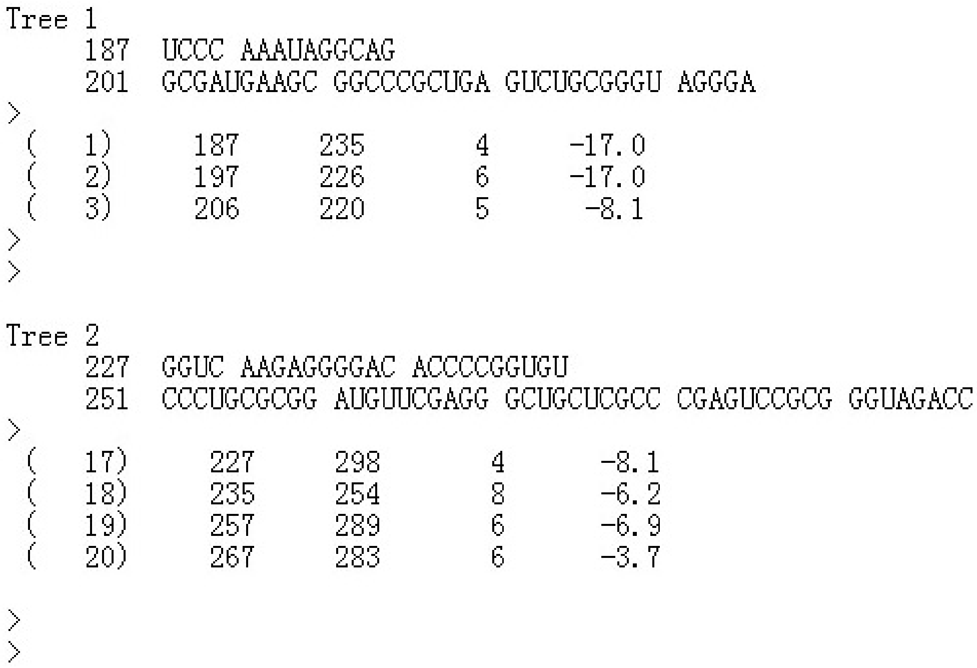
In the result file, the bases marked with '*' are the regions determined.



```
RNA Score Matrix -- similarity

     -  A  G  C  U   AA   AG   AC   AU   GA   GG   GC   GU   CA   CG   CC   CU   UA   UG   UC   UU
-
A   -2  2
G   -2 -1  2
C   -2 -1 -1  2
U   -2 -1 -1 -1  2
AA  -4 -9 -9 -9 -9    4
AG  -4 -9 -9 -9 -9   -2    4
AC  -4 -9 -9 -9 -9   -2   -2    4
AU  -4 -9 -9 -9 -9   -2   -2   -2    4
GA  -4 -9 -9 -9 -9   -2   -2   -2   -2    4
GG  -4 -9 -9 -9 -9   -2   -2   -2   -2   -2    4
GC  -4 -9 -9 -9 -9   -2   -2   -2    1   -2   -2    4
GU  -4 -9 -9 -9 -9   -2   -2   -2    1   -2   -2    1    4
CA  -4 -9 -9 -9 -9   -2   -2   -2   -2   -2   -2   -2   -2    4
CG  -4 -9 -9 -9 -9   -2   -2   -2    1   -2   -2    3    1   -2    4
CC  -4 -9 -9 -9 -9   -2   -2   -2   -2   -2   -2   -2   -2   -2   -2    4
CU  -4 -9 -9 -9 -9   -2   -2   -2   -2   -2   -2   -2   -2   -2   -2   -2    4
UA  -4 -9 -9 -9 -9   -2   -2   -2    3   -2   -2    1    1   -2    1   -2   -2    4
UG  -4 -9 -9 -9 -9   -2   -2   -2    1   -2   -2    1    3   -2    1   -2   -2    1    4
UC  -4 -9 -9 -9 -9   -2   -2   -2   -2   -2   -2   -2   -2   -2   -2   -2   -2   -2   -2    4
UU  -4 -9 -9 -9 -9   -2   -2   -2   -2   -2   -2   -2   -2   -2   -2   -2   -2   -2   -2   -2    4
```

Figure 5.2: Score matrix file.

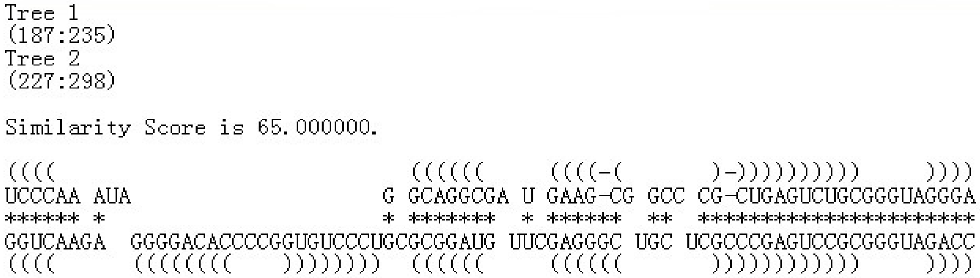
Experiment 1
Forest file:



```
Tree 1
          187  UCCC AAAUAGGCAG
          201  GCGAUGAAGC GGCCCGCUGA GUCUGCGGGU AGGGA
>
  (   1)      187     235        4     -17.0
  (   2)      197     226        6     -17.0
  (   3)      206     220        5      -8.1
>
>

Tree 2
          227  GGUC AAGAGGGGAC ACCCCGGUGU
          251  CCCUGCGCGG AUGUUCGAGG GCUGCUCGCC CGAGUCCGCG GGUAGACC
>
  (  17)      227     298        4      -8.1
  (  18)      235     254        8      -6.2
  (  19)      257     289        6      -6.9
  (  20)      267     283        6      -3.7

>
>
```

Figure 5.3: forest file for experiment 1.

Result file:



```
Tree 1
(187:235)
Tree 2
(227:298)

Similarity Score is 65.000000.

((((                            ((((((     ((((-(       )-))))))))))     ))))
UCCCAA AUA                    G GCAGGCGA U GAAG-CG GCC CG-CUGAGUCUGCGGGUAGGGA
****** *                      * *******  * ******  **  **********************
GGUCAAGA  GGGGACACCCCGGUGUCCCUGCGCGGAUG UUCGAGGGC UGC UCGCCCGAGUCCGCGGGUAGACC
((((      ((((((((    ))))))))  ((((((     ((((((       ))))))))))))     ))))
```

Figure 5.4: Result file for experiment 1.

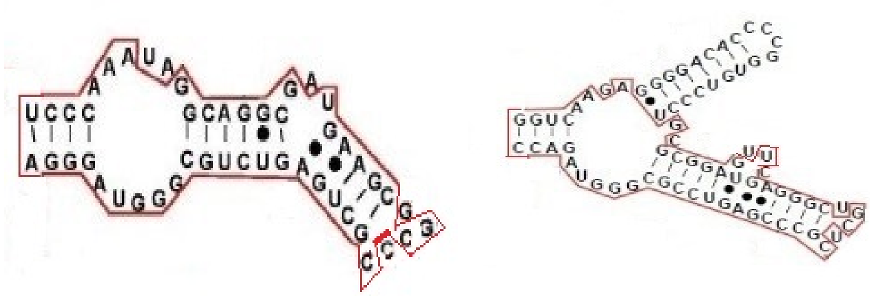
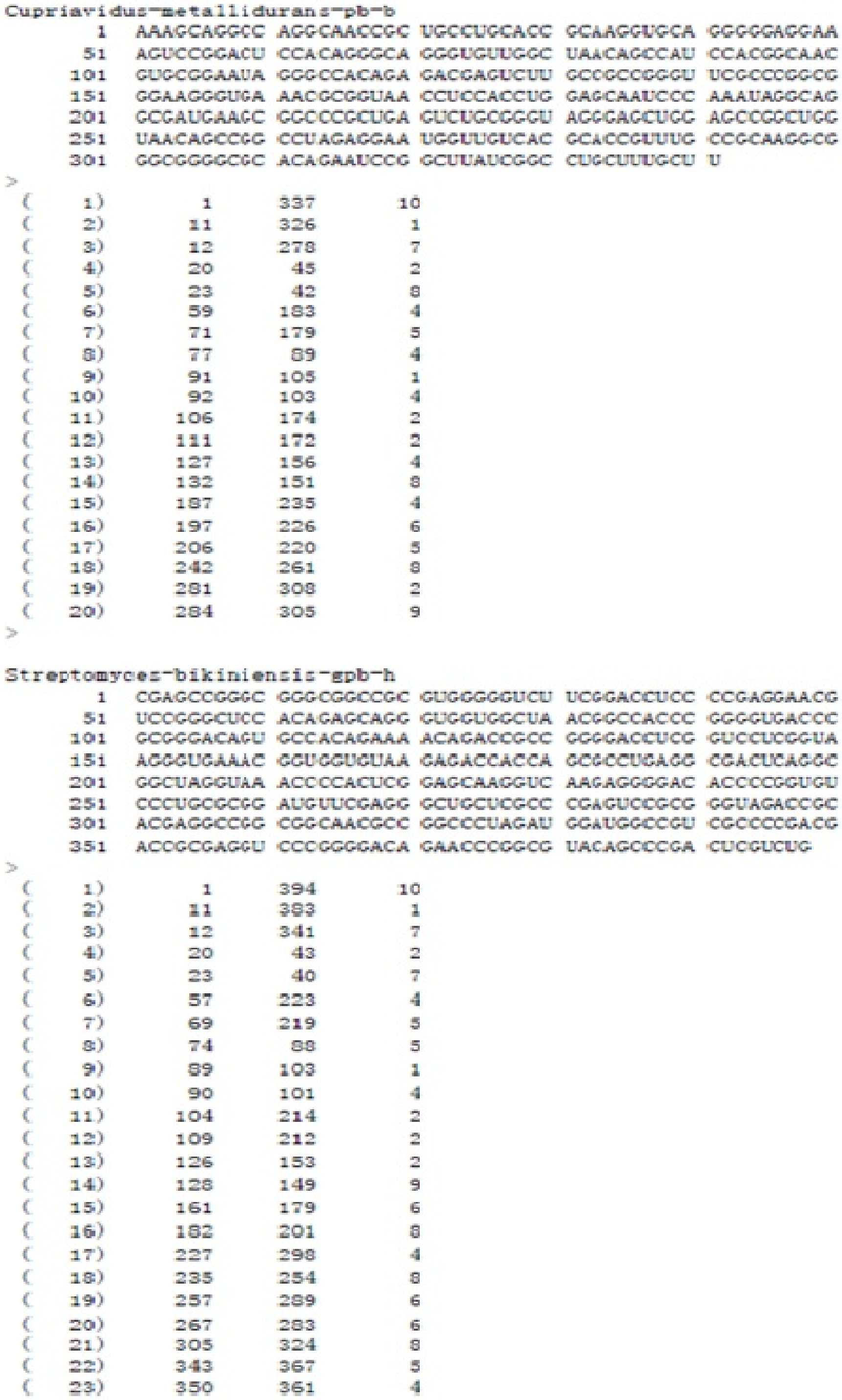


Figure 5.5: Result of experiment 1.

In experiment 1, we use two small trees to test the program. Figure 5.5 shows the result. The program determines one sub-forest from each tree and the maximum similarity score is 70. 5 sub-trees with one single base are removed from Tree 1 and 6 from Tree 2. One base pair of Tree 1 maps to insertion/deletion. One complete sub-tree of Tree 2 is totally removed. As the graph shows, the program determines large areas from the two trees which are most similar.

Experiment 2

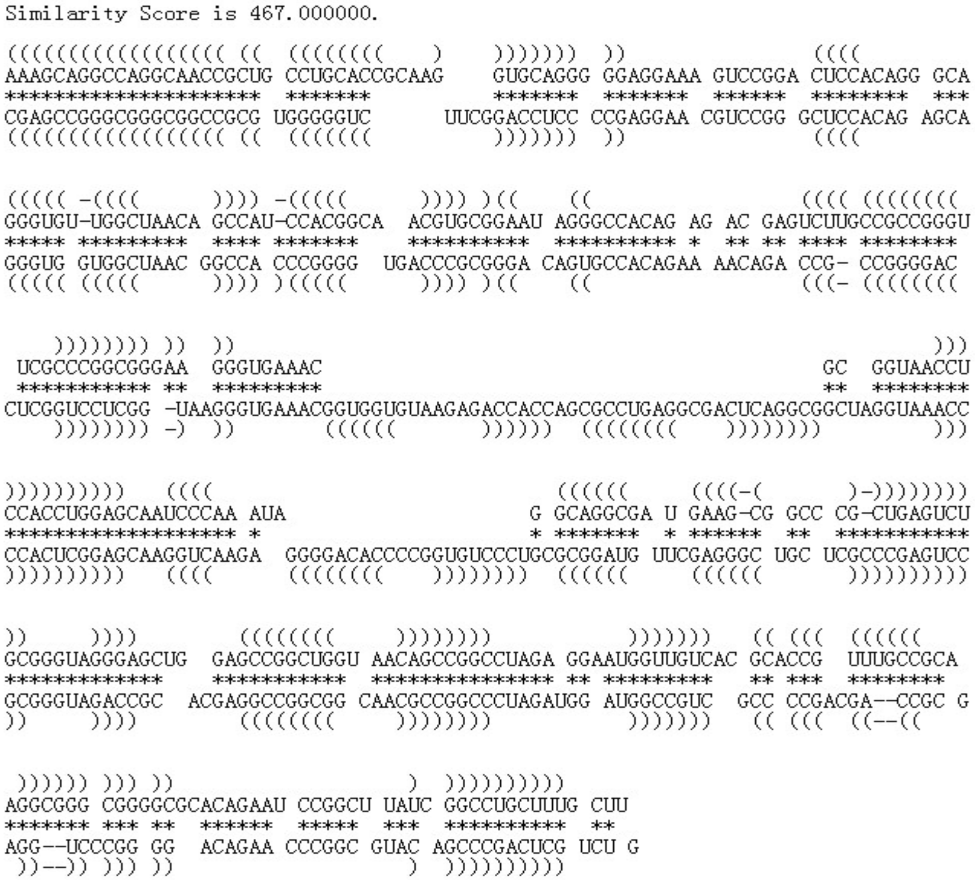
Forest file:



```
Cupriavidus-metallidurans-pb-b
       1   AAAGCAGGCC AGGCAACCGC UGCCUGCACC GCAAGGUGCA GGGGGAGGAA
      51   AGUCCGGACU CCACAGGGCA GGGUGUUGGC UAACAGCCAU CCACGGCAAC
     101   GUGCGGAAUA GGGCCACAGA GACGAGUCUU GCCGCCGGGU UCGCCCGGCG
     151   GGAAGGGUGA AACGCGGUAA CCUCCACCUG GAGCAAUCCC AAAUAGGCAG
     201   GCGAUGAAGC GGCCCGCUGA GUCUGCGGCU ACGGCAGCUGC AGCCCGCUGC
     251   UAACAGCCGG CCUAGAGGAA UGGUUGUCAC GCACCGUUUG CCGCAAGGCG
     301   GGCGGGGCGC ACAGAAUCCG GCUUAUCGGC CUGCUUUGCU U
>
  (     1)         1       337        10
  (     2)        11       326         1
  (     3)        12       278         7
  (     4)        20        45         2
  (     5)        23        42         8
  (     6)        59       183         4
  (     7)        71       179         5
  (     8)        77        89         4
  (     9)        91       105         1
  (    10)        92       103         4
  (    11)       106       174         2
  (    12)       111       172         2
  (    13)       127       156         4
  (    14)       132       151         8
  (    15)       187       235         4
  (    16)       197       226         6
  (    17)       206       220         5
  (    18)       242       261         8
  (    19)       281       308         2
  (    20)       284       305         9
>

Streptomyces-bikiniensis-gpb-h
       1   CGAGCCGGGC GGGCGGCCGC GUGGGGGUCU UCGGACCUCC CCGAGGAACG
      51   UCCGGGCUCC ACAGAGCAGG GUGGUGGCUA ACGGCCACCC GGGGUGACCC
     101   GCGGGACAGU GCCACAGAAA ACAGACCGCC GGGGACCUCG GUCCUCGGUA
     151   AGGGUGAAAC GGUGGUGUAA GAGACCACCA GCGCCUGAGG CGACUCAGGC
     201   GGCUAGGUAA ACCCCACUCG GAGCAAGGUC AAGAGGGGAC ACCCCGGUGU
     251   CCCUGCGCGG AUGUUCGAGG GCUGCUCGCC CGAGUCCGCG GGUAGACCGC
     301   ACGAGGCCGG CGGCAACGCC GGCCCUAGAU GGAUGGCCGU CGCCCCGACG
     351   ACCGCGAGGU CCCGGGGACA GAACCCGGCG UACAGCCCGA CUCGUCUG
>
  (     1)         1       394        10
  (     2)        11       383         1
  (     3)        12       341         7
  (     4)        20        43         2
  (     5)        23        40         7
  (     6)        57       223         4
  (     7)        69       219         5
  (     8)        74        88         5
  (     9)        89       103         1
  (    10)        90       101         4
  (    11)       104       214         2
  (    12)       109       212         2
  (    13)       126       153         2
  (    14)       128       149         9
  (    15)       161       179         6
  (    16)       182       201         8
  (    17)       227       298         4
  (    18)       235       254         8
  (    19)       257       289         6
  (    20)       267       283         6
  (    21)       305       324         8
  (    22)       343       367         5
  (    23)       350       361         4
```

Figure 5.6: forest file for experiment 2.

Result file:



```
Similarity Score is 467.000000.

(((((((((((((((((( ((  ((((((((    )    )))))))  ))               ((((
AAAGCAGGCCAGGCAACCGCUG CCUGCACCGCAAG    GUGCAGGG GGAGGAAA GUCCGGA CUCCACAGG GCA
*********************  *******          *******  *******  ******  ********  ***
CGAGCCGGGCGGGCGGCCGCG UGGGGGUC      UUCGGACCUCC CCGAGGAA CGUCCGG GCUCCACAG AGCA
(((((((((((((((((( ((  (((((((          )))))))  ))               ((((

((((( -((((      )))) -(((((      )))) )((     ((                 (((( ((((((((
GGGUGU-UGGCUAACA GCCAU-CCACGGCA  ACGUGCGGAAU AGGGCCACAG AG AC GAGUCUUGCCGCCGGGU
***** *********  **** *******    **********  ********** *  ** ** **** ********
GGGUG GUGGCUAAC GGCCA CCCGGGG  UGACCCGCGGGA CAGUGCCACAGAA AACAGA CCG- CCGGGGAC
((((( (((((      )))) )(((((      )))) )((     ((                 (((- ((((((((

    )))))))) ))  ))                                                         )))
 UCGCCCGGCGGGAA  GGGUGAAAC                                         GC  GGUAACCU
 *********** **  *********                                         **  ********
CUCGGUCCUCGG -UAAGGGUGAAACGGUGGUGUAAGAGACCACCAGCGCCUGAGGCGACUCAGGCGGCUAGGUAAACC
    )))))))) -)  ))       ((((((       ))))))  ((((((((    ))))))))         )))

))))))))))   ((((                            ((((((     ((((-(       )-))))))))
CCACCUGGAGCAAUCCCAA AUA                    G GCAGGCGA U GAAG-CG GCC CG-CUGAGUCU
******************* *                      * *******  * ******  **  ***********
CCACUCGGAGCAAGGUCAAGA  GGGGACACCCCGGUGUCCCUGCGCGGAUG UUCGAGGGC UGC UCGCCCGAGUCC
))))))))))   ((((      ((((((((    ))))))))  ((((((     ((((((       ))))))))))

))     ))))        ((((((((     ))))))))           )))))))   (( (((  ((((((
GCGGGUAGGGAGCUG  GAGCCGGCUGGU AACAGCCGGCCUAGA GGAAUGGUUGUCAC GCACCG  UUUGCCGCA
*************    ***********  *************** ** *********   ** ***  ********
GCGGGUAGACCGC  ACGAGGCCGGCGG CAACGCCGGCCCUAGAUGG AUGGCCGUC  GCC CCGACGA--CCGC G
))     ))))        ((((((((     ))))))))           )))))))   (( (((  ((--((

 )))))) ))) ))                   )  ))))))))))
AGGCGGG CGGGGCGCACAGAAU CCGGCU UAUC GGCCUGCUUUG CUU
******* *** **  ******  *****  ***  **********  **
AGG--UCCCGG GG  ACAGAA CCCGGC GUAC AGCCCGACUCG UCU G
 ))--)) ))) ))                   )  ))))))))))
```

Figure 5.7: Result file for experiment 2.

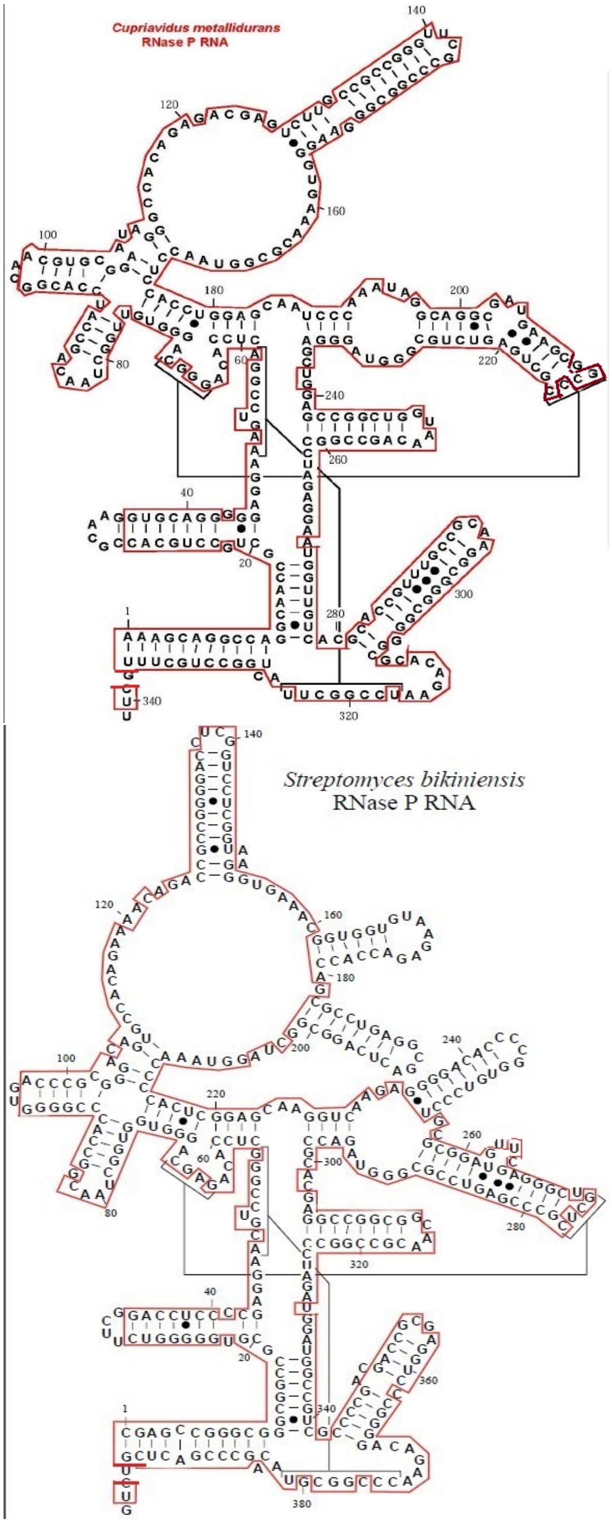


Figure 5.8: Result of experiment 2.

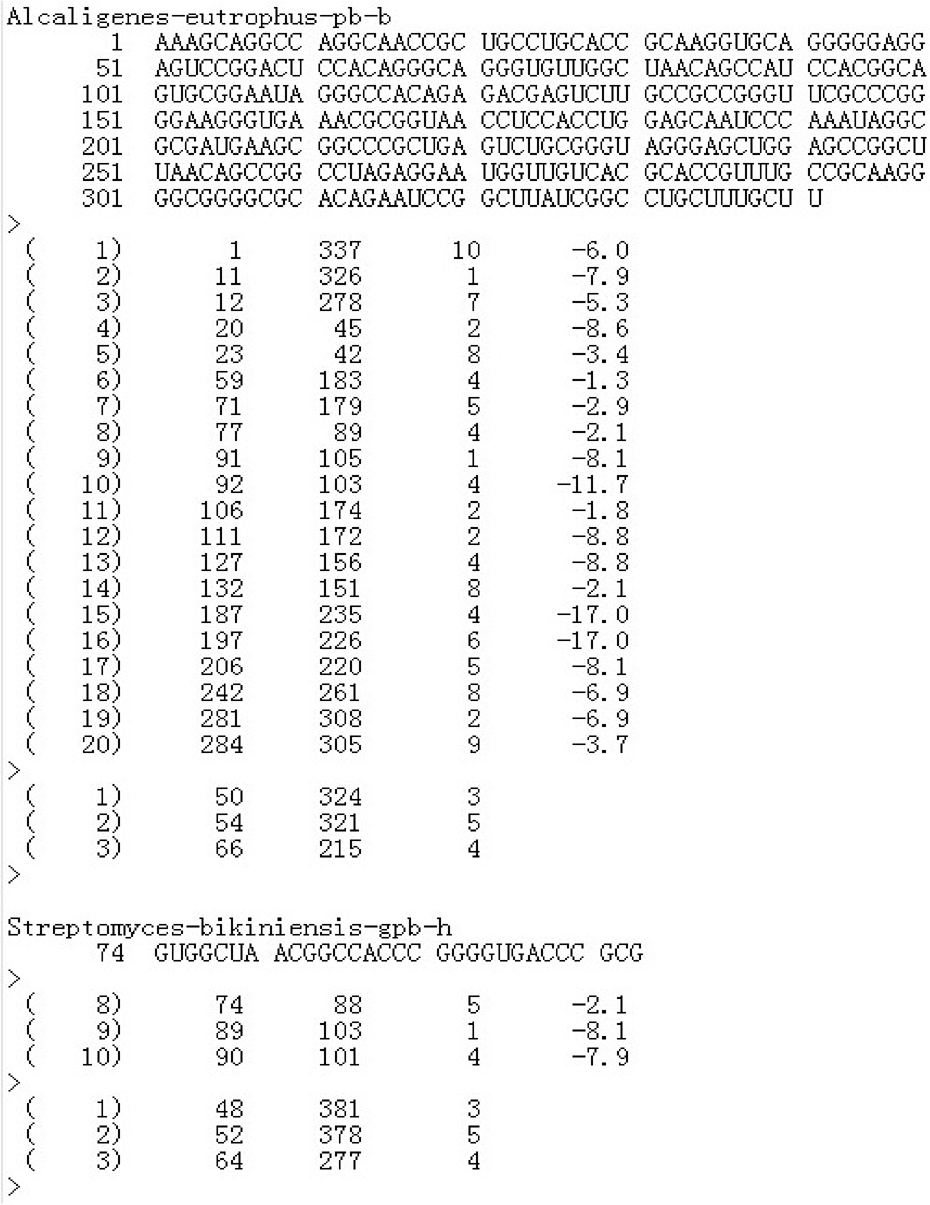
In experiment 2, we input two whole large RNAs, cupriavidus metallidurans and streptomyces bikiniensis.

The result is shown in Figure 5.8. Three sub-trees are determined from each RNA. Two of them are sub-trees which have only one single base. The other ones are large trees with some of its sub-trees removed.

From the detail, we can see there are other solutions, especially at loop structure (multiple loop, hairpin loop etc.). In fact, for all kinds of similarity calculation, the optimal solution may not be unique. To LSS, the maximum score may appear more than once in the matrix. In LFS, for each point in the matrix, operations may get the same score. But all these solutions are correct at the algorithm level. Our algorithm just offers one of the solutions.

Experiment 3

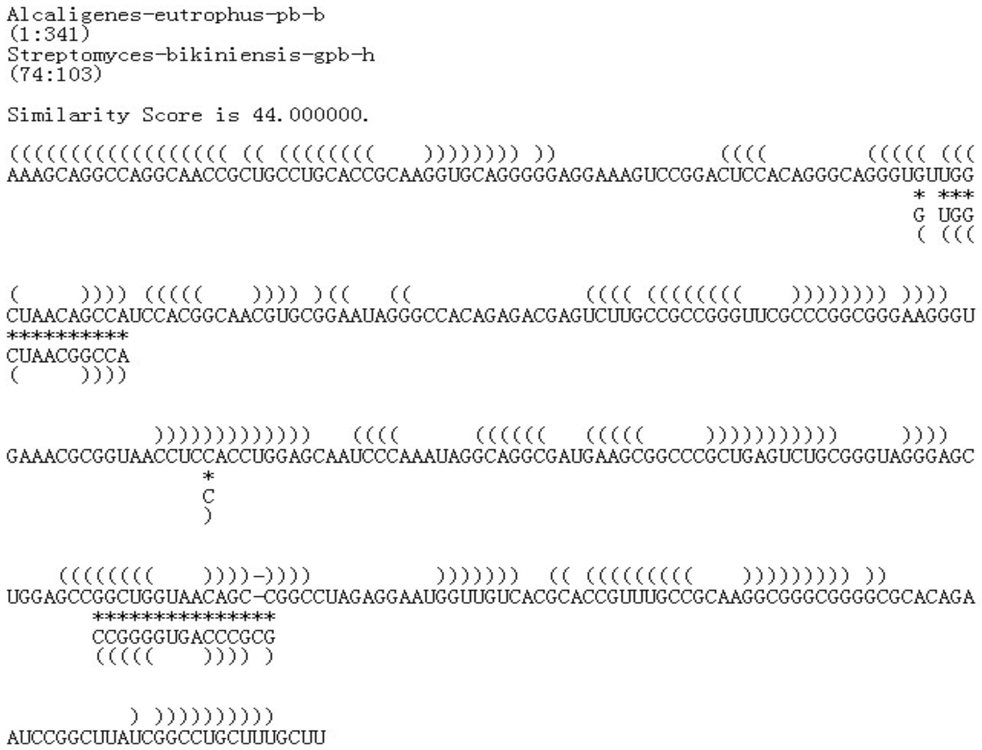
Forest file:



```
Alcaligenes-eutrophus-pb-b
       1  AAAGCAGGCC AGGCAACCGC UGCCUGCACC GCAAGGUGCA GGGGGAGG
      51  AGUCCGGACU CCACAGGGCA GGGUGUUGGC UAACAGCCAU CCACGGCA
     101  GUGCGGAAUA GGGCCACAGA GACGAGUCUU GCCGCCGGGU UCGCCCGG
     151  GGAAGGGUGA AACGCGGUAA CCUCCACCUG GAGCAAUCCC AAAUAGGC
     201  GCGAUGAAGC GGCCCGCUGA GUCUGCGGGU AGGGAGCUGG AGCCGGCU
     251  UAACAGCCGG CCUAGAGGAA UGGUUGUCAC GCACCGUUUG CCGCAAGG
     301  GGCGGGGCGC ACAGAAUCCG GCUUAUCGGC CUGCUUUGCU U
>
 (    1)       1     337      10      -6.0
 (    2)      11     326       1      -7.9
 (    3)      12     278       7      -5.3
 (    4)      20      45       2      -8.6
 (    5)      23      42       8      -3.4
 (    6)      59     183       4      -1.3
 (    7)      71     179       5      -2.9
 (    8)      77      89       4      -2.1
 (    9)      91     105       1      -8.1
 (   10)      92     103       4     -11.7
 (   11)     106     174       2      -1.8
 (   12)     111     172       2      -8.8
 (   13)     127     156       4      -8.8
 (   14)     132     151       8      -2.1
 (   15)     187     235       4     -17.0
 (   16)     197     226       6     -17.0
 (   17)     206     220       5      -8.1
 (   18)     242     261       8      -6.9
 (   19)     281     308       2      -6.9
 (   20)     284     305       9      -3.7
>
 (    1)      50     324       3
 (    2)      54     321       5
 (    3)      66     215       4
>

Streptomyces-bikiniensis-gpb-h
      74  GUGGCUA ACGGCCACCC GGGGUGACCC GCG
>
 (    8)      74      88       5      -2.1
 (    9)      89     103       1      -8.1
 (   10)      90     101       4      -7.9
>
 (    1)      48     381       3
 (    2)      52     378       5
 (    3)      64     277       4
>
```

Figure 5.9: forest file for experiment 3.

Result file:



```
Alcaligenes-eutrophus-pb-b
(1:341)
Streptomyces-bikiniensis-gpb-h
(74:103)

Similarity Score is 44.000000.

(((((((((((((((((( (( ((((((((    )))))))) ))             ((((        ((((( (((
AAAGCAGGCCAGGCAACCGCUGCCUGCACCGCAAGGUGCAGGGGGAGGAAAGUCCGGACUCCACAGGGCAGGGUGUUGG
                                                                          * ***
                                                                          G UGG
                                                                          ( (((


(     )))) (((((    )))) )((   ((              (((( ((((((((    )))))))) ))))
CUAACAGCCAUCCACGGCAACGUGCGGAAUAGGGCCACAGAGACGAGUCUUGCCGCCGGGUUCGCCCGGCGGGAAGGGU
**********
CUAACGGCCA
(     ))))


            )))))))))))))   ((((      ((((((   (((((     )))))))))))     ))))
GAAACGCGGUAACCUCCACCUGGAGCAAUCCCAAAUAGGCAGGCGAUGAAGCGGCCCGCUGAGUCUGCGGGUAGGGAGC
                *
                C
                )


    ((((((((    ))))-))))          )))))))  (( (((((((((    ))))))))) ))
UGGAGCCGGCUGGUAACAGC-CGGCCUAGAGGAAUGGUUGUCACGCACCGUUUGCCGCAAGGCGGGCGGGGCGCACAGA
       ***************
       CCGGGGUGACCCGCG
       (((((    )))) )


          ) ))))))))))
AUCCGGCUUAUCGGCCUGCUUUGCUU
```

Figure 5.10: Result file for experiment 3.

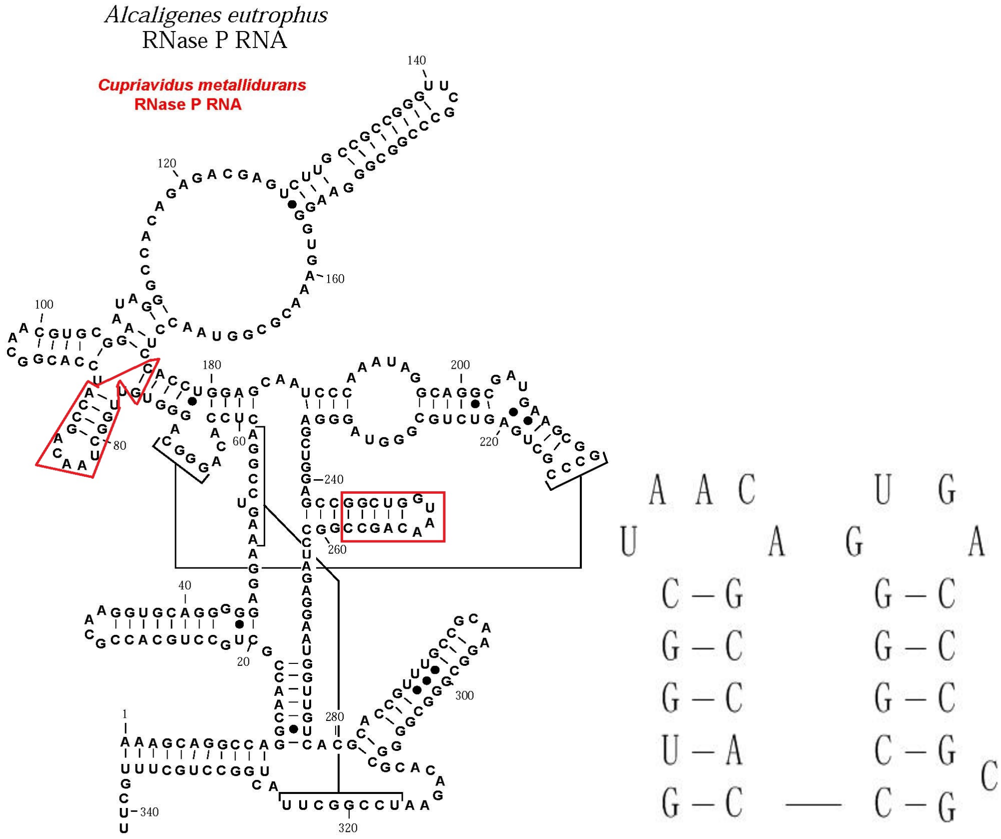
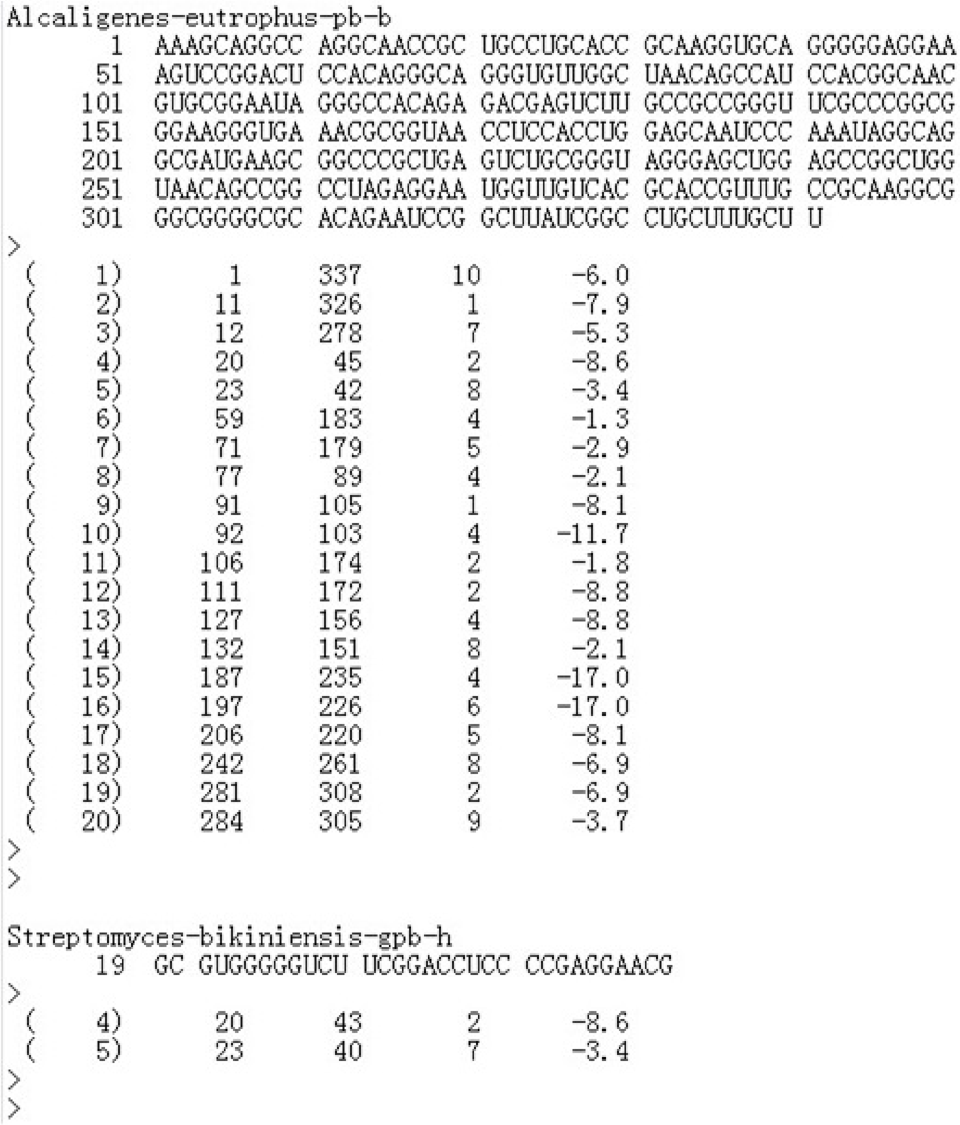


Figure 5.11: Result of experiment 3.

In experiment 3, we pick up two complete sub-forests and combine them to a forest. Then fit it to alcaligenes eutrophus. Figure 5.11 shows the result of experiment 3. Two parts of the structure are determined. One single base from the first part is removed. One single base from the pattern is deleted.

Experiment 4

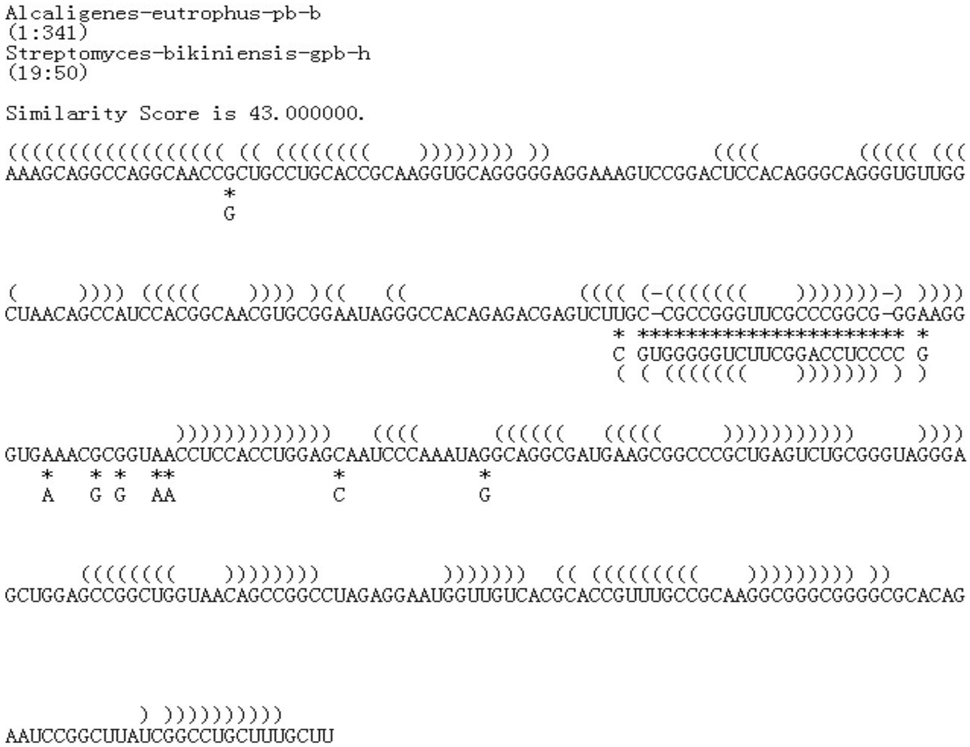
Forest file:



```
Alcaligenes-eutrophus-pb-b
       1   AAAGCAGGCC AGGCAACCGC UGCCUGCACC GCAAGGUGCA GGGGGAGGAA
      51   AGUCCGGACU CCACAGGGCA GGGUGUUGGC UAACAGCCAU CCACGGCAAC
     101   GUGCGGAAUA GGGCCACAGA GACGAGUCUU GCCGCCGGGU UCGCCCGGCG
     151   GGAAGGGUGA AACGCGGUAA CCUCCACCUG GAGCAAUCCC AAAUAGGCAG
     201   GCGAUGAAGC GGCCCGCUGA GUCUGCGGGU AGGGAGCUGG AGCCGGCUGG
     251   UAACAGCCGG CCUAGAGGAA UGGUUGUCAC GCACCGUUUG CCGCAAGGCG
     301   GGCGGGGCGC ACAGAAUCCG GCUUAUCGGC CUGCUUUGCU U
>
 (     1)        1      337       10       -6.0
 (     2)       11      326        1       -7.9
 (     3)       12      278        7       -5.3
 (     4)       20       45        2       -8.6
 (     5)       23       42        8       -3.4
 (     6)       59      183        4       -1.3
 (     7)       71      179        5       -2.9
 (     8)       77       89        4       -2.1
 (     9)       91      105        1       -8.1
 (    10)       92      103        4      -11.7
 (    11)      106      174        2       -1.8
 (    12)      111      172        2       -8.8
 (    13)      127      156        4       -8.8
 (    14)      132      151        8       -2.1
 (    15)      187      235        4      -17.0
 (    16)      197      226        6      -17.0
 (    17)      206      220        5       -8.1
 (    18)      242      261        8       -6.9
 (    19)      281      308        2       -6.9
 (    20)      284      305        9       -3.7
>
>

Streptomyces-bikiniensis-gpb-h
      19   GC GUGGGGGUCU UCGGACCUCC CCGAGGAACG
>
 (     4)       20       43        2       -8.6
 (     5)       23       40        7       -3.4
>
>
```

Figure 5.12: forest file for experiment 4.

Result file:



```
Alcaligenes-eutrophus-pb-b
(1:341)
Streptomyces-bikiniensis-gpb-h
(19:50)

Similarity Score is 43.000000.

(((((((((((((((((( (( ((((((((    )))))))) ))             ((((        ((((( (((
AAAGCAGGCCAGGCAACCGCUGCCUGCACCGCAAGGUGCAGGGGGAGGAAAGUCCGGACUCCACAGGGCAGGGUGUUGG
                  *
                  G


(     )))) (((((    )))) )((   ((              (((( (-(((((((    )))))))-) ))))
CUAACAGCCAUCCACGGCAACGUGCGGAAUAGGGCCACAGAGACGAGUCUUGC-CGCCGGGUUCGCCCGGCG-GGAAGG
                                                  * ********************** *
                                                  C GUGGGGGUCUUCGGACCUCCCC G
                                                  ( ( (((((((    ))))))) ) )


              )))))))))))))   ((((      ((((((   (((((     )))))))))))     ))))
GUGAAACGCGGUAACCUCCACCUGGAGCAAUCCCAAAUAGGCAGGCGAUGAAGCGGCCCGCUGAGUCUGCGGGUAGGGA
   *   * *  **             *           *
   A   G G  AA             C           G


      ((((((((    ))))))))          )))))))  (( (((((((((    ))))))))) ))
GCUGGAGCCGGCUGGUAACAGCCGGCCUAGAGGAAUGGUUGUCACGCACCGUUUGCCGCAAGGCGGGCGGGGCGCACAG


           ) ))))))))))
AAUCCGGCUUAUCGGCCUGCUUUGCUU
```

Figure 5.13: Result file for experiment 4.

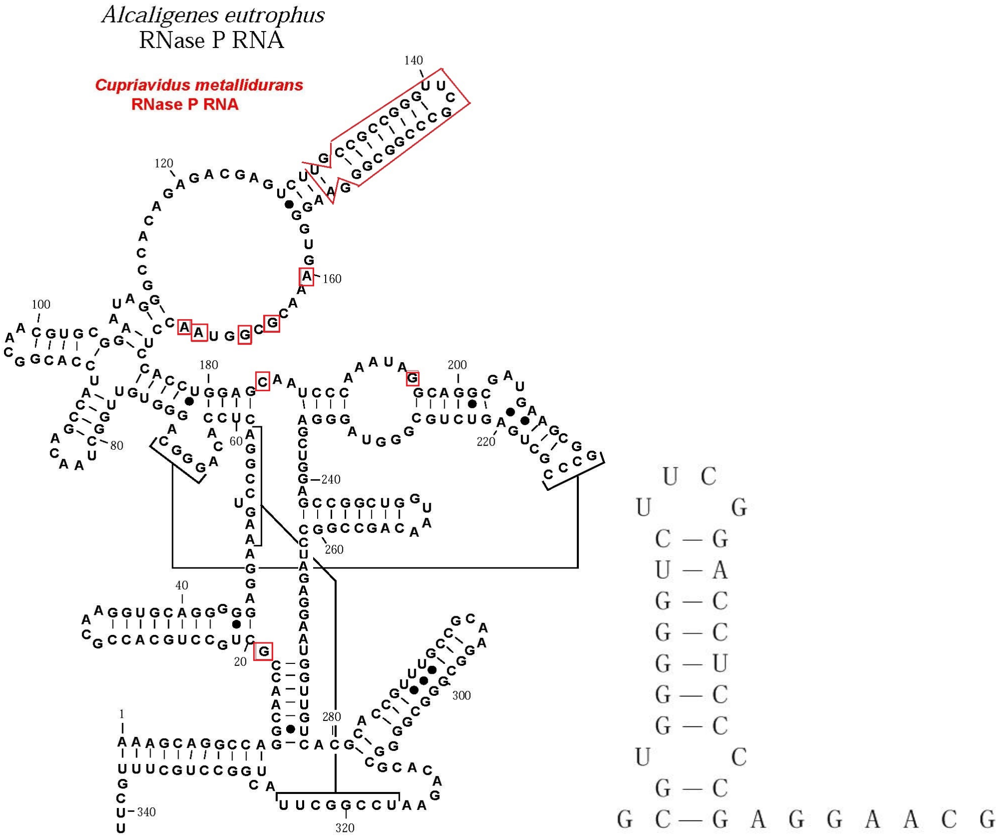
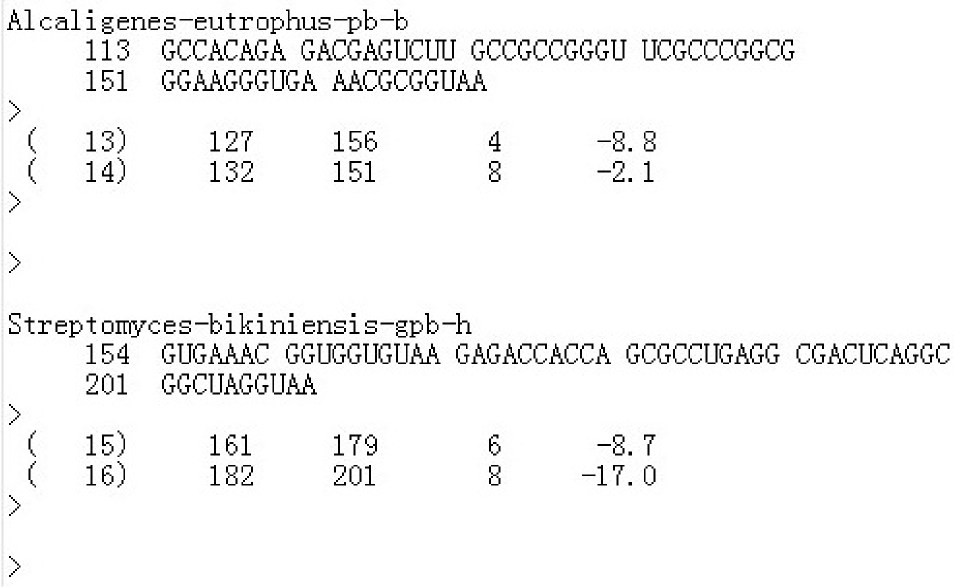


Figure 5.14: Result of experiment 4.

In experiment 4, we fit a closed sub-forest from streptomyces bikiniensis to alcaligenes eutrophus. Figure 5.14 shows the result of experiment 4. 9 sub-trees are determined. Two single bases from the sub-trees are removed. Two single bases from the pattern are deleted.

Experiment 5

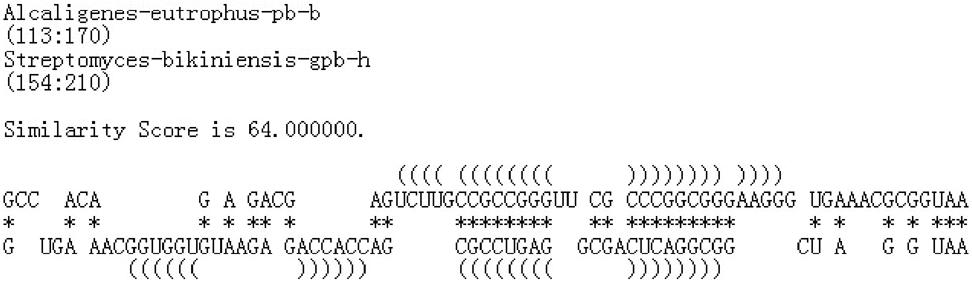
Forest file:



```
Alcaligenes-eutrophus-pb-b
     113  GCCACAGA GACGAGUCUU GCCGCCGGGU UCGCCCGGCG
     151  GGAAGGGUGA AACGCGGUAA
>
 (   13)     127     156       4      -8.8
 (   14)     132     151       8      -2.1
>

>

Streptomyces-bikiniensis-gpb-h
     154  GUGAAAC GGUGGUGUAA GAGACCACCA GCGCCUGAGG CGACUCAGGC
     201  GGCUAGGUAA
>
 (   15)     161     179       6      -8.7
 (   16)     182     201       8     -17.0
>

>
```

Figure 5.15: forest file for experiment 5.

Result file:



```
Alcaligenes-eutrophus-pb-b
(113:170)
Streptomyces-bikiniensis-gpb-h
(154:210)

Similarity Score is 64.000000.

                                (((( ((((((((      )))))))) ))))
GCC  ACA        G A GACG      AGUCUUGCCGCCGGGUU CG CCCGGCGGGAAGGG UGAAACGCGGUAA
*    * *        * * ** *      **     ********   ** *********      * *   * * ***
G  UGA AACGGUGGUGUAAGA GACCACCAG     CGCCUGAG  GCGACUCAGGCGG     CU A   G G UAA
          ((((((        ))))))       ((((((((      ))))))))
```

Figure 5.16: Result file for experiment 5.

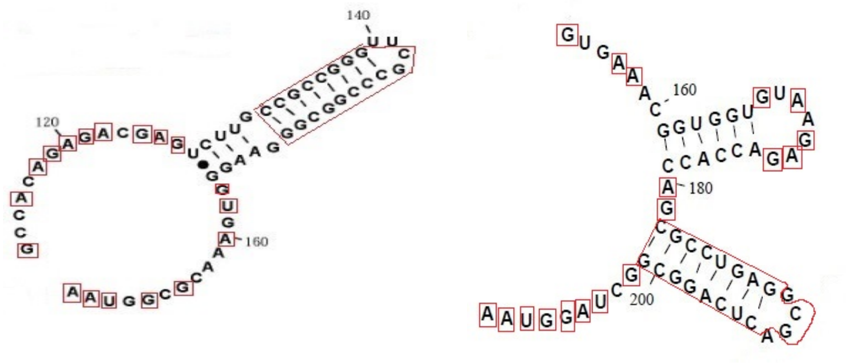


Figure 5.17: Result of experiment 5.

Experiment 5 is for LFS problem. We pick a small sub-forest from Alcaligenes eutrophus and Streptomyces bikiniensis respectively. 19 sub-trees are found out of each sub-forest.

As Figure 5.17 shows, the 1st to 10th sub-trees are siblings in the first sub-forest. But the 1st to 10th sub-trees are not siblings in the second sub-forest. The mapping of the 4th to 8th sub-trees may not make sense in RNA structure. And also, the similar situation also appears in Experiment 4 that the first 'G' is located far away from others. All these solutions are correct at the algorithm level. The result of Experiment 5 is a typical solution that is correct in algorithm but may not be reasonable in RNA structure which is worth further research.

Chapter 6

Conclusion

In this thesis, we studied local forest similarity and forest pattern matching problems. We developed efficient algorithms for general cases of these two problems.

For the local forest similarity problem, based on the well known Smith-Waterman method for local sequence similarity and longest common sub-sequence algorithm, we designed new algorithms to extend them from sequence to forest. Many algorithm techniques can be used for the tree removing similarity algorithm and we chose the Zhang-Shasha method in our algorithms. Our algorithm can identify locally similar regions in two forests.

For the forest pattern matching, based on sequence pattern matching we designed new algorithms to extend it from sequence to forest. Our algorithm can identify regions from one large forest which are most similar to another small forest.

Both the local forest similarity algorithm and the forest pattern matching algorithm can be applied to RNA structure comparison since pseudoknot-free RNA can be represented as forests.

Our algorithm can determine the general case of local forest similarity and forest pattern matching problems. In some cases, as the experiments showed, sub-forest determination is split over the forest. But a compact result may be more reasonable in RNA structure comparison. A possible way is to add the gap penalty during the computation. We will continue to improve our algorithm in the future research.

Bibliography

[1] B. A. Shapiro and K. Zhang. Comparing multiple RNA secondary structures using tree comparisons. *Computer Applications in the Biosciences*, 6(4):309–318, 1990.

[2] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

[3] K.-C. Tai. The tree-to-tree correction problem. *Journal of the Association for Computing Machinery* (JACM), 26(3):422–433, 1979.

[4] R. A. Wagner and M. J. Fischer. The string–to–string correction problem. *Journal of the Association for Computing Machinery* (JACM), 21(1):168–173, 1974.

[5] J. Wang, B. A. Shapiro, D. Shasha, K. Zhang, and K. M. Currey. An algorithm for finding the largest approximately common substructures of two trees. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(8):889–895, 1998.

[6] K. Zhang. Computing similarity between RNA secondary structures. In *Proceedings of IEEE International Joint Symposia on Intelligence and Systems*,Rockville, Maryland, May 1998, pages 126–132.

[7] 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.

[8] K. Zhang and Y. Zhu. Algorithms for Forest Pattern Matching. In *Proceedings of the 21th Symposium on Combinatorial Pattern Matching* (CPM 2010), pages 1–12, 2010.

[9] Y. Zhu. Algorithms for Forest Pattern Matching and Local Forest Similarity. Thesis(M.Sc), School of Graduate and Postdoctoral Studies, University of Western Ontario, London, Ontario, Canada, 2010.

[10] Liang, Z Efficient algorithms for local forest similarity Thesis (M. Sc), School of Graduate and Postdoctoral Studies, University of Western Ontario, London, Ontario, Canada 2011.

[11] Waterman, Michael S Introduction to computational biology: maps, sequences and genomes Book, CRC Press, 1995.

[12] J. W. Brown. The Ribonuclease P Database. *Nucleic Acids Research*, 27(1):314, 1999.

[13] R. Backofen and S. Will. Local Sequence-structure Motifs in RNA. *Journal of Bioinformatics and Computational Biology*, 2(4):681–698, 2004.

[14] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, and F. Yergeau, Extensible markup language (XML) 1.0. *W3C recommendation*, 6, 2000.

[15] S. Chen. Topics in Computing Similarity and Distance. Thesis(Ph.D), School of Graduate and Postdoctoral Studies, University of Western Ontario, London, Ontario, Canada, 2008.

[16] S. Chen, B. Ma, and K. Zhang On the similarity metric and the distance metric. *Theor. Comput. Sci.* 410(24-25):2365–2376, 2009.

[17] E. D. Demaine, S. Mozes, B. Rossman, and O. Weimann. An optimal decomposition algorithm for tree edit distance. In *Proceedings of the 34th International Colloquium on Automata, Languages and Programming* (ICALP), pages 146–157, 2007.

[18] M. Höchsmann, T. Töller, R. Giegerich, and S. Kurtz. Local similarity in RNA secondary structures. In *Proceedings of the IEEE Computational Systems Bioinformatics Conference*, pages 159–168, 2003.

[19] J. Jansson, N. T. Hieu, and W.K. Sung. Local Gapped Subforest Alignment and Its Application in Finding RNA Structural Motifs. *Journal of Computational Biology*, 13(3): 702–718, 2006.

[20] J. Jansson and Z. Peng. Algorithms for Finding a Most Similar Subforest. In *Proceedings of the 17th Symposium on Combinatorial Pattern Matching*, pages 377–388, 2006.

[21] T. Jiang, L. Wang, and K. Zhang. Alignment of trees - an alternative to tree edit. *Theoretical Computer Science*, 143:137–148, 1995.

[22] 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.

[23] Motifs database. http://subviral.med.uottawa.ca/cgi-bin/motifs.cgi.

[24] Z. Peng. Algorithms for Local Forest Similarity. In *Proceedings of the 16th International Symposium on Algorithms and Computation* (ISAAC 2005), pages 704–713, 2005.

Curriculum Vitae

Name:	Fang Han
Post-Secondary Education and Degrees:	University of Western Ontario London, ON 2011 - present M.Sc. candidate
	Tianjin University of Technology Tianjin, China 2006 - 2010 B.Eng
Honours and Awards:	University people's Scholarship 2006-2010
Related Work Experience:	Teaching Assistant The University of Western Ontario 2011 - resent