Topics in Computing Similarity and Distance

(Spine title: Topics in Computing Similarity and Distance)
(Thesis format: Integrated Article)

by

Shihyen Chen

Graduate Program in Computer Science

A thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

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

©Shihyen Chen 2008
THE UNIVERSITY OF WESTERN ONTARIO
School of Graduate and Postdoctoral Studies

CERTIFICATE OF EXAMINATION

Supervisor
Dr. Kaizhong Zhang

Co-Supervisor
Dr. Bin Ma

Supervisory Committee
Examiners

Dr. Sheng Yu
Dr. Lucian Ilie
Dr. Pei Yu
Dr. Jason Wang

The thesis by

Shihyen Chen

entitled:

Topics in Computing Similarity and Distance

is accepted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Date ___________________________  Chair of the Thesis Examination Board
Abstract

The central theme that permeates this thesis is about comparing abstract objects to reveal their similarity. Attention is given to the following topics: (1) the similarity between abstract objects in a specific context, namely ordered labeled trees, (2) a general framework of similarity and distance metrics, and (3) normalized local similarity.

For the first topic, the goal is to improve the algorithmic performance for the state of the art. For the second topic, the goal is to construct general metrics. For the third topic, the goal is to design ways for computing normalized local similarity.

Aside from the theoretical interests, the practical importance of this study is exemplified by the need to compare objects that frequently arises in diverse fields.

The goal of the first topic is accomplished by incorporating structural linearity into the state-of-the-art algorithms. The goal of the second topic is accomplished by forming a general definition for similarity metric as well as relating it to distance metric, thereby providing a basis for formulating general metrics. The goal of the third topic is accomplished by combining various algorithmic techniques while observing the metrics defined in the second topic.

Keywords: tree edit distance, dynamic programming, fractional programming, sequence comparison, RNA secondary structure comparison, text comparison, similarity metric, distance metric, normalized similarity metric, normalized distance metric, local similarity.
To my parents
Acknowledgments

First of all, I am profoundly indebted to Dr. Kaizhong Zhang and Dr. Bin Ma for their invaluable mentorship as well as generous and persistent support of my research endeavors. I consider myself fortunate to have them.

My appreciation extends to current and past fellow students and departmental members for fostering an amiable and stimulating environment for doing interesting and quality research work. Moreover, the education I received at this institute has served me well, academically or otherwise. It is not an overstatement that my most fruitful years to date were spent here at Western.

Last but not least, I sincerely thank the members of the examination board for their precious time and helpful opinions.

The works in this thesis were partially supported by grants from the Natural Sciences and Engineering Research Council of Canada, and MITACS, a Network of Centres of Excellence for the Mathematical Sciences.
Contents

Certificate of Examination ii

Abstract iii

Acknowledgments v

Table of Contents vi

1 Introduction 1
   Bibliography .......................................................... 4

2 Algorithmic Improvements for Tree Edit Distance 5
   2.1 Introduction ......................................................... 5
   2.2 Preliminaries ....................................................... 8
      2.2.1 Notations ...................................................... 8
      2.2.2 Recursion Strategy ........................................... 8
   2.3 Linearity ........................................................... 15
   2.4 Incorporating Vertical Linearity ................................. 16
      2.4.1 The Zhang-Shasha Algorithm ............................... 17
      2.4.2 Properties .................................................... 20
      2.4.3 Algorithmic Improvements ................................. 23
      2.4.4 Application ................................................... 29
   2.5 Incorporating Horizontal Linearity .............................. 31
3 The Similarity Metric and the Distance Metric
3.1 Introduction ........................................ 55
3.2 Similarity Metric and Distance Metric ............... 56
   3.2.1 Preliminaries .................................. 56
   3.2.2 Relationship between Similarity Metric and Distance Metric . 64
3.3 Normalized Similarity Metric .......................... 69
3.4 Normalized Distance Metric ............................ 79
3.5 Examples ............................................. 82
   3.5.1 Set Similarity and Distance ...................... 83
   3.5.2 Information Similarity and Distance ............ 84
   3.5.3 Sequence Edit Distance and Similarity .......... 89
3.6 Conclusions ......................................... 90
Bibliography ............................................ 91

4 Normalized Local Similarity: Sequences and RNA Structures 94
4.1 Introduction .......................................... 94
4.2 Normalized Local Sequence Similarity ................ 95
4.3 Normalized Local RNA Structural Similarity .......... 105
4.4 Conclusions ......................................... 107
Bibliography ............................................ 108

5 Conclusions ........................................... 110

Vita .................................................... 112
Chapter 1

Introduction

We consider the following discrete yet related topics: (1) computing the tree edit distance, (2) a general framework for similarity metric and distance metric, and (3) normalized local similarity.

A common theme shared by them concerns comparing abstract objects for their mutual similarity. The practical importance is that these abstract objects may be useful representations of real objects which we want to compare. Due to various reasons, direct comparison on real objects may be difficult. A common approach is to form an abstract representation of the real objects that captures the essential attributes of the objects under consideration and compare the abstract representations. The similarity between the real objects can thus be implied by the similarity between their abstract representations.

The comparison may be based on two types of measures: a dissimilarity measure, namely distance, or a similarity measure, namely similarity. Distance and similarity are complementary concepts in that shorter distance implies higher similarity, and vice versa. There are situations in which one type of measure is more convenient than the other type, hence the need for both types of measures.

With respect to the first topic, we aim to improve the algorithmic performance for the state of the art. With respect to the second topic, we aim at the following
goals:

- to give a definition of the similarity metric,
- to establish the relationship between the similarity metric and the distance metric,
- to construct general formulae (normalized or otherwise) for computing similarity and distance.

We approach the task in a general setting. With respect to the third topic, we combine various algorithmic techniques to design ways for computing normalized local similarity metrics.

We give brief sketches for the individual topics as follows.

**Tree edit distance:** Many kinds of objects can be represented by trees. As such, the problem of comparing trees is pervasive in diverse fields such as structured databases, computer vision, compiler optimization, natural language processing, and computational biology. The tree edit distance metric is a common dissimilarity measure for ordered trees. It was introduced in 1979 as a generalization of the string edit distance problem. Since then, a number of landmark works have been done on the problem of computing the tree edit distance. As to the state of the art, it comes down to three algorithms all of which based on dynamic programming with various styles of recursion. We explore the possibility of taking advantage of certain types of structural regularity in the trees so as to reduce the running time. We develop techniques for such purpose and show that they can be incorporated in the state-of-the-art algorithms. Preliminary work can be found in [3].

**The similarity metric and the distance metric:** Similarity and distance measures are widely used in many diverse fields. When a measure satisfies a set of well defined properties, we call it a metric. Distance metric is a well-defined concept. The
concept of similarity, in contrast, has not been formally defined although similarity measures are widely used and their properties are studied and discussed. We give a formal definition for the concept, and establish the relationship between similarity and distance that allows interconversion between the two types of metrics. Based on the metric definition, we present general formulae for similarity and distance metrics and show how to normalize these metrics. We demonstrate that some existing well-known metrics in various fields are special cases of our result. The content for this part is based on [2, 4].

**Normalized local similarity:** Local similarity is an important concept in the context concerning relations among biological species. Some of the metrics defined in the preceding topic are put to use here. The task is to design algorithms for computing normalized local similarity metrics for sequences and for RNA secondary structures. This part is primarily based on [1].

**Organization of the thesis:** Chapter 2 deals with the topic of computing tree edit distance. Chapter 3 deals with the topic of the general metric framework. Chapter 4 deals with normalized local similarity metric and its applications. Chapter 5 presents concluding remarks.
Bibliography


Chapter 2

Algorithmic Improvements for Tree Edit Distance

2.1 Introduction

An ordered labeled tree is a tree in which the nodes are labeled and the left-to-right order among siblings is significant. Trees can represent many phenomena, such as grammar parses, image descriptions and structured texts, to name a couple. In many applications where trees are useful representations of objects, the need for comparing trees frequently arises.

The tree edit distance metric was introduced by Tai [9] as a generalization of the string edit distance problem [11]. Given two trees $T_1$ and $T_2$, the tree edit distance between $T_1$ and $T_2$ is the minimum cost of transforming one tree into the other, with the sibling and ancestor orders preserved, by a sequence of edit operations on the nodes (substitution, insertion and deletion) as shown in Figure 2.1. In the same article, an algorithm was given with a time complexity $O(|T_1| \times |T_2| \times \prod_{i=1}^2 \text{depth}^2(T_i))$.

Later, Zhang and Shasha presented an algorithm [12] which improved the running time to $O(|T_1| \times |T_2| \times \prod_{i=1}^2 \min\{\text{depth}(T_i), \text{leaves}(T_i)\})$. Subsequent developments were made by Klein in [5] and by Demaine et al. in [3] where the ideas were similar
variants to that of Zhang and Shasha. The Klein algorithm runs in $O(|T_1|^2 \times |T_2| \times \log |T_2|)$ time and the Demaine et al. algorithm runs in $O(|T_1|^2 \times |T_2| \times (1 + \log |T_2|/|T_1|))$ time. These three algorithms require quadratic space. Table 2.1 summarizes these results.

<table>
<thead>
<tr>
<th>Time Complexity</th>
<th>Worst-Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) $O(</td>
<td>T_1</td>
</tr>
<tr>
<td>(b) $O(</td>
<td>T_1</td>
</tr>
<tr>
<td>(c) $O(</td>
<td>T_1</td>
</tr>
<tr>
<td>(d) $O(</td>
<td>T_1</td>
</tr>
</tbody>
</table>

Table 2.1: Time complexities of known algorithms. (a): Tai. (b): Zhang-Shasha. (c): Klein. (d): Demaine et al..

For arbitrary trees, the relative running times of the last three algorithms depend on the shapes of the trees, which are not necessarily in accord with the relative order in the worst case. It is fair to say that these three algorithms represent the state of
the art. Therefore, our focus is on these three algorithms.

All these three algorithms are based on dynamic programming with various recursion strategies prescribing the way in which a dynamic program builds up the solution. In [4], the algorithmic behaviors for each of these algorithms were formalized and the term *cover strategy* was used to refer to these recursion strategies. Given an instance $I$ of the tree edit distance problem, a *cover* refers to a way of tree decomposition which results in a set of disjoint paths. This set of disjoint paths, which we call *special paths*, induces a subset of all the subproblems of $I$ such that knowing the solutions of which is sufficient to build up the solution of $I$. Each special path is associated with the smallest subtree in which it is contained. The dynamic program proceeds in a bottom-up order with respect to the special paths such that for any node $i$ on a special path, the portion of the tree below $i$ has been processed before the node $i$ is reached. For sibling subtrees hanging off on the sides of a special path, the decision as to which one takes precedence is referred to as *strategy*.

In the Zhang-Shasha algorithm [12], the special paths are chosen to be the leftmost paths. In Klein’s algorithm [5] as well as that of Demaine et al. [3], the special paths are chosen such that every node on a special path is the root of a largest subtree over its sibling subtrees. These special paths are referred to as *heavy paths* [8]. Examples of various types of special paths are shown in Figure 2.2.

![Various types of special paths](http://example.com/fig2.2.png)

**Figure 2.2:** Various types of special paths (in bold).

In these algorithms, no consideration is given to any structural features which
may have an impact on the running time. In this paper, we investigate the possibility of utilizing certain linear features within the trees to speed up the computation. In particular, we consider two types of linearity, namely vertical linearity and horizontal linearity. We develop techniques to incorporate these types of linearity into the strategy-based algorithms. We show that the algorithmic running times of all the strategy-based algorithms may be substantially reduced in this way when there is a high degree of linearity in the input trees.

2.2 Preliminaries

2.2.1 Notations

Given a tree $T$, we denote by $t[i]$ the $i$th node in the left-to-right post-order numbering. The index of the leftmost leaf of the subtree rooted at $t[i]$ is denoted by $l(i)$. We denote by $F[i \cdots j]$ the ordered sub-forest of $T$ induced by the nodes indexed $i$ to $j$ inclusive. The subtree rooted at $t[i]$ in $T$ is denoted by $T[i]$, i.e., $T[i] = F[l(i) \cdots i]$. The sub-forest induced by removing $t[i]$ from $T[i]$ is denoted by $F[i]$, i.e., $F[i] = F[l(i) \cdots i-1]$. When referring to the children of a specific node, we adopt a subscript notation in accordance with the left-to-right sibling order. For example, the children of $t[i]$, from left to right, may be denoted by $(t[i_1], t[i_2], \ldots, t[i_k])$. Denote by $F \bullet G$ the forest composed of forests $F$ and $G$.

2.2.2 Recursion Strategy

The tree edit distance between two trees rooted respectively at the $i^{th}$ and the $j^{th}$ nodes is recursively computed according to Equation 2.1.

$$d(T_1[i], T_2[j]) = \min \left\{ \begin{array}{l}
d(F_1[i], T_2[j]) + \delta(t_1[i], \emptyset), \\
d(T_1[i], F_2[j]) + \delta(\emptyset, t_2[j]), \\
d(F_1[i], F_2[j]) + \delta(t_1[i], t_2[j])
\end{array} \right\}. \quad (2.1)$$
In order to compute this value, we need values of some tree-to-tree distances and forest-to-forest distances that are subproblems of the given instance of problem. The tree-to-tree subproblems are computed in the same recursive fashion as in Equation 2.1. As to the forest-to-forest distance, it is recursively computed according to Equation 2.5 with base cases given in Equation 2.2, 2.3 and 2.4.

\[
d(\emptyset, \emptyset) = 0 .
\]  

(2.2)

\[
d(F, \emptyset) = d(F - t[i], \emptyset) + \delta(t[i], \emptyset) .
\]  

(2.3)

\[
d(\emptyset, G) = d(\emptyset, G - t[j]) + \delta(\emptyset, t[j]) .
\]  

(2.4)

\[
d(F, G) = \min \left\{ \begin{array}{l} 
\quad d(F - t[i], G) + \delta(t[i], \emptyset), \\
\quad d(F, G - t[j]) + \delta(\emptyset, t[j]), \\
\quad d(F - T[i], G - T[j]) + d(T[i], T[j]) \end{array} \right\} . 
\]  

(2.5)

There are two distinctive ways by which the forest-to-forest distance may recurse, namely the rightmost recursion where both \(t[i]\) and \(t[j]\) are the rightmost tree roots and the leftmost recursion where both \(t[i]\) and \(t[j]\) are the leftmost tree roots. Figure 2.3 and Figure 2.4 depict the rightmost recursion and the leftmost recursion, respectively.

The subproblems with respect to the forest-to-forest distance \(d(F, G)\), excluding the trivial ones associated with single-node insertion or deletion cost as denoted by the \(\delta\) terms, are represented by the following terms: \(d(F - t[i], G)\), \(d(F, G - t[j])\), \(d(F - T[i], G - T[j])\) and \(d(T[i], T[j])\).

Note that the set of sub-forests which are relevant to the computation of the tree edit distance is a subset of all possible sub-forests. We call this kind of sub-forest a relevant sub-forest, defined as follows.
Figure 2.3: Rightmost recursion.

Figure 2.4: Leftmost recursion.
Definition 2.1 (Relevant Sub-forest). A sub-forest appearing in a subproblem is a relevant sub-forest.

Analogously, a relevant subtree is a special case of relevant sub-forest. Therefore in the preceding context, the terms $F - t[i]$, $F - T[i]$ and $T[i]$ represent the relevant sub-forests with respect to $F$.

The Zhang-Shasha algorithm is based on the rightmost recursion. Figure 2.5 gives an example showing the relevant sub-forests resulted from successive deletion of nodes in the Zhang-Shasha algorithm. In the context of dynamic programming, one would simply reverse the directions of the arrows in the example to obtain the bottom-up orderings. Figure 2.6 gives an example of the entire set of relevant sub-forests generated by the recursion in the Zhang-Shasha algorithm.

![Figure 2.5: An example showing the relevant sub-forests resulted from successive deletion of nodes in the Zhang-Shasha algorithm. The black nodes belong to the special path.](image)

The idea of the Klein algorithm is a similar variant of the Zhang-Shasha algorithm. The main difference is in the choice of key roots which yields special paths that may lie in the middle of a subtree. Suppose that the dynamic program encounters two forests $(\{T_1[i_1], \ldots, T_1[i_k]\}, \{T_2[j_1], \ldots, T_2[j_l]\})$ during the computation and the
Figure 2.6: An example showing the entire set of relevant sub-forests generated in the Zhang-Shasha algorithm. The black nodes belong to a special path.
heavy paths are identified for $T_1$. Let $t_1[i_p], i_{k'} \leq i_p \leq i_k$, be a special child. The Klein algorithm works as follows. If $i_p = i_{k'}$, the algorithm applies rightmost recursion. If $i_p \neq i_{k'}$, the algorithm applies leftmost recursion. Figure 2.7 gives an example showing the relevant sub-forests resulted from successive deletion of nodes in the Klein algorithm. Figure 2.8 gives an example of the entire set of relevant sub-forests generated by the recursion in the Klein algorithm.

![Figure 2.7: An example showing the relevant sub-forests resulted from successive deletion of nodes in the Klein algorithm. The black nodes belong to the special path.](image)

In order to guarantee that the solutions to all the subproblems are available when needed during the course of the dynamic program, for each relevant sub-forest of $T_1$ we need to consider all possible sub-forests in $T_2$.

Demaine et al. gave a new algorithm based on a subtle modification of the Klein algorithm. Note that the Klein algorithm identifies all the special paths of the larger tree which give rise to the relevant subtrees in the larger tree. The relevant subtrees, together with all the possible sub-forests of the smaller tree form the set of subproblems. For every subproblem, the subtree of the initially larger tree guides the recursion of the dynamic program. The modification made by Demaine et al. is that, for a subproblem $d(T_1[i], T_2[j])$ with $|T_1[i]| \geq |T_2[j]|$, instead of identifying all
Figure 2.8: An example showing the entire set of relevant sub-forests generated in the Klein algorithm. The black nodes belong to a special path.
the special paths of $T_1[i]$ only the longest special path of $T_1[i]$ is identified which gives rise to the top-level relevant subtrees of $T_1[i]$. On the $T_2[j]$ side, we consider all possible sub-forests. This produces a set of smaller subproblems. The same procedure is performed recursively for each subproblem in this set. For each subproblem, the larger subtree guides the recursion of the dynamic program.

2.3 Linearity

We formally define what we mean by linearity. Based on this definition, we construct compact representation for trees. The use of such compact representation can aid in improving the running time for computing the tree edit distance.

Definition 2.2 (V-Component). Given a tree $T$, a path $\pi$ of $T$ is a v-component (i.e., vertically linear component) if $\pi$ is a maximal non-branching path.

Definition 2.3 (V-Reduction). The v-reduction on a tree is to replace every v-component in the tree by a single node.

Definition 2.4 (H-Component). Given a tree $T$ and another tree $\tilde{T}$ obtained by a v-reduction on $T$, any set of connected subgraphs of $T$ corresponding to a set of leaves $\tilde{L}$ in $\tilde{T}$ form an h-component (i.e., horizontally linear component) if $\tilde{L}$ is a maximal contiguous subset of siblings such that $|\tilde{L}| \geq 2$.

Definition 2.5 (H-Reduction). The h-reduction on a tree is to replace every h-component in the tree by a single node.

A tree possesses vertical (horizontal) linearity if it contains any v-component (h-component). A tree is v-reduced if it is obtained by a v-reduction only. A tree is vh-reduced if it is obtained by a v-reduction followed by an h-reduction.

In Figure 2.9, we give an example showing the v-components and h-components of a tree and the corresponding reduced trees. Note that an h-component can also contain v-components.
Each node in a v-reduced tree corresponds to either a v-component or a single node in the corresponding full tree. Given a v-reduced tree $\tilde{T}$, we define two functions $\alpha(i)$ and $\beta(i)$ which respectively map a node $\tilde{t}[i]$ to the highest indexed node $t[\alpha(i)]$ and the lowest indexed node $t[\beta(i)]$ of the corresponding v-component in the full tree $T$. In the special case when $\tilde{t}[i]$ corresponds to a single node in $T$, $t[\alpha(i)] = t[\beta(i)]$. An example of this mapping is given in Figure 2.10. When $\tilde{T}$ is h-reduced to yield $\hat{T}$, $\alpha(i)$ and $\beta(i)$ apply in the same way to the mapping between $\hat{T}$ and $\tilde{T}$.

It is important to note that a reduced tree is just a compact representation of the original tree. Therefore, given trees $(T_1, T_2)$ and their reduced trees $(\tilde{T}_1, \tilde{T}_2)$, the relation $d(\tilde{T}_1, \tilde{T}_2) = d(T_1, T_2)$ is implied.

### 2.4 Incorporating Vertical Linearity

The Zhang-Shasha algorithm is conceptually the simplest to understand. Therefore we choose this algorithm to demonstrate the techniques of incorporating vertical linearity. We will then show that the techniques can apply to the entire family of
cover strategy-based algorithms.

2.4.1 The Zhang-Shasha Algorithm

The Zhang-Shasha algorithm is based on the following recursions.

Lemma 2.1. \cite{12}

1. \( d(\emptyset, \emptyset) = 0 \).

2. \( \forall i \in T_1, \forall i' \in \{l(i), \ldots, i\} \),

\[
d(F_1[l(i) \cdots i'], \emptyset) = d(F_1[l(i) \cdots i' - 1], \emptyset) + d(t_1[i'], \emptyset) .
\]

3. \( \forall j \in T_2, \forall j' \in \{l(j), \ldots, j\} \),

\[
d(\emptyset, F_2[l(j) \cdots j']) = d(\emptyset, F_2[l(j) \cdots j' - 1]) + d(\emptyset, t_2[j']) .
\]

Lemma 2.2. \cite{12} \( \forall (i, j) \in (T_1, T_2), \forall i' \in \{l(i), \ldots, i\} \) and \( \forall j' \in \{l(j), \ldots, j\} \),
if \( l(i') = l(i) \) and \( l(j') = l(j) \),

\[
d(F_1[l(i) \cdots i'], F_2[l(j) \cdots j']) = \min \left\{ \begin{array}{l}
d(F_1[l(i) \cdots i' - 1], F_2[l(j) \cdots j']) + d(t_1[i'], \emptyset), \\
d(F_1[l(i) \cdots i'], F_2[l(j) \cdots j' - 1]) + d(\emptyset, t_2[j']), \\
d(F_1[l(i) \cdots i' - 1], F_2[l(j) \cdots j' - 1]) + d(t_1[i'], t_2[j'])
\end{array} \right\};
\]

otherwise,

\[
d(F_1[l(i) \cdots i'], F_2[l(j) \cdots j']) = \min \left\{ \begin{array}{l}
d(F_1[l(i) \cdots i' - 1], F_2[l(j) \cdots j']) + d(t_1[i'], \emptyset), \\
d(F_1[l(i) \cdots i'], F_2[l(j) \cdots j' - 1]) + d(\emptyset, t_2[j']), \\
d(F_1[l(i) \cdots l(i' - 1)], F_2[l(j) \cdots l(j') - 1]) + d(T_1[i'], T_2[j'])
\end{array} \right\}.
\]

The main loop body of the Zhang-Shasha algorithm is shown in Algorithm 1.

Each loop computes a subtree-subtree distance where the roots of the subtrees are referred to as LR-keyroots.

**Algorithm 1**: Computing \( d(T_1, T_2) \)

| input : \( (T_1, T_2) \) | output: \( d(T_1[i], T_2[j]), \) where \( 1 \leq i \leq |T_1| \) and \( 1 \leq j \leq |T_2| \) |
|---|---|
| 1 compute \( keyroots(T_1) \) and \( keyroots(T_2) \) |
| 2 sort \( (keyroots(T_1), keyroots(T_2)) \) in increasing order into arrays \( (K_1, K_2) \) |
| 3 for \( i' \leftarrow 1 \) to \( |keyroots(T_1)| \) do |
| 4 \hspace{1em} for \( j' \leftarrow 1 \) to \( |keyroots(T_2)| \) do |
| 5 \hspace{2.5em} \( i \leftarrow K_1[i'] \) |
| 6 \hspace{2.5em} \( j \leftarrow K_2[j'] \) |
| 7 \hspace{2.5em} LRKeyRoots\((i, j)\) |
| 8 \hspace{1em} endfor |
| 9 endfor |
Algorithm 2: LRKeyRoots$(i, j)$

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$d(\emptyset, \emptyset) \leftarrow 0$</td>
</tr>
<tr>
<td>2</td>
<td>for $i' \leftarrow l(i)$ to $i$ do</td>
</tr>
<tr>
<td>3</td>
<td>$d(F_1[l(i) \cdots i'], \emptyset) \leftarrow d(F_1[l(i) \cdots i' - 1], \emptyset) + d(t_1[i'], \emptyset)$</td>
</tr>
<tr>
<td>4</td>
<td>endfor</td>
</tr>
<tr>
<td>5</td>
<td>for $j' \leftarrow l(j)$ to $j$ do</td>
</tr>
<tr>
<td>6</td>
<td>$d(\emptyset, F_2[l(j) \cdots j']) \leftarrow d(\emptyset, F_2[l(j) \cdots j' - 1]) + d(\emptyset, t_2[j'])$</td>
</tr>
<tr>
<td>7</td>
<td>endfor</td>
</tr>
<tr>
<td>8</td>
<td>for $i' \leftarrow l(i)$ to $i$ do</td>
</tr>
<tr>
<td>9</td>
<td>for $j' \leftarrow l(j)$ to $j$ do</td>
</tr>
<tr>
<td>10</td>
<td>if $l(i') = l(i)$ and $l(j') = l(j)$ then</td>
</tr>
<tr>
<td>11</td>
<td>$d(F_1[l(i) \cdots i'], F_2[l(j) \cdots j']) \leftarrow$</td>
</tr>
<tr>
<td></td>
<td>$\min \left{ \begin{array}{l} d(F_1[l(i) \cdots i' - 1], F_2[l(j) \cdots j']) + d(t_1[i'], \emptyset), \ d(F_1[l(i) \cdots i'], F_2[l(j) \cdots j' - 1]) + d(\emptyset, t_2[j']), \ d(F_1[l(i) \cdots i' - 1], F_2[l(j) \cdots j' - 1]) + d(t_1[i'], t_2[j']) \end{array} \right}$</td>
</tr>
<tr>
<td>12</td>
<td>$d(T_1[i'], T_2[j']) \leftarrow d(F_1[l(i) \cdots i'], F_2[l(j) \cdots j'])$</td>
</tr>
<tr>
<td>13</td>
<td>endif</td>
</tr>
<tr>
<td>14</td>
<td>else</td>
</tr>
<tr>
<td>15</td>
<td>$d(F_1[l(i) \cdots i'], F_2[l(j) \cdots j']) \leftarrow$</td>
</tr>
<tr>
<td></td>
<td>$\min \left{ \begin{array}{l} d(F_1[l(i) \cdots i' - 1], F_2[l(j) \cdots j']) + d(t_1[i'], \emptyset), \ d(F_1[l(i) \cdots i'], F_2[l(j) \cdots j' - 1]) + d(\emptyset, t_2[j']), \ d(F_1[l(i) \cdots i' - 1], F_2[l(j) \cdots (j' - 1)]) + d(T_1[i'], T_2[j']) \end{array} \right}$</td>
</tr>
<tr>
<td>16</td>
<td>endif</td>
</tr>
<tr>
<td>17</td>
<td>endfor</td>
</tr>
<tr>
<td>18</td>
<td>endfor</td>
</tr>
</tbody>
</table>
2.4.2 Properties

We denote by \( d(x, y) \) the edit distance between \( x \) and \( y \). The following lemmas incorporate vertical linearity in the Zhang-Shasha algorithm.

**Lemma 2.3.**

1. \( d(\emptyset, \emptyset) = 0 \).

2. \( \forall i \in \widetilde{T}_1, \forall i' \in \{l(i), \cdots, i\}, \)

\[
d(\widetilde{F}_1[l(i) \cdots i'], \emptyset) = d(\widetilde{F}_1[l(i) \cdots i' - 1], \emptyset) + d(i', \emptyset).
\]

3. \( \forall j \in \widetilde{T}_2, \forall j' \in \{l(j), \cdots, j\}, \)

\[
d(\emptyset, \widetilde{F}_2[l(j) \cdots j']) = d(\emptyset, \widetilde{F}_2[l(j) \cdots j' - 1]) + d(\widetilde{F}_2[j'], \emptyset).
\]

**Proof.** Case 1 requires no edit operation. In case 2 and case 3, the distances correspond to the costs of deleting and inserting the nodes in \( \widetilde{F}_1[l(i) \cdots i'] \) and \( \widetilde{F}_2[l(j) \cdots j'] \), respectively. \( \square \)

**Lemma 2.4.** \( \forall (i, j) \in (\widetilde{T}_1, \widetilde{T}_2), \forall i' \in \{l(i), \cdots, i\} \) and \( \forall j' \in \{l(j), \cdots, j\}, \)

if \( l(i') = l(i) \) and \( l(j') = l(j) \),

\[
d(\widetilde{F}_1[l(i) \cdots i'], \widetilde{F}_2[l(j) \cdots j']) = d(\widetilde{T}_1[i'], \widetilde{T}_2[j']) ;
\]

otherwise,

\[
d(\widetilde{F}_1[l(i) \cdots i'], \widetilde{F}_2[l(j) \cdots j']) = \min \left\{ \begin{array}{l} 
d(\widetilde{F}_1[l(i) \cdots i' - 1], \widetilde{F}_2[l(j) \cdots j']), \\d(\widetilde{T}_1[i'], \emptyset), \end{array} \right\}.
\]
Proof. The condition “\(l(i') = l(i)\) and \(l(j') = l(j)\)” implies that the two forests are simply two trees and the equality clearly holds. We now consider the other condition in which “\(l(i') \neq l(i)\) or \(l(j') \neq l(j)\)”. If \(t_1[\alpha(i')] = t_1[\beta(i')]\) and \(t_2[\alpha(j')] = t_2[\beta(j')]\), the formula holds as an obvious result. Otherwise, at least one of \(\tilde{t}_1[i']\) and \(\tilde{t}_2[j']\) corresponds to a \(v\)-component in \((T_1[\alpha(i)], T_2[\alpha(j)])\). Consider the components in \((T_1[\alpha(i)], T_2[\alpha(j)])\) corresponding to \((\tilde{t}_1[i'], \tilde{t}_2[j'])\). There are two cases to consider: either (1) there is no occurrence of node-to-node match between the components; or (2) there is at least one occurrence of node-to-node match between the components. In case 1, one of the components must be entirely deleted which implies that either \(\tilde{t}_1[i']\) must be deleted or \(\tilde{t}_2[j']\) must be inserted. In case 2, in order to preserve the ancestor-descendant relationship \(\tilde{T}_1[i']\) and \(\tilde{T}_2[j']\) must be matched. \(\square\)

Note. In Lemma 2.4 for the condition “\(l(i') \neq l(i)\) or \(l(j') \neq l(j)\)” the value of \(d(\tilde{T}_1[i'], \tilde{T}_2[j'])\) would already be available if implemented in a bottom-up order, since it involves a subproblem of \(d(\tilde{F}_1[l(i) \cdots i'], \tilde{F}_2[l(j) \cdots j'])\) and would have been computed. For the condition “\(l(i') = l(i)\) and \(l(j') = l(j)\)” however, we encounter the problem involving \((\tilde{T}_1[i'], \tilde{T}_2[j'])\) for the first time and must compute its value.

We show how to compute \(d(\tilde{T}_1[i'], \tilde{T}_2[j'])\) in the following lemmas.

Lemma 2.5. \(\forall u \in \{\beta(i'), \cdots, \alpha(i')\}\),

\[
d(T_1[u], F_2[\beta(j')]) = \
\min \left\{ \begin{array}{l}
d(F_1[u], F_2[\beta(j')]) + d(t_1[u], \emptyset), \\
\min_{j_1 \leq j \leq j'} \{d(T_1[u], T_2[\alpha(q)]) - d(\emptyset, T_2[\alpha(q)])\} + d(\emptyset, F_2[\beta(j')])
\end{array} \right\}.
\]

Proof. This is the edit distance between the tree \(T_1[u]\) and the forest \(F_2[\beta(j')]\). There are two cases. In the first case, \(t_1[u]\) is constrained to be deleted and the remaining substructure \(F_1[u]\) is matched to \(F_2[\beta(j')]\). In the second case, \(t_1[u]\) is constrained to be matched to a node somewhere in \(F_2[\beta(j')]\). This is equivalent to stating that \(T_1[u]\) is constrained to be matched to a subtree in \(F_2[\beta(j')]\). The question thus becomes...
finding a subtree in $F_2[\beta(j')]$ to be matched to $T_1[u]$ so as to minimize the distance between $T_1[u]$ and $F_2[\beta(j')]$ under such constraint. This can be done by considering the set of all combinations in which exactly one tree in $F_2[\beta(j')]$ is matched to $T_1[u]$ while the remainder of $F_2[\beta(j')]$ is deleted. The minimum in this set is the edit distance for the second case.

Lemma 2.6. $\forall v \in \{\beta(j'), \cdots, \alpha(j')\}$,

$$d(F_1[\beta(i')], T_2[v]) = \min \left\{ d(F_1[\beta(i')], F_2[v]) + d(\emptyset, t_2[v]), \quad \min_{1 \leq p \leq t_2} \{ d(T_1[\alpha(p)], T_2[v]) - d(T_1[\alpha(p), \emptyset]) + d(F_1[\beta(i')], \emptyset) \} \right\}.$$

Proof. This is symmetric to that of Lemma 2.5.

Lemma 2.7. $\forall u \in \{\beta(i'), \cdots, \alpha(i')\}$ and $\forall v \in \{\beta(j'), \cdots, \alpha(j')\}$,

$$d(T_1[u], T_2[v]) = \min \left\{ d(F_1[u], T_2[v]) + d(t_1[u], \emptyset), \quad d(T_1[u], F_2[v]) + d(\emptyset, t_2[v]), \quad d(F_1[u], F_2[v]) + d(t_1[u], t_2[v]) \right\}.$$

Proof. This is a known result for the tree-to-tree edit distance.

Note. In the computation for every $d(\widetilde{T}_1[i'], \widetilde{T}_2[j'])$, we save the values of $d(T_1[u], T_2[\alpha(j')])$ for all $u \in \{\beta(i'), \cdots, \alpha(i')\}$ and $d(T_1[\alpha(i')], T_2[v])$ for all $v \in \{\beta(j'), \cdots, \alpha(j')\}$. This ensures that when $d(T_1[u], F_2[\beta(j')])$ in Lemma 2.5 and $d(F_1[\beta(i')], T_2[v])$ in Lemma 2.6 are evaluated in a bottom-up order the values of the terms involving $d(T_1[u], T_2[\alpha(q)])$ and $d(T_1[\alpha(p)], T_2[v])$ would be available.

Lemma 2.8. $d(\widetilde{T}_1[i'], \widetilde{T}_2[j']) = d(T_1[\alpha(i')], T_2[\alpha(j')])$.

Proof. The result follows from the tree definitions.

We give figures to illustrate the above lemmas for computing $d(\widetilde{T}_1[i'], \widetilde{T}_2[j'])$ as follows. When this subproblem comes into scene for the first time during the compu-
tation, $d(F_1[\beta(\ell')], F_2[\beta(j')])$ has already been computed. This situation is depicted in Figure 2.11. The computation proceeds by executing the lemmas in the following order: Lemma 2.5, Lemma 2.6, Lemma 2.7. The scenarios for these lemmas are depicted in Figure 2.12, 2.13 and 2.14.

![Figure 2.11: $F_1[\beta(\ell')]$ and $F_2[\beta(j')]$.](image1)

![Figure 2.12: $T_1[\alpha(\ell')]$ and $F_2[\beta(j')]$ in Lemma 2.5.](image2)

### 2.4.3 Algorithmic Improvements

With dynamic programming, it is possible that the solution of one subproblem may be obtained as a by-product of computing another subproblem. For example, in the rightmost recursion strategy the solutions of two subproblems may be obtained in one run of computation if the relevant sub-forests in the smaller subproblem are the prefixes of the relevant sub-forests in the larger subproblem. In this case, the solution of the smaller subproblem is obtained as a by-product of the computation of the
larger subproblem. If, however, the solution of a subproblem can not be obtained as a by-product of solving another subproblem, then it must be computed separately. We refer to this kind of subproblems as disjoint subproblems.

**Definition 2.6** (Disjoint Subproblems). *The set of disjoint subproblems is such that the solution of any member of this set can not be obtained as a by-product of solving another subproblem.*

For the sake of efficiency, it is crucial to identify the set of disjoint subproblems as such set forms a cover of all the subproblems the solutions of which are necessary for building the final solution. We describe an important concept, namely key roots, for the purpose of identifying the set of disjoint subproblems, as follows.

For every node $i$ of tree $T$, we designate a child of $i$, if any, to be its special child, denoted by $sc(i)$. Note that in the Zhang-Shasha algorithm $sc(i)$ is the leftmost child of $i$ whereas in a different recursion strategy the choice of $sc(i)$ may be different.
Denote by \( p(i) \) the parent of \( i \). We define a set of nodes, called \textit{key roots}, for tree \( T \) as follows.

\[
\text{keyroots}(T) = \{ k \mid k = \text{root}(T) \text{ or } k \neq \text{sc}(p(k)) \}.
\]

This is a generalized version of the \textit{LR_keyroots} used in [12] and is suitable for any known recursion strategy as in [3, 5, 12]. Referring to Figure 2.2, in every special path the highest numbered node in a left-to-right post-order is a key root. Here is the intuitive meaning of key root. Suppose, for example, that the recursion strategy is rightmost and let \( i \in \text{keyroots}(T_1) \) and \( j \in \text{keyroots}(T_2) \), then \( d(T_1[i], T_2[j]) \) is a disjoint subproblem; that is, it requires a separate computation.

We now present the new algorithm. The entry point is Algorithm 3. Algorithm 3 contains the main loop which repeatedly calls the procedure \textit{KeyRoots} to compute \( d(\tilde{T}_1[i], \tilde{T}_2[j]) \) where \((i, j)\) are key roots in \((\tilde{T}_1, \tilde{T}_2)\). The procedure \textit{KeyRoots} is divided into several parts with each part handled in a separate procedure. The following notations are used.

- \( D_t \): a \((|T_1| + 1) \times (|T_2| + 1)\) two-dimensional permanent array.
- \( \tilde{D}_t \): a \((|	ilde{T}_1| + 1) \times (|	ilde{T}_2| + 1)\) two-dimensional permanent array.
- \( \tilde{D}_f \): a \((|	ilde{T}_1| + 1) \times (|	ilde{T}_2| + 1)\) two-dimensional temporary array.
- \( A_1, A_2 \): temporary one-dimensional arrays of lengths \((|T_1| + 1)\) and \((|T_2| + 1)\), respectively.

\( D_t \) is used to store distances with respect to the \((T_1, T_2)\) representation. \( \tilde{D}_t \) is used to store distances with respect to the \((\tilde{T}_1, \tilde{T}_2)\) representation. The computation for forest-to-forest distances is done in \( \tilde{D}_f \). \( A_1 \) and \( A_2 \) are used in conjunction with \( D_t \) to handle boundary initializations. The temporary arrays \( \tilde{D}_f \), \( A_1 \) and \( A_2 \) allow rewriting of their contents.

\textbf{Lemma 2.9.} The new algorithm correctly computes \( d(T_1, T_2) \).
Algorithm 3: Computing $d(T_1, T_2)$

| input : $(T_1, T_2)$ | output: matrix $\tilde{D}_t$ containing $d(T_1[\alpha(i)], T_2[\alpha(j)])$, where $1 \leq i \leq |\tilde{T}_1|$, $1 \leq j \leq |\tilde{T}_2|$ |
|----------------------|------------------------------------------------|
| 1 build $(\tilde{T}_1, \tilde{T}_2)$ | 2 compute $\textit{keyroots}(\tilde{T}_1)$ and $\textit{keyroots}(\tilde{T}_2)$ |
| 3 sort $(\textit{keyroots}(\tilde{T}_1), \textit{keyroots}(\tilde{T}_2))$ in increasing order into arrays $(K_1, K_2)$ | 4 for $i' \leftarrow 1$ to $|\textit{keyroots}(\tilde{T}_1)|$ do |
| 5 | for $j' \leftarrow 1$ to $|\textit{keyroots}(\tilde{T}_2)|$ do |
| 6 | $i \leftarrow K_1[i']$ |
| 7 | $j \leftarrow K_2[j']$ |
| 8 | $\textit{KeyRoots}(i, j)$ |
| 9 | endfor |
| 10 | endfor |

Algorithm 4: $\textit{KeyRoots}(i, j)$

1 $\textit{BoundaryConditions}(i, j)$
2 $\textit{MainRecursion}(i, j)$

Algorithm 5: $\textit{BoundaryConditions}(i, j)$

1 $\tilde{D}_f[l(i) - 1, l(j) - 1] \leftarrow 0$
2 for $i' \leftarrow l(i)$ to $i$ do
3 | $D_f[i', l(j) - 1] \leftarrow \tilde{D}_f[i' - 1, l(j) - 1] + \delta(l_1[i'], \emptyset)$
4 endfor
5 for $j' \leftarrow l(j)$ to $j$ do
6 | $D_f[l(i) - 1, j'] \leftarrow \tilde{D}_f[l(i) - 1, j' - 1] + \delta(\emptyset, \tilde{t}_2[j'])$
7 endfor

Algorithm 6: $\textit{MainRecursion}(i, j)$

1 for $i' \leftarrow l(i)$ to $i$ do
2 | for $j' \leftarrow l(j)$ to $j$ do
3 | | if $l(i') = l(i)$ and $l(j') = l(j)$ then
4 | | | $\textit{TreeDist}(i', j')$
5 | | else
6 | | | $\textit{ForestDist}(i', j')$
7 | | endif
8 | endfor
9 | endfor
### Algorithm 7: TreeDist($i', j'$)

1. for $u \leftarrow \beta(i') - 1$ to $\alpha(i')$
   2. \hspace{1em} $A_1[u] \leftarrow D_t[u, \beta(j') - 1]$
   3. endfor
4. for $v \leftarrow \beta(j')$ to $\alpha(j')$
   5. \hspace{1em} $A_2[v] \leftarrow D_t[\beta(i') - 1, v]$
   6. endfor
7. $D_t[\beta(i') - 1, \beta(j') - 1] \leftarrow \tilde{D}_t[i' - 1, j' - 1]$
8. for $u \leftarrow \beta(i')$ to $\alpha(i')$
   9. \hspace{1em} $D_t[u, \beta(j') - 1] \leftarrow$
   10. \hspace{2em} $\min \left\{ D_t[u - 1, \beta(j') - 1] + \delta(t_1[u], \emptyset),$
   11. \hspace{3em} $\min_{\beta \leq q \leq \alpha} \{ D_t[u, \alpha(q)] - \delta(\emptyset, T_2[\alpha(q)]) \} + \delta(\emptyset, F_2[\beta(j')]) \right\}$
   12. endfor
13. for $v \leftarrow \beta(j')$ to $\alpha(j')$
   14. \hspace{1em} $D_t[\beta(i') - 1, v] \leftarrow$
   15. \hspace{2em} $\min \left\{ D_t[\beta(i') - 1, v - 1] + \delta(\emptyset, t_2[v]),$
   16. \hspace{3em} $\min_{\beta \leq p \leq \alpha} \{ D_t[\alpha(p), v] - \delta(T_1[\alpha(p)], \emptyset) \} + \delta(F_1[\beta(i')], \emptyset) \right\}$
   17. endfor
18. for $u \leftarrow \beta(i')$ to $\alpha(i')$
   19. \hspace{1em} for $v \leftarrow \beta(j')$ to $\alpha(j')$
   20. \hspace{2em} $D_t[u, v] \leftarrow \min \left\{ D_t[u - 1, v] + \delta(t_1[u], \emptyset),$
   21. \hspace{3em} $D_t[u, v - 1] + \delta(\emptyset, t_2[v]),$
   22. \hspace{4em} $D_t[u - 1, v - 1] + \delta(t_1[u], t_2[v]) \right\}$
   23. endfor
19. $\tilde{D}_t[i', j'] \leftarrow D_t[\alpha(i'), \alpha(j')]$
20. $\tilde{D}_t[i', j'] \leftarrow D_t[\alpha(i'), \alpha(j')]$
21. for $u \leftarrow \beta(i') - 1$ to $\alpha(i')$
22. \hspace{1em} $D_t[u, \beta(j') - 1] \leftarrow A_1[u]$
23. endfor
24. for $v \leftarrow \beta(j')$ to $\alpha(j')$
25. \hspace{1em} $D_t[\beta(i') - 1, v] \leftarrow A_2[v]$
26. endfor

### Algorithm 8: ForestDist($i', j'$)

\[
\tilde{D}_f[i', j'] \leftarrow \min \left\{ \begin{array}{l}
\tilde{D}_f[i' - 1, j'] + \delta(\tilde{t}_1[i'], \emptyset), \\
\tilde{D}_f[i', j' - 1] + \delta(\emptyset, \tilde{t}_2[j']), \\
\tilde{D}_f[l(i') - 1, l(j') - 1] + \tilde{D}_t[i', j'] \end{array} \right\}
\]
Proof. The correctness of all the computed values in the new algorithm follows from the lemmas. By Lemma 2.8, \( d(\tilde{T}_1, \tilde{T}_2) = d(T_1, T_2) \) when \((i', j')\) are set to be the roots of \((\tilde{T}_1, \tilde{T}_2)\). Since these roots are key roots, \( d(T_1, T_2) \) is always computed by the algorithm.

Lemma 2.10. Given \( T_1 \) and \( T_2 \), the new algorithm runs in \( \mathcal{O}(|T_1| \times |T_2| + |\tilde{T}_1| \times |\tilde{T}_2| \times \min\{\text{depth}(\tilde{T}_1), \text{leaves}(\tilde{T}_1)\} \times \min\{\text{depth}(\tilde{T}_2), \text{leaves}(\tilde{T}_2)\}) \) time and \( \mathcal{O}(|T_1| \times |T_2|) \) space, with \( |\tilde{T}_1| \leq |T_1| \) and \( |\tilde{T}_2| \leq |T_2| \).

Proof. We first consider the time complexity. \( \tilde{T}_1 \) and \( \tilde{T}_2 \) can be built in linear time in a preprocess. Identifying and sorting the key roots can be done in linear time. All the values associated with insertion or deletion of a subtree or a sub-forest, as in Lemma 2.5 and Lemma 2.6, can be obtained beforehand in linear time during a tree traversal. The Zhang-Shasha algorithm runs in \( \mathcal{O}(|\tilde{T}_1| \times |\tilde{T}_2| \times \min\{\text{depth}(\tilde{T}_1), \text{leaves}(\tilde{T}_1)\} \times \min\{\text{depth}(\tilde{T}_2), \text{leaves}(\tilde{T}_2)\}) \) time for \((\tilde{T}_1, \tilde{T}_2)\). We consider the loops in the procedure \( \text{TreeDist} \) which involve computations related to the \((i', j')\) pairs on leftmost paths, based on Lemma 2.5, 2.6 and 2.7. This is the part of the algorithm that explicitly traverses in \((T_1, T_2)\). For each such \((i', j')\) pair, this part takes \( \mathcal{O}((\alpha(i') - \beta(i') + 1) \times (j' - j'_1 + 1) + (\alpha(j') - \beta(j') + 1) \times (i' - i'_1 + 1) + (\alpha(i') - \beta(i') + 1) \times (\alpha(j') - \beta(j') + 1)) \). All the subproblems associated with these \((i', j')\) pairs are disjoint. Summing over all these pairs, we can bound the complexity by \( \mathcal{O}(|T_1| \times |T_2|) \). Hence, the overall time complexity follows.

By considering the sizes of the data structures used, the space complexity clearly follows.

Theorem 2.1. Given \((T_1, T_2)\), the edit distance \( d(T_1, T_2) \) can be computed in \( \mathcal{O}(|T_1| \times |T_2| + \mathcal{T}(A, \tilde{T}_1, \tilde{T}_2)) \) time where \( \mathcal{T}(A, \tilde{T}_1, \tilde{T}_2) \) is the time complexity of any strategy-based algorithm \( A \) applied to \((\tilde{T}_1, \tilde{T}_2)\).

Proof. Since a v-component corresponds to a simple path, there is only one way a dynamic program may recurse along this path regardless which strategy is used.
Hence, Lemma 2.5 to 2.8 are valid for all cover strategies. Lemma 2.3 and 2.4, after a proper adjustment of the subtree orderings in each forest to adapt to the given strategy, are also valid. The theorem is implied from Lemma 2.9 and 2.10 when the relevant lemmas are properly embedded in any strategy-based algorithm.

The above result is an improvement over the original algorithms. This can be easily seen as $|T_1| \times |T_2| < \mathcal{T}(A, T_1, T_2)$ and $\mathcal{T}(A, \tilde{T}_1, \tilde{T}_2) \leq \mathcal{T}(A, T_1, T_2)$.

### 2.4.4 Application

We describe one application which would benefit from our result, namely RNA secondary structure comparison. RNA is a molecule consisting of a single strand of nucleotides (abbreviated as $A$, $C$, $G$ and $U$) which folds back upon itself by means of hydrogen bonding between distant complementary nucleotides, giving rise to a so-called secondary structure. The secondary structure of an RNA molecule can be topologically represented by a tree. An example is depicted in Figure 2.15.

In this representation, an internal node represents a pair of complementary nucleotides interacting via hydrogen bonding. When a number of such pairs are stacked up, they form a local structure called stem, which corresponds to a v-component in the tree representation. The secondary structure plays an important role in the functions of RNA [7]. Therefore, comparing the secondary structures of RNA molecules can help understand their comparative functions. One way to compare two trees is to compute the edit distance between them.

As an example, we list the size reductions for a set of tRNA molecules in Table 2.2. The last column shows the size reductions in percentage. On average, we observe a size reduction by nearly one third of the original size, which roughly translates into a one-half decrease of running time for the known strategy-based algorithms.
Figure 2.15: The secondary structure of RNA and the corresponding tree representation. The dotted lines in the secondary structure represent hydrogen bonds.

Table 2.2: Vertical reduction of tree sizes for selected tRNA molecules [10].
2.5 Incorporating Horizontal Linearity

The incorporation of horizontal linearity relies on certain properties pertaining to the edit distance between two forests both suffixed with a horizontal component. These properties allow us to relate the current problem to the problem of finding row minima in a totally monotone matrix.

In the following, we describe the problem of finding row minima in a totally monotone matrix. We then identify and discuss the properties that enable us to establish the connection between this problem and the problem we want to solve. This connection makes it feasible to solve our problem by taking advantage of existing efficient algorithms for finding row minima in a totally monotone matrix.

2.5.1 Properties

Definition 2.7 (Monge Condition [6]). A cost function $w$ satisfies the (convex) Monge condition if $\forall \ a < b$ and $c < d$,

$$w(a, c) + w(b, d) \leq w(b, c) + w(a, d) \ .$$

Since $w(a, c) - w(a, d) \leq w(b, c) - w(b, d)$, therefore $w(a, c) - w(a, d) \geq 0$ implies $w(b, c) - w(b, d) \geq 0$. This leads to the following property.

Definition 2.8 (Totally Monotone Matrix [1]). An $m \times n$ matrix $A$ is (convex) totally monotone if and only if $\forall \ a < b$ and $c < d$,


We now define some shorthand notations related to an $m \times n$ matrix $A$. Denote by $row(A, i)$ and $col(A, j)$ the $i$th row and the $j$th column of $A$, respectively. Denote by $A[i \cdots i', j \cdots j']$ the sub-matrix of $A$ that is the intersection of rows $row(A, i), \cdots, row(A, i')$ and columns $col(A, j), \cdots, col(A, j')$. 
Denote by \( \text{indmin}(\text{row}(A, i)) \) the minimum column index such that \( A[i, \text{indmin}(\text{row}(A, i))] \) equals the minimum value in \( \text{row}(A, i) \). An important consequence of total monotonicity is

\[
\text{indmin}(\text{row}(A, 1)) \leq \text{indmin}(\text{row}(A, 2)) \leq \cdots \leq \text{indmin}(\text{row}(A, n))
\]

One problem which is closely related to our purpose and can be solved efficiently by taking advantage of the total monotonicity is the problem of finding row minima for a totally monotone matrix. To find the row minima of a totally monotone matrix \( A_{m \times n} \), a linear time solution is available [1]. The solution is given in Algorithm 10 which executes Algorithm 9 in the first step. The time complexity is \( O(m + n) \) and the space complexity is \( O(\max\{m, n\}) \).

---

**Algorithm 9: [1] Reduce\((A_{m \times n})\)**  

1. \( B \leftarrow A \)  
2. \( i \leftarrow 1 \)  
3. \( \text{while number of columns in } B > m \text{ do} \)  
   4. \( \text{if } B[i, i] \geq B[i, i + 1] \text{ then} \)  
   5. \( \quad \text{delete } \text{col}(B, i) \)  
   6. \( \quad i \leftarrow \min \{1, i - 1\} \)  
   7. \( \text{else} \)  
   8. \( \quad \text{if } i < m \text{ then} \)  
   9. \( \quad \quad i \leftarrow i + 1 \)  
  10. \( \quad \text{else} \)  
  11. \( \quad \quad \text{delete } \text{col}(B, i + 1) \)  
  12. \( \quad \text{endif} \)  
  13. \( \text{endif} \)  
14. \( \text{endw} \)  
15. \( \text{return } B \)  

---

In our problem setting, we are concerned with a pair of forests of the form \((F \bullet h_1[1 \cdots l_1], G \bullet h_2[1 \cdots l_2])\) where \( h_1 \) and \( h_2 \) represent two sequences of nodes of lengths \( l_1 \) and \( l_2 \), respectively. The dynamic programming table corresponding to this part of the computation is depicted in Figure 2.16. The lower-right block of the table corresponds to \( d(F \bullet h_1[1 \cdots i], G \bullet h_2[1 \cdots j]) \) for \( 1 \leq i \leq l_1 \) and \( 1 \leq j \leq l_2 \). Recursively
Algorithm 10: \[1\text{] RowMin}(A_{m \times n})

1. \(B \leftarrow \text{Reduce}(A)\)
2. \(\text{if } m = 1 \text{ then}\)
3. \(|\) output the minimum and return
4. \(\text{endif}\)
5. \(C \leftarrow \{\text{row}(B, 2), \text{row}(B, 4), \cdots, \text{row}(B, 2 \times \lfloor m/2 \rfloor)\}\)
6. \(\text{RowMin}(C)\)
7. from the known positions of the minima in the even rows of \(B\), find the minima in its odd rows

Figure 2.16: Dynamic programming table for \(d(F \bullet h_1, G \bullet h_2)\).
throughout the entire \((F \bullet h_1[1 \cdots l_1], G \bullet h_2[1 \cdots l_2])\), our objective is that instead of computing all the \((l_1 \times l_2)\) entries corresponding to \(d(F \bullet h_1[1 \cdots i], G \bullet h_2[1 \cdots j])\) for \(1 \leq i \leq l_1\) and \(1 \leq j \leq l_2\), we compute only \((l_1 + l_2 - 1)\) entries corresponding to \(d(F \bullet h_1[1 \cdots i], G \bullet h_2[1 \cdots j])\) for \(1 \leq i \leq l_1\) and \(d(F \bullet h_1[1 \cdots i], G \bullet h_2[1 \cdots i])\) for \(1 \leq i \leq l_2\) as highlighted by the shaded region in the lower-right block. The matter of fact is that computing only these entries instead of the entire block is sufficient for the computation to continue beyond this pair of forests. However, this also incurs an additional treatment in order to carry out the computation in this fashion. The justification of this approach is that the time saved on the omitted computation exceeds the time spent in carrying out the additional treatment.

We describe this approach as follows. The next two lemmas describe what information is involved in computing \(d(F \bullet h_1[1 \cdots i], G \bullet h_2[1 \cdots l_2])\) for \(1 \leq i \leq l_1\) and \(d(F \bullet h_1[1 \cdots l_1], G \bullet h_2[1 \cdots i])\) for \(1 \leq i \leq l_2\).

**Lemma 2.11.** \(\forall i \in \{1, \cdots, l_1\}\),

\[
\begin{align*}
\min \left\{ \min_{0 \leq j \leq i} \{d(F \bullet h_1[1 \cdots j], G) + d(h_1[j + 1 \cdots i], h_2[1 \cdots l_2])\}, \min_{0 \leq j \leq i} \{d(F, G \bullet h_2[1 \cdots j]) + d(h_1[1 \cdots i], h_2[j + 1 \cdots l_2])\} \right\} = d(F \bullet h_1[1 \cdots i], G \bullet h_2[1 \cdots l_2])
\end{align*}
\]

(2.6)

**Proof.** Denote by \(\langle F, G \rangle\) the edit script for \(d(F, G)\). We partition \((F \bullet h_1[1 \cdots i], G \bullet h_2[1 \cdots l_2])\) into \(P_1 = (F \bullet h_1[1 \cdots p], G \bullet h_2[1 \cdots q])\) and \(P_2 = (h_1[p + 1 \cdots i], h_2[q + 1 \cdots l_2])\) such that the leftmost edit step \(\langle a, b \rangle\) in \(\langle F \bullet h_1[1 \cdots i], G \bullet h_2[1 \cdots l_2]\rangle\) where \(a \in h_1[1 \cdots i]\) and \(b \in h_2[1 \cdots l_2]\), if one exists, is \(\langle h_1[p + 1], h_2[q + 1] \rangle\). If such edit step does not exist, then \(P_2 = \emptyset\). As to \(h_1[1 \cdots p]\) and \(h_2[1 \cdots q]\), since there can not exist any match between them (otherwise, \(\langle h_1[p + 1], h_2[q + 1] \rangle\) would not be the leftmost edit step we are seeking), they can either involve indels only or matches with \(G\) or \(F\). There are three cases.
1. \(h_1[1 \ldots p]\) and \(h_2[1 \ldots q]\) involve only indels:

\[
d(F \bullet h_1[1 \ldots i], G \bullet h_2[1 \ldots l]) = d(F \bullet h_1[1 \ldots p], G) + d(h_1[p + 1 \ldots i], h_2[1 \ldots l])
\]

2. \(h_1[1 \ldots p]\) involves matches with \(G\) and \(h_2[1 \ldots q]\) involves indels only: 

\[
d(F \bullet h_1[1 \ldots i], G \bullet h_2[1 \ldots l]) = d(F \bullet h_1[1 \ldots p], G) + d(h_1[p + 1 \ldots i], h_2[1 \ldots l])
\]

3. \(h_1[1 \ldots p]\) involves indels only and \(h_2[1 \ldots q]\) involves matches with \(F\):

\[
d(F \bullet h_1[1 \ldots i], G \bullet h_2[1 \ldots l]) = d(F, G \bullet h_2[1 \ldots q]) + d(h_1[1 \ldots i], h_2[q + 1 \ldots l])
\]

Note that it is not possible that both \(h_1[1 \ldots p]\) and \(h_2[1 \ldots q]\) involve matches with \(G\) and \(F\), respectively, since this would violate the basic requirement that the ordering of the tree nodes be preserved in the edit script.

Putting together these cases, we have

\[
d(F \bullet h_1[1 \ldots i], G \bullet h_2[1 \ldots l]) = \min \left\{ d(F \bullet h_1[1 \ldots p], G) + d(h_1[p + 1 \ldots i], h_2[1 \ldots l]), d(F, G \bullet h_2[1 \ldots q]) + d(h_1[1 \ldots i], h_2[q + 1 \ldots l]) \right\}
\]

For a fixed \(i\), let

\[
A = \{d(F \bullet h_1[1 \ldots j], G) + d(h_1[j + 1 \ldots i], h_2[1 \ldots l]) \mid 0 \leq j \leq i\}
\]

\[
B = \{d(F, G \bullet h_2[1 \ldots j]) + d(h_1[1 \ldots i], h_2[j + 1 \ldots l]) \mid 0 \leq j \leq l\}
\]

The elements of \(A \cup B\) are generated from all possible partitions on \((F \bullet h_1[1 \ldots i], G \bullet h_2[1 \ldots l])\). Therefore, \(d(F \bullet h_1[1 \ldots i], G \bullet h_2[1 \ldots l])\) is the minimum element in \(A \cup B\). \(\square\)
Lemma 2.12. \( \forall i \in \{1, \ldots, l_2\} \),

\[
d(F \bullet h_1[1 \cdots l_1], G \bullet h_2[1 \cdots i]) = \\
\min \left\{ 
\begin{array}{l}
\min_{0 \leq j \leq i} \{d(F, G \bullet h_2[1 \cdots j]) + d(h_1[1 \cdots l_1], h_2[j + 1 \cdots i])\} , \\
\min_{0 \leq j \leq l_2} \{d(F \bullet h_1[1 \cdots j], G) + d(h_1[j + 1 \cdots l_1], h_2[1 \cdots i])\}
\end{array}
\right\}.
\] (2.7)

Proof. Symmetrical to Lemma 2.11. \(\square\)

To obtain \( d(F \bullet h_1[1 \cdots i], G \bullet h_2[1 \cdots l_2]) \) or \( d(F \bullet h_1[1 \cdots l_1], G \bullet h_2[1 \cdots i]) \), we need the values of the set of individual terms from which the minima are obtained. As will be shown later, the crucial task is obtaining the following terms efficiently:

\( d(h_1[j + 1 \cdots i], h_2[1 \cdots l_2]) \), \( d(h_1[1 \cdots i], h_2[j + 1 \cdots l_2]) \), \( d(h_1[1 \cdots l_1], h_2[j + 1 \cdots i]) \) and \( d(h_1[j + 1 \cdots l_1], h_2[1 \cdots i]) \).

We now discuss some crucial relations among these terms that allow us to transform this part of computation to the problem of finding row minima for totally monotone matrices which can be solved by Algorithm 10.

Lemma 2.13. Let \( D_{ij} = d(F \bullet h_1[1 \cdots j], G) + d(h_1[j + 1 \cdots i], h_2[1 \cdots l_2]) \). For any \( 0 \leq j \leq j' \leq i \leq i' \leq l_1 \), \( D_{ij} + D_{i'j'} \leq D_{ij'} + D_{i'j} \).

Proof. Define

\[
d_{ij} = d(h_1[j + 1 \cdots i], h_2[1 \cdots l_2]),
\]

then

\[
D_{ij} = d(F \bullet h_1[1 \cdots j], G) + d_{ij}.
\]

Since

\[
D_{ij} + D_{i'j'} = d(F \bullet h_1[1 \cdots j], G) + d(F \bullet h_1[1 \cdots j'], G) + d_{ij} + d_{i'j'},
\]

and

\[
D_{ij'} + D_{i'j} = d(F \bullet h_1[1 \cdots j], G) + d(F \bullet h_1[1 \cdots j'], G) + d_{ij'} + d_{i'j},
\]

...
after cancellation of terms our task reduces to proving

\[ d_{ij} + d'_{j'i'} \leq d'_{ij'} + d_{i'j}, \]

that is,

\[
\begin{align*}
&d(h_1[j + 1 \cdots i], h_2[1 \cdots l_2]) + d(h_1[j' + 1 \cdots i'], h_2[1 \cdots l_2]) \\
&\leq d(h_1[j' + 1 \cdots i], h_2[1 \cdots l_2]) + d(h_1[j + 1 \cdots i'], h_2[1 \cdots l_2]).
\end{align*}
\]

Denote by \( p_{ij} \) a minimum-cost edit path for \((h_1[j + 1 \cdots i], h_2[1 \cdots l_2])\) with a cost equal \( d_{ij} \). Since \( h_1[j + 1 \cdots i'] \) contains \( h_1[j' + 1 \cdots i] \), \( p_{ij'} \) and \( p_{i'j} \) must meet at some point. This means that \( \exists (k, l) \) where \( j' \leq k \leq i \) and \( 0 \leq l \leq l_2 \), such that

\[
\begin{align*}
d_{ij'} &= d(h_1[j' + 1 \cdots i], h_2[1 \cdots l_2]) \\
&= d(h_1[j' + 1 \cdots k], h_2[1 \cdots l]) + d(h_1[k + 1 \cdots i], h_2[l + 1 \cdots l_2])
\end{align*}
\]

and

\[
\begin{align*}
d_{i'j} &= d(h_1[j + 1 \cdots i'], h_2[1 \cdots l_2]) \\
&= d(h_1[j + 1 \cdots k], h_2[1 \cdots l]) + d(h_1[k + 1 \cdots i'], h_2[l + 1 \cdots l_2]).
\end{align*}
\]

See Figure 2.17 for an example.

Therefore,

\[
\begin{align*}
d_{ij'} + d_{i'j} \\
&= d(h_1[j' + 1 \cdots k], h_2[1 \cdots l]) + d(h_1[k + 1 \cdots i], h_2[l + 1 \cdots l_2]) \\
&\quad + d(h_1[j + 1 \cdots k], h_2[1 \cdots l]) + d(h_1[k + 1 \cdots i'], h_2[l + 1 \cdots l_2]) \\
&= (d(h_1[j' + 1 \cdots k], h_2[1 \cdots l]) + d(h_1[k + 1 \cdots i'], h_2[l + 1 \cdots l_2])) \\
&\quad + (d(h_1[j + 1 \cdots k], h_2[1 \cdots l]) + d(h_1[k + 1 \cdots i], h_2[l + 1 \cdots l_2])).
\end{align*}
\]
Note that $d(h_1[j' + 1 \cdots k], h_2[1 \cdots l]) + d(h_1[k + 1 \cdots i'], h_2[l + 1 \cdots l_2])$ and $d(h_1[j + 1 \cdots k], h_2[1 \cdots l]) + d(h_1[k + 1 \cdots i], h_2[l + 1 \cdots l_2])$ correspond to the costs of some edit paths for $(h_1[j' + 1 \cdots k], h_2[1 \cdots l_2])$ and $(h_1[j + 1 \cdots k], h_2[1 \cdots l_2])$, respectively. Since the minimum costs for $(h_1[j' + 1 \cdots k], h_2[1 \cdots l_2])$ and $(h_1[j + 1 \cdots k], h_2[1 \cdots l_2])$ are $d_{ij'}$ and $d_{ij}$, respectively, it follows that $d_{ij} + d_{ij'} \leq d_{ij'} + d_{ij}$, hence $D_{ij} + D_{ij'} \leq D_{ij'} + D_{ij}$. \hfill \qed

**Lemma 2.14.** Let $D_{ij} = d(F, G \circ h_2[1 \cdots j]) + d(h_1[1 \cdots i], h_2[j + 1 \cdots l_2])$. For any $1 \leq i \leq i' \leq l_1$ and $0 \leq j \leq j' \leq l_2$, $D_{ij} + D_{ij'} \geq D_{ij'} + D_{ij}$.

**Proof.** Define

$$d_{ij} = d(h_1[1 \cdots i], h_2[j + 1 \cdots l_2])$$

then

$$D_{ij} = d(F, G \circ h_2[1 \cdots j]) + d_{ij}.$$
Since

\[ D_{ij} + D_{i'j'} = d(F, G \bullet h_2[1 \cdots j]) + d(F, G \bullet h_2[1 \cdots j']) + d_{ij} + d_{i'j'}, \]

and

\[ D_{i'j} + D_{ij} = d(F, G \bullet h_2[1 \cdots j]) + d(F, G \bullet h_2[1 \cdots j']) + d_{i'j} + d_{ij}, \]

after cancellation of terms our task reduces to proving

\[ d_{ij} + d_{i'j'} \geq d_{i'j} + d_{ij}. \]

that is,

\[ d(h_1[1 \cdots i], h_2[j+1 \cdots l_2]) + d(h_1[1 \cdots i'], h_2[j' + 1 \cdots l_2]) \geq d(h_1[1 \cdots i], h_2[j' + 1 \cdots l_2]) + d(h_1[1 \cdots i'], h_2[j + 1 \cdots l_2]) . \]

Denote by \( p_{ij} \) a minimum-cost edit path for \((h_1[1 \cdots i], h_2[j+1 \cdots l_2])\) with a cost equal \(d_{ij}\). Since \(h_1[1 \cdots i']\) contains \(h_1[1 \cdots i]\) and \(h_2[j+1 \cdots l_2]\) contains \(h_2[j' + 1 \cdots l_2]\), \( p_{ij}\) and \( p_{i'j'}\) must meet at some point. This means that \( \exists (k, l) \) where \( 0 \leq k \leq i \) and \( j' \leq l \leq l_2 \), such that

\[ d_{ij} = d(h_1[1 \cdots i], h_2[j + 1 \cdots l_2]) \]

\[ = d(h_1[1 \cdots k], h_2[j + 1 \cdots l]) + d(h_1[k + 1 \cdots i], h_2[l + 1 \cdots l_2]) \]

and

\[ d_{i'j'} = d(h_1[1 \cdots i'], h_2[j' + 1 \cdots l_2]) \]

\[ = d(h_1[1 \cdots k], h_2[j' + 1 \cdots l]) + d(h_1[k + 1 \cdots i'], h_2[l + 1 \cdots l_2]) . \]
See Figure 2.18 for an example.

Therefore,

\[
d_{ij} + d_{i'j'}
= d(h_1[1 \cdots k], h_2[j + 1 \cdots l]) + d(h_1[k + 1 \cdots i], h_2[l + 1 \cdots l_2])
\]
\[
+ d(h_1[1 \cdots k], h_2[j' + 1 \cdots l]) + d(h_1[k + 1 \cdots i'], h_2[l + 1 \cdots l_2])
\]
\[
= (d(h_1[1 \cdots k], h_2[j + 1 \cdots l]) + d(h_1[k + 1 \cdots i'], h_2[l + 1 \cdots l_2]))
\]
\[
+ (d(h_1[1 \cdots k], h_2[j' + 1 \cdots l]) + d(h_1[k + 1 \cdots i], h_2[l + 1 \cdots l_2])).
\]

Note that \(d(h_1[1 \cdots k], h_2[j + 1 \cdots l]) + d(h_1[k + 1 \cdots i'], h_2[l + 1 \cdots l_2])\) and \(d(h_1[1 \cdots k], h_2[j' + 1 \cdots l]) + d(h_1[k + 1 \cdots i], h_2[l + 1 \cdots l_2])\) correspond to the costs of some edit paths for \((h_1[1 \cdots i'], h_2[j + 1 \cdots l_2])\) and \((h_1[1 \cdots i], h_2[j' + 1 \cdots l_2])\), respectively. Since the minimum costs for \((h_1[1 \cdots i'], h_2[j + 1 \cdots l_2])\) and \((h_1[1 \cdots i], h_2[j' + 1 \cdots l_2])\) are \(d_{i'j}\) and \(d_{ij'}\), respectively, it follows that \(d_{ij} + d_{i'j'} \geq d_{i'j} + d_{ij'}\), hence \(D_{ij} + D_{i'j'} \geq D_{i'j} + D_{ij'}\). \(\square\)
Lemma 2.15. Let \( D_{ij} = d(F, G \cdot h_1[1 \cdots j]) + d(h_1[1 \cdots l_1], h_2[j + 1 \cdots i]) \). For any \( 0 \leq j \leq j' \leq i \leq l_1 \), \( D_{ij} + D_{ij'} \leq D_{ij'} + D_{ij} \).


Lemma 2.16. Let \( D_{ij} = d(F \cdot h_1[1 \cdots j], G) + d(h_1[j + 1 \cdots l_1], h_2[1 \cdots i]) \). For any \( 1 \leq i \leq i' \leq l_1 \) and \( 0 \leq j \leq j' \leq l_2 \), \( D_{ij} + D_{ij'} \geq D_{ij'} + D_{ij} \).


Corollary 2.1. A matrix with entries defined as follows is a totally monotone matrix.

\[
\begin{align*}
A_1[i, j] &= \begin{cases} 
  d(F \cdot h_1[1 \cdots j], G) + d(h_1[j + 1 \cdots i], h_2[1 \cdots l_2]), & \text{if } 0 \leq j \leq i, \\
  A_1[i, i], & \text{if } j > i.
\end{cases} \\
A_2[i, j] &= d(F, G \cdot h_2[1 \cdots l_2 - j]) + d(h_1[1 \cdots i], h_2[l_2 - j + 1 \cdots l_2]) . \\
A_3[i, j] &= \begin{cases} 
  d(F, G \cdot h_2[1 \cdots j]) + d(h_1[1 \cdots l_1], h_2[j + 1 \cdots i]), & \text{if } 0 \leq j \leq i, \\
  A_3[i, i], & \text{if } j > i.
\end{cases} \\
A_4[i, j] &= d(F \cdot h_1[1 \cdots l_1 - j], G) + d(h_1[l_1 - j + 1 \cdots l_1], h_2[1 \cdots i]) .
\end{align*}
\]

Proof. This is a consequence that follows directly from Lemma 2.13, 2.14, 2.15 and 2.16.

Therefore, to compute the values of \( d(F \cdot h_1[1 \cdots i], G \cdot h_2[1 \cdots l_2]) \) and \( d(F \cdot h_1[1 \cdots l_1], G \cdot h_2[1 \cdots i]) \), we may first compute the row minima for \( A_1 \), \( A_2 \), \( A_3 \) and \( A_4 \) by means of the procedure RowMin of Algorithm 10, store the minima in arrays \( R_1 \), \( R_2 \), \( R_3 \) and \( R_4 \), respectively, where \( R_k[i] = A_k[i, \text{indmin}(\text{row}(A_k, i))] \) with \( k \in \{1, 2, 3, 4\} \), then obtain \( d(F \cdot h_1[1 \cdots i], G \cdot h_2[1 \cdots l_2]) \) and \( d(F \cdot h_1[1 \cdots l_1], G \cdot h_2[1 \cdots i]) \) as follows:

\[
\begin{align*}
  d(F \cdot h_1[1 \cdots i], G \cdot h_2[1 \cdots l_2]) &= \min \{ R_1[i], R_2[i] \} , \\
  d(F \cdot h_1[1 \cdots l_1], G \cdot h_2[1 \cdots i]) &= \min \{ R_3[i], R_4[i] \} .
\end{align*}
\]
2.5.2 Algorithmic Improvements

We have shown that computing distances of the forms \(d(F \bullet h_1[1 \cdots i], G \bullet h_2[1 \cdots l_2])\) and \(d(F \bullet h_1[1 \cdots l_1], G \bullet h_2[1 \cdots i])\) can be transformed into computing the row minima for a totally monotone matrix where the matrix entries correspond to the terms from which \(d(F \bullet h_1[1 \cdots i], G \bullet h_2[1 \cdots l_2])\) and \(d(F \bullet h_1[1 \cdots l_1], G \bullet h_2[1 \cdots i])\) are computed. The only issue that remains to be resolved is that we need to have all the matrix entries ready before computing the row minima. This requires an additional treatment. Note that all the relevant distances of the forms \(d(h_1[i], h_2[j]), d(F \bullet h_1[1 \cdots j], G)\) and \(d(F, G \bullet h_2[1 \cdots j])\) would be available due to the bottom-up order of computation. Therefore, the terms that concern us are of the forms: \(d(h_1[j + 1 \cdots i], h_2[1 \cdots l_2]), d(h_1[1 \cdots i], h_2[j + 1 \cdots l_2]), d(h_1[1 \cdots l_1], h_2[j + 1 \cdots i]),\) and \(d(h_1[j + 1 \cdots l_1], h_2[1 \cdots i])\).

The values of these terms can be efficiently obtained using techniques similar to a sequential version of a parallel algorithm found in [2].

In the following, we give a general description of the procedure for obtaining the values of these terms.

First, we note that our task at hand is essentially finding the edit distance between substrings. This kind of problem can be casted in a graph-theoretic setting. Consider a \((m + 1) \times (n + 1)\) grid \(G\) where the points on the left boundary \((0, 0), (1, 0), \cdots, (m, 0)\) correspond to \(\emptyset, s_1, \cdots, s_m\), respectively, and the points on the top boundary \((0, 0), (0, 1), \cdots, (0, n)\) correspond to \(\emptyset, t_1, \cdots, t_n\), respectively. On this grid, directed edges are drawn such that for any grid point \((i, j)\), the only allowed outgoing edges are to \((i, j + 1), (i + 1, j)\) and \((i + 1, j + 1)\). The resulting grid with the directed edges is essentially a directed acyclic graph (DAG). An example of such DAG is given in Figure 2.19. From now on, when the context is clear we shall simply refer to this kind of grid as DAG.

Given two strings \(s\) and \(t\) of lengths \(m\) and \(n\), respectively, we associate a \((m + 1) \times (n + 1)\) DAG \(G\) to the pair of strings. An edge \(e((i, j), (i, j + 1))\) in \(G\) represents insertion of \(t_{j+1}\) in the edit script converting \(s\) to \(t\). An edge \(e((i, j), (i + 1, j))\) in \(G\)
Figure 2.19: An example of a DAG.

represents deletion of $s_{i+1}$ in the edit script converting $s$ to $t$. An edge $e((i, j), (i + 1, j + 1))$ in $G$ represents substitution of $s_{i+1}$ by $t_{j+1}$ in the edit script converting $s$ to $t$. The edit script between two sub-strings $s_i \cdots s_{i'}$ and $t_j \cdots t_{j'}$ corresponds to a minimum-cost path in $G$ between two grid points $(i - 1, j - 1)$ and $(i', j')$.

Recall that our goal is to obtain values for the terms of the following forms:

\[
d(h_1[j + 1 \cdots i], h_2[1 \cdots l_2]), \quad d(h_1[1 \cdots i], h_2[j + 1 \cdots l]), \quad d(h_1[1 \cdots l_1], h_2[j + 1 \cdots i]), \quad \text{and} \quad d(h_1[j + 1 \cdots l_1], h_2[1 \cdots i]).
\]

If we associate a $(l_1 + 1) \times (l_2 + 1)$ DAG $G'$ to $h_1[1 \cdots l_1]$ and $h_2[1 \cdots l_2]$, then any such term actually corresponds to a minimum-cost edit path in $G'$ which starts at a point on the top or left border and ends at a point on the right or bottom border. Therefore, we would like to have an efficient way of obtaining the values of $d(u, v)$ for all $u$ on the top and left border and all $v$ on the right and bottom border of $G'$. Once we have these values, the terms of the forms $d(F \bullet h_1[1 \cdots i], G \bullet h_2[1 \cdots l_2])$ and $d(F \bullet h_1[1 \cdots l_1], G \bullet h_2[1 \cdots i])$ can be readily constructed in constant time.

We first consider the case when $m = n$. We divide a $m \times m$ DAG $G$ into four smaller squares, $G_{tl}$, $G_{tr}$, $G_{bl}$ and $G_{br}$, as shown in Figure 2.20. The subscripts indicate the common boundaries that a sub-DAG shares with $G$, where $t$ and $b$ denote top and bottom, respectively, and $l$ and $r$ denote left and right, respectively. For example, $G_{tl}$ is the sub-DAG of which the top and left boundaries are contained in the boundaries
Figure 2.20: A DAG is divided into sub-DAG’s.

of $G$. Suppose that for each sub-DAG the values of minimum-cost paths between all the points on the left and top boundaries and all the points on the right and bottom boundaries are known. Then to obtain for $G$ the values between all the points on the left and top boundaries and all the points on the right and bottom boundaries, we do the following:

1. Use the values for $G_{tl}$ and $G_{bl}$ to obtain the values for the sub-DAG $G_{tl} \cup G_{bl}$.
2. Use the values for $G_{tr}$ and $G_{br}$ to obtain the values for the sub-DAG $G_{tr} \cup G_{br}$.
3. Use the values for $G_{tl} \cup G_{bl}$ and $G_{tr} \cup G_{br}$ to obtain the values for $G$.

Therefore, the overall procedure is:

1. Recursively solve the problem for $G_{tl}$, $G_{bl}$, $G_{tr}$ and $G_{br}$.
2. Use the values for $G_{tl}$ and $G_{bl}$ to obtain the values for the sub-DAG $G_{tl} \cup G_{bl}$.
3. Use the values for $G_{tr}$ and $G_{br}$ to obtain the values for the sub-DAG $G_{tr} \cup G_{br}$.
4. Use the values for $G_{tl} \cup G_{bl}$ and $G_{tr} \cup G_{br}$ to obtain the values for $G$.

The main challenge is how to do the combining steps efficiently. Note that a minimum-cost path originating at a point $u$ on the left or top boundary of a
sub-DAG and ending at a point \( v \) on the right or bottom boundary of an adjacent sub-DAG must pass through the common boundary of the two sub-DAG's. For any pair of \((u, v)\), if we know this intersecting point, say \( w \), then the value of 

\[
\text{cost}(u, v) = \text{cost}(u, w) + \text{cost}(w, v).
\]

Therefore, the problem is essentially a matter of computing these intersecting points. Denote by \( \theta(u, v) \) the leftmost intersecting point. Consider two adjacent sub-DAG's \( G_1 \) and \( G_2 \) where \( G_1 \) is either to the left of or above \( G_2 \). Denote by \( L \) an ordered list for the points on the bottom and the right boundaries of \( G_2 \). Let \( b_1, b_2, \cdots, b_m \) be the points on the bottom boundary of \( G_2 \) from left to right. Let \( r_1, r_2, \cdots, r_m \) be the points on the right boundary of \( G_2 \) from bottom to top. Then \( L = \{b_1, b_2, \cdots, b_m, r_1, r_2, \cdots, r_m\} \). Define a linear ordering \( \prec_L \) on \( L \) such that for any two distinct points \( v, v' \in L \), 

\[
v \prec_L v' \text{ if } v \text{ precedes } v' \text{ in } L.
\]

Let \( L_c = \{c_1, c_2, \cdots, c_m\} \) be the points on the common boundary in the left-to-right order if \( G_1 \) is above \( G_2 \), or in the bottom-to-top order if \( G_1 \) is to the left of \( G_2 \). A linear ordering \( \prec_c \) on \( L_c \) is analogously defined as for \( L \). The following lemma states an important property by which the \( \theta \) values can be computed efficiently.

**Lemma 2.17.** [2] Let \( G_1 \) and \( G_2 \) be two adjacent square DAG's as described above. Consider a point \( u \) on the left or top boundary of \( G_1 \) and two points \( v \) and \( v' \) on the right or bottom boundary of \( G_2 \). 

\[
v \prec_L v' \Rightarrow \theta(u, v) \prec_c \theta(u, v') \text{ or } \theta(u, v) = \theta(u, v').
\]

Lemma 2.17 implies that the ordering of the \( \theta \)'s for a fixed \( u \) with respect to all the \( v \)'s is an instance of total monotonicity (see Figure 2.21). Therefore, the problem of finding \( \theta \)'s between any \( u \) on the left or top boundary of \( G_1 \) and all the \( v \)'s on the right and bottom boundaries of \( G_2 \) can be handled as the problem of finding row minima in a totally monotone matrix where the rows are ordered according to the ordering of the list \( L \). As already mentioned, the \( \theta \)'s for one \( u \) can be found in \( \mathcal{O}(m) \) time. Hence, the \( \theta \)'s for all the \( u \)'s can be found in \( \mathcal{O}(m^2) \) time.

The total running time for the above divide-and-conquer procedure is expressed
Figure 2.21: The minimum-cost edit paths corresponding to $d(u,v_1)$, $d(u,v_2)$ and $d(u,v_3)$. Here, $\theta_1 = \theta(u,v_1)$, $\theta_2 = \theta(u,v_2)$, and $\theta_3 = \theta(u,v_3)$.

by the following recurrence relation:

$$T(m) \leq 4T(m/2) + k \times (m/2)^2.$$ 

Therefore, the total time complexity is $O(m^2 \times \log m)$.

For a $m \times n$ DAG assuming $m \leq n$, the problem is solved in the following steps:

1. Divide the DAG into $\lceil \frac{n}{m} \rceil \times m$ sub-DAG’s and solve each of them. If $m$ does not divide $n$, add extra vertices and zero-cost edges so as to make $n$ divisible by $m$.

2. Recursively, combine the leftmost $\lceil \frac{n}{2 \times m} \rceil$ sub-DAG’s, and combine the rightmost $\lceil \frac{n}{2 \times m} \rceil$ sub-DAG’s. Then, combine the two resulting $m \times \lceil \frac{n}{2} \rceil$ sub-DAG’s.

For the first step, there are $O(\frac{n}{m})$ sub-DAG’s each taking $O(m^2 \times \log m)$ time. There-
fore, the running time is
\[
\mathcal{O}\left(\frac{n}{m} \times m^2 \times \log m\right) = \mathcal{O}(m \times n \times \log m).
\]

For the second step, there are \(\mathcal{O}\left(\frac{n}{m} \times \sum_{i=1}^{\log \frac{n}{m}} \frac{1}{2^i}\right)\) combining steps. For any \(i\) in this range, the time taken is \(\mathcal{O}((2 \times m + m \times i)^2)\) since there are \((2 \times m + m \times i)\) boundary points to consider for each sub-DAG. Therefore,
\[
\frac{n}{m} \times \sum_{i=1}^{\log \frac{n}{m}} \frac{(2 \times m + m \times i)^2}{2^i}
= m \times n \times \sum_{i=1}^{\log \frac{n}{m}} \frac{(2 + i)^2}{2^i}
\le m \times n \times \sum_{i=1}^{\log \frac{n}{m}} \frac{(2 + \log \frac{n}{m})^2}{2^i}
= m \times n \times (2 + \log \frac{n}{m})^2 \times \sum_{i=1}^{\log \frac{n}{m}} \frac{1}{2^i}
\le m \times n \times (2 + \log \frac{n}{m})^2.
\]

The running time for the second step is \(\mathcal{O}(m \times n \times (\log^2 \frac{n}{m}))\).

The total time complexity can be bounded by \(\mathcal{O}(n^2 \times \log n)\). Therefore, the time complexity in a general form can be written as \(\mathcal{O}\left(\max\{m^2 \times \log m, n^2 \times \log n\}\right)\).

The techniques described in the preceding text may be readily applied to the problem of our concern, namely the computation involving \((F \bullet h_1[1 \cdots l_1], G \bullet h_2[1 \cdots l_2])\). The following lemma states this.

**Lemma 2.18.** Given \(F \bullet h_1[1, \cdots, l_1]\) and \(G \bullet h_2[1, \cdots, l_2]\) where \(d(h_1[l], h_2[l'])\) is known for all \(l \in \{1, \cdots, l_1\}\) and \(l' \in \{1, \cdots, l_2\}\), the entire set of the following terms in Equation 2.6 and 2.7 for all possible \((i, j)\) values, can be computed in \(\mathcal{O}(\max\{l_1^2 \times \log l_1, l_2^2 \times \log l_2\})\) time and \(\mathcal{O}(l_1 \times l_2)\) space: \(d(h_1[j+1 \cdots i], h_2[1 \cdots l_2])\), \(d(h_1[1 \cdots i], h_2[j+1 \cdots l_2])\), \(d(h_1[1 \cdots l_1], h_2[j+1 \cdots i])\), and \(d(h_1[j+1 \cdots l_1], h_2[1 \cdots i])\).
Proof. By applying the techniques presented in the preceding text, this result readily follows.

Suppose that \( F \) and \( G \) are induced by the indices \( f_1, \ldots, f_k \) and \( g_1, \ldots, g_l \), respectively. In the computation involving \((F \bullet h_1[1 \cdots l_1], G \bullet h_2[1 \cdots l_2])\), all the subproblems may be partitioned into the following sets:

1. \( d(F[f_1 \cdots i], G[g_1 \cdots j]), f_1 \leq i \leq f_k \) and \( g_1 \leq j \leq g_l \),
2. \( d(F[f_1 \cdots i], G \bullet h_2[1 \cdots j]), f_1 \leq i \leq f_k \) and \( 1 \leq j \leq l_2 \),
3. \( d(F \bullet h_1[1 \cdots i], G[g_1 \cdots j]), 1 \leq i \leq l_1 \) and \( g_1 \leq j \leq g_l \),
4. \( d(F \bullet h_1[1 \cdots i], G \bullet h_2[1 \cdots j]), 1 \leq i \leq l_1 \) and \( 1 \leq j \leq l_2 \).

These four sets of subproblems correspond to the four blocks of the dynamic programming table in Figure 2.16. For the last set, however, we are only concerned with a subset:

- \( d(F \bullet h_1[1 \cdots i], G \bullet h_2[1 \cdots l_2]), 1 \leq i \leq l_1 \),
- \( d(F \bullet h_1[1 \cdots l_1], G \bullet h_2[1 \cdots j]), 1 \leq j \leq l_2 \).

To compute this subset of distances, we distinguish among the necessary terms by the following types:

- type-1: \( d(F \bullet h_1[1 \cdots i], G) \) and \( d(F, G \bullet h_2[1 \cdots j]) \),

- type-2: \( d(h_1[j + 1 \cdots i], h_2[1 \cdots l_2]) \), \( d(h_1[1 \cdots i], h_2[j + 1 \cdots l_2]) \), \( d(h_1[1 \cdots l_1], h_2[j + 1 \cdots i]) \), and \( d(h_1[j + 1 \cdots l_1], h_2[1 \cdots i]) \).

The computation involving \((F \bullet h_1[1 \cdots l_1], G \bullet h_2[1 \cdots l_2])\) is arranged in a bottom-up order such that the first three sets of subproblems are computed before the last set. Since the type-1 terms are located beside the top and left borders of the lower-right block, the above order of computation permits the type-1 terms to be computed.
in $O(l_1 + l_2)$ time. The type-2 terms can be computed in $O(\max\{l_1^2 \times \log l_1, l_2^2 \times \log l_2\})$ time as stated in Lemma 2.18. The values of type-1 terms are written to a temporary table in which the computation of forest distance takes place. The type-2 terms for each $(h_1[1 \cdots l_1], h_2[1 \cdots l_2])$ pair are computed only once, that is, when the computation process encounters this pair for the first time; and, their values are stored in a permanent array so as to be readily fetched in subsequent encounters of the same $(h_1[1 \cdots l_1], h_2[1 \cdots l_2])$ pair. Therefore, the overall time for each $(h_1[1 \cdots l_1], h_2[1 \cdots l_2])$ pair is $O(\max\{l_1^2 \times \log l_1, l_2^2 \times \log l_2\})$. When Algorithm 10 is applied to computing the row minima, values of relevant terms can be fetched separately and combined before being passed to the relevant procedure. In this way, the row minima can be computed in $O(l_1 + l_2)$ time and $O(\max\{l_1, l_2\})$ space.

In the following text, a symbol $S$ associated with a tree is denoted by $\tilde{S}$ as a result of vertical reduction, and $\hat{S}$ as a result of both vertical reduction and horizontal reduction. As well, let $\text{collapsed\_depth}(T)$ stand for $\min\{\text{depth}(T), \text{leaves}(T)\}$.

Lemma 2.19. The edit distance $d(T_1, T_2)$ for $T_1$ and $T_2$ can be computed in $O(|T_1| \times |T_2| + \max\{|\tilde{T}_1|^2 \times \log |\tilde{T}_1|, |\tilde{T}_2|^2 \times \log |\tilde{T}_2|\} + (|\tilde{T}_1| \times |\tilde{T}_2| + |\tilde{T}_1| \times |\tilde{T}_2|) \times \prod_{i=1}^2 \text{collapsed\_depth}(\tilde{T}_i))$ time and $O(|T_1| \times |T_2|)$ space.

Proof. The set of all subproblems can be divided into separate groups as follows:

1. subproblems involving vertically linear components,
2. subproblems involving horizontally linear components where vertical linearity has been removed,
3. subproblems involving tree nodes which are not contained in any vertical or horizontal component.

The running time for group 1 is bounded by $O(|T_1| \times |T_2|)$ as has been shown before.

Group 2 incurs two types of computation:
that which computes the terms specified in Lemma 2.18,

that which computes the row minima.

For the first type of computation, the running time is bounded by $O(\max\{|\tilde{T}_1|^2 \times \log |\tilde{T}_1|, |\tilde{T}_2|^2 \times \log |\tilde{T}_2|\})$ since for each pair of horizontally linear components $(h_1[1 \cdots l_1], h_2[1 \cdots l_2])$ the computation executes exactly once in $O(\max\{l_1^2 \times \log l_1, l_2^2 \times \log l_2\})$ time. For the second type of computation, the running time is bounded by $O(\left(\sum_{h_1i \in \tilde{T}_1} \sum_{h_2j \in \tilde{T}_2} (l_{1i} + l_{2j}) \times \prod_{k=1}^{\text{collapsed depth}(\tilde{T}_k)}\right) = O(|\tilde{T}_1| \times |\tilde{T}_2| \times |\tilde{T}_1| \times |\tilde{T}_2|))$, where a symbol such as $l_{1i}$ denotes the length of the $i$th horizontally linear component $h_{1i}$ in $\tilde{T}_1$.

The running time for group 3 is $O(|\hat{T}_1| \times |\hat{T}_2| \times \prod_{i=1}^{\text{collapsed depth}(\tilde{T}_i)})$. This is because there are at most $|\hat{T}_1| \times |\hat{T}_2|$ pair of nodes to be considered, and each pair of nodes may relate to at most $\prod_{i=1}^{\text{collapsed depth}(\tilde{T}_i)}$ disjoint subproblems.

Put together all the above contributions, the time complexity follows.

As to the space complexity, it is clear from previous discussion that any group of subproblems can be computed in quadratic space. 

To extend the preceding results to all the strategy-based algorithms, we first define a conceptually unified notation for expressing the time complexities of these algorithms. Recall that the time complexity for the Zhang-Shasha algorithm is $O(|T_1| \times |T_2| \times \prod_{i=1}^{\text{collapsed depth}(\tilde{T}_i)})$. The term $\text{collapsed depth}(T_i)$ essentially represents the upper-bound on the number of key roots in which any tree node of $T_i$ may be contained. Therefore, $\prod_{k=1}^{\text{collapsed depth}(\tilde{T}_k)}$ represents the upper-bound on the number of disjoint subproblems in which any pair of nodes $(t_1[i], t_2[j])$ could appear.

Denote by $P(A, T_1, T_2)$ the upper-bound on the number of disjoint subproblems in which any pair of nodes $(t_1[i], t_2[j])$ could appear in the computation of $d(T_1, T_2)$ using algorithm $A$. Let $A_1$ stand for the Zhang-Shasha algorithm, $A_2$ the Klein algorithm, and $A_3$ the Demaine et al. algorithm. Assume that $|T_1| \leq |T_2|$. The number of
disjoint subproblems for each strategy-based algorithm can be written as:

\[
P(A_1, T_1, T_2) = \prod_{i=1}^{2} \text{collapsed_depth}(\tilde{T}_i),
\]

\[
P(A_2, T_1, T_2) = |T_1| \times \log |T_2|,
\]

\[
P(A_3, T_1, T_2) = |T_1| \times (1 + \log \frac{|T_2|}{|T_1|}).
\]

The relation between \(\mathcal{T}(A, T_1, T_2)\) and \(\mathcal{P}(A, T_1, T_2)\) is

\[
\mathcal{T}(A, T_1, T_2) = |T_1| \times |T_2| \times \mathcal{P}(A, T_1, T_2).
\]

**Theorem 2.2.** For a chosen strategy-based algorithm \(A\), the edit distance \(d(T_1, T_2)\) for \(T_1\) and \(T_2\) can be computed in \(O(|T_1| \times |T_2| + \max\{|\tilde{T}_1|^2 \times \log |\tilde{T}_1|, |\tilde{T}_2|^2 \times \log |\tilde{T}_2|\}) + \left(\frac{|\tilde{T}_1|}{|T_1|} + \frac{|\tilde{T}_2|}{|T_2|}\right) \times \mathcal{T}(A, \tilde{T}_1, \tilde{T}_2))\) time and \(O(|T_1| \times |T_2|)\) space.

**Proof.** The statement follows directly from all previous results. \(\Box\)

In practice, \(\mathcal{T}(A, \tilde{T}_1, \tilde{T}_2) \geq \max\{|\tilde{T}_1|^2 \times \log |\tilde{T}_1|, |\tilde{T}_2|^2 \times \log |\tilde{T}_2|\}\). Since \(\frac{|\tilde{T}_1|}{|T_1|} \leq 1\) and \(\frac{|\tilde{T}_2|}{|T_2|} \leq 1\), the improvement is easily seen.

### 2.5.3 Application

An example of suitable application related to horizontal linearity is text comparison. This is because the structure of a text in its corresponding tree representation normally is comprised of a large number of leaf nodes and a relatively small number of internal nodes. An internal node may represent a group of words such as chapter, section, etc, whereas a leaf node may represent a word.
2.6 Conclusions

In this work, we consider the problem of computing the edit distance between two rooted trees that are ordered and labeled. We have presented algorithmic improvements for speeding up a family of tree-distance algorithms of which the recursion styles have been formalized and referred to as cover strategy. The improvements are based on two types of structural linearity, namely, vertical linearity and horizontal linearity. By incorporating these types of linearity into the algorithms, the running times are reduced.

While these improvements may be potentially useful in a variety of situations, we have given two concrete examples. The incorporation of vertical linearity is beneficial to applications involving comparison of RNA secondary structures. Further incorporation of horizontal linearity is beneficial to such application as text comparison.
Bibliography


Chapter 3

The Similarity Metric and the Distance Metric

3.1 Introduction

Distance and similarity measures are widely used in bioinformatics research and other fields. Here, we give some examples.

Distance: A partial list of applications involving distance is given as follows.

- In many areas such as those mentioned in [12, 26] where sequence edit distance and tree edit distance are employed.
- Constructing phylogenetic trees [17, 19, 21].
- Improving database search [16].
- Describing the relationship between words [3].
- Comparing graphs [2] and attributed trees [22].
- Comparing information contents using information distance [7].
- Evaluating the importance of attributes [9, 15, 23] in data mining.

Similarity: A partial list of applications involving similarity is given as follows.
• Protein sequence comparison [18] using protein sequence similarity based on BLOSUM matrices.

• Evaluating the importance of attributes [4, 5, 6, 8, 10, 14] in data mining.

When a measure satisfies a set of well defined properties, we call it a metric. Distance metric is a well defined concept. In contrast, although similarity measures are widely used and their properties are studied and discussed [20, 24], it seems that there is no formal definition for the concept. In this paper, we propose a formal definition of similarity metric and show its properties and its relationship to the distance metric. Based on the metric definitions, we present general formulae for similarity and distance metrics.

We also consider the problem of normalized distance metric and normalized similarity metric. Although there are studies on normalizing specific similarity and distance metrics [2, 4, 6, 7, 9, 10, 15, 22], there is no general solution. We give general formulae to normalize a similarity metric or a distance metric.

We show, with examples, how the general solutions are useful in constructing metrics suitable for various contexts.

The rest of the paper is organized as follows. Section 3.2 is devoted to the formal definitions of distance metric and similarity metric, their properties, and their relationship. Section 3.3 presents several similarity metrics and their normalized versions. Section 3.4 presents several distance metrics and their normalized versions. Section 3.5 shows how the general solutions are useful in constructing metrics suitable for various contexts. Section 3.6 gives concluding remarks.

3.2 Similarity Metric and Distance Metric

3.2.1 Preliminaries

Recall the formal definition of a distance metric as follows.
**Definition 3.1** (Distance Metric). Given a set $X$, a real-valued function $d(x, y)$ on the Cartesian product $X \times X$ is a distance metric if for any $x, y, z \in X$, it satisfies the following conditions:

1. $d(x, y) \geq 0$ (nonnegativity),

2. $d(x, y) = d(y, x)$ (symmetry),

3. $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality),

4. $d(x, y) = 0$ if and only if $x = y$ (identity of indiscernibles).

To our knowledge, there is no formal metric definition for similarity. In the following, we present a formal definition for the similarity metric.

**Definition 3.2** (Similarity Metric). Given a set $X$, a real-valued function $s(x, y)$ on the Cartesian product $X \times X$ is a similarity metric if for any $x, y, z \in X$, it satisfies the following conditions:

1. $s(x, y) = s(y, x)$,

2. $s(x, x) \geq 0$,

3. $s(x, x) \geq s(x, y)$,

4. $s(x, y) + s(y, z) \leq s(x, z) + s(y, y)$,

5. $s(x, x) = s(y, y) = s(x, y)$ if and only if $x = y$.

Condition 1 states that $s(x, y)$ is symmetric. Condition 2 states that for any $x$ the self similarity is nonnegative. Although it is not mandatory to set this lower bound at zero, this is a common and reasonable choice. Condition 3 states that for any $x$ the self similarity is no less than the similarity between $x$ and any $y$. Condition 4 states that the similarity between $x$ and $z$ through $y$ is no greater than the direct similarity between $x$ and $z$ plus the self similarity of $y$. This property is the equivalent
of the triangle inequality in distance metric. Condition 5 states that the statements
\( s(x, x) = s(y, y) = s(x, y) \) and \( x = y \) are equivalent.

With the possible exceptions of Condition 4 and 5, the remaining conditions clearly
agree with the intuitive meaning of similarity. As to Condition 4 and 5, although their
relevance to similarity may not be intuitively clear, we explain in the following that
they are indeed indispensable properties for similarity.

Consider Condition 4. At first sight, this inequality might appear unnatural since
from the triangle inequality one might expect it to be \( s(x, y) + s(y, z) \leq s(x, z) \)
without the \( s(y, y) \) term. In a deeper analysis as follows, we shall see why this \( s(y, y) \)
term should be included.

Intuitively, the notion of similarity serves as a means to quantify the common
information shared by two objects. Two scenarios arise. In the first scenario, only
non-negative terms are used to quantify similarity. In the second scenario, real-valued
terms are used to quantify similarity. In the current discussion, we borrow notations
from set theory due to its convenience in conveying the intuition underlying similarity.

For non-negative quantification, the similarity between \( x \) and \( y \) may be informally
expressed as \( |x \cap y| \). That is, we are concerned with only that which is commonly
shared by both objects. Moreover, note that \( |x \cap y| = |x \cap y \cap z| + |x \cap y \cap \bar{z}| \). In
this scenario, we are concerned with the following inequality:

\[
|x \cap y| + |y \cap z| \leq |x \cap z| + |y| .
\]

The validity of this inequality is justified in the following:

\[
|x \cap y| + |y \cap z| \\
= |x \cap y \cap z| + |x \cap y \cap \bar{z}| + |x \cap y \cap z| + |\bar{x} \cap y \cap z| \\
\leq |x \cap z| + |y| .
\]
The inequality between (3.1) and (3.2) holds since \( |x \cap y \cap z| \leq |x \cap z| \) and 
\[ |x \cap y \cap z| + |x \cap y \cap z| + |\bar{x} \cap y \cap z| \leq |y| . \]
Without the presence of \(|y|\), one can not say that \(|x \cap z|\) alone is enough to bound all the terms on the other side of the inequality. A simple example is when \(|x \cap z| = \emptyset\) while \(|x \cap y| \neq \emptyset\) and \(|y \cap z| \neq \emptyset\).

For real-valued quantification, the similarity between \(x\) and \(y\) may be informally expressed as 
\[ k \times |x \cap y| - k' \times (|x \setminus y| + |y \setminus x|) . \]
That is, we take into account both common and non-common contributions. In this scenario, we are concerned with the following inequality:

\[
k \times (|x \cap y| + |y \cap z|) - k' \times (|x \setminus y| + |y \setminus x| + |z \setminus y|) \leq k \times (|x \cap z| + |y|) - k' \times (|x \setminus z| + |z \setminus x|) .
\]

From the results in the non-negative quantification, if we can show the validity of the following inequality then the validity of the above inequality follows.

\[
|x \setminus y| + |y \setminus x| + |y \setminus z| + |z \setminus y| \geq |x \setminus z| + |z \setminus x| .
\]

As shown in the following, this is indeed true.

\[
|x \setminus y| + |y \setminus x| + |y \setminus z| + |z \setminus y| \\
= |x \cap \bar{y}| + |y \cap \bar{x}| + |y \cap \bar{z}| + |z \cap \bar{y}| \\
\geq |x \cap \bar{y} \cap \bar{z}| + |\bar{x} \cap y \cap z| + |x \cap y \cap \bar{z}| + |\bar{x} \cap \bar{y} \cap z| \\
= (|x \cap y \cap \bar{z}| + |x \cap \bar{y} \cap \bar{z}|) + (|\bar{x} \cap y \cap z| + |\bar{x} \cap \bar{y} \cap z|) \\
= |x \cap \bar{z}| + |z \cap \bar{x}| \\
= |x \setminus z| + |z \setminus x| .
\]

Now consider Condition 5. The “if” part is clear. The “only-if” part, which states
that if $s(x, x) = s(y, y) = s(x, y)$ then $x = y$, is justified by Lemma 3.1.

**Lemma 3.1.** Let $s(x, y)$ be a real function satisfying similarity metric conditions 1, 2, 3 and 4. If $s(x, x) = s(y, y) = s(x, y)$ then for any $z$, $s(x, z) = s(y, z)$.

**Proof.** From $s(x, y) + s(y, z) \leq s(x, z) + s(y, y)$, we have $s(y, z) \leq s(x, z)$. From $s(y, x) + s(x, z) \leq s(y, z) + s(x, x)$, we have $s(x, z) \leq s(y, z)$. This means that for any $z$, $s(x, z) = s(y, z)$. □

Following the definitions of distance and similarity, the normalized metrics are defined as follows.

**Definition 3.3** (Normalized Distance Metric). A distance metric $d(x, y)$ is a normalized distance metric if $d(x, y) \leq 1$.

**Definition 3.4** (Normalized Similarity Metric). A similarity metric $s(x, y)$ is a normalized similarity metric if $|s(x, y)| \leq 1$.

**Corollary 3.1.** If $s(x, y)$ is a normalized similarity metric such that $s(x, x) = 1$, then $\frac{1}{2} \times (1 - s(x, y))$ is a normalized distance metric. If, in addition, $s(x, y) \geq 0$, then $1 - s(x, y)$ is a normalized distance metric. If $d(x, y)$ is a normalized distance metric, then $1 - d(x, y)$ is a normalized similarity metric.

**Proof.** The statements follow directly from the basic definitions. □

The next two lemmas consider the result of adding or multiplying two similarity metrics.

**Lemma 3.2.** Let $s_1(x, y) \geq 0$ and $s_2(x, y) \geq 0$ be two similarity metrics, then $s_1(x, y) + s_2(x, y)$ is a similarity metric.

**Proof.** Trivial. □

**Lemma 3.3.** Let $s_1(x, y) \geq 0$ and $s_2(x, y) \geq 0$ be two similarity metrics, then $s_1(x, y) \times s_2(x, y)$ is a similarity metric.
Proof. We only show the proof for condition 4 as the other conditions can be proved trivially.

Condition 4: Let \( d_{xz} = \max\{s_2(x, y) + s_2(y, z) - s_2(y, y), 0\} \), then \( d_{xz} \leq s_2(x, z) \), \( d_{xz} \leq s_2(y, y) \) and \( s_2(x, y) + s_2(y, z) \leq s_2(y, y) + d_{xz} \). Without loss of generality, we assume that \( s_1(x, y) \geq s_1(y, z) \). Therefore,

\[
\begin{align*}
    s_1(x, y) \times s_2(x, y) + s_1(y, z) \times s_2(y, z) \\
    &= (s_1(x, y) - s_1(y, z)) \times s_2(x, y) + s_1(y, z) \times (s_2(x, y) + s_2(y, z)) \\
    &\leq (s_1(x, y) - s_1(y, z)) \times s_2(y, y) + s_1(y, z) \times (s_2(y, y) + d_{xz}) \\
    &= s_1(x, y) \times s_2(y, y) + s_1(y, z) \times d_{xz} \\
    &= s_1(x, y) \times (s_2(y, y) - d_{xz}) + (s_1(x, y) + s_1(y, z)) \times d_{xz} \\
    &\leq s_1(y, y) \times (s_2(y, y) - d_{xz}) + (s_1(y, y) + s_1(x, z)) \times d_{xz} \\
    &= s_1(y, y) \times s_2(y, y) + s_1(x, z) \times d_{xz} \\
    &\leq s_1(y, y) \times s_2(y, y) + s_1(x, z) \times s_2(x, z) .
\end{align*}
\]

Therefore, if \( s_i(x, y) \geq 0 \), \( 1 \leq i \leq n \), are normalized similarity metrics, then \( \prod_i^n s_i(x, y) \) is a normalized similarity metric and \( 1 - \prod_i^n s_i(x, y) \) is a normalized distance metric.

In the following, we discuss some properties of concave and convex functions that will be useful later.

Definition 3.5. A function \( f \) is concave over an interval \([a, b]\) if for every \( x_1, x_2 \in [a, b] \) and \( 0 \leq \lambda \leq 1 \),

\[
\lambda \times f(x_1) + (1 - \lambda) \times f(x_2) \leq f(\lambda \times x_1 + (1 - \lambda) \times x_2) . \tag{3.3}
\]

Definition 3.6. A function \( f \) is convex over an interval \([a, b]\) if for every \( x_1, x_2 \in [a, b] \)
and \( 0 \leq \lambda \leq 1 \),

\[
\lambda \times f(x_1) + (1 - \lambda) \times f(x_2) \geq f(\lambda \times x_1 + (1 - \lambda) \times x_2) .
\]  \hspace{1cm} (3.4)

**Lemma 3.4.** If a function \( f \) is concave over interval \((-\infty, \infty)\), then for any \( a, b \geq 0 \) and \( c \geq 0 \),

\[
f(a) + f(a + b + c) \leq f(a + b) + f(a + c) .
\]  \hspace{1cm} (3.5)

**Proof.** Let \( a + b = \lambda \times a + (1 - \lambda) \times (a + b + c) \) and \( a + c = \lambda' \times a + (1 - \lambda') \times (a + b + c) \), then \( \lambda = \frac{c}{b+c} \) and \( \lambda' = \frac{b}{b+c} \). Therefore \( \lambda + \lambda' = 1 \). From Equation 3.3, we have

\[
\lambda \times f(a) + (1 - \lambda) \times f(a + b + c) \\
\leq f(\lambda \times a + (1 - \lambda) \times (a + b + c)) \\
= f(a + b)
\]

and

\[
\lambda' \times f(a) + (1 - \lambda') \times f(a + b + c) \\
\leq f(\lambda' \times a + (1 - \lambda') \times (a + b + c)) \\
= f(a + c) .
\]

Hence, \( f(a) + f(a + b + c) \leq f(a + b) + f(a + c) \). \( \square \)

**Lemma 3.5.** If a function \( f \) is convex over interval \((-\infty, \infty)\), then for any \( a, b \geq 0 \) and \( c \geq 0 \),

\[
f(a) + f(a + b + c) \geq f(a + b) + f(a + c) .
\]  \hspace{1cm} (3.6)

**Proof.** Symmetric to Lemma 3.4. \( \square \)
Lemma 3.6. Let \( f \) be a non-negative concave function over \([0, \infty)\). Then

\[
\frac{x}{f(b + x)} \leq \frac{y}{f(b + y)}
\]

where \( 0 \leq x \leq y \) and \( 0 \leq b \).

Proof. Let \( 0 \leq \lambda \leq 1 \) such that \( \lambda \times b + (1 - \lambda) \times (b + y) = b + x \). Then \( (1 - \lambda) \times y = x \) and

\[
\begin{align*}
\frac{f(b + x)}{x} & \geq \frac{\lambda \times f(b) + (1 - \lambda) \times f(b + y)}{x} \\
& = \frac{\lambda \times f(b)}{x} + \frac{f(b + y)}{y} \\
& \geq \frac{f(b + y)}{y}.
\end{align*}
\]

The next lemma states the consequence of setting a similarity metric as an argument of a convex function.

Lemma 3.7. Let \( s(x, y) \) be a similarity metric, and \( f \) a convex function such that 

\( f(0) \geq 0 \), and \( f(x) < f(y) \) if \( x < y \). Then \( f(s(x, y)) \) is a similarity metric.

Proof.

Condition 1: It is clear that \( f(s(x, y)) = f(s(y, x)) \) since \( s(x, y) = s(y, x) \).

Condition 2: Since \( s(x, x) \geq 0 \), \( f(0) \geq 0 \), and \( f(x) < f(y) \) if \( x < y \), we have \( f(s(x, x)) \geq 0 \).

Condition 3: Since \( s(x, x) \geq s(x, y) \), and \( f(x) < f(y) \) if \( x < y \), we have \( f(s(x, x)) \geq f(s(x, y)) \).

Condition 4: Let \( a = s(x, y) + s(y, z) - s(y, y) \), \( b = s(y, y) - s(x, y) \) and \( c = s(y, y) - s(y, z) \), then \( f(s(x, y)) + f(s(y, z)) = f(a + c) + f(a + b) \leq f(a) + f(a + b + c) = f(s(x, y) + s(y, z) - s(y, y)) + f(s(y, y)) \leq f(s(x, z)) + f(s(y, y)) \). The first inequality
is due to convexity and the last inequality is due to the condition that \( f(x) < f(y) \) if \( x < y \).

Condition 5: If \( x = y \) then clearly \( f(s(x, x)) = f(s(y, y)) = f(s(x, y)) \). Conversely, \( f(s(x, x)) = f(s(y, y)) = f(s(x, y)) \) implies \( s(x, x) = s(y, y) = s(x, y) \) due to the condition that \( f(x) < f(y) \) if \( x < y \), hence \( x = y \). \( \square \)

**Note.** If the functional condition in Lemma 3.7 becomes “\( f(x) \leq f(y) \) if \( x < y \)”, then by partitioning the set into equivalence classes such that \( x \) and \( y \) are in the same class if and only if \( f(s(x, x)) = f(s(y, y)) = f(s(x, y)) \), \( f(s(x, y)) \) is still a similarity metric on the quotient set.

**Corollary 3.2.** Given a similarity metric \( s(x, y) \) on \( X \), we define \( s^+(x, y) \) as follows.

\[
    s^+(x, y) = \begin{cases} 
        s(x, y), & s(x, y) \geq 0, \\
        0, & s(x, y) < 0.
    \end{cases}
\]

Then \( s^+(x, y) \) is a similarity metric on \( X' \) where all \( x \in X \) such that \( s(x, x) = 0 \) correspond to a single element in \( X' \).

**Proof.** The result follows directly from the preceding note. \( \square \)

### 3.2.2 Relationship between Similarity Metric and Distance Metric

We consider the relationship between the similarity metric and the distance metric. In particular, we establish transformations that transform a given similarity metric to a distance metric and vice versa.

We first consider transformations from similarity metric to distance metric. Given a similarity metric \( s(x, y) \), we define two transformations, \( F_p(s) = d_p \) and \( F_m(s) = d_m \), as follows.
\[ F_p(s(x,y)) = \frac{s(x,x) + s(y,y)}{2} - s(x,y) \tag{3.7} \]
\[ F_m(s(x,y)) = \max\{s(x,x), s(y,y)\} - s(x,y) \tag{3.8} \]

In the following, we prove that these transformations produce distance metrics.

**Lemma 3.8.** Let \( s(x,y) \) be a similarity metric. Then,

\[ d_p(x,y) = \frac{s(x,x) + s(y,y)}{2} - s(x,y) \]

is a distance metric.

**Proof.**

Condition 1:

\[
d_p(x,y) = \frac{s(x,x) + s(y,y)}{2} - s(x,y)
\]

\[ = \frac{1}{2} \times (s(x,x) + s(y,y) - 2 \times s(x,y)) \]

\[ = \frac{1}{2} \times ((s(x,x) - s(x,y)) + (s(y,y) - s(x,y))) \tag{3.9} \]

\[ \geq 0 . \tag{3.10} \]

The inequality between (3.9) and (3.10) is due to similarity metric condition 3.

Condition 2: \( d_p(x,y) = d_p(y,x) \) clearly holds.

Condition 3:

\[
d_p(x,z) = \frac{s(x,x) + s(z,z) - 2 \times s(x,z)}{2}
\]

\[ = \frac{s(x,x) + s(z,z) + 2 \times s(y,y) - 2 \times s(x,z) - 2 \times s(y,y)}{2} \]
\[
\leq \frac{s(x, x) + s(z, z) + 2 \times s(y, y) - 2 \times s(x, y) - 2 \times s(y, z)}{2} = \frac{s(x, x) + s(y, y) - 2 \times s(x, y) + s(y, y) + s(z, z) - 2 \times s(y, z)}{2} = d_p(x, y) + d_p(y, z). 
\]

Condition 4: If \( x = y \) then clearly \( d_p(x, y) = 0 \). Conversely, \( d_p(x, y) = 0 \) means \( s(x, x) - s(x, y) + s(y, y) - s(x, y) = 0 \). Since \( s(x, x) \geq s(x, y) \) and \( s(y, y) \geq s(x, y) \), we must have \( s(x, x) = s(x, y) \) and \( s(y, y) = s(x, y) \) for \( s(x, x) - s(x, y) + s(y, y) - s(x, y) = 0 \) to hold, that is, \( s(x, x) = s(y, y) = s(x, y) \). Hence, \( x = y \).

**Lemma 3.9.** Let \( s(x, y) \) be a similarity metric. Then,

\[
d_m(x, y) = \max\{s(x, x), s(y, y)\} - s(x, y)
\]

is a distance metric.

**Proof.**

Condition 1: \( d_m(x, y) \geq 0 \) is trivially true.

Condition 2: \( d_m(x, y) = d_m(y, x) \) clearly holds.

Condition 3:

\[
d_m(x, z) = \max\{s(x, x), s(z, z)\} - s(x, z)
\]

\[
= \max\{s(x, x), s(z, z)\} + s(y, y) - s(x, z) - s(y, y)
\]

\[
\leq \max\{s(x, x), s(z, z)\} + s(y, y) - s(x, y) - s(y, z)
\]

\[
\leq \max\{s(x, x), s(y, y)\} + \max\{s(y, y), s(z, z)\} - s(x, y) - s(y, z)
\]

\[
= d_m(x, y) + d_m(y, z).
\]

Condition 4: If \( x = y \), then clearly \( d_m(x, y) = 0 \). Conversely, \( d_m(x, y) = 0 \) means \( \max\{s(x, x), s(y, y)\} - s(x, y) = 0 \). Since \( s(x, x) \geq s(x, y) \) and \( s(y, y) \geq s(x, y) \), this implies \( s(x, x) = s(y, y) = s(x, y) \), hence \( x = y \).  \(\square\)
Next, we consider transformations from distance metric to similarity metric. Given a distance metric $d(x, y)$ on $X$, we define, for any fixed $w \in X$, transformations $G^k_p(d) = s^k_p$ with $k \geq 1$, and $G^k_m(d) = s^k_m$ with $k > 0$, as follows.

$$G^k_p(d(x, y)) = \frac{d(x, w) + d(y, w)}{k} - d(x, y) \quad (3.11)$$

$$G^k_m(d(x, y)) = k \times \min\{d(x, w), d(y, w)\} - d(x, y) \quad (3.12)$$

In the following, we prove that these transformations produce similarity metrics.

**Lemma 3.10.** Let $d(x, y)$ be a distance metric on $X$. Then for any fixed $w \in X$ and $k \geq 1$,

$$s^k_p(x, y) = \frac{d(x, w) + d(y, w)}{k} - d(x, y)$$

is a similarity metric.

**Proof.**

Condition 1: Clearly, $s^k_p(x, y) = s^k_p(y, x)$ holds since $d(x, y) = d(y, x)$.

Condition 2: Clearly, $s^k_p(x, x) \geq 0$ holds since $d(x, y) \geq 0$.

Condition 3:

$$s^k_p(x, x) - s^k_p(x, y)$$

$$= \frac{d(x, w) + d(x, w)}{k} - d(x, x) - \frac{d(x, w) + d(y, w)}{k} + d(x, y)$$

$$= \frac{1}{k} \times (d(x, w) + k \times d(x, y) - d(y, w))$$

$$\geq 0.$$
Condition 4:

\[
\begin{align*}
    s^k_p(x, y) + s^k_p(y, z) &= \frac{d(x, w) + d(y, w)}{k} - \frac{d(x, y)}{k} + \frac{d(y, w) + d(z, w)}{k} - \frac{d(y, z)}{k} \\
    &\leq \frac{d(x, w) + d(z, w)}{k} + \frac{d(y, w) + d(y, w)}{k} - (d(x, y) + d(y, z)) \\
    &= \frac{d(x, w) + d(z, w)}{k} - \frac{d(x, z)}{k} + \frac{d(y, w) + d(y, w)}{k} - \frac{d(y, y)}{k} \\
    &= s^k_p(x, z) + s^k_p(y, y).
\end{align*}
\]

Condition 5: If \( x = y \) then \( s^k_p(x, x) = s^k_p(y, y) = s^k_p(x, y) \) holds trivially. Conversely, \( s^k_p(x, x) = s^k_p(y, y) = s^k_p(x, y) \) implies \( 2 \times d(x, w) = 2 \times d(y, w) = d(x, w) + d(y, w) - k \times d(x, y) \). We obtain

\[
d(x, w) - d(y, w) + k \times d(x, y) = 0
\]

and

\[
d(y, w) - d(x, w) + k \times d(x, y) = 0
\]

which yields \( d(x, y) = 0 \), hence \( x = y \).

\[\square\]

**Lemma 3.11.** Let \( d(x, y) \) be a distance metric on \( X \). Then for any fixed \( w \in X \),

\[
s^k_m(x, y) = k \times \min\{d(x, w), d(y, w)\} - d(x, y), \quad k > 0,
\]

is a similarity metric.

**Proof.**

Condition 1: Clearly, \( s^k_m(x, y) = s^k_m(y, x) \) holds since \( d(x, y) = d(y, x) \).

Condition 2: \( s^k_m(x, x) \geq 0 \) since \( k > 0 \) and \( d(x, w) \geq 0 \).

Condition 3: \( s^k_m(x, y) - s^k_m(x, x) = k \times \min\{d(x, w), d(y, w)\} - d(x, y) - k \times d(x, w) \leq 0 \) since \( k > 0 \).
Condition 4:

\[ s^k_m(x, y) + s^k_m(y, z) = k \times \min \{d(x, w), d(y, w)\} - d(x, y) + k \times \min \{d(y, w), d(z, w)\} - d(y, z) \]

\[ = k \times \min \{d(x, w), d(y, w)\} + k \times \min \{d(y, w), d(z, w)\} - (d(x, y) + d(y, z)) \]

\[ \leq k \times \min \{d(x, w), d(y, w)\} + k \times \min \{d(y, w), d(z, w)\} - (d(x, z) + d(y, y)) \]

\[ \leq k \times \min \{d(x, w), d(z, w)\} + k \times d(y, w) - (d(x, z) + d(y, y)) \]

\[ = k \times \min \{d(x, w), d(z, w)\} - d(x, z) + k \times \min \{d(y, w), d(y, w)\} - d(y, y) \]

\[ = s^k_m(x, z) + s^k_m(y, y) . \]

Condition 5: If \( x = y \) then \( s^k_m(x, x) = s^k_m(y, y) = s^k_m(x, y) \) clearly holds. Conversely, \( s^k_m(x, x) = s^k_m(y, y) = s^k_m(x, y) \) implies \( k \times d(x, w) = k \times d(y, w) = k \times \min \{d(x, w), d(y, w)\} - d(x, y) \). This means \( d(x, y) = 0 \), hence \( x = y \). \( \Box \)

Note. Given a distance metric \( d \), we have \( F_p(G^k_p(d)) = d \). Given a similarity metric \( s \), in general \( G^k_p(F_p(s)) \neq s \). Only when there exists a fixed \( w \in X \), such that \((k - 1) \times (s(x, x) + s(y, y)) = 2 \times (s(w, w) - s(x, w) - s(y, w))\), we have \( G^k_p(F_p(s)) = s \). Note that this means that for any given \( d \), let \( s = G^k_p(d) \), then \( G^k_p(F_p(s)) = s \).

### 3.3 Normalized Similarity Metric

We first present several similarity metrics. Note that these metrics are not necessarily normalized according to definition 3.4, although they may be considered as weakly normalized. Following these, we give the condition on which these metrics are normalized.

**Theorem 3.1.** Let \( s(x, y) \) be a similarity metric, and \( f \) a concave function over
\[ [0, \infty) \text{ satisfying } f(0) \geq 0, \ f(x) > 0 \text{ if } x > 0, \text{ and } f(x) \leq f(y) \text{ if } x < y. \text{ Then} \]

\[
\bar{s}(x, y) = \frac{s(x, y)}{f(s(x, x) + s(y, y) - s(x, y))}
\]

is a similarity metric.

Proof.

Condition 1: Since \( s(x, y) = s(y, x) \), clearly \( \bar{s}(x, y) = \bar{s}(y, x) \).

Condition 2: Since \( s(x, x) \geq 0 \) and \( s(x, x) + s(y, y) - s(x, y) \geq 0 \), \( \bar{s}(x, y) \geq 0 \).

Condition 3: Since \( s(x, x) \geq s(x, y) \) and \( s(x, x) \leq s(x, x) + s(y, y) - s(x, y) \), \( \bar{s}(x, x) \geq \bar{s}(x, y) \).

Condition 4: Define

\[
\begin{align*}
  f_1 &= f(s(x, x) + s(y, y) + s(z, z) - s(x, y) - s(y, z)) \\
  f_2 &= f(s(x, x) + s(y, y) - s(x, y)) \\
  f_3 &= f(s(y, y) + s(z, z) - s(y, z)) \\
  f_4 &= f(s(y, y))
\end{align*}
\]

Clearly, \( f_1 \geq \{f_2, f_3\} \geq f_4 \). Further, define

\[
\begin{align*}
  a &= s(y, y) \\
  b &= s(x, x) - s(x, y) \\
  c &= s(z, z) - s(y, z)
\end{align*}
\]

Therefore,

\[
\begin{align*}
  \bar{s}(x, y) + \bar{s}(y, z) - \bar{s}(y, y) - \bar{s}(x, z) \\
  = \bar{s}(x, y) + \bar{s}(y, z) - \bar{s}(y, y) - \frac{s(x, z)}{f(s(x, x) + s(z, z) - s(x, z))} \tag{3.13}
\end{align*}
\]
The inequality between (3.13) and (3.14) clearly holds for \( s(x, z) \geq 0 \). When \( s(x, z) < 0 \), this inequality also holds due to Lemma 3.6. The inequality between (3.15) and (3.16) holds due to Lemma 3.4.

Condition 5: If \( x = y \), clearly \( \bar{s}(x, x) = \bar{s}(y, y) = \bar{s}(x, y) \). Conversely, if \( \bar{s}(x, x) = \bar{s}(y, y) = \bar{s}(x, y) \), then \( \frac{s(x, x)}{f(s(x, x))} = \frac{s(y, y)}{f(s(y, y))} = \frac{s(x, y)}{f(s(x, y))} \). Since \( s(y, y) \geq s(x, y) \) and \( f(s(y, y)) \leq f(s(x, x) + s(y, y) - s(x, y)) \), in order for \( \frac{s(y, y)}{f(s(y, y))} = \frac{s(x, y)}{f(s(x, x) + s(y, y) - s(x, y))} \) to hold we must have \( s(y, y) = s(x, y) \). Since \( s(x, x) \geq s(x, y) \) and \( f(s(x, x)) \leq f(s(x, x) + s(y, y) - s(x, y)) \), in order for \( \frac{s(x, x)}{f(s(x, x))} = \frac{s(x, y)}{f(s(x, x) + s(y, y) - s(x, y))} \) to hold we must have \( s(x, x) = s(x, y) \). This means \( s(x, x) = s(y, y) = s(x, y) \), hence \( x = y \).

\[ \square \]

**Theorem 3.2.** Let \( f \) be a function satisfying \( f(0) \geq 0 \), \( f(x) > 0 \) if \( x > 0 \), and \( f(x) \leq f(y) \) if \( x < y \). Then, given a similarity metric \( s(x, y) \geq 0 \),

\[ \bar{s}(x, y) = \frac{s(x, y)}{f(\max\{s(x, x), s(y, y)\})} \]
is a similarity metric.

Proof.

Condition 1: Since $s(x, y) = s(y, x)$, clearly $\bar{s}(x, y) = \bar{s}(y, x)$.

Condition 2: Clearly, $\bar{s}(x, x) \geq 0$.

Condition 3: Since $s(x, x) \geq s(x, y)$ and $\max\{s(x, x), s(x, x)\} \leq \max\{s(x, x), s(y, y)\}$, we have $\bar{s}(x, x) \geq \bar{s}(x, y)$.

Condition 4: To show $\bar{s}(x, y) + \bar{s}(y, z) \leq \bar{s}(y, y) + \bar{s}(x, z)$, there are three cases.

1. $s(z, z) \leq s(x, x) \leq s(y, y)$:

$$
\frac{s(x, y)}{f(\max\{s(x, x), s(y, y)\})} + \frac{s(y, z)}{f(\max\{s(y, y), s(z, z)\})} = \frac{s(x, y)}{f(s(y, y))} + \frac{s(y, z)}{f(s(y, y))} \quad (3.17)
$$

$$
\leq \frac{s(x, z)}{f(s(x, x))} + \frac{s(y, y)}{f(s(y, y))} \quad (3.18)
$$

$$
\leq \frac{s(x, z)}{f(s(x, x))} + \frac{s(y, y)}{f(s(y, y))} \quad (3.19)
$$

The inequality between 3.17 and 3.18 is due to $s(x, y) + s(y, z) \leq s(x, z) + s(y, y)$.

The inequality between 3.18 and 3.19 is due to $s(x, z) \geq 0$.

2. $s(z, z) \leq s(y, y) \leq s(x, x)$:

$$
\frac{s(x, y)}{f(\max\{s(x, x), s(y, y)\})} + \frac{s(y, z)}{f(\max\{s(y, y), s(z, z)\})} = \frac{s(x, y)}{f(s(x, x))} + \frac{s(y, z)}{f(s(y, y))}
$$

$$
= \frac{f(s(y, y)) \times s(x, y) + f(s(x, x)) \times s(y, z)}{f(s(x, x)) \times f(s(y, y))}
$$
The inequality between 3.20 and 3.21 holds because \( s(y, z) \leq f(s(x, x)) \times (s(x, y) + s(y, z)) \)
\[ f(s(x, x)) \times f(s(y, y)) \]
\[ + \frac{(f(s(x, x)) - f(s(y, y))) \times s(y, z)}{f(s(x, x)) \times f(s(y, y))} \]
\[ \leq \frac{f(s(y, y)) \times (s(x, z) + s(y, y))}{f(s(x, x)) \times f(s(y, y))} \]
\[ + \frac{(f(s(x, x)) - f(s(y, y))) \times s(y, y)}{f(s(x, x)) \times f(s(y, y))} \]
\[ = \frac{f(s(y, y)) \times s(x, z) + f(s(x, x)) \times s(y, y)}{f(s(x, x)) \times f(s(y, y))} \]
\[ = \frac{s(x, z)}{f(s(x, x))} + \frac{s(y, y)}{f(s(y, y))} \]
\[ = \frac{s(x, y)}{f(\max\{s(x, x), s(y, y)\})} + \frac{s(y, z)}{f(\max\{s(y, y), s(z, z)\})} + \frac{s(y, y)}{f(\max\{s(y, y), s(y, y)\})} . \]

The inequality between 3.20 and 3.21 holds because \( s(x, y) + s(y, z) \leq s(x, z) + s(y, y) \) and \( s(y, y) \geq s(y, z) \).

3. \( s(y, y) \leq s(z, z) \leq s(x, x) \):

\[
\frac{s(x, y)}{f(\max\{s(x, x), s(y, y)\})} + \frac{s(y, z)}{f(\max\{s(y, y), s(z, z)\})} + \frac{s(y, y)}{f(\max\{s(y, y), s(y, y)\})} .
\]
Condition 5: It is clear that \( \bar{s}(x, x) = \bar{s}(y, y) = \bar{s}(x, y) \) if \( x = y \). Conversely, if \( \bar{s}(x, x) = \bar{s}(y, y) = \bar{s}(x, y) \), then \( \frac{s(x, x)}{f(s(x, x))} = \frac{s(y, y)}{f(s(y, y))} = \frac{s(x, y)}{f(\max\{s(x, x), s(y, y)\})} \). Since \( s(y, y) \geq s(x, y) \) and \( f(s(y, y)) \leq f(\max\{s(x, x), s(y, y)\}) \), in order for \( \frac{s(y, y)}{f(s(y, y))} = \frac{s(x, x)}{f(s(x, x))} \) to hold we must have \( s(y, y) = s(x, y) \). Since \( s(x, x) \geq s(x, y) \) and \( f(s(x, x)) \leq f(\max\{s(x, x), s(y, y)\}) \), in order for \( \frac{s(x, x)}{f(s(x, x))} = \frac{s(x, y)}{f(\max\{s(x, x), s(y, y)\})} \) to hold we must have \( s(x, x) = s(x, y) \). This means \( s(x, x) = s(y, y) = s(x, y) \), hence \( x = y \). 

\[ \square \]

**Theorem 3.3.** Let \( f \) be a concave function over \([0, \infty)\) satisfying \( f(0) \geq 0 \), \( f(x) > 0 \) if \( x > 0 \), and \( f(x) \leq f(y) \) if \( x < y \). Then, given a similarity metric \( s(x, y) \geq 0 \), for \( 0 \leq k \leq 1 \),

\[
\bar{s}(x, y) = \frac{s(x, y)}{f(\max\{s(x, x), s(y, y)\}) + k \times (\min\{s(x, x), s(y, y)\} - s(x, y))}
\]

is a similarity metric.

**Proof.** We only give the proof for Condition 4. The other conditions are proved similarly as in previous theorems. Define

\[
\begin{align*}
f_1 & = f(\max\{s(x, x), s(z, z)\}) + k \times (\min\{s(x, x), s(z, z)\} - s(x, y) - s(y, z) + s(y, y)) , \\
f'_1 & = f(\max\{s(x, x), s(z, z)\}) + k \times (\min\{s(x, x), s(z, z)\} - s(x, z)) , \\
f_2 & = f(\max\{s(x, x), s(y, y)\}) + k \times (\min\{s(x, x), s(y, y)\} - s(x, y)) , \\
f_3 & = f(\max\{s(y, y), s(z, z)\}) + k \times (\min\{s(y, y), s(z, z)\} - s(y, z)) , \\
f_4 & = f(s(y, y)).
\end{align*}
\]
Note that \( f_1 \geq f'_1 \) due to \( s(x, y) + s(y, z) \leq s(x, z) + s(y, y) \). Therefore,

\[
\begin{align*}
\bar{s}(x, y) + \bar{s}(y, z) - \bar{s}(y, y) - \bar{s}(x, z) \\
= \bar{s}(x, y) + \bar{s}(y, z) - \bar{s}(y, y) - \frac{s(x, z)}{f'_1} \\
\leq \bar{s}(x, y) + \bar{s}(y, z) - \bar{s}(y, y) - \frac{s(x, z)}{f_1} \\
\leq \bar{s}(x, y) + \bar{s}(y, z) - \bar{s}(y, y) - \frac{(s(x, y) + s(y, z) - s(y, y))}{f_1} \\
= \frac{s(x, y)}{f_2} - \frac{s(x, y)}{f_1} + \frac{s(y, z)}{f_3} - \frac{s(y, z)}{f_1} - \frac{s(y, y)}{f_4} + \frac{s(y, y)}{f_1} \\
= \frac{s(x, y) \times (f_1 - f_2)}{f_1 \times f_2} + \frac{s(y, z) \times (f_1 - f_3)}{f_1 \times f_3} - \frac{s(y, y) \times (f_1 - f_4)}{f_1 \times f_4}.
\end{align*}
\]

There are three cases to consider, namely \( s(x, x) \geq s(z, z) \geq s(y, y), s(x, x) \geq s(y, y) \geq s(z, z) \) and \( s(y, y) \geq s(x, x) \geq s(z, z) \). A partial order \( f_1 \geq \{f_2, f_3\} \geq f_4 \) is obtained for \( s(x, x) \geq s(z, z) \geq s(y, y) \) and \( s(x, x) \geq s(y, y) \geq s(z, z) \). For these two cases, continuing the previous derivation for \( \bar{s}(x, y) + \bar{s}(y, z) - \bar{s}(y, y) - \bar{s}(x, z) \), we have

\[
\begin{align*}
\bar{s}(x, y) + \bar{s}(y, z) - \bar{s}(y, y) - \bar{s}(x, z) \\
\leq \frac{s(x, y) \times (f_1 - f_2)}{f_1 \times f_2} + \frac{s(y, z) \times (f_1 - f_3)}{f_1 \times f_3} - \frac{s(y, y) \times (f_1 - f_4)}{f_1 \times f_4} \\
\leq \frac{s(y, y)}{f_1 \times f_4} \times ((f_1 - f_2) + (f_1 - f_3) - (f_1 - f_4)) \\
= \frac{s(y, y)}{f_1 \times f_4} \times (f_1 + f_4 - f_2 - f_3)
\end{align*}
\]

where \( \frac{s(y, y)}{f_1 \times f_4} \geq 0 \). Therefore, in the following analysis concerning these two cases it suffices to show that

\[
f_1 + f_4 - f_2 - f_3 \leq 0 \, .
\]
$\bullet$ \( s(x, x) \geq s(y, y) \geq s(z, z) \): Define

\[
\begin{align*}
a &= s(y, y) \\
b &= s(x, x) - s(y, y) + k \times (s(y, y) - s(x, y)) \\
c &= k \times (s(z, z) - s(y, z)).
\end{align*}
\]

Therefore,

\[
\begin{align*}
f_1 + f_4 - f_2 - f_3 &= f(s(x, x) + k \times (s(z, z) + s(y, y) - s(x, y) - s(y, z))) + f(s(y, y)) \\
&\quad - f(s(x, x) + k \times (s(y, y) - s(x, y))) \\
&\quad - f(s(y, y) + k \times (s(z, z) - s(y, z))) \\
&= f(s(y, y) + s(x, x) - s(y, y) \\
&\quad + k \times (s(y, y) - s(x, y)) + k \times (s(z, z) - s(y, z))) \\
&\quad + f(s(y, y)) \\
&\quad - f(s(y, y) + s(x, x) - s(y, y) + k \times (s(y, y) - s(x, y))) \\
&\quad - f(s(y, y) + k \times (s(z, z) - s(y, z))) \\
&= f(a + b + c) + f(a) - f(a + b) - f(a + c) \\
&\leq 0.
\end{align*}
\]

The last inequality holds because \( f \) is concave and Lemma 3.4 may be applied.

$\bullet$ \( s(x, x) \geq s(z, z) \geq s(y, y) \): Define

\[
\begin{align*}
a &= s(y, y) \\
b &= s(x, x) - s(y, y) + k \times (s(y, y) - s(x, y)) \\
c &= s(z, z) - s(y, y) + k \times (s(y, y) - s(y, z)) \\
c' &= k \times (s(z, z) - s(y, z)).
\end{align*}
\]
From $c - c' = (1 - k) \times (s(z, z) - s(y, y)) \geq 0$, we have $c \geq c'$. Therefore,

$$f_1 + f_4 - f_2 - f_3$$

$$= f(s(x, x) + k \times (s(z, z) + s(y, y) - s(x, y) - s(y, z)))$$

$$+ f(s(y, y))$$

$$- f(s(x, x) + k \times (s(y, y) - s(x, y)))$$

$$- f(s(z, z) + k \times (s(y, y) - s(y, z)))$$

$$\leq f(a + b + c') + f(a) - f(a + b) - f(a + c)$$

$$\leq f(a + b + c) + f(a) - f(a + b) - f(a + c)$$

$$\leq 0 .$$

• $s(y, y) \geq s(x, x) \geq s(z, z)$: In this case, the dominant term associated with $f_1$ is $s(x, x)$ whereas for $f_2$, $f_3$ and $f_4$ it is $s(y, y)$. This gives rise to the following possible partial orders: $f_1 \geq \{f_2, f_3\} \geq f_4$, $\{f_2, f_3\} \geq f_1 \geq f_4$, $\{f_2, f_3\} \geq f_4 \geq f_1$, and $f_2 \geq f_1 \geq f_3 \geq f_4$.

- $f_1 \geq \{f_2, f_3\} \geq f_4$: Similar as above.

- $f_2 \geq f_1 \geq f_3 \geq f_4$: It is easy to verify the following inequality by considering the relative magnitudes and the resulting signs of the terms.

\[
\begin{align*}
\bar{s}(x, y) + \bar{s}(y, z) - \bar{s}(y, y) - \bar{s}(x, z) & \leq \frac{s(x, y) \times (f_1 - f_2)}{f_1 \times f_2} + \frac{s(y, z) \times (f_1 - f_3)}{f_1 \times f_3} \\
& - \frac{s(y, y) \times (f_1 - f_4)}{f_1 \times f_4} \\
\leq & \frac{s(x, y) \times (f_1 - f_2)}{f_1 \times f_2} + \frac{s(y, y) \times (f_1 - f_3)}{f_1 \times f_3} \\
& - \frac{s(y, y) \times (f_1 - f_4)}{f_1 \times f_4}
\end{align*}
\]
\[
\begin{align*}
&= \frac{s(x, y) \times (f_1 - f_2)}{f_1 \times f_2} + \frac{s(y, y) \times (f_4 - f_3)}{f_4 \times f_4} \times (f_4 - f_3) \\
&\leq 0 .
\end{align*}
\]

- \( f_3 \geq f_1 \geq f_2 \geq f_4 \): Similar as above.

- \( \{f_2, f_3\} \geq f_1 \geq f_4 \) or \( \{f_2, f_3\} \geq f_4 \geq f_1 \): Using the following facts

\[
\begin{align*}
\min\{f_2, f_3\} &\geq \max\{f_1, f_4\} , \\
f_1 &\geq f'_1 , \\
s(x, y) + s(y, z) &\leq s(y, y) + s(x, z) , \\
s(x, z) &\geq 0 ,
\end{align*}
\]

we have

\[
\begin{align*}
&= \frac{s(x, y) + s(y, z) - s(y, y) - s(x, z)}{f_2} + \frac{s(y, y) - s(x, z)}{f_3} - \frac{s(y, y) - s(x, z)}{f_4} - \frac{s(y, y) - s(x, z)}{f_1} \\
&\leq \frac{s(x, y) + s(y, z) - s(y, y) - s(x, z)}{f_2} + \frac{s(y, y) - s(x, z)}{f_3} - \frac{s(y, y) - s(x, z)}{f_4} - \frac{s(y, y) - s(x, z)}{f_1} \\
&\leq \frac{s(x, y) + s(y, z) - s(y, y) - s(x, z)}{\max\{f_1, f_4\}} \\
&\leq 0 .
\end{align*}
\]

Therefore, we have proved that

\[
\bar{s}(x, y) + \bar{s}(y, z) \leq \bar{s}(y, y) + \bar{s}(x, z) .
\]

\(\square\)

**Corollary 3.3.** The metrics \( \bar{s}(x, y) \) in Theorem 3.1, 3.2 and 3.3 are normalized metrics with an additional condition, \( f(x) \geq x \).

**Proof.** Trivial. \(\square\)
Note. In Theorem 3.1, 3.2 and 3.3, if both the numerator and the denominator of \( \bar{s}(x, y) \) are 0, we may define \( \bar{s}(x, y) = 0 \).

Remark. We see that both \( \frac{s(x, y)}{f(s(x, x) + s(y, y) - s(x, y))} \) and \( \frac{s(x, y)}{f(\max\{s(x, x), s(y, y)\})} \) can be obtained from \( \frac{s(x, y)}{f(\max\{s(x, x), s(y, y)\} + k \times (\min\{s(x, x), s(y, y)\} - s(x, y)))} \) with \( k = 1 \) and \( k = 0 \), respectively. A comparison of their respective metric conditions is listed in Table 3.1. When \( k \) is in between 0 and 1 the condition requirement is more stringent than when \( k \) takes on the limits, i.e. 0 or 1. When \( k = 0 \) the condition for \( f \) is relaxed, whereas when \( k = 1 \) the condition for \( s(x, y) \) is relaxed.

<table>
<thead>
<tr>
<th>Formula</th>
<th>( k )</th>
<th>( s(x, y) )</th>
<th>( f )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \frac{s(x, y)}{f(s(x, x) + s(y, y) - s(x, y))} )</td>
<td>1</td>
<td>any</td>
<td>concave</td>
</tr>
<tr>
<td>( \frac{s(x, y)}{f(\max{s(x, x), s(y, y)} + k \times (\min{s(x, x), s(y, y)} - s(x, y)))} )</td>
<td>0 &lt; ( k ) &lt; 1</td>
<td>≥ 0</td>
<td>concave</td>
</tr>
<tr>
<td>( \frac{s(x, y)}{f(\max{s(x, x), s(y, y)})} )</td>
<td>0</td>
<td>≥ 0</td>
<td>any</td>
</tr>
</tbody>
</table>

Table 3.1: A comparison of metric conditions.

### 3.4 Normalized Distance Metric

We first present distance metrics. Like in the previous section, these metrics are not necessarily normalized according to definition 3.3, although they may be considered as weakly normalized. Following these, we give the condition on which these metrics are normalized.

**Theorem 3.4.** Let \( d(x, y) \) be a distance metric on \( X \). Let \( f \) be a concave function on \( [0, \infty) \) such that \( f(0) \geq 0 \), \( f(x) > 0 \) if \( x > 0 \), and \( f(x) \leq f(y) \) if \( x < y \). Then for any fixed \( w \in X \),

\[
\tilde{d}(x, y) = \frac{(1 + \frac{1}{k}) \times d(x, y)}{f(d(x, y)) + \frac{d(x, w) + d(y, w)}{k}}
\]

is a distance metric, where \( k \geq 1 \).
Proof. We prove that $\tilde{d}(x, y) \leq \tilde{d}(x, z) + \tilde{d}(y, z)$ as the other conditions clearly hold.

$$
\tilde{d}(x, y) = \frac{(1 + \frac{1}{k}) \times d(x, y)}{f(d(x, y) + \frac{d(x, w) + d(y, w)}{k})} \\
\leq \frac{(1 + \frac{1}{k}) \times (d(x, z) + d(y, z))}{f(d(x, z) + d(y, z) + \frac{d(x, w) + d(y, w)}{k})} \\
\leq \frac{(1 + \frac{1}{k}) \times d(x, z)}{f(d(x, z) + \frac{d(x, w) + d(z, w)}{k})} + \frac{(1 + \frac{1}{k}) \times d(y, z)}{f(d(y, z) + \frac{d(y, w) + d(z, w)}{k})} \\
= \tilde{d}(x, z) + \tilde{d}(y, z).
$$

Theorem 3.5. Let $d(x, y)$ be a distance metric on $X$. Let $f$ be a concave function over $[0, \infty)$ such that $f(0) \geq 0$, $f(x) > 0$ if $x > 0$, and $f(x) \leq f(y)$ if $x \leq y$. Then for any fixed $w \in X$,

$$
\tilde{d}(x, y) = \frac{\max\{d(x, w), d(y, w)\} - \min\{d(x, w), d(y, w)\} + d(x, y)}{2 \times f(\max\{d(x, w), d(y, w)\})}
$$

is a distance metric.

Proof.

Condition 1: Clearly, $\tilde{d}(x, y) \geq 0$.

Condition 2: Clearly, $\tilde{d}(x, y) = \tilde{d}(y, x)$.

Condition 3: To show $\tilde{d}(x, z) \leq \tilde{d}(x, y) + \tilde{d}(y, z)$, there are three cases to consider.

1. $d(y, w) \leq d(x, w) \leq d(z, w)$:

$$
\frac{d(z, w) - d(x, w) + d(x, z)}{2 \times f(d(z, w))} \\
\leq \frac{d(x, w) - d(y, w) + d(z, w) - d(y, w) + d(x, y) + d(y, z)}{2 \times f(d(z, w))} \\
= \frac{d(x, w) - d(y, w) + d(x, y)}{2 \times f(d(z, w))} + \frac{d(z, w) - d(y, w) + d(y, z)}{2 \times f(d(z, w))} \\
\leq \frac{d(x, w) - d(y, w) + d(x, y)}{2 \times f(d(x, w))} + \frac{d(z, w) - d(y, w) + d(y, z)}{2 \times f(d(z, w))}.
$$
2. \(d(x, w) \leq d(y, w) \leq d(z, w)\):

\[
\begin{align*}
\frac{d(z, w) - d(x, w) + d(x, z)}{2 \times f(d(z, w))} & \leq \frac{d(y, w) - d(x, w) + d(z, w) - d(y, w) + d(x, y) + d(y, z)}{2 \times f(d(z, w))} \\
= \frac{d(y, w) - d(x, w) + d(x, y)}{2 \times f(d(z, w))} & + \frac{d(z, w) - d(y, w) + d(y, z)}{2 \times f(d(z, w))} \\
\leq \frac{d(y, w) - d(x, w) + d(x, y)}{2 \times f(d(y, w))} & + \frac{d(z, w) - d(y, w) + d(y, z)}{2 \times f(d(z, w))} 
\end{align*}
\]

3. \(d(x, w) \leq d(z, w) \leq d(y, w)\):

\[
\begin{align*}
\frac{d(z, w) - d(x, w) + d(x, z)}{2 \times f(d(z, w))} & = \frac{f(d(y, w)) \times (d(z, w) - d(x, w) + d(x, z))}{2 \times f(d(z, w)) \times f(d(y, w))} \\
& \leq \frac{2 \times f(d(z, w)) \times f(d(y, w))}{2 \times f(d(z, w)) \times f(d(y, w))} \\
& \leq \frac{2 \times f(d(z, w)) \times f(d(y, w))}{2 \times f(d(z, w)) \times f(d(y, w))} \\
= \frac{d(x, y) + d(y, z)}{2 \times f(d(y, w))} & + \frac{(f(d(y, w)) - f(d(z, w))) \times (d(x, w) + d(z, w))}{2 \times f(d(z, w)) \times f(d(y, w))} \\
& + \frac{f(d(y, w)) \times (d(z, w) - d(x, w))}{2 \times f(d(z, w)) \times f(d(y, w))} \\
& \leq \frac{d(x, y) + d(y, z)}{2 \times f(d(y, w))} & + \frac{(f(d(y, w)) - f(d(z, w))) \times (d(x, w) + d(z, w))}{2 \times f(d(z, w)) \times f(d(y, w))} \\
& + \frac{2 \times f(d(z, w)) \times f(d(y, w))}{2 \times f(d(z, w)) \times f(d(y, w))} \\
& \leq \frac{d(x, y) + d(y, z)}{2 \times f(d(y, w))} & + \frac{d(y, w) - d(x, w) + d(x, y) + d(y, w)}{2 \times f(d(y, w))} \\
& + \frac{d(y, w) - d(z, w) + d(y, z)}{2 \times f(d(y, w))} 
\end{align*}
\]

(3.22)
The inequality between (3.22) and (3.23) is due to Lemma 3.6. Condition 4: It is clear that $\bar{d}(x, y) = 0$ if $x = y$. Conversely, if $\bar{d}(x, y) = 0$, then $\max\{d(x, w), d(y, w)\} - \min\{d(x, w), d(y, w)\} + d(x, y) = 0$. Since $\max\{d(x, w), d(y, w)\} - \min\{d(x, w), d(y, w)\} \geq 0$ and $d(x, y) \geq 0$, in order for $\max\{d(x, w), d(y, w)\} - \min\{d(x, w), d(y, w)\} + d(x, y) = 0$ to hold we must have $d(x, y) = 0$, hence $x = y$.

**Corollary 3.4.** *The metrics $\bar{d}(x, y)$ in Theorem 3.4 and 3.5 are normalized metrics with an additional condition, $f(x) \geq x$.***

*Proof.* We prove for Theorem 3.5 as the case for Theorem 3.4 is trivial. It is clear that $\bar{d}(x, y) \geq 0$. Since

$$\max\{d(x, w), d(y, w)\} + \min\{d(x, w), d(y, w)\} \geq d(x, y) ,$$

we have

$$\max\{d(x, w), d(y, w)\} - \min\{d(x, w), d(y, w)\} + d(x, y)$$

$$\leq \max\{d(x, w), d(y, w)\} + \max\{d(x, w), d(y, w)\}$$

$$= 2 \times \max\{d(x, w), d(y, w)\}$$

$$\leq 2 \times f(\max\{d(x, w), d(y, w)\}) .$$

Hence, $0 \leq \bar{d}(x, y) \leq 1$. 

## 3.5 Examples

Several similarity and distance metrics have been proposed, for example, in maximal common subgraph, in information distance based on the notion of Kolmogorov complexity, and in evaluating the importance of attributes. These are special solutions of which each is only suitable for a specific context from which it is derived. In the
following, we show that by casting the general solution to each of these contexts, these metrics readily follow.

### 3.5.1 Set Similarity and Distance

**Graph Distance:** An example of graph distance metric [2], based on the notion of maximal common subgraph, is $1 - \frac{|\text{mcs}(G_1, G_2)|}{\max\{|G_1|, |G_2|\}}$ where $\text{mcs}(G_1, G_2)$ denotes the maximal common subgraph between the graphs $G_1$ and $G_2$. For the sake of comparison, we can rewrite this distance as $\frac{\max\{|G_1 \setminus G_2|, |G_2 \setminus G_1|\}}{\max\{|G_1|, |G_2|\}}$ where $|G_i \setminus G_j| = |\text{mcs}(G_i, G_i)| - |\text{mcs}(G_i, G_j)| = |G_i| - |\text{mcs}(G_i, G_j)|$. It is easy to verify that $|\text{mcs}(G_1, G_2)|$ is a similarity metric.

**Attributed Tree Distance:** An attributed tree is a tree of which every node is associated with a vector of attributes. A way of defining a distance metric between two attributed trees is based on maximum similarity subtree isomorphism [22]. Examples are

1. $|T_1| + |T_2| - 2 \times W_{12}$,
2. $\max\{|T_1|, |T_2|\} - W_{12}$,
3. $1 - \frac{W_{12}}{|T_1| + |T_2| - W_{12}}$,
4. $1 - \frac{W_{12}}{\max\{|T_1|, |T_2|\}}$,

where $W_{12}$ is a similarity metric for two attributed trees $T_1$ and $T_2$, with $W_{ii} = |T_i|$, $i \in \{1, 2\}$.

Note that the formulation of the metrics in the above examples is essentially based on the notion of set similarity and distance. We now cast the general solution in this context.
Given finite sets $A$, $B$ and $C$, we have

$$|A \cap B| + |B \cap C| - |A \cap C| \leq |B \cap B| = |B|.$$  

Note that this inequality is the equivalent of that in similarity condition 4. It is easy to verify that $|A \cap B|$ is a similarity metric. Since

$$|A| + |B| - 2|A \cap B| = |A \cup B| - |A \cap B| = |A \backslash B| + |B \setminus A|,$$

and

$$\max\{|A|, |B|\} - |A \cap B| = \max\{|A \setminus B|, |B \setminus A|\},$$

from Lemma 3.8 and 3.9 it follows that both $|A \setminus B| + |B \setminus A|$ and $\max\{|A \setminus B|, |B \setminus A|\}$ are distance metrics.

Referring to Theorem 3.1, it follows that $\frac{|A \cap B|}{|A| + |B| - |A \cap B|} = \frac{|A \cap B|}{|A \cup B|}$ is a similarity metric and $1 - \frac{|A \cap B|}{|A| + |B| - |A \cap B|} = 1 - \frac{|A \cap B|}{|A \cup B|} = \frac{|A \setminus B| + |B \setminus A|}{|A \cup B|}$ is a distance metric.

Referring to Theorem 3.2, it follows that $\frac{|A \cap B|}{\max\{|A|, |B|\}}$ is a similarity metric and $1 - \frac{|A \cap B|}{\max\{|A|, |B|\}} = \frac{\max\{|A|, |B|\} - |A \cap B|}{\max\{|A|, |B|\}} = \frac{\max\{|A \setminus B|, |B \setminus A|\}}{\max\{|A|, |B|\}}$ is a distance metric.

We summarize the results in Table 3.2.

Remark. Note that these are a subset of the metrics that can be obtained from the general solution. Evidently they encompass the metrics in the examples. For the formulae in fractional forms, we have chosen a simple concave function, $f(x) = x$. There are many other choices so long as they meet the functional conditions set forth in previous sections. Certainly, it is much easier to know whether a given function meets these conditions than to explicitly prove that a given formula involving this function is a metric.

3.5.2 Information Similarity and Distance
Data Mining: An attribute is deemed important in data mining if it partitions the database such that new patterns are revealed [25].

Denote by $H(X)$ the information entropy of a discrete random variable $X$, $H(Y|X)$ the entropy of $Y$ conditional on $X$, $H(X,Y)$ the joint entropy of $X$ and $Y$, and $I(X,Y)$ the mutual information between $X$ and $Y$.

Several similarity and distance metrics were proposed in the context of evaluating the importance of attributes. They are listed in Table 3.3.

<table>
<thead>
<tr>
<th>Similarity</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I(X, Y)$, [5, 8, 14]</td>
<td>$H(X</td>
</tr>
<tr>
<td>$\frac{I(X,Y)}{H(X,Y)}$, [10]</td>
<td>$\frac{H(X</td>
</tr>
<tr>
<td>$\frac{f(X,Y)}{\max{H(X),H(Y)}}$, [4, 6]</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.3: Summary: similarity and distance metrics for evaluating the importance of attributes.

The formulation of the metrics in the above examples is essentially based on the notion of information similarity and distance. We now cast the general solution in
this context.

From information theory, we have

\[ H(X|Y) \leq H(X, Z|Y) \leq H(Z|Y) + H(X|Z, Y) \leq H(X|Z) + H(Z|Y) . \]

The mutual information between \( X \) and \( Y \) is defined as

\[ I(X, Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) . \]

With the above, we have

\[
I(X, Y) + I(Y, Z) \\
= (H(X) - H(X|Y)) + (H(Y) - H(Y|Z)) \tag{3.24}
\]
\[
\leq (H(X) - H(X|Z)) + (H(Y) - H(Y|Y)) \tag{3.25}
\]
\[
= I(X, Z) + I(Y, Y) .
\]

Note that from (3.24) to (3.25) the presence of \( H(Y|Y) \) has no effect since \( H(Y|Y) = 0 \). Then, it is easy to verify that \( I(X, Y) \) is a similarity metric.

Since

\[
I(X, X) + I(Y, Y) - 2 \times I(X, Y) \\
= H(X) + H(Y) - H(X) + H(X|Y) - H(Y) + H(Y|X) \\
= H(X|Y) + H(Y|X) ,
\]

and

\[
\max\{I(X, X), I(Y, Y)\} - I(X, Y) \\
= \max\{I(X, X) - I(X, Y), I(Y, Y) - I(Y|X)\} \\
= \max\{H(X|Y), H(Y|X)\} ,
\]
from Lemma 3.8 and 3.9 it follows that both $H(X|Y) + H(Y|X)$ and \( \max\{H(X|Y), H(Y|X)\} \) are distance metrics.

The joint entropy of $X$ and $Y$ is defined as

$$H(X, Y) = H(X) + H(Y|X) = H(Y) + H(X|Y).$$

Referring to Theorem 3.1, we have

$$\frac{I(X, Y)}{I(X, X) + I(Y, Y) - I(X, Y)} = \frac{I(X, Y)}{H(X) + H(Y) - (H(Y) - H(Y|X))} = \frac{I(X, Y)}{H(X) + H(Y|X)} = \frac{I(X, Y)}{H(X, Y)}.$$

Therefore, $\frac{I(X,Y)}{H(X,Y)}$ is a similarity metric, and $1 - \frac{I(X,Y)}{H(X,Y)} = \frac{H(X|Y) + H(Y|X)}{H(X,Y)}$ is a distance metric.

Referring to Theorem 3.2, it follows that

$$\frac{I(X, Y)}{\max\{I(X, X), I(Y, Y)\}} = \frac{I(X, Y)}{\max\{H(X), H(Y)\}}$$

is a similarity metric. Since

$$H(X) - H(X|Y) = H(Y) - H(Y|X)$$

implies

$$\max\{H(X), H(Y)\} - \min\{H(X), H(Y)\} = \max\{H(X|Y), H(Y|X)\} - \min\{H(X|Y), H(Y|X)\},$$
we have

\[ 1 - \frac{I(X,Y)}{\max\{H(X), H(Y)\}} = \frac{2 \times \max\{H(X), H(Y)\} - H(X) - H(Y) + H(X|Y) + H(Y|X)}{2 \times \max\{H(X), H(Y)\}} = \frac{\max\{H(X), H(Y)\} - \min\{H(X), H(Y)\} + H(X|Y) + H(Y|X)}{2 \times \max\{H(X), H(Y)\}} = \frac{\max\{H(X|Y), H(Y|X)\} - \min\{H(X|Y), H(Y|X)\}}{2 \times \max\{H(X), H(Y)\}} + \frac{H(X|Y) + H(Y|X)}{2 \times \max\{H(X|Y), H(Y|X)\}} = \frac{2 \times \max\{H(X), H(Y)\}}{\max\{H(X), H(Y)\}} = \frac{\max\{H(X|Y), H(Y|X)\}}{\max\{H(X), H(Y)\}} \]

which is a distance metric.

We summarize the results in Table 3.4.

<table>
<thead>
<tr>
<th>Similarity</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I(X,Y)$</td>
<td>$H(X</td>
</tr>
<tr>
<td>$\frac{I(X,Y)}{H(X,Y)}$</td>
<td>$\frac{H(X</td>
</tr>
<tr>
<td>$\frac{I(X,Y)}{\max{H(X), H(Y)}}$</td>
<td>$\frac{\max{H(X</td>
</tr>
</tbody>
</table>

**Table 3.4:** Summary: information similarity metrics and distance metrics.

**Remark.** Note the resemblance between the above metrics and those for the case of set, both constructed from the general solution. Furthermore, it is evident that the metrics in these examples can all be obtained from the same principle. In the context of Kolmogorov complexity, basic quantities such as $K(x)$, $K(x,y)$, $K(x|y)$ and $I(x : y)$ are similar to $H(X)$, $H(X,Y)$, $H(X|Y)$ and $I(X,Y)$, respectively. Their respective formulae take on equivalent forms. Analogous to $I(X,Y)$, $I(x : y)$ is a similarity metric. With this, the two distance metrics readily follow from the general solution.
3.5.3 Sequence Edit Distance and Similarity

It is well known that if the cost for basic operations of insertion, deletion, and substitution is a distance metric, then the sequence edit distance \( d(s_1, s_2) \), defined between two sequences \( s_1 \) and \( s_2 \) and derived from finding the minimum-cost sequence of operations that transform \( s_1 \) to \( s_2 \), is also a distance metric.

Several normalized edit distances have been proposed and studied [11, 13]. Examples are

- \( \frac{d(s_1, s_2)}{|s_1| + |s_2|} \),
- \( \frac{d(s_1, s_2)}{\max\{|s_1|, |s_2|\}} \),
- \( n(s_1, s_2) = \min\{\frac{d(s_1, s_2)}{|p|} | p \text{ is a path that changes } s_1 \text{ to } s_2\} \).

Although these are referred to as normalized edit distance, they are not distance metric.

From the results of Section 3.4, choosing \( w \) as the empty sequence, we have two normalized edit distance metrics. If the indel cost is 1, then the following is a distance metric:

\[
\frac{\max\{|s_1|, |s_2|\} - \min\{|s_1|, |s_2|\}}{2 \times \max\{|s_1|, |s_2|\}} + d(s_1, s_2) \]

For sequence similarity, one popular measurement is protein sequence similarity based on BLOSUM matrices using Smith-Waterman algorithm [18]. In fact, based on the original score without rounding, any BLOSUM-\( N \) matrix with \( N \geq 55 \) is a similarity metric. Therefore protein sequence similarity based on those BLOSUM matrices with Smith-Waterman algorithm is a similarity metric.

For normalized sequence similarity, an example is \( \frac{s(s_1, s_2)}{|s_1| + |s_2| + k} \) where \( k > 0 \) [1]. This, however, is not a similarity metric since condition 4 of the similarity metric is not satisfied.
3.6 Conclusions

We have given formal definition for the similarity metric. We have shown the relation between the similarity metric and the distance metric. We have given general formulae to normalize a similarity metric or a distance metric. We have shown, with examples, how the general solutions are useful in constructing metrics suitable for various contexts.
Bibliography


Chapter 4

Normalized Local Similarity:
Sequences and RNA Structures

4.1 Introduction

We apply the similarity metrics that we have developed to find the normalized local similarity for the following objects:

- sequences,
- RNA secondary structures.

These are important problems in biological researches. Two species may appear rather different, when compared globally, yet still be very closely related. This is due to the similarity that exists locally rather than globally between the two species. Although global similarity serves adequately in many situations, local similarity can reveal “hidden” connections between species which may yield significantly meaningful insight unattainable by means of a global measure. Take two sequences, “GGAGGAACG-TAGGAGG” and “CCCTTCACGTCTTCCC” for example. Judged globally, they appear to be unrelated due to the seemingly obvious discrepancy in their respective contents. However, a close inspection reveals a subsequence “ACGT” identically
shared by both sequences as the local similarity. Although it is not always true that
the significance of local similarity outweighs that of global similarity, neglecting such
information when it does matter will likely lead us to wrong conclusions.

In what follows, we discuss ways for computing the similarity between sequences
or RNA secondary structures that are local and normalized.

4.2 Normalized Local Sequence Similarity

In some applications involving finding local sequence similarity, it is more important
to find highly conserved local segments than those high-scoring segments that contain
poorly conserved internal segments. This is the case in comparison of long genomic
sequences and comparative gene prediction, for example. In this scenario, the Smith-
Waterman algorithm [9], which aims to find the subsequences with the highest additive
scores, can produce poor results [2]. In such context, the notion of normalization
becomes desirable. We explain this situation in more detail as follows.

In molecular biology, it is frequently necessary to know how closely two biologi-
cal sequences are related. A biological sequence may undergo mutations producing
descendant sequences with sequence content differing at the mutation site. After sev-
eral generations of mutations, two descendant sequences originating from the same
ancestral sequence may look quite different globally, and yet still share a commonly
conserved segment. This commonly conserved segment, referred to as local similarity,
serves as an important indication of the connection between various species; hence,
the importance of finding local sequence similarity.

Traditionally, local sequence similarity is defined in terms of optimal alignment
score with respect to a given scoring scheme. The Smith-Waterman algorithm was
designed to compute this optimal alignment score. Due to many successful applica-
tions in the biological context, it has become one of the most important techniques in
molecular biology. However, this similarity is defined in terms of the overall additive
score of an alignment, which does not necessarily imply a consistent and uniform distribution of high scores throughout this alignment. In certain situations, this may produce results which are computationally optimal but not the most desirable in the context of the application. Figure 4.1 illustrates this situation. Suppose that the Smith-Waterman algorithm returns the pair of subsequences depicted in 4.1(a) as the highest scoring pair. Within this local alignment, each of the two shaded border segments yields a positive score (60 and 50) whereas the internal segment yields a negative score (−45) which is in magnitude slightly less than the score of one of the two border segments (the right one). The overall score (65) is greater than any one of the three segments. Suppose that the corresponding additive score along this alignment is depicted in 4.1(b). The internal segment corresponds to a poorly conserved pair where the additive score tends to descend within this segment. Therefore, although this alignment is associated with the highest score, it is not an adequate candidate for representing a highly conserved segment. Either one of the border segments may be a preferred candidate. This example shows the importance of taking consistency into account when evaluating the quality of an alignment by scores.

The cause of this situation is known as mosaic effect or shadow effect as depicted in Figure 4.2 where an optimal path returned by the Smith-Waterman algorithm in the dynamic programming table is shown.

As a remedy to this situation, we need to normalize the scores. Instead of evaluating the quality of an alignment by additive scores, we resort to the normalized scores.

In what follows, we show applications of normalized similarity metrics in finding local sequence similarity. The optimization techniques we use here are adapted from those used in [2]; however, the normalized similarity formula used therein is not a metric.

Given a sequence \( S \), denote by \( S[i] \) the \( i \)th element of \( S \), and \( S[i, j] \) the subsequence of \( S \) beginning at \( S[i] \) and ending at \( S[j] \). Note that \( S[i, j] = \emptyset \) if \( i > j \). Given two
Figure 4.1: An example of non-uniform quality within a local alignment by the Smith-Waterman algorithm between $S_1$ and $S_2$ where the optimally aligned subsequences are $(S_1[i, j], S_2[k, l])$. 

(a) non-uniform alignment quality

(b) score distribution for the three alignment segments
Figure 4.2: Examples of mosaic effect and shadow effect.
sequences \( S_a \) and \( S_b \), denote by \( \theta_{ab}(i', i, j', j) \) the similarity of the subsequences \( S_a[i', i] \) and \( S_b[j', j] \). The local sequence similarity between \( S_a \) and \( S_b \), of lengths \( m \) and \( n \) respectively, is defined as \( \max\{\theta_{ab}(i', i, j', j) \mid 1 \leq i' \leq i \leq m, 1 \leq j' \leq j \leq n\} \).

The problem of computing the local sequence similarity is solvable in quadratic time using standard dynamic programming techniques [10]. Our interest is in solving the normalized version of the local sequence similarity problem. Applying the formulae for normalized similarity in previous sections, we formulate normalized local sequence similarity for \( S_1 \) and \( S_2 \) as follows.

1. \( \max\{\frac{\theta_{12}(i', i, j', j)}{h+\theta_{11}(i', i, i)+\theta_{22}(j', j, j)-\theta_{12}(i', i, j', j)} \mid 1 \leq i' \leq i \leq m, 1 \leq j' \leq j \leq n, h \geq 0\} \).  

2. \( \max\{\frac{\theta_{12}(i', i, j', j)}{f(\theta_{11}(i', i, i)+\theta_{22}(j', j, j)-\theta_{12}(i', i, j', j))} \mid 1 \leq i' \leq i \leq m, 1 \leq j' \leq j \leq n, f(x) \text{ is concave over } [0, \infty), f(x) \geq x, f(x) \geq f(y) \text{ if } x \geq y\} \).

3. \( \max\{\frac{\theta_{12}(i', i, j', j)}{h+\max\{\theta_{11}(i', i, i), \theta_{22}(j', j, j), \theta_{12}(i', i, j', j)\}} \mid 1 \leq i' \leq i \leq m, 1 \leq j' \leq j \leq n, h \geq 0\} \).

4. \( \max\{\frac{\theta_{12}(i', i, j', j)}{g(h+\max\{\theta_{11}(i', i, i), \theta_{22}(j', j, j), \theta_{12}(i', i, j', j)\}))} \mid 1 \leq i' \leq i \leq m, 1 \leq j' \leq j \leq n, g(x) \geq x, g(x) \geq g(y) \text{ if } x \geq y\} \).

In this paper, we present methods for computing the normalized sequence similarity based on formula 1 and 2.

The method based on formula 1 calls for a technique known as fractional programming [2, 7, 11]. It casts the normalization problem into a parametric problem with the parameter to be optimized being the normalized similarity. It then progressively solves the parametric problem in iterations until the solution reaches an acceptable stability, at which point the solution obtained from the parametric problem is also the solution for the normalization problem.

We now describe the parametric problem and its relation to the normalization problem. For subsequences \( S_1[i', i] \) and \( S_2[j', j] \), define \( \lambda = \frac{\theta_{12}(i', i, j', j)}{h+\theta_{11}(i', i, i)+\theta_{22}(j', j, j)-\theta_{12}(i', i, j', j)} \). The corresponding parametric problem is defined
as \( P(\lambda) = \max_{1 \leq i', j' \leq n} \{(1 + \lambda) \times \theta_{12}(i', i, j') - \lambda \times (\theta_{11}(i', i, i') + \theta_{22}(j', j, j'))\} \).

The procedure works as follows. Solve \( P(\lambda) \) with \( \lambda = 0 \) and obtain the corresponding \( \theta_{12}, \theta_{11}, \text{ and } \theta_{22} \) with which a new \( \lambda \) is obtained. Solve \( P(\lambda) \) again with the new \( \lambda \) and obtain new \( \theta_{12}, \theta_{11}, \text{ and } \theta_{22} \) with which \( \lambda \) is updated again. Continue this process until two consecutive \( \lambda \) values agree within a preset threshold. This final \( \lambda \) is taken as the normalized local similarity for \( S_1 \) and \( S_2 \) according to the formula. The procedure is illustrated in Algorithm 11.

**Algorithm 11:** computing normalized local sequence similarity using formula 1 and fractional programming

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>((i', i, j', j) \leftarrow \arg \max_{1 \leq i', j' \leq n} {\theta_{12}(i', i, j')} )</td>
</tr>
<tr>
<td>2</td>
<td>( \lambda \leftarrow \frac{h + \theta_{11}(i', i', i') + \theta_{22}(j', j', j') - \theta_{12}(i', i, j', j')}{} )</td>
</tr>
<tr>
<td>3</td>
<td>( \lambda' \leftarrow \lambda - 1 )</td>
</tr>
<tr>
<td>4</td>
<td><strong>while</strong> ( \lambda - \lambda' &gt; \delta ) <strong>do</strong></td>
</tr>
<tr>
<td>5</td>
<td>( \lambda' \leftarrow \lambda )</td>
</tr>
<tr>
<td>6</td>
<td>((i', i, j', j) \leftarrow \arg \max_{1 \leq i', j' \leq n} {(1 + \lambda') \times \theta_{12}(i', i, j') - \lambda' \times (\theta_{11}(i', i, i') + \theta_{22}(j', j, j'))} )</td>
</tr>
<tr>
<td>7</td>
<td>( \lambda \leftarrow \frac{h + \theta_{11}(i', i', i') + \theta_{22}(j', j', j') - \theta_{12}(i', i, j', j')}{} )</td>
</tr>
<tr>
<td>8</td>
<td><strong>endw</strong></td>
</tr>
<tr>
<td>9</td>
<td>return ( \lambda )</td>
</tr>
</tbody>
</table>

For convenience, we define the following notations. We order the series of \( \lambda \) values in the same order as they are obtained in the iterative procedure and denote by \( \lambda^i \) the \( i \)th \( \lambda \) in the series. Denote by \( \theta_{ab}^i \), with \( a, b \in \{1, 2\} \), the values obtained by solving \( P(\lambda^i) \). We can now write:

\[
P(\lambda^i) = (1 + \lambda^i) \times \theta_{12}^i - \lambda^i \times (\theta_{11}^i + \theta_{22}^i) \quad (4.1)
\]

\[
\lambda^{i+1} = \frac{\theta_{12}^i}{h + \theta_{11}^i + \theta_{22}^i - \theta_{12}^i} \quad (4.2)
\]

**Lemma 4.1.** For any \( \lambda^i \), \( P(\lambda^i) \geq \lambda^i \times h \).
Proof. Assume that \( \exists i \) such that \( P(\lambda^i) < \lambda^i \times h \). That is,

\[
(1 + \lambda^i) \times \theta_{12}^i - \lambda^i \times (\theta_{11}^i + \theta_{22}^i) < \lambda^i \times h.
\]

By Equation 4.2,

\[
\lambda^i = \frac{\theta_{12}^{i-1}}{(h + \theta_{11}^{i-1} + \theta_{22}^{i-1} - \theta_{12}^{i-1})}.
\]

That is,

\[
(1 + \lambda^i) \times \theta_{12}^{i-1} - \lambda^i \times (\theta_{11}^{i-1} + \theta_{22}^{i-1}) = \lambda^i \times h.
\]

However, this means that

\[
P(\lambda^i) = \lambda^i \times h.
\]

Based on the definition of \( P(\lambda^i) \), we have a contradiction. \( \square \)

**Lemma 4.2.** For any \( i \), \( \lambda^{i+1} \geq \lambda^i \).

**Proof.** By Lemma 4.1,

\[
P(\lambda^i) \geq \lambda^i \times h
\]

\[
\Rightarrow (1 + \lambda^i) \times \theta_{12}^i - \lambda^i \times (\theta_{11}^i + \theta_{22}^i) \geq \lambda^i \times h
\]

\[
\Rightarrow \frac{\theta_{12}^i}{h + \theta_{11}^i + \theta_{22}^i - \theta_{12}^i} \geq \lambda^i
\]

\[
\Rightarrow \lambda^{i+1} \geq \lambda^i
\]

\( \square \)

The convergence of the procedure is guaranteed by Lemma 4.2.

Each instance of the parametric problem can be solved in \( \mathcal{O}(m \times n) \) time, therefore the time complexity of Algorithm 11 is \( \mathcal{O}(k \times m \times n) \) where \( k \) is the number of iterations. Somewhat uneasy about this result is the fact that there is no clear theoretical bound established for the number of iterations. The situation can be improved by adapting a technique introduced in [1]. The central idea comes from the
observation that the difference between any two possible values for \( \lambda \) is at least as large as \( \frac{1}{c^x(m+n)^2} \) where \( c \) is a constant dependent on the similarity scoring function. We first pick a range large enough to contain the optimal \( \lambda \) value. Partition this range into smaller ranges each equal \( \frac{1}{c^x(m+n)^2} \). The optimal \( \lambda \) value must fall within one of the smaller ranges. With this, in the initial stage we can use binary search to quickly eliminate a large number of candidates, until we get fairly close to where the optimal value is, then we may slow down the iteration speed by switching to fractional programming. This leads to the following theorem. The procedure is illustrated in Algorithm 12.

**Theorem 4.1.** The normalized local similarity between two sequences \( S_1 \) and \( S_2 \) of lengths \( m \) and \( n \), respectively, can be computed in \( \mathcal{O}(m \times n \times \log(m + n)) \) time and \( \mathcal{O}(\min\{m, n\}) \) space, according to the definition of lengths \( 1 \) and \( \lambda \) is at least as large as \( \frac{1}{c^x(m+n)^2} \).

We now turn to the method based on formula 2. Note that the fractional programming approach is not feasible in this case as the presence of the function \( f \) prevents us from casting the normalization problem into a parametric problem as previously described. For arbitrary integral scoring schemes, the problem is solvable in cubic time by using three-dimensional dynamic programming as follows. Define \( \Gamma(i, j, k) \) to be the similarity for \( (S_1[i', i], S_2[j', j]) \) such that \( \theta_{11}(i', i, i, i) + \theta_{22}(j', j, j, j) = k \). Then the normalized local similarity we are seeking can be written as \( \max\left\{ \frac{\Gamma(i, j, k)}{f(k-\Gamma(i, j, k))} \mid 1 \leq i' \leq i \leq m, 0 \leq j' \leq j \leq n, h \geq 0 \right\} \).

**Lemma 4.3.** \( \Gamma(i, j, k) = \max \{0, \Gamma(i', j', k') + \theta_{12}(i, i, i, j) \times (i - i') \times (j - j') + \theta_{12}(i, 0, 0, j) \times (i - i') \times (1 - j + j') + \theta_{22}(0, 0, j, j) \times (1 - i + i') \times (j - j') \mid (i', j') \in \{(i - 1, j - 1), (i - 1, j), (i, j - 1)\}, k' = k - \theta_{11}(i, i, i, i) \times (i - i') - \theta_{22}(j, j, j, j) \times (j - j'), k' \geq 0 \} \).

**Proof.** Each \( (i, j) \) of \( \Gamma(i, j, k) \) comes from exactly one of three possible directions: \( (i - 1, j - 1) \), \( (i - 1, j) \), or \( (i, j - 1) \). Associated with each direction is a single-element similarity, i.e., \( \theta_{12}(i, i, j, j) \), \( \theta_{12}(i, i, 0, 0) \), and \( \theta_{12}(0, 0, j, j) \), respectively. For
Algorithm 12: computing normalized local sequence similarity using formula 1 and binary search

**Input:** $S_1[1, m]$, $S_2[1, n]$, $h > 0$, $0 \leq (\delta_1, \delta_2) < 1$, $c \in \mathbb{N}$

**Output:** $\max_{1 \leq i' \leq i \leq m, 1 \leq j' \leq j \leq n} \{ (1 + \lambda') \times \theta_{12}(i', i, j', j) - \lambda \times (\theta_{11}(i', i, i) + \theta_{22}(j', j, j)) \}$

1. $\beta \leftarrow \frac{1}{c \times (m + n)^2}$
2. $[a, b] \leftarrow [0, c \times (m + n)^2]$
3. $e = 0$
4. while $e = 0$ and $b - a > 1$
5. \[ \alpha \leftarrow \frac{a + b}{2} \]
6. $\lambda \leftarrow \alpha \times \beta$
7. $x \leftarrow \max_{1 \leq i' \leq i \leq m} \{ (1 + \lambda) \times \theta_{12}(i', i, j', j) - \lambda \times (\theta_{11}(i', i, i) + \theta_{22}(j', j, j)) \}$
8. if $x - \lambda \times h > \delta_1$ then $a \leftarrow \alpha$
9. else if $\lambda \times h - x > \delta_1$ then $b \leftarrow \alpha$
10. else $e \leftarrow 1$
11. endw
12. $\lambda \leftarrow a \times \beta$
13. $\lambda' \leftarrow \lambda - 1$
14. while $\lambda - \lambda' > \delta_2$
15. \[ \lambda' \leftarrow \lambda \]
16. \[ (i', \lambda', j', j) \leftarrow \arg \max_{1 \leq i' \leq i \leq m} \{ (1 + \lambda') \times \theta_{12}(i', i, j', j) - \lambda' \times (\theta_{11}(i', i, i) + \theta_{22}(j', j, j)) \} \]
17. $\lambda \leftarrow \frac{h + \theta_{11}(i', i, i') + \theta_{22}(j', j, j') - \theta_{12}(i', i, j', j)}{h + \theta_{11}(i', i, i'} + \theta_{22}(j', j, j') - \theta_{12}(i', i, j', j')}$
18. endw
19. return $\lambda$
each direction, \( k \) comes from a nonnegative value \( k' \) which equals \( k \) subtracting the corresponding \( \theta \) value. If \( \Gamma(i, j, k) \) is negative, the entry is set to 0 since it means total dissimilarity.

**Algorithm 13:** computing normalized local sequence similarity using formula 2

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( x \leftarrow 0 )</td>
</tr>
<tr>
<td>2</td>
<td>( K \leftarrow \theta_{11}(1, m, 1, m) + \theta_{22}(1, n, 1, n) )</td>
</tr>
<tr>
<td>3</td>
<td>for ( k \leftarrow 0 ) to ( K ) do</td>
</tr>
<tr>
<td>4</td>
<td>( \Gamma(0, 0, k) \leftarrow 0 )</td>
</tr>
<tr>
<td>5</td>
<td>for ( i \leftarrow 1 ) to ( m ) do</td>
</tr>
<tr>
<td>6</td>
<td>( \Gamma(i, 0, k) \leftarrow 0 )</td>
</tr>
<tr>
<td>7</td>
<td>endfor</td>
</tr>
<tr>
<td>8</td>
<td>for ( j \leftarrow 1 ) to ( n ) do</td>
</tr>
<tr>
<td>9</td>
<td>( \Gamma(0, j, k) \leftarrow 0 )</td>
</tr>
<tr>
<td>10</td>
<td>endfor</td>
</tr>
<tr>
<td>11</td>
<td>for ( i \leftarrow 1 ) to ( m ) do</td>
</tr>
<tr>
<td>12</td>
<td>for ( j \leftarrow 1 ) to ( n ) do</td>
</tr>
<tr>
<td>13</td>
<td>if ( k = 0 ) then ( \Gamma(i, j, k) \leftarrow 0 )</td>
</tr>
<tr>
<td>14</td>
<td>compute ( \Gamma(i, j, k) ) according to Lemma 4.3</td>
</tr>
<tr>
<td>15</td>
<td>if ( \Gamma(i, j, k) &gt; x ) then ( x \leftarrow \Gamma(i, j, k) )</td>
</tr>
<tr>
<td>16</td>
<td>endfor</td>
</tr>
<tr>
<td>17</td>
<td>endfor</td>
</tr>
<tr>
<td>18</td>
<td>( a[k] \leftarrow x )</td>
</tr>
<tr>
<td>19</td>
<td>endfor</td>
</tr>
<tr>
<td>20</td>
<td>( x \leftarrow 0 )</td>
</tr>
<tr>
<td>21</td>
<td>for ( k \leftarrow 1 ) to ( K ) do</td>
</tr>
<tr>
<td>22</td>
<td>if ( \frac{a[k]}{f(k-a[k])} &gt; x ) then ( x \leftarrow \frac{a[k]}{f(k-a[k])} )</td>
</tr>
<tr>
<td>23</td>
<td>endfor</td>
</tr>
<tr>
<td>24</td>
<td>return ( x )</td>
</tr>
</tbody>
</table>

The procedure is illustrated in Algorithm 13. Using dynamic programming, we compute \( \Gamma(i, j, k) \) according to the recurrence formula in Lemma 4.3. This step takes \( \mathcal{O}(m \times n \times (m + n)) \) time. The results are stored in a three-dimensional array. The normalized local similarity comes from an entry which maximizes \( \frac{\Gamma(i, j, k)}{f(k-\Gamma(i, j, k))} \). This entry can be identified in \( \mathcal{O}(m + n) \) time if the maximum values with respect to each
$k$ have been previously stored. Note that although the procedure in Algorithm 13 uses cubic space, the space usage can be reduced to quadratic if we rearrange the order of the loops while using linear space for the $ij$-dimension. This leads to the following result.

**Theorem 4.2.** The normalized local similarity between two sequences $S_1$ and $S_2$ of lengths $m$ and $n$, respectively, can be computed in $O(m \times n \times (m + n))$ time and $O(m \times n)$ space, according to the definition $\max \{ \theta_{12}(i',i,j,j') \mid 1 \leq i' \leq i \leq m, \ 1 \leq j' \leq j \leq n, \ f(x) \text{ is concave over } [0, \infty), \ f(x) \geq x, \ f(x) \geq f(y) \text{ if } x \geq y \}$.

### 4.3 Normalized Local RNA Structural Similarity

There are good reasons for finding common features between RNA molecules that takes into consideration both the sequential and the structural information. The secondary structure of an RNA molecule determines a major fraction of its biological function [8]. Furthermore, the structure of an RNA molecule is often more conserved than its sequence during evolution [4].

The secondary structure of RNA can be represented in two ways with a tree or an annotated sequence as in Figure 4.3.

We have presented operations for tree editing. For annotated sequences, the edit operations are shown in Figure 4.4. The two approaches are equivalent.

There are algorithms for computing the local similarity between two RNA structures [3, 5, 6]. They are generalization of the Smith-Waterman algorithm. To normalize the local similarity for RNA structures, the normalizing methods presented in previous section are readily applicable with these algorithms.
Figure 4.3: Representations for RNA.

Figure 4.4: Edit operations for annotated RNA sequences.
4.4 Conclusions

We have presented algorithmic techniques for normalizing the local similarity between two sequences as well as between two RNA structures.
Bibliography


Chapter 5

Conclusions

Results regarding discrete yet related topics have been presented.

Firstly, we concerned ourselves with computing the tree edit distance which serves as a metric for measuring the similarity between trees. We have developed techniques for reducing the algorithmic running times for the state-of-the-art algorithms by incorporating structural linearity in the algorithms.

Secondly, we concerned ourselves with the construction of general similarity and distance metrics. This consists of formal metric definitions as well as general formulae for similarity and distance. We have shown that a collection of metrics used in various diverse fields are special cases of our result. It has also been revealed that some existing similarity formulae are not metrics. Suggestions have been given for modifying these formulae so as to be metrics.

Thirdly, we concerned ourselves with normalized local similarity for sequences as well as RNA secondary structures. We have designed algorithms for computing such information.

As to future work, we envision two directions.

The first direction concerns continuation of the analytical and algorithmic framework laid out in this thesis. In the context of tree edit distance, one aspect that has not been probed in this work is the analysis of the averaged complexities for the
strategy-based algorithms. It is a known fact that the averaged case is in general harder to analyze than the worst case. Nonetheless, the averaged complexity often conveys more useful information than the worst-case complexity and thus deserves efforts. In the context of the similarity metric and the distance metric, what has been done here is a generalization of some existing specialized metrics. It is possible, however, that certain existing metrics are not derivable from this framework. Should this be the case, it may be necessary to expand the current framework. This is an aspect worthy of investigation. Another related issue is the design of efficient algorithms for computing similarity or distance based on available metrics. In a practical sense, the more algorithms at our disposal the better.

The second direction concerns the synergic aspects stemming from the current works. As pointed out in the text, the importance of the works is exemplified in many diverse fields. Conversely, the value of the solutions obtained here can be best exemplified through applications in other fields. To this end, it calls for collaborative efforts to bring together various fields so as to facilitate a synergism.

To sum up future endeavors: Possibilities abound.
Vita

Name: Shi hyen Chen

Post-secondary Education and Degrees:
- Chung-Hsing University, Taichung, Taiwan, B.Sc.
- The University of Memphis, Memphis, Tennessee, USA, M.Sc., Ph.D.
- The University of Western Ontario, London, Ontario, Canada, M.Sc.

Related Work Experience:
- Teaching and Research Assistant, The University of Memphis
- Teaching and Research Assistant, The University of Western Ontario

Publications:


