Information Processing Letters 116 (2016) 744–749

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

An efficient algorithm for computing non-overlapping inversion and transposition distance

Toan Thang Ta, Cheng-Yao Lin, Chin Lung Lu *

Department of Computer Science, National Tsing Hua University, Hsinchu 30013, Taiwan

ARTICLE INFO

Article history:
Received 17 March 2015
Received in revised form 8 March 2016
Accepted 16 July 2016
Available online 21 July 2016
Communicated by M. Chrobak

Keywords:
Algorithms
Computational biology
Inversion
Transposition
Mutation distance

ABSTRACT

Given two strings of the same length n, the non-overlapping inversion and transposition distance (also called mutation distance) between them is defined as the minimum number of non-overlapping inversion and transposition operations used to transform one string into the other. In this study, we present an $O(n^3)$ time and $O(n^2)$ space algorithm to compute the mutation distance of two input strings.

1. Introduction

The dissimilarity of two strings is usually measured by the so-called edit distance, which is defined as the minimum number of edit operations necessary to convert one string into the other. The commonly used edit operations are character insertions, deletions and substitutions. In biological application, the aforementioned edit operations correspond to point mutations of DNA sequences (i.e., mutations at the level of individual nucleotides). From evolutionary point of view, however, DNA sequences may evolve by large-scale mutations (also called rearrangements, i.e., mutations at the level of sequence fragments) [6], such as inversions (i.e., replacing a fragment of DNA sequence by its reverse complement) and transpositions (i.e., moving a fragment of DNA sequence from one location to another or, equivalently, exchanging two adjacent and non-overlapping fragments on DNA sequence). Note that a large-scale mutation that replaces a fragment of DNA sequence only by its reverse (without complement) is called a reversal. Based on large-scale mutation operations, the dissimilarity (or mutation distance) between two strings can be defined to be the minimum number of large-scale mutation operations used to transform one string to the other. Cantone et al. [1] introduced an $O(nm)$ time and $O(m^2)$ space algorithm to solve an approximate string matching problem with non-overlapping reversals, which is to find all locations of a given text that match a given pattern with non-overlapping reversals, where n is the length of the text and m is the length of the pattern. In this problem, two equal-length strings are said to have a *match with non-overlapping reversals* if one string can be transformed into the other using any finite sequence of non-overlapping reversals. It should be noted that the number of the used non-overlapping reversals in the algorithm proposed by Cantone et al. [1] is not required to be less than or equal to a fixed non-negative integer. In [1], Cantone et al. also presented another algorithm whose average-case time complexity is $O(n)$. Cantone et al. [2] studied the same problem by considering both non-overlapping rever-

* Corresponding author.
E-mail addresses: toanthanghy@gmail.com (T.T. Ta), begoingto0830@gmail.com (C.-Y. Lin), cllu@cs.nthu.edu.tw (C.L. Lu).

http://dx.doi.org/10.1016/j.ipl.2016.07.004
0020-0190/

sals and transpositions, where they called transpositions as translocations and the lengths of two exchanged adjacent fragments are constrained to be equivalent. They finally designed an algorithm to solve this problem in $O(nm^2)$ time and $O(m^2)$ space. For the above problem, Grabowski et al. [4] gave another algorithm whose worst-case time and space are $O(nm^2)$ and $O(m)$, respectively. Moreover, they proved that their algorithm has an $O(n)$ average time complexity. Recently, Huang et al. [5] studied the above approximate string matching problem under non-overlapping reversals by further restricting the number of the used reversals not to exceed a given positive integer k. They proposed a dynamic programming algorithm to solve this problem in $O(nm^2)$ time and $O(m^2)$ space.

In this work, we are interested in the computation of the mutation distance between two strings of the same length under non-overlapping inversions and transpositions (i.e., non-overlapping inversion and transposition distance), where the lengths of two adjacent and non-overlapping fragments exchanged by a transposition can be different. For this problem, we devise an algorithm whose time and space complexities are $O(n^3)$ and $O(n^2)$, respectively, where n is the length of two input strings. The rest of the paper is organized as follows. In Section 2, we provide some notations that are helpful when we present our algorithm later. Next, we develop the main algorithm and also analyze its time and space complexities in Section 3. Finally, we have a brief conclusion in Section 4.

2. Preliminaries

Let x be a string of length n over a finite alphabet Σ. A character at position i of x is represented with x_i, where $1 \le i \le n$. A substring of x from position i to j is indicated as $x_{i,j}$, i.e., $x_{i,j} = x_i x_{i+1} \dots x_j$, for $1 \le i \le j \le n$. In biological sequences, $\Sigma = \{A, C, G, T\}$, in which A–T and C–G are considered as complementary base pairs. We use $\theta(x)$ to denote an inversion operation acting on a string x, resulting in a reverse and complement of x. For example, $\theta(A) = T$, $\theta(T) = A$, $\theta(G) = C$, $\theta(C) = G$ and $\theta(CGA) = TCG$. In addition, we utilize $\tau(uv) = vu$ to represent a transposition operation to exchange two non-empty strings u and v. Note that the lengths of u and v are required to be identical in some previous works [2–4], but they may be different in this study. For convenience, we call θ and τ as *mutation operations*. We also let $\theta_{i,j}(x) = \theta(x_{i,j})$ for $1 \le i \le j \le n$ and $\tau_{i,j,k}(x) = x_{k,j} x_{i,k-1}$ for $1 \le i < k \le j \le n$, where $[i, j]$ is called a *mutation range* for $\theta_{i,j}$ and $\tau_{i,j,k}$.

For an integer $1 \le t \le n$, we say that a mutation operation $\theta_{i,j}$ or $\tau_{i,j,k}$ *covers* t if $i \le t \le j$. Given two mutation operations, they are *non-overlapping* if the intersection of their mutation ranges is empty. In this study, we are only interested in sets of non-overlapping mutation operations. Given a set Θ of non-overlapping mutation operations and a string x, let $\Theta(x)$ be the resulting string after consecutively applying the mutation operations in Θ on x. For example, suppose that $\Theta = \{\tau_{1,3,2}, \theta_{5,5}\}$ and $x = TAGAC$. Then we have $\Theta(x) = AGTAG$.

$M_1[i,j]$	$i=1$	2	3	4	5
$j=1$	$(1,1,A)$	$(2,1,A)$	$(3,1,A)$	$(4,1,A)$	$(5,1,A)$
2	$(1,2,T)$	$(2,2,T)$	$(3,2,T)$	$(4,2,T)$	$(5,2,T)$
3	$(1,3,C)$	$(2,3,C)$	$(3,3,C)$	$(4,3,C)$	$(5,3,C)$
4	$(1,4,T)$	$(2,4,T)$	$(3,4,T)$	$(4,4,T)$	$(5,4,T)$
5	$(1,5,G)$	$(2,5,G)$	$(3,5,G)$	$(4,5,G)$	$(5,5,G)$

$M_2[i,j]$	$i=1$	2	3	4	5
$j=1$	$(1,1,T)$	$(2,1,T)$	$(3,1,T)$	$(4,1,T)$	$(5,1,T)$
2	$(1,2,A)$	$(2,2,A)$	$(3,2,A)$	$(4,2,A)$	$(5,2,A)$
3	$(1,3,G)$	$(2,3,G)$	$(3,3,G)$	$(4,3,G)$	$(5,3,G)$
4	$(1,4,A)$	$(2,4,A)$	$(3,4,A)$	$(4,4,A)$	$(5,4,A)$
5	$(1,5,C)$	$(2,5,C)$	$(3,5,C)$	$(4,5,C)$	$(5,5,C)$

Fig. 1. Mutation tables M_1 and M_2 for a given string $x = TAGAC$, where the column is indexed by i and the row by j. Shaded entries respectively represent the inversion $\theta_{1,3}(x)$ on M_1 and the transposition operation $\tau_{1,5,4}(x)$ on M_2.

Definition 1 *(Non-overlapping inversion and transposition distance).* Given two strings x and y of the same length, the *non-overlapping inversion and transposition distance* (simply called *mutation distance*) between x and y, denoted by $md(x, y)$, is defined as the minimum number of non-overlapping inversion and transposition operations used to transform x into y. If there does not exist any set of non-overlapping mutation operations that converts x into y, then $md(x, y)$ is infinite. Formally,

$$md(x,y) = \begin{cases} \min\{|\Theta| : \Theta(x) = y\} \\ \qquad \text{if } \exists \Theta \text{ such that } \Theta(x) = y \\ \infty \quad \text{otherwise} \end{cases}$$

For example, let $x = TAGAC$ and $y = TAACG$. Clearly, there are only two sets of mutation operations $\Theta_1 = \{\theta_{1,2}, \tau_{3,5,4}\}$ and $\Theta_2 = \{\tau_{3,5,4}\}$ such that $\Theta_1(x) = y$ and $\Theta_2(x) = y$. Therefore, $md(x, y) = |\Theta_2| = 1$.

3. The algorithm

Basically, a transposition (respectively, inversion) operation acting on a string x can be considered as a permutation of characters in x (respectively, complement of x). From this view point, a mutation operation actually comprises several sub-operations, called as *mutation fragments*, each of which is denoted either by a tuple (i, j, x_j) or $(i, j, \theta(x_j))$. The mutation fragment (i, j, x_j) (respectively, $(i, j, \theta(x_j))$) means that x_j (respectively, complement of x_j) is moved into the position i in the resulting string obtained when applying the mutation operation on x. For convenience, we arrange all possible mutation fragments in two $n \times n$ two-dimensional tables M_1 and M_2, called *mutation tables* of x, as follows.

- $M_1[i, j] = (i, j, \theta(x_j))$ for $i, j = 1, 2, \dots, n$.
- $M_2[i, j] = (i, j, x_j)$ for $i, j = 1, 2, \dots, n$.

For example, let $x = TAGAC$. Then its mutation tables are shown in Fig. 1.

When an inversion operation $\theta_{i,j}$ applies on a string x, we can decompose it into $(j - i + 1)$ mutation fragments $\mathcal{F}(\theta_{i,j}, x, t)$ for $i \le t \le j$, where $\mathcal{F}(\theta_{i,j}, x, t) =$