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 $\mathcal{F}(\theta_{i,j}, x, t) =$

$(t, i+j-t, \theta(x_{i+j-t}))$. Similarly, a transposition operation $\tau_{i,j,k}$ can be also decomposed into $(j-i+1)$ mutation fragments $\mathcal{F}(\tau_{i,j,k}, x, t)$ for $i \le t \le j$, where if $i \le t \le i+j-k$, then $\mathcal{F}(\tau_{i,j,k}, x, t) = (t, k+t-i, x_{k+t-i})$; otherwise (that is, when $i+j-k < t \le j$), $\mathcal{F}(\tau_{i,j,k}, x, t) = (t, t-j+k-1, x_{t-j+k-1})$. For the purpose of brevity, we let $\tau_{i,j,k}(x,1) = \{(t, k+t-i, x_{k+t-i}) : i \le t \le i+j-k\}$ and $\tau_{i,j,k}(x,2) = \{(t, t-j+k-1, x_{t-j+k-1}) : i+j-k < t \le j\}$.

The aforementioned decomposition can be extended to apply on a set Θ of non-overlapping mutation operations. That is, when Θ acts on a string x, it can be decomposed into a sequence $\mathcal{F}(\Theta, x) = \langle F_1, F_2, \ldots, F_n\rangle$ of mutation fragments by the following formula: For $t = 1, 2, \ldots, n$,

$$F_t = \begin{cases} \mathcal{F}(\theta_{i,j}, x, t) & \text{if } \exists \theta_{i,j} \in \Theta \text{ that covers } t \\ \mathcal{F}(\tau_{i,j,k}, x, t) & \text{if } \exists \tau_{i,j,k} \in \Theta \text{ that covers } t \\ (t, t, x_t) & \text{otherwise} \end{cases}$$

For instance, given $\Theta = \{\tau_{1,2,2}, \theta_{4,5}\}$ and $x = TAGAC$, we have $\mathcal{F}(\Theta, x) = \langle(1,2,\mathrm{A}), (2,1,\mathrm{T}), (3,3,\mathrm{G}), (4,5,\mathrm{G}), (5,4,\mathrm{T})\rangle$.

Observation 1. Given a string x and its mutation tables M_1 and M_2, the result of an inversion $\theta_{i,j}(x)$ can be obtained by concatenating $(j-i+1)$ mutation fragments starting at $M_1[i,j]$ and continuing to move in the anti-diagonal direction to $M_1[j,i]$ (see the shaded entries of M_1 in Fig. 1 for an example). Moreover, the result of a transposition $\tau_{i,j,k}(x)$ is obtained by concatenating the mutation fragments on the following two paths, one starting at $M_2[i,k]$ and continuing to move in the diagonal direction to $M_2[i+j-k, j]$ and the other beginning at $M_2[i+j-k+1, i]$ and also continuing to move in the diagonal direction to $M_2[j, k-1]$ (see the shaded entries of M_2 in Fig. 1 for an example).

For a mutation fragment $F = (i, j, \sigma)$, we say that F *yields* the character σ, denoted by $\ell(F) = \sigma$. If a sequence $S = \langle F_1, F_2, \ldots, F_m\rangle$ consists of m mutation fragments (m is also considered as the *length* of S) with $\ell(F_i) = \sigma_i$ for $1 \le i \le m$, then we say that S yields a string $\sigma_1\sigma_2\ldots\sigma_m$, which is further written as $\ell(S) = \sigma_1\sigma_2\ldots\sigma_m$. A subsequence P containing first t elements of S, i.e., $P = \langle F_1, F_2, \ldots, F_t\rangle$, is called a *prefix* of S and denoted by $P \sqsubset S$, where $1 \le t \le m$.

Definition 2 *(Agreed sequence).* A sequence $S = \langle F_1, F_2, \ldots, F_m\rangle$ of m mutation fragments is an *agreed sequence* if there exists a set Θ of non-overlapping mutation operations such that $S \sqsubset \mathcal{F}(\Theta, x)$, where $m \le n$.

Given a mutation operation $\theta_{i,j}$ or $\tau_{i,j,k}$, we call i and j as the *left end point* and *right end point* of the mutation operation, respectively. We also say that this mutation operation (i.e., $\theta_{i,j}$ or $\tau_{i,j,k}$) *intersects* with a range $[a,b]$ if the intersection of $[i,j]$ and $[a,b]$ is non-empty. The number of mutation operations in Θ that intersects with the range $[1,m]$, where $m \le n$, is denoted as $\rho(\Theta, m)$. For example, if $\Theta = \{\theta_{1,2}, \tau_{4,6,5}, \theta_{7,9}\}$, then $\rho(\Theta, 4) = 2$. Suppose that a mutation fragment $F = (i, j, \sigma)$ is in $\mathcal{F}(\Theta, x)$ for some set Θ of non-overlapping mutation operations. Then F is said to be *covered* by a mutation operation $\theta_{l,r}$ or $\tau_{l,r,k}$ in Θ if $F = \mathcal{F}(\theta_{l,r}, x, i)$ or $F = \mathcal{F}(\tau_{l,r,k}, x, i)$. If F is not covered by any mutation operation in Θ, that is $F = (i, i, x_i)$, then we can consider that F is still covered by a *virtual mutation operation*. We use $\mathcal{L}(\Theta, x, F)$ and $\mathcal{R}(\Theta, x, F)$ to denote the left and right end points of a mutation operation in Θ respectively that covers F. Formally,

- $$\mathcal{L}(\Theta, x, F) = \begin{cases} l & \text{if } \exists \theta_{l,r} \text{ or } \tau_{l,r,k} \in \Theta \text{ such that} \\ & \quad F = \mathcal{F}(\theta_{l,r}, x, i) \\ & \quad \text{or } F = \mathcal{F}(\tau_{l,r,k}, x, i) \\ i & \text{otherwise} \end{cases}$$
- $$\mathcal{R}(\Theta, x, F) = \begin{cases} r & \text{if } \exists \theta_{l,r} \text{ or } \tau_{l,r,k} \in \Theta \text{ such that} \\ & \quad F = \mathcal{F}(\theta_{l,r}, x, i) \\ & \quad \text{or } F = \mathcal{F}(\tau_{l,r,k}, x, i) \\ i & \text{otherwise} \end{cases}$$

For example, suppose that $\Theta = \{\tau_{1,3,2}, \theta_{4,5}\}$, $x = TAGAC$ and $\mathcal{F}(\Theta, x) = \langle(1,2,\mathrm{A}), (2,3,\mathrm{G}), (3,1,\mathrm{T}), (4,5,\mathrm{G}), (5,4,\mathrm{T})\rangle$. For $F = (2,3,G)$, we have $\mathcal{L}(\Theta, x, F) = 1$ and $\mathcal{R}(\Theta, x, F) = 3$.

For an integer i, let $A_i = \{(i+1, t, \theta(x_t)) : i+1 < t \le n\}$ and $B_i = \{(i+1, t, x_t) : i+1 < t \le n\}$, which are sets of mutation fragments covered by inversion and transposition operations respectively acting on x whose left end points are all $i+1$, where $0 \le i < n$.

Definition 3. Let $S = \langle F_1, F_2, \ldots, F_m\rangle$ be an agreed sequence of m mutation fragments of x, where $m \le n$, and Θ be a set of non-overlapping mutation operations such that $S \sqsubset \mathcal{F}(\Theta, x)$. Sequence S is called a *complete sequence* if $\mathcal{R}(\Theta, x, F_m) = m$.

Observation 2. Suppose that $S = \langle F_1, F_2, \ldots, F_m\rangle$ is a complete sequence, where $m < n$. Then $S' = \langle F_1, F_2, \ldots, F_m, F_{m+1}\rangle$ is an agreed sequence if $F_{m+1} \in A_m \cup M_1[m+1, m+1] \cup B_m \cup M_2[m+1, m+1]$.

Definition 4 *(Legal sequence).* An agreed sequence $S = \langle F_1, F_2, \ldots, F_m\rangle$ of m mutation fragments of x is *legal* if $\ell(S) = y_{1,m}$, where $m \le n$.

The main idea of our algorithm is as follows. First, the algorithm constructs all legal sequences of length 1 by examining all mutation fragments in $A_0 \cup M_1[1,1] \cup B_0 \cup M_2[1,1]$. A legal sequence is then created if the mutation fragment yields y_1. Next, for each $1 \le i < n$, the algorithm produces all possible legal sequences of length $i+1$, which can be obtained from all previously created legal sequences of length i. At the end, if there exists a legal sequence of length n, then the algorithm returns the minimum number of the used mutation operations among all legal sequences of length n. Otherwise, the algorithm returns infinity.

To distinguish all the legal sequences of length i, where $1 \le i \le n$, we first define four sets of *extended* mutation fragments I_i, H_i, T_i and V_i of x as follows:

- $I_i = \{(g, d, (i, j, \theta(x_j)))\}$ for $0 \le g \le n-1$, $0 \le d \le n$ and $1 \le j \le n$.

- $H_i = \{(s, d, (i, j, x_j))\}$ for $1 \le s \le n-1$, $0 \le d \le n$ and $i < j \le n$.
- $T_i = \{(g, d, (i, j, x_j))\}$ for $0 \le g \le n-1$, $0 \le d \le n$ and $1 \le j < i$.
- $V_i = \{(0, d, (i, i, x_i))\}$ for $0 \le d \le n$.

Actually, each extended mutation fragment is used to indicate a "status" of a legal sequence $S = \langle F_1, F_2, \ldots, F_i \rangle$ of length i with some extra guiding information g, d and s. The value of g is the number of steps that are required to move towards the anti-diagonal (respectively, diagonal) direction on the mutation table M_1 (respectively, M_2) to complete an inversion (respectively, transposition) operation. That is, if the last mutation fragment F_i of S is covered by a mutation $\theta_{l,r}$ or $\tau_{l,r,k}$, then $g = r - i$. The value of d denotes the number of the currently used mutation operations in S. If F_i is covered by a transposition operation, say $\tau_{l,r,k}$, and $F_i \in \tau_{l,r,k}(x, 1)$, then s equals to the left end point of $\tau_{l,r,k}$ (i.e., $s = l$). An extended mutation fragment is further called an *ending fragment* (respectively, *continuing fragment*) if the value of its first component is zero (respectively, non-zero).

Next, we represent each legal sequence by an extended mutation fragment as follows. Given a legal sequence $S = \langle F_1, F_2, \ldots, F_i \rangle$ that consists of i mutation fragments of x, let Θ be any set of non-overlapping mutation operations such that $S \sqsubset \mathcal{F}(\Theta, x)$. We further let ω be the mutation operation in Θ that covers F_i. Note that ω may be a virtual mutation operation. We create an extended mutation fragment e by the following cases (each case is also called as a *type* of extended mutation fragment):

Case 1 (type I): ω is an inversion operation. Then $e = (\mathcal{R}(\Theta, x, F_i) - i, \rho(\Theta, i), F_i)$.

Case 2 (type H): ω is a transposition operation and $F_i \in \omega(x, 1)$. Then $e = (\mathcal{L}(\Theta, x, F_i), \rho(\Theta, i), F_i)$.

Case 3 (type T): is a transposition operation and $F_i \in \omega(x, 2)$. Then $e = (\mathcal{R}(\Theta, x, F_i) - i, \rho(\Theta, i), F_i)$.

Case 4 (type V): ω is a virtual mutation operation. Then $e = (0, \rho(\Theta, i), F_i)$.

Now, we briefly describe how to produce all possible legal sequences of length $i+1$ from all previously created legal sequences of length i. Let L_i be the set of all extended mutation fragments of x with each corresponding to a legal sequence of length i. Suppose that L_i is already known. Below, we construct L_{i+1} by examining all extended mutation fragments in L_i. For each extended mutation fragment $e \in L_i$, we first identify all candidate mutation fragments at column $i+1$ of the mutation tables M_1 and M_2 based on the guiding information contained in e. Let $N_i(e)$ be set of the candidate mutation fragments of x at column $i+1$ of M_1 and M_2. Then we compute $N_i(e)$ as follows:

Case 1. e is a continuing fragment. Let $F_i = (i, j, \sigma)$ be the mutation fragment contained in e. Then we consider three sub-cases to compute $N_i(e)$: (1) Suppose that e is of type I. Then $N_i(e) = \{(i+1, j-1, \theta(x_{j-1}))\}$. (2) Suppose that e is of type H, i.e., $e = (s, d, F_i)$. If $j < n$, then $N_i(e) = \{(i+1, j+1, x_{j+1}), (i+1, s, x_s)\}$; otherwise (i.e., $j = n$), $N_i(e) = \{(i+1, s, x_s)\}$. (3) Suppose that e is of type T, i.e., $e = (g, d, F_i)$. Then $N_i(e) = \{(i+1, j+1, x_{j+1})\}$.

Case 2. e is an ending fragment. We set $N_i(e) = A_i \cup M_1[i+1, i+1] \cup B_i \cup M_2[i+1, i+1]$.

Next, for any $F_{i+1} \in N_i(e)$ with $\ell(F_{i+1}) = y_{i+1}$, an extended mutation fragment e' is created to contain F_{i+1}. As to the values of the guiding information in e', they can easily be obtained from those in e based on Observations 1 and 2. Finally, we add e' into L_{i+1}.

Lemma 1. *Let P_1 and P_2 be two complete sequences of length m of x with $\ell(P_1) = \ell(P_2)$. Moreover, let S_1 be an agreed sequence of x with $P_1 \sqsubset S_1$ and S_2 be the resulting sequence obtained by replacing P_1 with P_2 in S_1. Then, S_2 is also an agreed sequence of x and $\ell(S_1) = \ell(S_2)$.*

Proof. Since S_1 and P_2 are agreed sequences of x, there exist sets of non-overlapping mutation operations Θ_1 and Θ_2 such that $S_1 \sqsubset \mathcal{F}(\Theta_1, x)$ and $P_2 \sqsubset \mathcal{F}(\Theta_2, x)$. Moreover, $P_1 \sqsubset \mathcal{F}(\Theta_1, x)$ since $P_1 \sqsubset S_1$. We construct a set Θ_3 of non-overlapping mutation operations from Θ_1 as follows. First, let Θ_3 be the resulting set obtained by removing all mutation operations in Θ_1 that intersect with the range $[1, m]$. Note that the ranges of these removed mutation operations are completely contained in the range $[1, m]$ because P_1 is a complete sequence of length m. Next, we insert those mutation operations in Θ_2 whose ranges entirely lie in the range $[1, m]$ into Θ_3. It can be verified that Θ_3 is a set of non-overlapping mutation operations and moreover $S_2 \sqsubset \mathcal{F}(\Theta_3, x)$ and $\ell(S_1) = \ell(S_2)$. □

Given an extended mutation fragment e, we use $d(e)$ to indicate the value of its second component (i.e., the number of the used mutation operations) for convenience. Based on Lemma 1, we have the following corollary.

Corollary 1. *Let e_1 and e_2 be two ending fragments in L_i. If $d(e_1) < d(e_2)$, then we can safely discard e_2 from L_i when computing the mutation distance between x and y.*

Lemma 2. *Let e_1 and e_2 be two continuing fragments of the same type T in L_i with $e_1 = (g, d_1, (i, j, x_j))$ and $e_2 = (g, d_2, (i, j, x_j))$. If $d(e_1) < d(e_2)$ (i.e., $d_1 < d_2$), then we can safely remove e_2 from L_i without affecting the computation of the mutation distance between x and y.*

Proof. Since two continuing fragments e_1 and e_2 of type T contain the same mutation fragment (i, j, x_j) and the same guiding information g, the right end point of the mutation operation covering (i, j, x_j) in e_1 is equal to that of the mutation operation covering (i, j, x_j) in e_2. Denote by t the above right end point. Clearly, for e_1 and e_2, their candidate mutation fragments at the columns from i to t on the mutation tables M_1 and M_2 are exactly equivalent. Hence, we can safely discard e_2 from L_i if $d_1 < d_2$ according to Corollary 1. □

Similar to the discussion in the proof of Lemma 2, it can also be verified that there do not exist two continuing fragments of same type I or H in L_i such that the values of all their components, excluding the second components

(i.e., the number of used mutation operations), are equal. Now, the details of our algorithm for computing the non-overlapping inversion and transposition distance between two equal-length strings x and y are described in Algorithm 1.

Algorithm 1. Computing the mutation distance between two strings.
Input: Two strings x and y of the same length n.
Output: The mutation distance $md(x, y)$.
1. Let $i = 0$, $d = 0$ and $L_1 = \varnothing$;
2. Construct L_1 by calling EMF_Creation(i, d);
3. **if** $L_1 = \varnothing$ **then** return ∞;
4. **for** $i = 1$ **to** $n - 1$ **do**
5. $\quad L_{i+1} =$ EMF_Extension(L_i);
6. $\quad$ **if** $L_{i+1} = \varnothing$ **then** return ∞;
7. Return $\min\{d(e) : e \in L_n\}$;

Procedure 1. EMF_Creation(i, d).
Input: i and d;
Output: Add extended mutation fragments to L_{i+1} when $L_i = \emptyset$ or there is an ending fragment in L_i.
1. **for** each $(i + 1, t, \theta(x_t)) \in A_i$ **do**
2. $\quad$ **if** $\theta(x_t) = y_{i+1}$ **then** add $(t - i - 1, d + 1, (i + 1, t, \theta(x_t)))$ of type I to L_{i+1};
3. **if** $\theta(x_{i+1}) = y_{i+1}$ **then** add $(0, d + 1, (i + 1, i + 1, \theta(x_{i+1})))$ of type I to L_{i+1};
4. **for each** $(i + 1, t, x_t) \in B_i$ **do**
5. $\quad$ **if** $x_t = y_{i+1}$ **then** add $(i + 1, d + 1, (i + 1, t, x_t))$ of type H to L_{i+1};
6. **if** $x_{i+1} = y_{i+1}$ **then** add $(0, d, (i + 1, i + 1, x_{i+1}))$ of type V to L_{i+1};

Procedure 2. EMF_Extension(L_i).
Input: L_i.
Output: L_{i+1}.
1. $L_{i+1} \leftarrow \varnothing$ and $E_i \leftarrow \varnothing$; /* E_i is used to keep all ending fragments in L_i */
2. **for** each extended mutation fragment e in L_i **do**
3. $\quad$ **case 1:** $e = (s, d, (i, j, x_j))$ is of type H.
4. $\quad\quad$ **if** $j < n$ and $x_{j+1} = y_{i+1}$ **then** add $(s, d, (i + 1, j + 1, x_{j+1}))$ of type H to L_{i+1};
5. $\quad\quad$ **if** $x_s = y_{i+1}$ **then** add $(j - i - 1, d, (i + 1, s, x_s))$ of type T to L_{i+1};
6. $\quad$ **case 2:** $e = (g, d, (i, j, x_j))$ is of type T.
7. $\quad\quad$ **if** $g > 0$ **then**
8. $\quad\quad\quad$ **if** $x_{j+1} = y_{i+1}$ **then** add $(g - 1, d, (i + 1, j + 1, x_{j+1}))$ of type T to L_{i+1};
9. $\quad\quad$ **else** /* that is, $g = 0$ */
10. $\quad\quad\quad$ Add e to E_i;
11. $\quad$ **case 3:** $e = (g, d, (i, j, \theta(x_j)))$ is of type I.
12. $\quad\quad$ **if** $g > 0$ **then**
13. $\quad\quad\quad$ **if** $\theta(x_{j-1}) = y_{i+1}$ **then** add $(g - 1, d, (i + 1, j - 1, \theta(x_{j-1})))$ of type I to L_{i+1};
14. $\quad\quad$ **else** /* that is, $g = 0$ */
15. $\quad\quad\quad$ Add e to E_i;
16. $\quad$ **case 4:** e is of type V.
17. $\quad\quad$ Add e to E_i;
18. **if** $E_i \neq \varnothing$ **then** $d = \min\{d(e) : e \in E_i\}$ and EMF_Creation(i, d);
19. Return L_{i+1};

For example, consider $x = TAGAC$, whose mutation tables are shown in Fig. 1, and $y = TAACG$. Then for simplicity, we just list the result of L_i for each $i = 1, 2, \ldots, 5$ as follows, where the superscript I, H, T or V indicates the type of the corresponding extended mutation fragment. According to the result of L_5, we finally can conclude that $md(x, y) = 1$.

- $L_1 = \{(1, 1, (1, 2, T))^I, (3, 1, (1, 4, T))^I, (0, 0, (1, 1, T))^V\}$.
- $L_2 = \{(0, 1, (2, 1, A))^I, (0, 0, (2, 2, A))^V, (2, 1, (2, 4, A))^H\}$.
- $L_3 = \{(3, 1, (3, 4, A))^H, (1, 1, (3, 2, A))^T\}$.
- $L_4 = \{(3, 1, (4, 5, C))^H\}$.
- $L_5 = \{(0, 1, (5, 3, G))^T\}$.

Theorem 1. *Algorithm 1 computes the mutation distance of two equal-length strings in $O(n^3)$ time and $O(n^2)$ space.*

Proof. The correctness of Algorithm 1 can be verified according to the discussion we mentioned before. Below, we analyze the complexities of Algorithm 1. For convenience, we use $|K|$ to denote the number of extended mutation fragments of type K in L_i. By Corollary 1 and Lemma 2, we have $O(|I|) = O(|H|) = O(|T|) = O(n^2)$ and $O(|V|) = O(1)$. Therefore, Procedure 2 can be done in $O(|I| + |H| + |T|) = O(n^2)$ time. It is not hard to see that Procedure 1 costs $O(n)$ time in the worst case. Moreover, Procedure 2 is called at most $O(n)$ times in Algorithm 1. As a result, the total time complexity of Algorithm 1 is $O(n^3)$. In addition, the space complexity of Algorithm 1 is $O(|I| + |H| + |T|) = O(n^2)$. □

4. Conclusions

In this work, we studied the computation of the non-overlapping inversion and transposition distance (also called mutation distance) between two equal-length strings, which can be useful applications especially in computational biology. As a result, we presented an $O(n^3)$ time and $O(n^2)$ space algorithm to compute this mutation distance. Note that the mutation operations we considered allow both inversion and transposition and the lengths of two substrings exchanged by a transposition are not necessarily identical. In fact, the algorithm we proposed in this study can be utilized to solve the approximate string matching problem under non-overlapping inversion and/or transposition distance by the sliding window technique. In future, it will be interesting to explore whether a space-time tradeoff is possible for the technique we used to compute the mutation distance between two equal-length strings.

References

[1] D. Cantone, S. Cristofaro, S. Faro, Efficient string-matching allowing for non-overlapping inversions, Theor. Comput. Sci. 483 (2013) 85–95.

[2] D. Cantone, S. Faro, E. Giaquinta, Approximate string matching allowing for inversions and translocations, in: Proceedings of the Prague Stringology Conference, 2010, pp. 37–51.
[3] D.-J. Cho, Y.-S. Han, H. Kim, Alignment with non-overlapping inversions and translocations on two strings, Theor. Comput. Sci. 575 (2015) 90–101.
[4] S. Grabowski, S. Faro, E. Giaquinta, String matching with inversions and translocations in linear average time (most of the time), Inf. Process. Lett. 111 (2011) 516–520.
[5] S.-Y. Huang, C.-H. Yang, T.T. Ta, C.L. Lu, Approximate string matching problem under non-overlapping inversion distance, in: Proceedings of the 2014 International Computer Symposium, ICS 2014, IOS Press, 2014, pp. 38–46.
[6] Y.-L. Huang, C.L. Lu, Sorting by reversals, generalized transpositions, and translocations using permutation groups, J. Comput. Biol. 17 (2010) 685–705.