Theoretical Computer Science 575 (2015) 90–101

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Alignment with non-overlapping inversions and translocations on two strings [☆]

Da-Jung Cho, Yo-Sub Han [*], Hwee Kim

Department of Computer Science, Yonsei University, Seoul 120-749, Republic of Korea

A R T I C L E   I N F O

Article history:
Received 16 April 2014
Received in revised form 22 September 2014
Accepted 21 October 2014
Available online 27 October 2014

Keywords:
Sequence alignment
Non-overlapping inversion
Translocation

A B S T R A C T

An inversion and a translocation are important in bio sequence analysis and motivate researchers to consider the sequence alignment problem using these operations. Based on inversion and translocation, we introduce a new alignment problem with non-overlapping inversions and translocations—given two strings x and y, find an alignment with non-overlapping inversions and translocations for x and y. This problem has interesting application for finding a common sequence from two mutated sequences. We, in particular, consider the alignment problem when non-overlapping inversions and translocations are allowed for both x and y. We design an efficient algorithm that determines the existence of such an alignment and retrieves an alignment, if exists.

1. Introduction

In modern biology, it is important to determine exact orders of DNA sequences, retrieve relevant information of DNA sequences and align these sequences [2,9,15,16]. A *chromosomal inversion* occurs when a single chromosome undergoes breakage and rearrangement within itself [14]. A *chromosomal translocation* is to relocate a piece of the DNA sequence from one place to another and, thus, rearrange the sequence [12]. The chromosomal inversion and translocation are crucial in DNAs since these operations alter a DNA sequence and often cause genetic diseases [11,13]. Fig. 1 shows an example of chromosomal inversion and translocation. String matching with inversion and translocation is defined as follows: given a text T and a pattern P, the string matching problem is to find all matching of a given pattern P in a text T allowing inversion and translocation. We can also consider an alignment problem that transforms given string x to another string y allowing inversion and translocation edit operations. Many researchers investigated efficient algorithms for the pattern matching and alignment problem with inversion and translocation [1–6,8,10,16,17]. See Table 1 for summary of related research.

Schöniger and Waterman [16] introduced the alignment problem with non-overlapping inversions under the assumption that all regions are not allowed to overlap. They presented an $O(n^6)$ algorithm that computes local alignments with inversions between two strings of length n and m based on the dynamic programming, where $n \geq m$. Gao et al. [5] designed a space-efficient dynamic programming algorithm for computing optimal alignment with inversions between two DNA sequences. Their algorithm maintains all possible ending positions of inversions and finds all alignments in $O(n^2m^2)$ time using $O(nm)$ space. Chen et al. [4] designed an $O(n^2m^2)$ algorithm that computes an optimal alignment between a pair of

[☆] A preliminary version of this paper appeared in *Proceedings of the 8th International Workshop on Algorithms and Computation*, WALCOM 2014, LNCS, vol. 8344, 261–272, 2014.
[*] Corresponding author.
E-mail addresses: dajung@cs.yonsei.ac.kr (D.-J. Cho), emmous@cs.yonsei.ac.kr (Y.-S. Han), kimhwee@cs.yonsei.ac.kr (H. Kim).

http://dx.doi.org/10.1016/j.tcs.2014.10.036
0304-3975/© 2014 Elsevier B.V. All rights reserved.

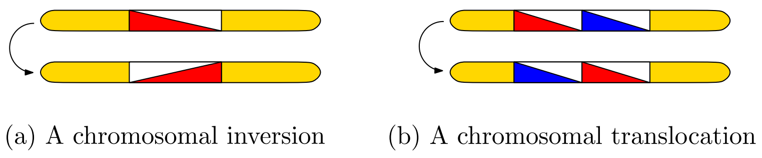


Fig. 1. An example of chromosomal inversion and translocation: the left image describes the chromosome inversion and the right image describes the chromosome translocation.

Table 1
Related research.

Problem	Authors	Time	Space
Alignment with inversions	Schöniger and Waterman (1992) [16]	$O(n^6)$	$O(n^2)$
	Gao et al. (2003) [5]	$O(n^2m^2)$	$O(nm)$
	Chen et al. (2004) [4]	$O(n^2m^2)$	$O(nm)$
	Alves et al. (2005) [1]	$O(n^3 \log n)$	$O(n^2)$
	Vellozo et al. (2006) [17]	$O(n^2m)$	$O(n^2)$
Pattern matching with inversions	Cantone et al. (2013) [2]	$O(nm)$	$O(m^2)$
Pattern matching with inversions and translocations	Cantone et al. (2010) [3]	$O(nm^2)$	$O(m^2)$
	Grabowski et al. (2011) [6]	$O(nm^2)$	$O(m)$

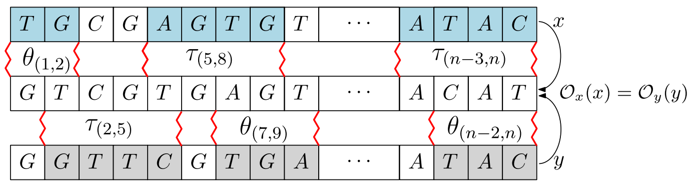

T G C G A G T G T ⋯ A T A C
x
$\theta_{(1,2)}$
$\tau_{(5,8)}$
$\tau_{(n-3,n)}$
G T C G T G A G T ⋯ A C A T
$\mathcal{O}_x(x) = \mathcal{O}_y(y)$
$\tau_{(2,5)}$
$\theta_{(7,9)}$
$\theta_{(n-2,n)}$
G G T T C G T G A ⋯ A T A C
y

Fig. 2. An example of non-overlapping inversions and translocations on both strings x and y, where $\mathcal{O}_x = \{\theta_{(1,2)}, \tau_{(5,8)}, \tau_{(n-3,n)}\}$ and $\mathcal{O}_y = \{\tau_{(2,5)}, \theta_{(7,9)}, \theta_{(n-2,n)}\}$. Note that $\mathcal{O}_x$ and $\mathcal{O}_y$ are sets of non-overlapping inversions and translocations, $\theta_{(i,j)}$ denotes an inversion from position i to j, and $\tau_{(i,j)}$ denotes a translocation that relocates the subsequence $(i, \frac{j-i+1}{2})$ to $(\frac{j-i+1}{2}+1, j)$.

DNA sequences with inversion operations. Vellozo et al. [17] presented an $O(n^2m)$ algorithm and improved the previous algorithm by Schöniger and Waterman [16]. Recently, Cantone et al. [2] introduced an $O(nm)$ algorithm using $O(m^2)$ space for the string matching problem, which is to find all locations of a pattern of length m with respect to a text of length n based on non-overlapping inversions. Furthermore, for the pattern matching problem allowing inversions and translocations, Cantone et al. [3] designed the first algorithm with $O(nm^2)$ time and $O(m^2)$ space. Grabowski et at. [6] investigated the previous problem and obtained an $O(n)$ average runtime algorithm.

Many diseases are often caused by genetic mutations, which can be inherited through generations and produce new sequences from a normal gene [7]. In other words, we may have two different sequences from a normal gene by different mutations. This motivates us to examine the problem of deciding whether or not two gene sequences are mutated from the same gene sequence. In particular, we consider an inversion and translocation mutation. See Fig. 2 for an example.

Our problem is different from the previous problem [17], where a non-overlapping inversion occurs only in one string and transforms the string to the other string. Our problem considers two non-overlapping operations (inversions and translocations) allowed on both x and y simultaneously. We first determine all existences of inversions and translocations on both strings. Once we know the existences, we retrieve the corresponding alignment efficiently. We rely on two set (a formal definition is provided in Section 2) $\mathcal{O}_1$ and $\mathcal{O}_2$ of non-overlapping inversions and translocations, and a string $\mathcal{O}_1(x)$ obtained by applying all inversions and translocations in $\mathcal{O}_1$ to a string x. Note that our problem is equivalent to searching two sets $\mathcal{O}_1$ and $\mathcal{O}_2$ of non-overlapping inversions and translocations such that $y = \mathcal{O}_2(\mathcal{O}_1(x))$.

2. Preliminaries

Let $A[a_1][a_2]\cdots[a_n]$ be an n-dimensional array, where the size of each dimension is a_i for $1 \le i \le n$. Let $A[i_1][i_2]\cdots[i_n]$ be the element of A with indices $(i_1, i_2, \ldots, i_n)$. Given a finite set Σ of characters and a string s over Σ, we use $|s|$ to denote the length of s and $s[i]$ to denote the symbol of s at position i. We use $s[i:j]$ to denote a substring $s[i]s[i+1]\cdots s[j]$, where $1 \le i \le j \le |s|$. We consider an *inversion* θ and denote by $\theta(s)$ the reversal[1] of a string s; namely, $\theta(s) = s[n]s[n-1]\cdots s[2]s[1]$, where $n = |s|$.

[1] In biology, inversion is a composition of a reversal operation and a complement operation. However, inversion is often regarded as reversal in the string matching and alignment literature for the simplicity of analysis. We also follow this convention.

We define an inversion operation $\theta_{(i,j)}$ for a given range i, j as follows:

$$\theta_{(i,j)}(s) = \theta\big(s[i:j]\big).$$

We also consider a *translocation* τ and define $\tau(s) = s_2 s_1$, where $s = s_1 s_2$ and $|s_1| = |s_2|$. We define a translocation operation $\tau_{(i,j)}$ for a given range i, j as follows:

$$\tau_{(i,j)}(s) = \tau\big(s[i:j]\big).$$

We say that $\frac{i+j}{2}$ is the *center of the operation* for $\theta_{(i,j)}$ or $\tau_{(i,j)}$. Given a positive constant n, we define a *set $\mathcal{O}$ of non-overlapping inversions and translocations* to be

$$\mathcal{O} = \big\{\theta_{(i_k, i_{k+1}-1)} \text{ or } \tau_{(i_k, i_{k+1}-1)} \;\big|\; i_k \in \mathbb{R},\ 1 \le k \le |\mathbb{R}| - 1\big\},$$

where $\mathbb{R} = \{i_k \mid i_k < i_{k+1} \text{ for } 1 \le k \le |\mathbb{R}| - 1,\ i_1 = 1,\ i_{|\mathbb{R}|} = n\}$. Then, for a string s of size n, we have $\mathcal{O}(s) = s'$, where

$$s'[i:j] = \begin{cases} \theta(s[i:j]) & \text{if } \theta_{(i,j)} \in \mathcal{O}, \\ \tau(s[i:j]) & \text{if } \tau_{(i,j)} \in \mathcal{O}. \end{cases}$$

From now on, we use a set of operations instead of a set of non-overlapping inversions and translocations since we only consider sets of non-overlapping inversions and translocations.

Definition 2.1. We define a new alignment problem with non-overlapping inversions and translocations on two strings as follows: given two strings x and y of the same length, find two sets $\mathcal{O}_x$ and $\mathcal{O}_y$ of operations such that $\mathcal{O}_x(x) = \mathcal{O}_y(y)$, if such two sets exist.

3. The algorithm

Before we present an algorithm, we first introduce some definitions, which lead to the definition of a *legal sequence*, and prove that finding an alignment on two strings is equivalent to finding legal sequences on two strings. Then, we design an algorithm that finds legal sequences on two strings (and their alignments).

3.1. Reformulating the problem using legal sequences

We use $x = AGCTCA$ and $y = CAGATC$ as our example strings for explaining the algorithm. Remark that $\theta(AG)\tau(CTCA) = GACACT = \tau(CAGA)\theta(TC)$ and, thus, we have two sets $\mathcal{O}_x = \{\theta_{(1,2)}, \tau_{(3,6)}\}$ and $\mathcal{O}_y = \{\tau_{(1,4)}, \theta_{(5,6)}\}$.

We start from building two tables in which each cell contains a pair of a range and a character. First, we define an *inversion fragment table* $T_x[n][n]$ (IFT for short) for x as follows:

$$T_x[i][j] = \begin{cases} ((j,i), x[j]) & \text{if } j \le i, \\ ((i,j), x[j]) & \text{if } j > i. \end{cases}$$

Next, we define a *translocation fragment table* $U_x[n][\lfloor \frac{n}{2} \rfloor][2]$ (TFT for short) for x as follows:

$$U_x[i][j][k] = \begin{cases} ((i-j-1, i+j), x[i+j]) & \text{if } k = 1 \text{ and } i+j \le n, \\ ((i-2j+1, i), x[i-j]) & \text{if } k = 2 \text{ and } 0 \le i-j. \end{cases}$$

We call all elements in T_x *inversion fragments* (IFs for short) of x and all elements in U_x *translocation fragments* (TFs for short) of x. We use *fragments* to mean both IFs and TFs. For a fragment $\mathbb{F} = ((p,q), \sigma)$, we say that $\mathbb{F}$ *yields* the character σ, $\frac{p+q}{2}$ is the *center* of the fragment and $q - p + 1$ is the *length* of the fragment. For a sequence of fragments $\mathbb{F}_1, \ldots, \mathbb{F}_n$, where $\mathbb{F}_i$ yields σ_i, we say that the sequence yields a string $\sigma_1 \cdots \sigma_n$. Figs. 3 and 4 show examples of IFT and TFT.

IFs become useful for computing a substring created by an inversion because of the following property of the inversion operation:

Observation 3.1. For a string x and its T_x,

(1) $\theta_{(i,j)}(x) = \theta(x[j])\theta(x[j-1]) \cdots \theta(x[i+1])\theta(x[i])$,
(2) A sequence $T_x[i][j], T_x[i+1][j-1], \ldots, T_x[j-1][i+1], T_x[j][i]$ of IFs yields $\theta_{(i,j)}(x)$. IFs in the sequence have the same center $\frac{i+j}{2}$.

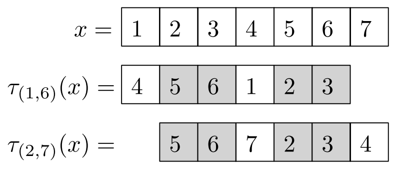
Fig. 3 shows an example of Observation 3.1(2). From Observation 3.1, we know that if we apply $\theta_{(\min(i,j),\max(i,j))}$ to x, then σ yielded by $T_x[i][j]$ becomes the ith character of the result string. Similarly, TFs are useful for computing a substring created by a translocation because of the following property of the translocation operation:

T_x	$i=1$	2	3	4	5	6
$j=1$	$((1,1),A)$	$((1,2),A)$	$((1,3),A)$	$((1,4),A)$	$((1,5),A)$	$((1,6),A)$
2	$((1,2),C)$	$((2,2),C)$	$((2,3),C)$	$((2,4),C)$	$((2,5),C)$	$((2,6),C)$
3	$((1,3),T)$	$((2,3),T)$	$((3,3),T)$	$((3,4),T)$	$((3,5),T)$	$((3,6),T)$
4	$((1,4),C)$	$((2,4),C)$	$((3,4),C)$	$((4,4),C)$	$((4,5),C)$	$((4,6),C)$
5	$((1,5),G)$	$((2,5),G)$	$((3,5),G)$	$((4,5),G)$	$((5,5),G)$	$((5,6),G)$
6	$((1,6),A)$	$((2,6),A)$	$((3,6),A)$	$((4,6),A)$	$((5,6),A)$	$((6,6),A)$

Fig. 3. IFT T_x for $x = AGCTCA$. Shaded cells denote IFs for the inversion $\theta_{(2,5)}$.

U_x		$i=1$	2	3	4	5	6
$k=1$	$j=2$	$((0,3),C)$	$((1,4),T)$	$((2,5),C)$	$((3,6),A)$		
	3	$((-1,4),T)$	$((0,5),C)$	$((1,6),A)$			
2	$j=2$			$((0,3),A)$	$((1,4),G)$	$((2,5),C)$	$((3,6),T)$
	3				$((-1,4),A)$	$((0,5),G)$	$((1,6),C)$

Fig. 4. TFT U_x for $x = AGCTCA$. Shaded cells denote TFs for the translocation $\tau_{(2,5)}$.



$x =$ 1 2 3 4 5 6 7
$\tau_{(1,6)}(x) =$ 4 5 6 1 2 3
$\tau_{(2,7)}(x) =$ 5 6 7 2 3 4

Fig. 5. Graphical representation of Observation 3.2(1).

Observation 3.2. For a string x and its U_x,

(1) Let $w = \tau_{(i,j)}(x)$. Then $\tau_{(i+1,j+1)}(x) = w[2 : \frac{j-i+1}{2}]x[j+1]w[\frac{j-i+5}{2} : j-i+1]x[\frac{j-i+3}{2}]$.
(2) A sequence $U_x[i][\frac{l}{2}][1], U_x[i+1][\frac{l}{2}][1], \ldots, U_x[i+\frac{l}{2}][\frac{l}{2}][1], U_x[i+\frac{l}{2}+1][\frac{l}{2}][2], U_x[i+\frac{l}{2}+2][\frac{l}{2}][2], \ldots, U_x[j][\frac{l}{2}][2]$ of TFs yields $\tau_{(i,j)}(x)$, where $l = j-i-1$. TFs in the sequence have the same length l.

Fig. 5 shows an example of Observation 3.2(1). Fig. 4 shows an example of Observation 3.2(2). From Observation 3.2, we know that if we apply a translocation of length $2j$ to x, then σ yielded by $U_x[i][j][k]$ becomes the ith character of the result string if index i is in the kth half of the translocation.

It is easy to verify from the construction that we can construct T_x and U_x in $O(n^2)$ time and the size of T_x and U_x is $O(n^2)$, where $|x| = n$. We also construct T_y and U_y for y. Next, we define an *agreed sequence* and a *partially agreed sequence* of fragments.

Definition 3.3. Given a sequence $\mathbb{F}_1, \mathbb{F}_2, \ldots, \mathbb{F}_l$ of l fragments of a string x, we say that the sequence is an *agreed sequence* from index i_1 to $i_2 = i_1 + l - 1$ of x if one of the following conditions holds:

(1) $\mathbb{F}_h = T_x[i_1 + h - 1][i_1 + l - h]$ for all h or
(2) $\mathbb{F}_h = U_x[i_1 + h - 1][\frac{l}{2}][1]$ if $h \le i_1 + \frac{l}{2} - 1$ and $U_x[i_1 + h - 1][\frac{l}{2}][2]$ if $h \ge i_1 + \frac{l}{2}$.

We also say that for an agreed sequence $\mathbb{F}_1, \mathbb{F}_2, \ldots, \mathbb{F}_l$, a sequence $\mathbb{F}_1, \mathbb{F}_2, \ldots, \mathbb{F}_m$ is a *partially agreed sequence* if $m \le l$.

Shaded cells in Figs. 3 and 4 represent two agreed sequences. From Observations 3.1 and 3.2, we establish the following result:

Lemma 3.4. *If a sequence is an agreed sequence of IFs of x from index i_1 to i_2, then the sequence yields $\theta_{(i_1,i_2)}(x)$. If a sequence is an agreed sequence of TFs of x from index i_1 to i_2, then the sequence yields $\tau_{(i_1,i_2)}(x)$.*

Next, we define a *connected sequence* and a *partially connected sequence* of fragments.

Definition 3.5. A *connected sequence* is defined recursively as follows:

(1) If a sequence is an agreed sequence from i_1 to i_2, then the sequence is a connected sequence from i_1 to i_2.
(2) If a sequence $\mathbb{F}_1, \mathbb{F}_2, \ldots, \mathbb{F}_{l_1}$ is a connected sequence from i_1 to i_2 and another sequence $\mathbb{F}'_1, \mathbb{F}'_2, \ldots, \mathbb{F}'_{l_2}$ is a connected sequence from i_2+1 to i_3, then the sequence $\mathbb{F}_1, \ldots, \mathbb{F}_{l_1}, \mathbb{F}'_1, \ldots, \mathbb{F}'_{l_2}$ is a connected sequence from i_1 to i_3.

We also define a *partially connected sequence*: if a sequence $\mathbb{F}_1, \mathbb{F}_2, \ldots, \mathbb{F}_{l_1}$ is a connected sequence from i_1 to i_2 and another sequence $\mathbb{F}'_1, \mathbb{F}'_2, \ldots, \mathbb{F}'_{l_2}$ is a partially agreed sequence from i_2+1 to i_3, then the sequence $\mathbb{F}_1, \ldots, \mathbb{F}_{l_1}, \mathbb{F}'_1, \ldots, \mathbb{F}'_{l_2}$ is a partially connected sequence from i_1 to i_3.

Finally, we define a *legal sequence* of fragments.

Definition 3.6. A sequence is a *legal sequence* of fragments of x (or y, respectively) if the sequence is a connected sequence from 1 to $n = |x|$ and there exists a connected sequence of fragments of y (or x, respectively) from 1 to n that yields the same string as the sequence.

From Lemma 3.4, Definitions 3.5 and 3.6, we have:

Lemma 3.7. *Given two strings x and y of the same length, there exist two sets $\mathcal{O}_x$ and $\mathcal{O}_y$ of operations such that $s = \mathcal{O}_x(x) = \mathcal{O}_y(y)$ if and only if there exist two legal sequences $\mathbb{S}_x$ and $\mathbb{S}_y$ that yield s for x and y, respectively.*

It follows from Lemma 3.7 that we have an alignment for x and y if and only if we find two legal sequences $\mathbb{S}_x$ and $\mathbb{S}_y$ for x and y.

3.2. Searching for legal sequences

The main idea of our algorithm is to keep tracking of all possible agreed sequences, append them to connected sequences and ensure that connected sequences of fragments of x and y generate the same substring. Assume that we inspect the $(i+1)$th index and the agreed sequence that we track is $\mathbb{F}_{i_1}, \mathbb{F}_{i_1+1}, \cdots, \mathbb{F}_{i_2}$ from i_1 to i_2, where $i_1 \le i \le i_2$. Note that the agreed sequence is a part of a connected sequence from 1 to i_2. For index $i+1$, we need to check the following five cases to build a partially connected sequence from 1 to $i+1$ (see Fig. 6 for an illustration):

1. $i = i_2$: since the agreed sequence that we have tracked ends at i, we append the first fragment of a new agreed sequence to the agreed sequence that we have tracked to construct a partially connected sequence from 1 to $i+1$. Note that we can add an inversion or a translocation starting from $i+1$ to a set of operations equivalent to a connected sequence from 1 to i. For an inversion, we can choose IF $T_x[i+1][j] = ((i+1, j), x[j])$ for a new agreed sequence of IFs from $i+1$ to j where $j \ge i+1$. For a translocation, we can choose TF $U_x[i+1][j][1] = ((i-j+2, i+j+1), x[i+j+1])$ for a sequence of TFs from $i+1$ to $i+2j$ where $j \ge i+1$ and $i+2j \le n$.
2. $i < i_2$ and the agreed sequence is for an inversion: if $\mathbb{F}_i = T_x[i][j]$, then $\mathbb{F}_{i+1} = T_x[i+1][j-1]$ from Definition 3.3.
3. $i < \frac{i_1+i_2-1}{2}$ and the agreed sequence is for a translocation: the current index $i+1$ is in the first half of the translocation that the sequence represents. If $\mathbb{F}_i = U_x[i][j][1]$, then $\mathbb{F}_{i+1} = U_x[i+1][j][1]$ from Definition 3.3.
4. $i = \frac{i_1+i_2-1}{2}$ and the agreed sequence is for a translocation: the current index $i+1$ is the start index of the second half of the translocation that the sequence represents. If $\mathbb{F}_i = U_x[i][j][1]$, then $\mathbb{F}_{i+1} = U_x[i+1][j][2]$ from Definition 3.3.
5. $\frac{i_1+i_2+1}{2} \le i < i_2$ and the agreed sequence is for a translocation: the current index $i+1$ is in the second half of the translocation that the sequence represents. If $\mathbb{F}_i = U_x[i][j][2]$, then $\mathbb{F}_{i+1} = U_x[i+1][j][2]$ from Definition 3.3.

We use tables $S_x^i[2][n]$ and $R_x^i[2][\lfloor \frac{n}{2} \rfloor][2]$ to record all partial agreed sequences of IFs and TFs that we are tracking, respectively. Namely, if $(i_1, \sigma_1\sigma_2) \in S_x^i[2][j]$, then there is a partially agreed sequence of IFs from i_1 to i, where the last IF of the sequence is $T_x[i][j]$ and the last two IFs yield $\sigma_1\sigma_2$. Similarly, if $(i_1, \sigma_1\sigma_2) \in R_x^i[2][j][k]$, then there is a partially agreed sequence of TFs from i_1 to i, where the last TF of the sequence is $U_x[i][j][k]$ and the last two TFs yield $\sigma_1\sigma_2$. We ensure that these partially agreed sequences are appended to any connected sequence from 1 to $i_1 - 1$ and, thus, we now have partially connected sequences from 1 to i by checking the first case (the case when $i = i_2$; See Case 1 in Fig. 6). For an element $(i_1, \sigma_1\sigma_2) \in S_x^i[2][j]$ or $R_x^i[2][j][k]$, we say that the element is a *sequence indicator* (indicator for short) with a *start index* i_1 and the sequence indicator *yields* $\sigma_1\sigma_2$. For each index, we also need to check whether or not partially connected sequences of fragments of x and y generate the same set of substrings. We use a table $C_x^i[|\Sigma|][|\Sigma|]$ to record two characters yielded by the ith and $(i+1)$th fragments of partially connected sequences of x. Moreover, we use tables $I_x[n][n]$ and $H_x[n][n]$ to record inversions and translocations corresponding to agreed sequences in partially connected sequences. Namely, if $I_x[i][j]$ ($H_x[i][j]$) is *true*, then $\theta_{(i,j)}$ ($\tau_{(i,j)}$) corresponds to an agreed sequence in a partially connected sequence. We design an algorithm that computes S_x^i, R_x^i, S_y^i and R_y^i for each index and checks whether or not there exist legal sequences for x and y.

Before the algorithm starts, we set initial data for S_x^1 and R_x^1; we add $(1, A\sigma)$ to $S_x^1[1][j]$ if $T_x[1][j]$ yields σ and set $I_x[1][j]$ *true*. We also add $(1, A\sigma)$ to $R_x^1[1][j][1]$ if $U_x[1][j][1]$ yields σ and set $H_x[1][2j]$ *true*. We also set S_y^1, I_y, R_y^1 and

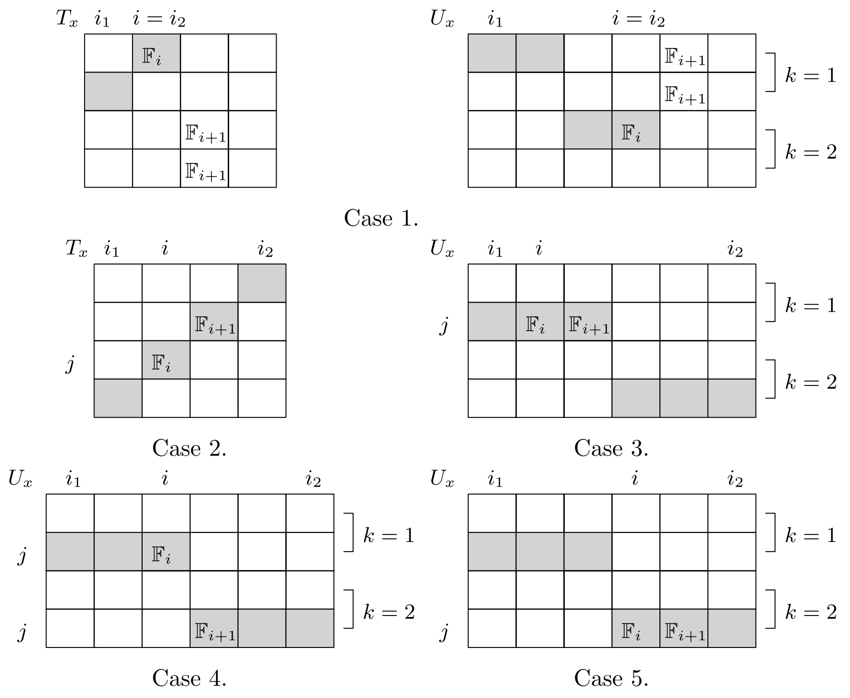


Fig. 6. The five cases to build a partially connected sequence from 1 to $i + 1$. Shaded cells in T_x denote IFs for the inversion $\theta_{(i_1,i_2)}$ and shaded cells in U_x denote TFs for the translocation $\tau_{(i_1,i_2)}$.

H_y similarly. Note that each cell of S^i_x and R^i_x is a sequence with $O(n)$ indicators. The algorithm sorts indicators in each cell of S^i_x and R^i_x in ascending order with respect to the start index, which is crucial in the runtime analysis. Now we execute the following steps from index 1 to $n - 1$ for x and y. We only illustrate the case for x. (The case for y is similar.)

The first two steps process agreed sequences of IFs.

STEP-1. We check all agreed sequences of IFs ending at i and record the first fragment of a new agreed sequence that can be appended to any connected sequence from 1 to i. When $(j, \sigma_0\sigma_1) \in S^i_x[1][j]$, we execute the following procedure (see Fig. 7 for an illustration):

1. Append $(i + 1, \sigma_1\sigma_2)$ to $S^i_x[2][j']$ and set $I[i + 1][j']$ *true* for $j' \geq i + 1$, where $T_x[i + 1][j']$ yields σ_2.
2. Append $(i + 1, \sigma_1\sigma_2)$ to $R^i_x[2][j'][1]$ and set $H[i + 1][i + 2j']$ *true* for $j' \geq i + 1$ and $i + 2j' \leq n$, where $U_x[i + 1][j'][1]$ yields σ_2.

STEP-2. We check all agreed sequences of IFs not ending at i (which are partially agreed sequences) and find IFs for all partially connected sequences from 1 to $i + 1$. When $(i_1, \sigma_0\sigma_1) \in S^i_x[1][j]$ and $i_1 \neq j$, we prepend $(i_1, \sigma_1\sigma_2)$ to $S^i_x[2][j - 1]$, where $T_x[i + 1][j - 1]$ yields σ_2 (see Fig. 8 for an illustration).

The next two steps process agreed sequences of TFs.

STEP-3. We check all agreed sequences of TFs ending at i and record the first fragment of a new agreed sequence that can be appended to any connected sequence from 1 to i. When $(i - 2j + 1, \sigma_0\sigma_1) \in R^i_x[1][j][2]$, we execute the following procedure (see Fig. 9 for an illustration):

1. Append $(i + 1, \sigma_1\sigma_2)$ to $S^i_x[2][j']$ and set $I[i + 1][j']$ *true* for $j' \geq i + 1$, where $T_x[i + 1][j']$ yields σ_2.
2. Append $(i + 1, \sigma_1\sigma_2)$ to $R^i_x[2][j'][1]$ and set $H[i + 1][i + 2j']$ *true* for $j' \geq i + 1$ and $i + 2j' \leq n$, where $U_x[i + 1][j'][1]$ yields σ_2.

STEP-4. We check all agreed sequences of TFs not ending at index i (which are partially agreed sequences) and find TFs for all partially connected sequences from 1 to $i + 1$. Then, we check the following three conditions (see Fig. 10 for an illustration):

1. If $(i_1, \sigma_0\sigma_1) \in R^i_x[1][j][1]$ and $i < i_1 + j - 1$: prepend $(i_1, \sigma_1\sigma_2)$ to $R^i_x[2][j][1]$ where $U_x[i + 1][j][1]$ yields σ_2.
2. If $(i_1, \sigma_0\sigma_1) \in R^i_x[1][j][1]$ and $i = i_1 + j - 1$: prepend $(i_1, \sigma_1\sigma_2)$ to $R^i_x[2][j][2]$ where $U_x[i + 1][j][2]$ yields σ_2.
3. If $(i_1, \sigma_0\sigma_1) \in R^i_x[1][j][1]$ and $i < i_1 + 2j - 1$: prepend $(i_1, \sigma_1\sigma_2)$ to $R^i_x[2][j][2]$ where $U_x[i + 1][j][2]$ yields σ_2.

T_x	1	2
1	$((1,1),A)$	$((1,2),A)$
2	$((1,2),C)$	$((2,2),C)$
3	$((1,3),T)$	$((2,3),T)$
4	$((1,4),C)$	$((2,4),C)$
5	$((1,5),G)$	$((2,5),G)$
6	$((1,6),A)$	$((2,6),A)$

S_x^1	1	2
1	$(1,AA)$	
2	$(1,AC)$	$(2,AC)$
3	$(1,AT)$	$(2,AT)$
4	$(1,AC)$	$(2,AC)$
5	$(1,AG)$	$(2,AG)$
6	$(1,AA)$	$(2,AA)$

I_x	1	2
1	$true$	$false$
2	$true$	$true$
3	$true$	$true$
4	$true$	$true$
5	$true$	$true$
6	$true$	$true$

U_x		1	2
1	2	$((0,3),C)$	$((1,4),T)$
	3	$((-1,4),T)$	$((0,5),C)$
2	2		
	3		

R_x^i		1	2
1	2	$(1,AC)$	$(2,AT)$
	3	$(1,AT)$	$(2,AC)$
2	2		
	3		

H_x		1	2
1	2	$true$	$true$
	3	$true$	$true$
2	2	$false$	$false$
	3	$false$	$false$

Fig. 7. An illustration of **STEP-1** for $x = ACTCGA$.

T_x	1	2
1	$((1,1),A)$	$((1,2),A)$
2	$((1,2),C)$	$((2,2),C)$
3	$((1,3),T)$	$((2,3),T)$
4	$((1,4),C)$	$((2,4),C)$
5	$((1,5),G)$	$((2,5),G)$
6	$((1,6),A)$	$((2,6),A)$

S_x^1	1	2
1	$(1,AA)$	$(1,CA)$
2	$(1,AC)$	$(1,TC),(2,AC)$
3	$(1,AT)$	$(1,CT),(2,AT)$
4	$(1,AC)$	$(1,GC),(2,AC)$
5	$(1,AG)$	$(1,AG),(2,AG)$
6	$(1,AA)$	$(2,AA)$

I_x	1	2
1	$true$	$false$
2	$true$	$true$
3	$true$	$true$
4	$true$	$true$
5	$true$	$true$
6	$true$	$true$

U_x		1	2
1	2	$((0,3),C)$	$((1,4),T)$
	3	$((-1,4),T)$	$((0,5),C)$
2	2		
	3		

R_x		1	2
1	2	$(1,AC)$	$(2,AT)$
	3	$(1,AT)$	$(2,AC)$
2	2		
	3		

H_x		1	2
1	2	$true$	$true$
	3	$true$	$true$
2	2	$false$	$false$
	3	$false$	$false$

Fig. 8. An illustration of **STEP-2** for $x = ACTCGA$.

From **STEPS-1** to **4**, we find all partially connected sequences from 1 to $i+1$. The next two steps ensure that these partially connected sequences of fragments of x and y yield the same set of substrings.

STEP-5. For all $(i_1, \sigma_1\sigma_2) \in S_x^i[2][j]$ or $R_x^i[2][j]$, where $1 \le j \le n$, we set $C_x^i[\sigma_1][\sigma_2]$ *true*.

We, then, run the same procedures from **STEP-1** to **STEP-5** for a string y before we start **STEP-6**.

STEP-6. We set $C_x'^i$ as follows:

$$C_x'^i[\sigma_1][\sigma_2] = \begin{cases} true & \text{if } C_x^i[\sigma_1][\sigma_2] = true \text{ and } C_y^i[\sigma_1][\sigma_2] = true, \\ false & \text{otherwise.} \end{cases}$$

Once we compute $C_x'^i$, for each $(i_1, \sigma_1\sigma_2) \in S_x^i[2][j]$, we remove $(i_1, \sigma_1\sigma_2)$ from $S_x^i[2][j]$ and set $I_x[i_1][i+j+1-i_1]$ *false* if $C_x'^i[\sigma_1][\sigma_2] = false$. Moreover, for each $(i_1, \sigma_1\sigma_2) \in R_x^i[2][j][k]$, we remove $(i_1, \sigma_1\sigma_2)$ from $R_x^i[2][j][k]$ and set $H_x[i_1][i_1+2j-1]$ *false* if $C_x'^i[\sigma_1][\sigma_2] = false$ (see Fig. 11 for an illustration).

After we finish calculating S_x^i, R_x^i, S_y^i and R_y^i using **STEPS-1** to **6** from index 1 to $n-1$, we check whether or not the second columns of S_x^{n-1}, R_x^{n-1}, S_y^{n-1} and R_y^{n-1} are empty. If they are not empty, then there exist connected sequences for x and y that yield the same string and, thus, we have legal sequences. Otherwise—they are empty—there are no legal sequences for x and y.

We are now ready to present the whole procedure of our algorithm. Algorithm 1 is a pseudo description of the proposed algorithm.

We first prove the correctness of the algorithm.

T_x	1	2
1	$((1,1),A)$	$((1,2),A)$
2	$((1,2),C)$	$((2,2),C)$
3	$((1,3),T)$	$((2,3),T)$
4	$((1,4),C)$	$((2,4),C)$
5	$((1,5),G)$	$((2,5),G)$
6	$((1,6),A)$	$((2,6),A)$

S_x^1	1	2
1	$(1,AA)$	$(1,CA)$
2	$(1,AC)$	$(1,TC),(2,AC)$
3	$(1,AT)$	$(1,CT),(2,AT)$
4	$(1,AC)$	$(1,GC),(2,AC)$
5	$(1,AG)$	$(1,AG),(2,AG)$
6	$(1,AA)$	$(2,AA)$

I_x	1	2
1	$true$	$false$
2	$true$	$true$
3	$true$	$true$
4	$true$	$true$
5	$true$	$true$
6	$true$	$true$

U_x		1	2
1	2	$((0,3),C)$	$((1,4),T)$
	3	$((-1,4),T)$	$((0,5),C)$
2	2		
	3		

R_x		1	2
1	2	$(1,AC)$	$(2,AT)$
	3	$(1,AT)$	$(2,AC)$
2	2		
	3		

H_x		1	2
1	2	$true$	$true$
	3	$true$	$true$
2	2	$false$	$false$
	3	$false$	$false$

Fig. 9. An illustration of **STEP-3** for $x = ACTCGA$.

T_x	1	2
1	$((1,1),A)$	$((1,2),A)$
2	$((1,2),C)$	$((2,2),C)$
3	$((1,3),T)$	$((2,3),T)$
4	$((1,4),C)$	$((2,4),C)$
5	$((1,5),G)$	$((2,5),G)$
6	$((1,6),A)$	$((2,6),A)$

S_x^1	1	2
1	$(1,AA)$	$(1,CA)$
2	$(1,AC)$	$(1,TC),(2,AC)$
3	$(1,AT)$	$(1,CT),(2,AT)$
4	$(1,AC)$	$(1,GC),(2,AC)$
5	$(1,AG)$	$(1,AG),(2,AG)$
6	$(1,AA)$	$(2,AA)$

I_x	1	2
1	$true$	$false$
2	$true$	$true$
3	$true$	$true$
4	$true$	$true$
5	$true$	$true$
6	$true$	$true$

U_x		1	2
1	2	$((0,3),C)$	$((1,4),T)$
	3	$((-1,4),T)$	$((0,5),C)$
2	2		
	3		

R_x		1	2
1	2	$(1,AC)$	$(1,CT),(2,AT)$
	3	$(1,AT)$	$(1,TC),(2,AC)$
2	2		
	3		

H_x		1	2
1	2	$true$	$true$
	3	$true$	$true$
2	2	$false$	$false$
	3	$false$	$false$

Fig. 10. An illustration of **STEP-4** for $x = ACTCGA$.

Lemma 3.8. *If Algorithm 1 returns true, then there exist two sets $\mathcal{O}_x$ and $\mathcal{O}_y$ of operations such that $\mathcal{O}_x(x) = \mathcal{O}_y(y)$. Moreover, we can retrieve $\mathcal{O}_x$ and $\mathcal{O}_y$ as follows; let $\mathcal{R}_x = \{(i_k, i_{k+1} - 1)\}$ be a sequence of ranges covering the index 1 to n where $I_x[i_k][i_{k+1} - 1] = true$ or $H_x[i_k][i_{k+1} - 1] = true$. Then $\mathcal{O}_x = \{\theta_{(i,j)} \mid (i, j) \in \mathcal{R}_x$ and $I_x[i][j] = true\} \cup \{\tau_{(i,j)} \mid (i, j) \in \mathcal{R}_x$ and $H_x[i][j] = true\}$. $\mathcal{O}_y$ is generated similarly.*

Proof. For $(i+1, \sigma_1\sigma_2)$ added to $S_x^i[2][j']$ in lines 8 or 19, indicators with the same start index are maintained till i reaches j' (in line 6). Similarly, for $(i+1, \sigma_1\sigma_2)$ added to $R_x^i[2][j'][1]$ in lines 11 or 22, indicators with the same start index are maintained till i reaches $i + 2j - 1$ (in line 17). Therefore, if Algorithm 1 returns *true*, there exist an index i such that $I_x[i][n] = true$ or $H_x[i][n] = true$ and, thus, $\mathcal{O}_x$ and $\mathcal{O}_y$ exist.

Since σ_1 and σ_2 in lines 36 and 40 are yielded by fragments in T_x and U_x, there exist two sequences $\mathcal{S}_x$ and $\mathcal{S}_y$ of fragments that yield the same string s. For each operation in $\mathcal{O}_x$, we have two cases to consider:

1. For $\theta_{(i_k, i_{k+1}-1)}$, $I_x[i_k][i_{k+1} - 1] = true$. This follows that $T_x[i][i_k + i_{k+1} - 1 - i]$ yields $s[i]$ (from line 15) for $i_k \le i \le i_{k+1} - 1$ (from line 6). Therefore, $\theta_{(i_k, i_{k+1}-1)}$ yields $s[i_k : i_{k+1} - 1]$.
2. For $\tau_{(i_k, i_{k+1}-1)}$, $H_x[i_k][i_{k+1} - 1] = true$. Let $j = \frac{i_{k+1} - i_k}{2}$. This implies that $U_x[i][j][1]$ yields $s[i]$ for $i_k \le i \le i_k + j - 1$ (from line 26) and $U_x[i][j][2]$ yields $s[i]$ for $i_k + j \le i \le i_k + 2j - 1 = i_{k+1} - 1$ (from lines 26, 28 and 17). Therefore, $\tau_{(i_k, i_{k+1}-1)}$ yields $s[i_k : i_{k+1} - 1]$.

From the analysis of two cases for x and y, we conclude that $\mathcal{O}_x(x) = \mathcal{O}_y(y) = s$. □

Lemma 3.9. *If there exist two sets $\mathcal{O}_x$ and $\mathcal{O}_y$ of operations such that $\mathcal{O}_x(x) = \mathcal{O}_y(y)$, then Algorithm 1 returns true. Moreover, $I_x[i][j] = true$ for $\theta_{(i,j)} \in \mathcal{O}_x$ and $H_x[i][j] = true$ for $\tau_{(i,j)} \in \mathcal{O}_x$ (similar for $\mathcal{O}_y$).*

T_x	1	2
1	$((1,1),A)$	$((1,2),A)$
2	$((1,2),C)$	$((2,2),C)$
3	$((1,3),T)$	$((2,3),T)$
4	$((1,4),C)$	$((2,4),C)$
5	$((1,5),G)$	$((2,5),G)$
6	$((1,6),A)$	$((2,6),A)$

S_x^1	1	2
1	$(1,AA)$	$(1,CA)$
2	$(1,AC)$	$(2,AC)$
3	$(1,AT)$	$(1,CT),(2,AT)$
4	$(1,AC)$	$(2,AC)$
5	$(1,AG)$	$(1,AG),(2,AG)$
6	$(1,AA)$	

I_x	1	2
1	*true*	*false*
2	*true*	*true*
3	*false*	*true*
4	*true*	*true*
5	*false*	*true*
6	*true*	*false*

U_x		1	2
1	2	$((0,3),C)$	$((1,4),T)$
	3	$((-1,4),T)$	$((0,5),C)$
2	2		
	3		

R_x		1	2
1	2	$(1,AC)$	$(1,CT),(2,AT)$
	3	$(1,AT)$	$(2,AC)$
2	2		
	3		

H_x		1	2
1	2	*true*	*true*
	3	*false*	*true*
2	2	*false*	*false*
	3	*false*	*false*

T_y	1	2
1	$((1,1),C)$	$((1,2),C)$
2	$((1,2),A)$	$((2,2),A)$
3	$((1,3),G)$	$((2,3),G)$
4	$((1,4),A)$	$((2,4),A)$
5	$((1,5),T)$	$((2,5),T)$
6	$((1,6),C)$	$((2,6),C)$

S_y^1	1	2
1	$(1,AC)$	$(1,AC)$
2	$(1,AA)$	$(1,GA),(2,CA)$
3	$(1,AG)$	$(1,AG)$
4	$(1,AA)$	$(2,CA)$
5	$(1,AT)$	$(1,CT),(2,CT)$
6	$(1,AC)$	

I_y	1	2
1	*true*	*false*
2	*true*	*true*
3	*true*	*false*
4	*true*	*true*
5	*false*	*true*
6	*true*	*false*

U_y		1	2
1	2	$((0,3),G)$	$((1,4),A)$
	3	$((-1,4),A)$	$((0,5),T)$
2	2		
	3		

R_y		1	2
1	2	$(1,AG)$	$(2,CA)$
	3	$(1,AA)$	$(1,AT),(2,CT)$
2	2		
	3		

H_y		1	2
1	2	*false*	*true*
	3	*true*	*true*
2	2	*false*	*false*
	3	*false*	*false*

Fig. 11. An illustration of **STEP-6** for $x = ACTCGA$ and $y = CAGATC$.

Proof. Since $\mathcal{O}_x(x) = \mathcal{O}_y(y)$, there exist corresponding legal sequences $\mathbb{S}_x$ and $\mathbb{S}_y$ that yield the same string, say s. If the second columns of both S_x^i and R_x^i are empty after line 44 for an index i, these legal sequence cannot exist. Therefore, the second columns of S_x^{n-1} and R_x^{n-1} should be not empty and Algorithm 1 returns true. For each operation in $\mathcal{O}_x$, we consider two cases.

1. For $\theta_{(i_k,i_{k+1}-1)}$, $(i_k,\sigma_1\sigma_2)$ is added to $S_x^{i_k-1}[2][i_{k+1}-1]$ and $I[i_k-1+1][i_{k+1}-1]$ becomes *true* from lines 8, 9, 19 and 20.
2. For $\tau_{(i_k,i_{k+1}-1)}$, $(i_k,\sigma_1\sigma_2)$ is added to $R_x^{i_k-1}[2][\frac{i_{k+1}-i_k}{2}][1]$ and $T[i_k-1+1][i_k+2\cdot\frac{i_{k+1}-i_k}{2}]$ becomes *true* from lines 11, 12, 22 and 23.

For all two cases, the statement holds. □

Next, we prove that indicators in each cell of S_x^i and R_x^i are sorted by ascending order with respect to the start index.

Lemma 3.10. *In Algorithm 1, indicators $(i_2,\sigma_1\sigma_2)$ in each cell of $S_x^i[2][j]$ are sorted by ascending order with respect to i_2 and $i_2 \le i+1$ after line 33 for every i.*

Proof. We prove the statement by induction on i. Note that cells of $S_x^i[2][j]$ are updated in lines 8, 15 and 19 of Algorithm 1.

Base Case Since $i = 1$, $(2,\sigma_1\sigma_2)$ is appended to $S_x^1[2][j']$ in lines 8 and 19. In line 15, i_1 is always 1 from the initialization of S_x^1 and $(1,\sigma_1\sigma_2)$ is appended to $S^1[2][j-1]$. Therefore, indicators in each cell of $S_x^1[2][j]$ are sorted by ascending order after line 33. The maximum value of start indices is 2, which is $i+1$.

Induction Hypothesis Assume indicators $(i_2,\sigma_1\sigma_2)$ in each cell of $S_x^i[2][j]$ are sorted by ascending order with respect to i_2 and $i_2 \le i+1$ after line 33 for an index i.

Algorithm 1: FindLegalSequences.

```
   Input: Strings $x$ and $y$
   Output: Boolean (whether or not there exist $\mathcal{O}_x$ and $\mathcal{O}_y$ such that $\mathcal{O}_x(x) = \mathcal{O}_y(y)$.)
   /* time complexity: $O(n^3)$, space complexity: $O(n^2)$                                   */
1  make $T_x$, $U_x$, $T_y$, $U_y$
2  initialize $S_x^1$, $R_x^1$, $S_y^1$, $R_y^1$, $I_x$, $H_x$, $I_y$ and $H_y$
3  for $i \leftarrow 1$ to $n-1$ do
4      for strings x and y do
5          for $j \leftarrow 1$ to $n$ do                                                   // STEP-1
6              if $(j, \sigma_0\sigma_1) \in S_x^i[1][j]$ then
7                  for $j' \leftarrow i+1$ to $n$ do
8                      append $(i+1, \sigma_1\sigma_2)$ to $S_x^i[2][j']$, where $\sigma_2$ is yielded by $T_x[i+1][j']$
9                      $I_x[i+1][j'] \leftarrow true$
10                 for $j' \leftarrow 2$ to $\lfloor n/2 \rfloor$ do
11                     append $(i+1, \sigma_1\sigma_2)$ to $R_x^i[2][j'][1]$, where $\sigma_2$ is yielded by $U_x[i+1][j'][1]$
12                     $H_x[i+1][i+2j'] \leftarrow true$
13         for $j \leftarrow 2$ to $n$ do                                                   // STEP-2
14             for each $(i_1, \sigma_0\sigma_1) \in S_x^i[1][j]$, $i_1$ in descending order except $j$ do
15                 prepend $(i_1, \sigma_1\sigma_2)$ to $S_x^i[2][j-1]$, where $T_x[i+1][j-1]$ yields $\sigma_2$
16         for $j \leftarrow 1$ to $n$ do                                                   // STEP-3
17             if $(i-2j+1, \sigma_0\sigma_1) \in R_x^i[1][j][2]$ then
18                 for $j' \leftarrow i+1$ to $n$ do
19                     append $(i+1, \sigma_1\sigma_2)$ to $S_x^i[2][j']$, where $\sigma_2$ is yielded by $T_x[i+1][j']$
20                     $I_x[i+1][j'] \leftarrow true$
21                 for $j' \leftarrow 2$ to $\lfloor n/2 \rfloor$ do
22                     append $(i+1, \sigma_1\sigma_2)$ to $R_x^i[2][j'][1]$, where $\sigma_2$ is yielded by $U_x[i+1][j'][1]$
23                     $H_x[i+1][i+2j'] \leftarrow true$
24         for $j \leftarrow 1$ to $n$ do                                                   // STEP-4
25             for each $(i_1, \sigma_0\sigma_1) \in R_x^i[1][j][1]$, $i_1$ in descending order except $i-2j+1$ do
26                 if $i < i_1 + j - 1$ then prepend $(i_1, \sigma_1\sigma_2)$ to $R_x^i[2][j][1]$, where $U_x[i+1][j][1]$ yields $\sigma_2$ else prepend $(i_1, \sigma_1\sigma_2)$ to $R_x^i[2][j][2]$,
                   where $U_x[i+1][j][2]$ yields $\sigma_2$
27             for each $(i_1, \sigma_0\sigma_1) \in R_x^i[1][j][2]$, $i_1$ in ascending order except $i-2j+1$ do
28                 if $i < i_1 + 2j - 1$ then append $(i_1, \sigma_1\sigma_2)$ to the end of $R_x^i[2][j][2]$, where $U_x[i+1][j][2]$ yields $\sigma_2$
29         clear $C_x^i$
30         for $j \leftarrow 1$ to $n$ do                                                   // STEP-5
31             for each $(i_1, \sigma_1\sigma_2) \in S_x^i[2][j]$ or $R_x^i[2][j]$ do $C_x^i[\sigma_1][\sigma_2] \leftarrow true$
32     $C_x'^i, C_y'^i \leftarrow C_x^i \wedge C_y^i$                                       // STEP-6
33     for strings x and y do
34         for $j \leftarrow 1$ to $n$ do
35             for each $(i_1, \sigma_1\sigma_2) \in S_x^i[2][j]$ do
36                 if $C_x'^i[\sigma_1][\sigma_2] = false$ then
37                     remove $(i_1, \sigma_1\sigma_2)$
38                     $I_x[i_1][i+j+1-i_1] \leftarrow false$
39             for each $(i_1, \sigma_1\sigma_2) \in R_x^i[2][j][k]$ do
40                 if $C_x'^i[\sigma_1][\sigma_2] = false$ then
41                     remove $(i_1, \sigma_1\sigma_2)$
42                     $H_x[i_1][i_1+2j-1] \leftarrow false$
43     copy the second columns of $S_x^i$, $R_x^i$, $S_y^i$ and $R_y^i$ to the first columns of $S_x^{i+1}$, $R_x^{i+1}$, $S_y^{i+1}$ and $R_y^{i+1}$.
44 return the second columns of $S_x^{n-1}$, $R_x^{n-1}$, $S_y^{n-1}$ and $R_y^{n-1}$ are not empty
```

Inductive Step For an index $i+1$, indicators $(i_1, \sigma_0\sigma_1)$ in each cell of $S_x^{i+1}[1][j]$ is sorted by ascending order with respect to i_1 from the induction hypothesis and line 43. In lines 8 and 19, $(i+2, \sigma_1\sigma_2)$ is appended to $S_x^{i+1}[2][j']$. In line 15, $(i_1, \sigma_1\sigma_2)$ is prepended to $S_x^{i+1}[2][j-1]$ in descending order with respect to i_1, where $i_1 \le i+1$ from the induction hypothesis. Therefore, indicators $(i_2, \sigma_1\sigma_2)$ in each cell of $S_x^{i+1}[2][j]$ are sorted by ascending order with respect to i_2 after line 33. The maximum value of i_2 is $i+2$, which is $(i+1)+1$. □

Lemma 3.11. *In Algorithm 1, indicators $(i_2, \sigma_1\sigma_2)$ in each cell of $R_x^i[2][j][k]$ are sorted by ascending order with respect to i_2 and $i_2 \le i+1$ after line 33 for every i.*

Proof. We prove the statement by induction on i. Note that cells of $R_x^i[2][j][k]$ are updated in lines 11, 22, 26 and 28.

Base Case Since $i = 1$, $(2, \sigma_1\sigma_2)$ is appended to $R_x^1[2][j][1]$ in lines 11 and 22. In line 26, i_1 is always 1 from the initialization of R_x^1 and $(1, \sigma_1\sigma_2)$ is appended to $R^1[2][j][k]$. In line 28, there is no indicator in $R_x^1[1][j][2]$. Therefore, indicators in each cell of $R_x^1[2][j][k]$ are sorted by ascending order after line 33. The maximum value of start indices is 2, which is $i + 1$.

Induction Hypothesis Assume indicators $(i_2, \sigma_1\sigma_2)$ in each cell of $R_x^i[2][j][k]$ are sorted by ascending order with respect to i_2 and $i_2 \le i + 1$ after line 33 for an index i.

Inductive Step For an index $i + 1$, indicators $(i_1, \sigma_0\sigma_1)$ in each cell of $R_x^{i+1}[1][j][k]$ is sorted by ascending order with respect to i_1 from the induction hypothesis and line 43. In lines 11 and 22, $(i + 2, \sigma_1\sigma_2)$ is appended to $R_x^{i+1}[2][j'][1]$. In lines 26 and 28, $(i_1, \sigma_1\sigma_2)$ is prepended to $R^i[2][j][k]$ in descending order with respect to i_1, where $i_1 \le i + 1$ from the induction hypothesis. Therefore, indicators $(i_2, \sigma_1\sigma_2)$ in each cell of $R_x^{i+1}[2][j][k]$ are sorted by ascending order with respect to i_2 after line 33. The maximum value of i_2 is $i + 2$, which is $(i + 1) + 1$. □

Before we analyze the time and space complexity of Algorithm 1, we establish the following results.

Lemma 3.12. *In Algorithm 1, the maximum value of start indices of indicators in $S_x^i[1][j]$ in line 6 is at most j.*

Proof. From Lemma 3.10, the maximum value of start indices of indicators in $S_x^i[2][j]$ after line 33 is $i + 1$. Moreover, $(i + 1, \sigma_1\sigma_2)$ is added to $S_x^i[2][j']$ in lines 9 and 19, where $i + 1 \le j'$. Therefore, when $j \le i$, the maximum value of start indices of indicators in $S_x^i[2][j]$ is less than j. When $j \ge i + 1$, the maximum value of start indices of indicators in $S_x^i[2][j]$ is j. From line 43, the maximum value of start indices of indicators in $S_x^i[1][j]$ in line 6 is at most j. □

Lemma 3.13. *In Algorithm 1, the minimum value of start indices of indicators in $R^i[1][j][2]$ in line 17 is at least $i - 2j + 1$.*

Proof. The if-statement in line 17 checks the existence of $(i - 2j + 1, \sigma_0\sigma_1)$ in $R_x^i[1][j][2]$ and $(i - 2j + 1, \sigma_0\sigma_1)$ increases as i increases for fixed j. Therefore, if $(i' - 2j + 1, \sigma_0\sigma_1) \in R_x^i[1][j][2]$ where $i' \le i$, then the indicator should have been checked for the index i' and removed from $R_x^i[1][j][2]$. □

Now we are ready to analyze the time and space complexity of Algorithm 1.

Theorem 3.14. *Algorithm 1 runs in $O(n^3)$ time using $O(n^2)$ space, where $n = |x| = |y|$.*

Proof. From Lemmas 3.10 and 3.12, we need to check only the last indicator in $S_x^i[1][j]$ at line 6. Therefore, **STEP-1** requires $O(n^3)$ time. From Lemmas 3.11 and 3.13, we need to check only the first indicator in $R_x^i[1][j][2]$ at line 17. Therefore, **STEP-3** also requires $O(n^3)$ time. It is clear from iterations that **STEPS-2**, **-4**, **-5**, **-6** require at most $O(n^3)$ time. Therefore, the time complexity of Algorithm 1 is $O(n^3)$. For the space requirement, $T_x, U_x, S_x^i, R_x^i, I_x, H_x$ requires $O(n^2)$ space. Therefore, the space complexity of Algorithm 1 is $O(n^2)$. □

Algorithm 2 is a pseudo description of retrieving an alignment from I_x, H_x, I_y and H_y after Algorithm 1 finishes, based on Lemmas 3.8 and 3.9.

Lemma 3.15. *Algorithm 2 runs in $O(n)$ time using $O(n)$ space.*

Proof. For each iteration from line 4 to 13, i decreases by at least 1. Since $i = n$ before the iteration and the end condition is $i = 0$, the iteration runs in $O(n)$ time. Therefore, Algorithm 2 runs in $O(n)$ time. For the space requirement, $\mathcal{O}_x$ and $\mathcal{O}_y$ requires $O(n)$ space. Therefore, the space complexity of Algorithm 2 is $O(n)$. □

Based on Lemmas 3.8 and 3.9 and Algorithm 2, we establish the following result:

Theorem 3.16. *We can determine the existence of an alignment with non-overlapping inversions and translocations on two strings x and y in $O(n^3)$ time using $O(n^2)$ space, where n is the size of input strings. Moreover, we can find two sets $\mathcal{O}_x$ and $\mathcal{O}_y$ of operations such that $\mathcal{O}_x(x) = \mathcal{O}_y(y)$ in $O(n)$ time using $O(n)$ space if there exists an alignment.*

4. Conclusions

An inversion and a translocation are important for bio sequences including DNAs or RNAs and these operations are closely related to mutations. We have, in particular, considered non-overlapping inversions and translocations on both sequences, which is crucial to find the original common sequence from two mutated sequences. We have introduced a new

Algorithm 2: RetrieveAlignments.

```
   Input: Array $I_x[n][n]$, $H_x[n][n]$, $I_y[n][n]$ and $H_y[n][n]$ from Algorithm 1
   Output: Sets $\mathcal{O}_x$ and $\mathcal{O}_y$ of operations such that $\mathcal{O}_x(x) = \mathcal{O}_y(y)$
   /* time complexity: $O(n)$, space complexity: $O(n)$ */
1  make empty sets $\mathcal{O}_x$ and $\mathcal{O}_y$
2  for strings x and y do
3      $i, j \leftarrow n$
4      repeat
5          if $I_x[i][j] = true$ then
6              add $\theta_{(i,j)}$ to $\mathcal{O}_x$
7              $i, j \leftarrow i - 1$
8          else if $H_x[i][j] = true$ then
9              add $\tau_{(i,j)}$ to $\mathcal{O}_x$
10             $i, j \leftarrow i - 1$
11         else
12             $i \leftarrow i - 1$
13     until $i = 0$
14 return $\mathcal{O}_x$ and $\mathcal{O}_y$
```

problem, alignment with non-overlapping inversions and translocations on two strings, and presented a polynomial algorithm for the problem. Given two strings x and y, based on the properties of inversions and translocations, our algorithm decides whether or not there exist two sets $\mathcal{O}_x$ and $\mathcal{O}_y$ of operations for x and y such that $\mathcal{O}_x(x) = \mathcal{O}_y(y)$ in $O(n^3)$ time using $O(n^2)$ space, where $n = |x| = |y|$. Once we know the existence of such $\mathcal{O}_x$ and $\mathcal{O}_y$, we can retrieve $\mathcal{O}_x$ and $\mathcal{O}_y$ in $O(n)$ time using $O(n)$ space. One future work is to improve the current running time $O(n^3)$. As far as we are aware, this algorithm is the first try to find an alignment with non-overlapping inversions and translocations on both strings. Our alignment problem can be extended to approximate pattern matching or edit distance problem.

Acknowledgements

We wish to thank the referees for the careful reading of the paper and many valuable suggestions.

This research was supported by the Basic Science Research Program through NRF (National Research Foundation) of Korea funded by MEST (2012R1A1A2044562). Kim was supported by NRF Grant funded by the Korean Government (NRF-2013-Global Ph.D. Fellowship Program).

References

[1] C.E.R. Alves, A.P. do Lago, A.F. Vellozo, Alignment with non-overlapping inversions in $O(n^3 \log n)$-time, in: Proceedings of GRACO 2005, in: Electronic Notes in Discrete Mathematics, vol. 19, 2005, pp. 365–371.
[2] D. Cantone, S. Cristofaro, S. Faro, Efficient string-matching allowing for non-overlapping inversions, Theoret. Comput. Sci. 483 (2013) 85–95.
[3] D. Cantone, S. Faro, E. Giaquinta, Approximate String Matching Allowing for Inversions and Translocations, 2010, pp. 37–51.
[4] Z.-Z. Chen, Y. Gao, G. Lin, R. Niewiadomski, Y. Wang, J. Wu, A space-efficient algorithm for sequence alignment with inversions and reversals, Theoret. Comput. Sci. 325 (3) (2004) 361–372.
[5] Y. Gao, J. Wu, R. Niewiadomski, Y. Wang, Z.-Z. Chen, G. Lin, A space efficient algorithm for sequence alignment with inversions, in: Proceedings of COCOON 2003, in: Lecture Notes in Computer Science, vol. 2697, 2003, pp. 57–67.
[6] S. Grabowski, S. Faro, E. Giaquinta, String matching with inversions and translocations in linear average time (most of the time), Inform. Process. Lett. 111 (11) (2011) 516–520.
[7] Z. Ignatova, K. Zimmermann, I. Martinez-Perez, DNA Computing Models, Advances in Information Security, 2008.
[8] J.D. Kececioglu, D. Sankoff, Exact and approximation algorithms for the inversion distance between two chromosomes, in: Proceedings of CPM 1993, in: Lecture Notes in Computer Science, vol. 684, 1993, pp. 87–105.
[9] S.C. Li, Y.K. Ng, On protein structure alignment under distance constraint, in: Proceedings of ISAAC 2009, in: Lecture Notes in Computer Science, vol. 5878, 2009, pp. 65–76.
[10] O. Lipsky, B. Porat, E. Porat, B.R. Shalom, A. Tzur, Approximate string matching with swap and mismatch, in: Proceedings of ISAAC 2007, in: Lecture Notes in Computer Science, vol. 4835, 2007, pp. 869–880.
[11] J.R. Lupski, Genomic disorders: structural features of the genome can lead to DNA rearrangements and human disease traits, Trends Genet. 14 (10) (1998) 417–422.
[12] C.M. Ogilvie, P.N. Scriven, Meiotic outcomes in reciprocal translocation carriers ascertained in 3-day human embryos, European J. Hum. Genet. 10 (12) (2009) 801–806.
[13] M. Oliver-Bonet, J. Navarro, M. Carrera, J. Egozcue, J. Benet, Aneuploid and unbalanced sperm in two translocation carriers: evaluation of the genetic risk, Mol. Hum. Reprod. 8 (10) (2002) 958–963.
[14] T.S. Painter, A new method for the study of chromosome rearrangements and the plotting of chromosome maps, Science 78 (1933) 585–586.
[15] Y. Sakai, A new algorithm for the characteristic string problem under loose similarity criteria, in: Proceedings of ISAAC 2011, in: Lecture Notes in Computer Science, vol. 7074, 2011, pp. 663–672.
[16] M. Schöniger, M.S. Waterman, A local algorithm for DNA sequence alignment with inversions, Bull. Math. Biol. 54 (4) (1992) 521–536.
[17] A.F. Vellozo, C.E.R. Alves, A.P. do Lago, Alignment with non-overlapping inversions in $O(n^3)$-time, in: Proceedings of WABI 2006, in: Lecture Notes in Computer Science, vol. 4175, 2006, pp. 186–196.