Alignment with Non-overlapping Inversions on Two Strings*

Da-Jung Cho, Yo-Sub Han, and Hwee Kim

Department of Computer Science, Yonsei University
50, Yonsei-Ro, Seodaemun-Gu, Seoul 120-749, Republic of Korea
{dajung,emmous,kimhwee}@cs.yonsei.ac.kr

Abstract. The inversion is one of the important operations in bio sequence analysis and the sequence alignment problem is well-studied for efficient bio sequence comparisons. Based on inversion operations, we introduce the alignment with non-overlapping inversion problem: Given two strings x and y, does there exist an alignment with non-overlapping inversions for x and y. We, in particular, consider the alignment problem when non-overlapping inversions are allowed for both x and y. We design an efficient algorithm that determines the existence of non-overlapping inversions and present another efficient algorithm that retrieves such 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 [1, 7, 12, 13]. For a DNA sequence, a *chromosomal translocation* is to relocate a piece of the DNA sequence from one place to another and, thus, rearrange the sequence [9]. The chromosomal translocation is a crucial operation in DNAs since it alters a DNA sequence and often causes genetic diseases [10]. A *chromosomal inversion* occurs when a single chromosome undergoes breakage and rearrangement within itself [11]. Based on the important biological events such as translocation and inversion, there is a well-defined string matching problem: given two strings and translocation or inversion, the string matching problem is finding all matched strings allowing translocations or inversions. Moreover, people proposed an alignment with translocation or inversion problem, which is closely related to find similarity between two given strings; that is to obtain minimal occurrences of translocation or inversion that transform one to the other. Many researchers investigated efficient algorithms for this problem [1–4, 6, 8, 13, 14].

The inversions, which are one of the important biological operations, are not automatically detected by the traditional alignment algorithms [14]. Schöniger

* This research was supported by the Basic Science Research Program through NRF funded by MEST (2012R1A1A2044562).

S.P. Pal and K. Sadakane (Eds.): WALCOM 2014, LNCS 8344, pp. 261–272, 2014.

and Waterman [13] introduced the alignment problem with non-overlapping inversions, defined a simplification hypothesis that all regions will not be allowed to overlap, and showed 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 $n \geq m$. Vellozo et al. [14] presented an $O(n^2m)$ algorithm, which improved the previous algorithm by Schöniger and Waterman. They built a matrix for one string and partially inverted string using table filling method with regard to the extended edit graph. Recently, Cantone et al. [1] 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.

Many diseases are often caused by genetic mutations, which can be inherited through generations and can result in new sequences from a normal gene [5]. 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 mutation. See Fig. 1 for an example.

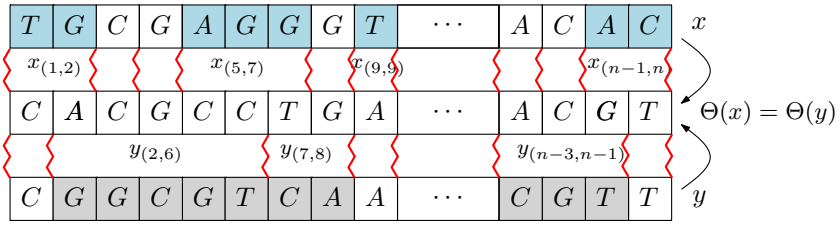

Fig. 1. An example of non-overlapping inversions on both strings x and y, where $\Theta_x = (1,2)(3,3)'(4,4)'(5,7)(8,8)'(9,9)\cdots(n-1,n)$ and $\Theta_y = (1,1)'(2,6)(7,8)(9,9)'\cdots(n-3,n-1)(n,n)'$. Note that $(i,i)'$ denotes the alignment at position i without complementing $x[i]$.

Note that this problem is different from the previous problem [13, 14], where a non-overlapping inversion occurs only in one string and transforms the string to the other string; namely $\Theta(x) = y$ for a set Θ of non-overlapping inversions. On the other hand, we consider more general case where inversions can occur in both x and y simultaneously. The problem is also equivalent to the string alignment problem allowing the inversions occurring at most two times at the same positions.

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 $1 \leq i \leq n$. 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 character 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 $1 \leq i \leq j \leq |s|$. We consider biological operation *inversion* θ and denote by $\theta(s)$ the reverse and complement of a string s. For example, $\theta(A) = T$ and $\theta(AGG) = \theta(G)\theta(G)\theta(A) = CCT$. We define an inversion operation $\theta_{(i,j)}$ for a given range i, j as follows:

$$\theta_{(i,j)}(s) = \theta(s_{(i,j)}).$$

For simplicity, we use (i, j) instead of $\theta_{(i,j)}$ if the notation is clear in the context. We say that $\frac{i+j}{2}$ is the *center of the inversion* for (i, j). We define a *set Θ of non-overlapping inversion* to be

$$\Theta = \{(i, j) \mid 1 \leq i \leq j \text{ and for } \forall (i', j') \neq (i, j) \in \Theta, j < i' \text{ or } j' < i\}.$$

Then, for a set Θ of non-overlapping inversions and a string s, we have $\Theta(s) = s'$, where

$$s'[i] = \begin{cases} \theta(s[j + k - i]) & \text{if } (j, k) \in \Theta \text{ and } j \leq i \leq k \\ s[i] & \text{otherwise.} \end{cases}$$

For example, given $\Theta = \{(1, 1), (2, 3)\}$ and $s = AGCC$, we have $\Theta(s) = \theta(A)\theta(GC)C = TGCC$. From now on, we use a set of inversions instead of a set of non-overlapping inversions since we only consider sets of non-overlapping inversions.

Definition 1. *We define a new alignment problem with non-overlapping inversions on two strings as follows: Given two strings x and y of the same length, can we determine whether or not there exist two sets Θ_x and Θ_y of inversions such that $\Theta_x(x) = \Theta_y(y)$?*

3 The Algorithm

We use $x = AGCT$ and $y = CGAA$ as our example strings for explaining the algorithm. Remark that $\theta(AG)C\theta(T) = CTCA = C\theta(GA)A$ and, thus, we have two sets $\Theta_x = \{(1, 2), (4, 4)\}$ and $\Theta_y = \{(2, 3)\}$.

We start from building a table in which each cell contains a pair of a range and a character. We define an array $T_x[n][n + 1]$ for x as follows:

$$T_x[i][j] = \begin{cases} ((j, i), \theta(x[j])) & \text{if } j < i, \\ ((i, i)', x[i]) & \text{if } j = i, \\ ((i, j - 1), \theta(x[j - 1])) & \text{if } j > i. \end{cases}$$

We call all elements in T_x *inversion fragments* of x. For an inversion fragment $\mathbb{F} = ((p, q), \sigma)$ or $((p, p)', \sigma)$, we say that $\mathbb{F}$ *yields* the character σ and $\frac{p+q}{2}$ is the *center* of the inversion fragment. For a sequence of inversion fragments $\mathbb{F}_1, \ldots, \mathbb{F}_n$, where $\mathbb{F}_i$ yields σ_i, we say that the sequence yields a string $\sigma_1 \cdots \sigma_n$.

Inversion fragments become useful to compute a substring created by any inversion because of the following property of the inversion operation:

$$\theta_{(i-1,j+1)}(x) = \theta(x[j+1])\theta_{(i,j)}(x)\theta(x[i-1]).$$

From a string x and its table T_x, we make the following observation:

Observation 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) $((i,j),\theta(x[j])),((i+1,j-1),\theta(x[j-1])),\ldots,((i+1,j-1),\theta(x[i+1])),((i,j),\theta(x[i]))$ *are all inversion fragments in the ith, $i+1$th, ..., $j-1$th, jth columns of T_x and have the same center.*

It is easy to verify from the construction that we can construct T_x in $O(n^2)$ time and the size of T_x is $O(n^2)$, where $|x| = n$. See Fig. 2 for an example. We also construct T_y for y.

T_x	1	2	3	4
1	$((1,1)', A)$	$((1,2), T)$	$((1,3), T)$	$((1,4), T)$
2	$((1,1), T)$	$((2,2)', G)$	$((2,3), C)$	$((2,4), C)$
3	$((1,2), C)$	$((2,2), C)$	$((3,3)', C)$	$((3,4), G)$
4	$((1,3), G)$	$((2,3), G)$	$((3,3), G)$	$((4,4)', T)$
5	$((1,4), A)$	$((2,4), A)$	$((3,4), A)$	$((4,4), A)$

Fig. 2. An example table T_x for $x = AGCT$. In this example, $T_x[3][3] = ((3,3)', C)$ means that we put C instead of $\theta(C) = G$ since the range is $(3,3)'$. On the other hand, we have $T_x[3][4] = ((3,3), G)$ because the range is $(3,3)$. Shaded cells denote inversion fragments that represent the inversion $\theta_{(1,3)}$.

Given a pair $(((p_1,p_2),\sigma_1),((q_1,q_2),\sigma_2))$ of two inversion fragments, we say that the pair is an *agreed pair* if $q_1 = p_2 + 1$ or $p_1 + p_2 = q_1 + q_2$. Otherwise, we call it a *disagreed pair*. Then, for two agreed pairs $(((p_1,p_2),\sigma_1),((q_1,q_2),\sigma_2))$ and $(((q_1,q_2),\sigma_2),((r_1,r_2),\sigma_3))$, we say that two pairs are *connected* by $((q_1,q_2),\sigma_2)$. We define an *agreed sequence* $\mathbb{S}$ to be a sequence of inversion fragments $((a_i,b_i),\sigma_i)$ for $1 \le i \le n$, where $a_1 = 1$, $b_n = n$ and $(((a_i,b_i),\sigma_i),((a_{i+1},b_{i+1}),\sigma_{i+1}))$ is an agreed pair for $1 \le i \le n-1$.

Given an agreed sequence $\mathbb{S}$, we define a set $\mathcal{F}_\Theta(\mathbb{S})$ of inversions from $\mathbb{S}$ as follows:

$$\mathcal{F}_\Theta(\mathbb{S}) = \{(p,q) \mid \exists \mathbb{S}[p] \text{ such that } \mathbb{S}[p] = ((p,q),\sigma)\}.$$

Given a set Θ of inversions, we can return an agreed sequence $\mathbb{S}$ from Θ as follows: (namely, $\mathbb{S} = \mathcal{F}_{\mathbb{S}}(\Theta)$.)

$$\mathbb{S}[i] = \begin{cases} ((i,j+k-i),\theta(x[j+k-i])) & \text{if } \exists(j,k) \in \Theta \text{ s.t. } j \le i \le k \text{ and } i < \frac{j+k}{2}, \\ ((j+k-i,i),\theta(x[j+k-i])) & \text{if } \exists(j,k) \in \Theta \text{ s.t. } j \le i \le k \text{ and } i \ge \frac{j+k}{2}, \\ ((i,i)', x[i]) & \text{otherwise.} \end{cases}$$

Observation 2. *Given a string x and its set Θ_x of non-overlapping inversions, $\mathcal{F}_{\mathbb{S}}(\Theta_x)$ yields $\Theta_x(x)$.*

Note that an agreed pair of two inversion fragments with the same center represents the same inversion in Θ. We define an agreed sequence $\mathbb{S}_x$ ($\mathbb{S}_y$, respectively) to be *legal* if there exist Θ_x and Θ_y such that $\Theta_x(x) = \Theta_y(y)$ and $\mathcal{F}_{\Theta}(\mathbb{S}_x) = \Theta_x$ ($\mathcal{F}_{\Theta}(\mathbb{S}_y) = \Theta_y$, respectively). Then, our problem is to determine whether or not there exist legal sequences $\mathbb{S}_x$ and $\mathbb{S}_y$ for two strings x and y.

The main idea of our algorithm is to keep tracking of all possible agreed pairs for adjacent indices and check if there exist two connected pairs $\mathbb{P}_x$ and $\mathbb{P}_y$ for x and y, that generate a common substring of x and y; namely, we check if there exist a common substring $\sigma_1\sigma_2\sigma_3$ and $\mathbb{P}_x = (\mathbb{F}_1, \mathbb{F}_2), (\mathbb{F}_2, \mathbb{F}_3)$ for x and $\mathbb{P}_y$ for y such that $\mathbb{F}_j$ yields σ_j. For instance, for $x = AGCT$ and $y = CGAA$, there exists a common substring CTC from index 1 to 3 such that

$$\mathbb{P}_x = (((1,2), C), ((1,2), T)), (((1,2), T), ((3,3)', C))$$

and

$$\mathbb{P}_y = (((1,1)', C), ((2,3), T)), (((2,3), T), ((2,3), C)).$$

Next, we define the following four sets for each index i:

Definition 2. *For a string x, its T_x and an index i, we define four sets as follows:*

(1) $AH_x^i = \{((p,i), \sigma) = T_x[i][p] \mid 1 \le p \le i \le n-1\} \cup \{((i,i)', x[i])\}$, *which is a set of all inversion fragments that end at i.*
(2) $AT_x^i = \{((i+1,q), \sigma) = T_x[i+1][q+1] \mid i < q \le n\} \cup \{((i+1,i+1)', x[i+1])\}$, *which is a set of all inversion fragments that start from $i+1$.*
(3) $BH_x^i = \{((p,q), \sigma) = T_x[i][j] \mid p > 1, 1 \le j \le n\}$, *which is a set of all inversion fragments that start before or from i.*
(4) $BT_x^i = \{((p,q), \sigma) = T_x[i+1][j] \mid q < n, 1 \le j \le n\}$, *which is a set of all inversion fragments that end after or at $i+1$.*

From these four sets, we establish the following observations:

Observation 3. *Given a string x and its four sets AH_x^i, AT_x^i, BH_x^i and BT_x^i, the following statements hold: For two inversion fragments $\mathbb{F}_1, \mathbb{F}_2$, if $(\mathbb{F}_1, \mathbb{F}_2)$ is an agreed pair, then*

(1) $\mathbb{F}_1 \in AH_x^i$ *and* $\mathbb{F}_2 \in AT_x^i$, *or*
(2) $\mathbb{F}_1 \in BH_x^i$ *and* $\mathbb{F}_2 \in BT_x^i$.

Based on Observation 3, it is possible to create all agreed pairs for an index i by comparing AH_x^i and AT_x^i, and BH_x^i and BT_x^i. If we can connect pairs through all indices, then we are able to generate all agreed sequences, which are essentially all sets of non-overlapping inversions.

We need additional tables $C_x^i[|\Sigma|][|\Sigma|][|\Sigma|]$ and $C_y^i[|\Sigma|][|\Sigma|][|\Sigma|]$ to record a sequence of three characters generated by connected pairs. For an index i,

$$C_x^i[\sigma_1][\sigma_2][\sigma_3] = \begin{cases} true & \text{if } \exists \mathbb{S}_x \text{ such that } \mathbb{S}_x[i-1]\mathbb{S}_x[i]\mathbb{S}_x[i+1] \text{ yields } \sigma_1\sigma_2\sigma_3, \\ false & \text{otherwise.} \end{cases}$$

We also define $S_x^i[2][n+1]$ and $S_y^i[2][n+1]$ as follows. For an index i and $1 \leq j \leq 2, 1 \leq k \leq n+1$, we say that $S_x^i[j][k] \ni (t, \sigma_1\sigma_2\sigma_3)$ if there exists $\mathbb{S}_x$ such that

- $T_x[i+j-1][k] = \mathbb{S}_x[i+j-1] = ((p_1, p_2), \sigma_3)$,
- $t \leq i+j-1$,
- $\mathbb{S}_x[t] = ((t, p_1+p_2-t), \omega)$ for some ω.

In other words, an element $(t, \sigma_1\sigma_2\sigma_3)$ in the first column (second column, respectively) of S_x^i for an index i represents that there exists an agreed sequence $\mathbb{S}_x$, where inversion (t, s) (or the identity function at index t) creates a suffix of the string yielded by the sequence.

We design an algorithm that computes S_x^i and S_y^i for each index and checks whether or not S_x^{n-1} and S_y^{n-1} are empty. If $T_x[1][j]$ yields σ, then we set $S_x^1[1][j] = \{(1, AA\sigma)\}$ as initial data. We also set S_y^1 similarly. Note that each cell in S_x^i and S_y^i has $O(n)$ elements. 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.)

STEP-1: We check all inversion fragments in the ith column of T_x. For $T_x[i][j] = ((j, i), \sigma_2)$ (or $((i, i)', \sigma_2)$), if $(j, \sigma_0\sigma_1\sigma_2) \in S_x^i[1][j]$, then we add $((j, i), \sigma_1\sigma_2)$ (or $((i, i)', \sigma_1\sigma_2)$) to a set AH_x^i. Namely, AH_x^i contains every inversion fragment that can be the ith element in a legal sequence and ends at i. We need to check $i+1$ inversion fragments for this step, and for each inversion fragment, we need to examine $O(n)$ elements. Therefore, we need $O(n^2)$ time for the step.

STEP-2: We check all inversion fragments in the $i+1$th column of T_x. For $T_x[i+1][i+1] = ((i+1, i+1),' \sigma_3)$, we add $((i+1, i+1)', \sigma_3)$ to a set AT_x^i. Moreover, for each $T_x[i+1][j] = ((i+1, j-1), \sigma_3)$, we add $((i+1, j-1), \sigma_3)$ to a set AT_x^i. In other words, AT_x^i contains every inversion fragment that can be the $i+1$th element in a legal sequence and starts from $i+1$. Note that AT_x^i has $n-i$ inversion fragments. Therefore, the total process takes $O(n^2)$ time and requires $O(n)$ space for storing all inversion fragments.

Once we have two set AH_x^i and AT_x^i, we can calculate all agreed pairs generated from AH_x^i and AT_x^i.

STEP-3: Based on Observation 3(1), for every $((p_1, i), \sigma_1\sigma_2)$ (or $((i, i)', \sigma_1\sigma_2)$) in AH_x^i and $((i+1, p_2), \sigma_3)$ (or $((i+1, i+1)', \sigma_3)$) in AT_x^i, we set $C_x^i[\sigma_1][\sigma_2][\sigma_3] = true$. We also add $(i+1, \sigma_1\sigma_2\sigma_3)$ to $S_x^i[2][p_2]$. Since $|AH_x^i| \leq i+1$ and $|AT_x^i| = n-i$, we repeat this step at most $(i+1)(n-i) = O(n^2)$ times. Thus, we can update C_x^i and S_x^i in $O(n^2)$ time.

T_x	1	2
1	$((1,1)', A)$	$((1,2), T)$
2	$((1,1), T)$	$((2,2)', G)$
3	$((1,2), C)$	$((2,2), C)$
4	$((1,3), G)$	$((2,3), G)$
5	$((1,4), A)$	$((2,4), A)$

S_x^1	1	2
1	$(1, AAA)$	
2	$(1, AAT)$	$(2, AAG), (2, ATG)$
3	$(1, AAC)$	$(2, AAC), (2, ATC)$
4	$(1, AAG)$	$(2, AAG), (2, ATG)$
5	$(1, AAA)$	$(2, AAA), (2, ATA)$

Fig. 3. An example for S_x^1 after **STEP-3**. Shaded cells in the first column of T_x are AH_x^1 and shaded cells in the second column of T_x are AT_x^1.

We have calculated all agreed pairs generated from AH_x^i and AT_x^i in **STEP-3**, and the other case of generating agreed pairs for an index i is to use the inversion fragments with the same center from BH_x^i and BT_x^i. Note that inversion fragments with the same center means the same inversion by Observation 1.

STEP-4: Based on Observation 3(2), for two elements $((p_1, p_2), \sigma_2) = T_x[i][j]$ and $((q_1, q_2), \sigma_3) = T_x[i+1][k]$, where $p_1 + p_2 = q_1 + q_2$ and $i \neq j$ and $i + 1 \neq k$, if $(t, \sigma_0\sigma_1\sigma_2) \in S_x^i[1][j]$ and $t \leq p_1$, then we set $C_x^i[\sigma_1][\sigma_2][\sigma_3] = true$ and add $(t, \sigma_1\sigma_2\sigma_3)$ to $S_x^i[2][k]$. For an inversion fragment in the ith column of T_x, we can find the inversion fragment in the i+1th column with the same center in the constant time. Since there are $O(n)$ inversion fragments in ith column of T_x and for each inversion fragment we need to examine $O(n)$ elements in S_x^i, the whole process takes $O(n^2)$ time.

T_x	1	2
1	$((1,1)', A)$	$((1,2), T)$
2	$((1,1), T)$	$((2,2)', G)$
3	$((1,2), C)$	$((2,2), C)$
4	$((1,3), G)$	$((2,3), G)$
5	$((1,4), A)$	$((2,4), A)$

S_x^1	1	2
1	$(1, AAA)$	$(1, ACT)$
2	$(1, AAT)$	$(2, AAG), (2, ATG)$
3	$(1, AAC)$	$(2, AAC), (2, ATC), (1, AGC)$
4	$(1, AAG)$	$(2, AAG), (2, ATG), (1, AAG)$
5	$(1, AAA)$	$(2, AAA), (2, ATA)$

Fig. 4. An example for S_x^1 after **STEP-4**. Shaded cells in T_x generates elements for S_x^1.

STEP-5: For all three letter strings $\sigma_1\sigma_2\sigma_3$ over Σ,

$$C_x^i[\sigma_1][\sigma_2][\sigma_3] = \begin{cases} true & \text{if } C_x^i[\sigma_1][\sigma_2][\sigma_3] = true \text{ and } C_y^i[\sigma_1][\sigma_2][\sigma_3] = true, \\ false & \text{otherwise.} \end{cases}$$

Once we recompute C_x^i, for each $(p, \sigma_1\sigma_2\sigma_3)$ in S_x, we remove $(p, \sigma_1\sigma_2\sigma_3)$ from S_x^i if $C_x^i[\sigma_1][\sigma_2][\sigma_3] = false$. The process ensures that S_x^i and S_y^i produce the same sequence of characters by connected pairs. Since the size of S_x^i is $O(n^2)$ and the size of C_x^i is constant, this step takes $O(n^2)$ time.

Algorithm 1

Input: Strings x and y
Output: Boolean (whether or not there exist Θ_x and Θ_y s.t. $\Theta_x(x) = \Theta_y(y)$.)
`/* time complexity:` $O(n^3)$`, space complexity:` $O(n^2)$ `*/`

make T_x and T_y.
initialize S_x^1 and S_y^1.
for $i \leftarrow 1$ **to** $n-1$ **do**
  for *strings* x *and* y **do**
    for $j \leftarrow 1$ **to** $i+1$ **do** `// STEP-1`
      σ_2 is yielded from $T_x[i][j]$
      if $(j, \sigma_0\sigma_1\sigma_2) \in S_x^i[1][j]$ **then** $((j,i), \sigma_1\sigma_2) \in AH_x^i$
    for $j \leftarrow i+1$ **to** $n+1$ **do** `// STEP-2`
      $T_x[i][j] \in AT_x^i$
    for each $((p_1,i), \sigma_1\sigma_2) \in AH_x^i$ *and* $((i+1,p_2),\sigma_3) \in AT_x^i$ **do** `// STEP-3`
      $C_x^i[\sigma_1][\sigma_2][\sigma_3] = true$
      $(i+1, \sigma_1\sigma_2\sigma_3) \in S_x^i[2][p_2]$
    for $j \leftarrow 1$ **to** $n+1$ **except** $min(i+1,1), i$ **do** `// STEP-4`
      if $j = i+1 \vee j = i+2$ **then**
        $k \leftarrow j-2$
      else
        $k \leftarrow j-1$
      $((p_1,p_2),\sigma_2) \leftarrow T_x[i][j], ((q_1,q_2),\sigma_3) \leftarrow T_x[i+1][k]$
      if $(t, \sigma_0\sigma_1\sigma_2) \in S_x^i[1][j] \wedge t \leq p_1$ **then**
        $C_x^i[\sigma_1][\sigma_2][\sigma_3] = true$
        $(r, \sigma_1\sigma_2\sigma_3) \in S_x^i[2][k]$
  $C_x^i, C_y^i \leftarrow C_x^i \wedge C_y^i$ `// STEP-5`
  for *strings* x *and* y **do**
    for each $(p, \sigma_1\sigma_2\sigma_3) \in S_x^i$ **do**
      if $C_x^i[\sigma_1][\sigma_2][\sigma_3] = false$ **then** remove $(p, \sigma_1\sigma_2\sigma_3)$ from S_x^i
  copy the second columns of S_x^i and S_y^i to the first column of S_x^{i+1} and S_y^{i+1}.
if *the second columns of* S_x^{n-1} *and* S_y^{n-1} *are not empty* **then**
  return true
else
  return false

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

Once we finish calculating S_x^i and S_y^i using **STEPS-1,2,3,4** and **5** from index 1 to $n-1$, we check whether or not the second columns in S_x^{n-1} and S_y^{n-1} are empty. If they are not empty, then there exist agreed sequences for x and y that generate the same string, which are legal sequences. On the other hand, if they are empty, then there are no legal sequences for x and y.

T_x	1	2
1	$((1,1)', A)$	$((1,2), T)$
2	$((1,1), T)$	$((2,2)', G)$
3	$((1,2), C)$	$((2,2), C)$
4	$((1,3), G)$	$((2,3), G)$
5	$((1,4), A)$	$((2,4), A)$

S_x^1	1	2
1	$(1, AAA)$	$(1, ACT)$
2	$(1, AAT)$	
3	$(1, AAC)$	$(2, ATC), (1, AGC)$
4	$(1, AAG)$	
5	$(1, AAA)$	

T_y	1	2
1	$((1,1)', C)$	$((1,2), G)$
2	$((1,1), G)$	$((2,2)', G)$
3	$((1,2), C)$	$((2,2), C)$
4	$((1,3), T)$	$((2,3), T)$
5	$((1,4), T)$	$((2,4), T)$

S_y^1	1	2
1	$(1, AAC)$	
2	$(1, AAG)$	
3	$(1, AAC)$	$(2, AGC), (1, ATC)$
4	$(1, AAT)$	$(2, ACT)$
5	$(1, AAT)$	$(2, ACT)$

Fig. 5. An example for S_x^1 and S_y^1 after **STEP-5**. Note that the second column of S_x^1 and S_y^1 generate same substrings, ACT, ATC and AGC.

T_x	3	4
1	$((1,3), T)$	$((1,4), T)$
2	$((2,3), C)$	$((2,4), C)$
3	$((3,3)', C)$	$((3,4), G)$
4	$((3,3), G)$	$((4,4)', T)$
5	$((3,4), A)$	$((4,4), A)$

S_x^3	1	2
1	$(1, GCT)$	
2		
3	$(3, CTC)$	
4	$(3, TCG)$	$(4, CTT), (4, TCT), (4, CGT)$
5		$(4, CTA), (4, TCA), (4, CGA)$

Fig. 6. An example for S_x^3 after **STEP-5**. Since the second column of S_x^3 (and S_y^3) is not empty, the algorithm returns true.

Theorem 4. *The proposed algorithm runs in $O(n^3)$ time using $O(n^2)$ space, where $n = |x| = |y|$.*

Lemmas 1 and 2 guarantee the correctness of our algorithm.

Lemma 1. *If $(t, \sigma_1\sigma_2\sigma_3) \in S_x^i[2][j]$ after completing* **STEP-5**, *then there exists $(t', \sigma_0\sigma_1\sigma_2) \in S_x^{i-1}[2][k]$ after completing* **STEP-5**.

Lemma 2. *If there exists a string $s = \sigma_0\sigma_1 \cdots \sigma_n$ such that $S_x^i[2][j_i] = (t_i, \sigma_{i-1}\sigma_i\sigma_{i+1})$, then there exists a sequence $\mathbb{S}$ of inversion fragments whose ith element $\mathbb{S}[i]$ is*

$$\mathbb{S}[i] = T_x[i][k], \textit{where } \begin{cases} k \in \{1,2\} \textit{ and } T_x[1][k] \textit{ yields } \sigma_1 & \textit{if } i = 1 \textit{ and } t_1 = 2, \\ T_x[i][k] = ((1, p+q-1), \sigma_1) & \textit{if } i = 1 \textit{ and } t_1 = 1, \\ k = j_i & \textit{otherwise.} \end{cases}$$

Then, $\mathbb{S}$ is an agreed sequence.

Theorem 5. *We can solve the alignment with non-overlapping inversion problem in $O(n^3)$ time using $O(n^2)$ space, where n is the size of input strings.*

Algorithm 2

Input: Strings x and s of length n
Output: Agreed sequence $\mathbb{S}_x$ for x such that $\mathbb{S}_x$ yields s

```
1  t ← 1                          // t is the length of the comparing substrings
2  for i ← 1 to n do
       // comparing substrings created from identity function
3      if t = 1 ∧ θ(x)[n + 1 − i] = θ(s[i]) then
4          add ((i, i)′, s[i]) to S_x
       // comparing substrings created from inversion
5      else if θ(x)_(n+1−i,n+t−i) = s_(i−t+1,i) then
6          for j ← 1 to ⌊t/2⌋ do
7              add ((i − t + j, i + 1 − j), s[i − t + j]) to S_x
8          for j ← ⌈t/2⌉ to t do
9              add ((i + 1 − j, i − t + j), s[i − t + j]) to S_x
10         t ← 1
11     else
12         t ← t + 1
13 return S_x
```

Next, we consider the problem of retrieving an alignment when we know that there exist two non-overlapping inversions for x and y. Note that **Algorithm 1** determines the existence of Θ_x and Θ_y.

Definition 3. *We define the alignment finding problem with non-overlapping inversions on two strings as follows: Given two strings x and y of the same length, find two sets Θ_x and Θ_y of inversions such that $\Theta_x(x) = \Theta_y(y)$.*

We tackle the problem in Definition 3 by retrieving the common string s such that $s = \Theta_x(x) = \Theta_y(y)$. After completing **STEP-5** for each i, we store every $\sigma_1\sigma_2\sigma_3$ to a set F^i, where $C_x^i[\sigma_1][\sigma_2][\sigma_3] = true$.

Observation 6. *For such sets F^i's, we have the following two observations:*

1. *The space requirement for F^i's is $O(n)$,*
2. *For any string s, where $s_{(i-2,i)} \in F^i$, for $3 \leq i \leq n$, there exist Θ_x and Θ_y such that $s = \Theta_x(x) = \Theta_y(y)$.*

Due to Observation 6, the problem becomes to find Θ_x for x and s such that $\Theta_x(x) = s$ (and Θ_y for y). **Algorithm 2** retrieves $\mathbb{S}_x$ equivalent to Θ_x from x and s. The idea of the algorithm is that every substring generated by an inversion on x is a substring of $\theta(x)$, and substrings do not overlap with each other on $\theta(x)$ if inversions do not overlap. Fig. 7 illustrates this idea.

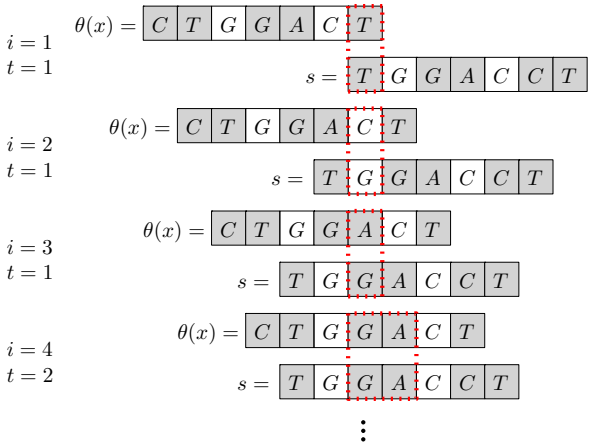

Fig. 7. An example of comparing $\theta(x)$ and s, where $x = AGTCCAG$ and $s = TGGACCT$. Dotted boxes are compared substrings in each i. In $\theta(x)$, substrings matched by inversions are same as substrings in s, and substrings matched by identity functions are complements of substrings in s. When $i = 3$, since $A \neq G$ and C, t is increased and the algorithm compares GA and GA when $i = 4$.

Theorem 7. *Once we solve the alignment with non-overlapping inversion problem, we can solve the alignment finding problem in $O(n^2)$ using additional $O(n)$ space.*

4 Conclusions

The inversion is an important operation for bio sequences such as DNA or RNA and is closely related to mutations. We have, in particular, considered non-overlapping inversions on both sequences, which is important to find the original common sequence from two mutated sequences. We have proposed a new problem, alignment with non-overlapping inversions on two strings, and presented a polynomial algorithm for the problem. Given two strings x and y, based on the inversion properties, our algorithm decides whether or not there exist two sets Θ_x and Θ_y of inversions for x and y such that $\Theta_x(x) = \Theta_y(y)$ in $O(n^3)$ time using $O(n^2)$ space, where $n = |x| = |y|$. Once we know the existence of Θ_x and Θ_y, we can retrieve Θ_x and Θ_y in $O(n^2)$ time using additional $O(n)$ 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 on both strings. The proposed problem is about the sequence alignment and can be extended to approximate pattern matching or edit distance problem.

References

1. Cantone, D., Cristofaro, S., Faro, S.: Efficient string-matching allowing for non-overlapping inversions. Theoretical Computer Science 483, 85–95 (2013)
2. Cantone, D., Faro, S., Giaquinta, E.: Approximate string matching allowing for inversions and translocations. In: Proceedings of the Prague Stringology Conference 2010, pp. 37–51 (2010)
3. Chen, Z.-Z., Gao, Y., Lin, G., Niewiadomski, R., Wang, Y., Wu, J.: A space-efficient algorithm for sequence alignment with inversions and reversals. Theoretical Computer Science 325(3), 361–372 (2004)
4. Grabowski, S., Faro, S., Giaquinta, E.: String matching with inversions and translocations in linear average time (most of the time). Information Processing Letters 111(11), 516–520 (2011)
5. Ignatova, Z., Zimmermann, K., Martinez-Perez, I.: DNA Computing Models. Advances in Information Security (2008)
6. Kececioglu, J.D., Sankoff, D.: Exact and approximation algorithms for the inversion distance between two chromosomes. In: Apostolico, A., Crochemore, M., Galil, Z., Manber, U. (eds.) CPM 1993. LNCS, vol. 684, pp. 87–105. Springer, Heidelberg (1993)
7. Li, S.C., Ng, Y.K.: On protein structure alignment under distance constraint. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 65–76. Springer, Heidelberg (2009)
8. Lipsky, O., Porat, B., Porat, E., Shalom, B.R., Tzur, A.: Approximate string matching with swap and mismatch. In: Tokuyama, T. (ed.) ISAAC 2007. LNCS, vol. 4835, pp. 869–880. Springer, Heidelberg (2007)
9. Ogilvie, C.M., Scriven, P.N.: Meiotic outcomes in reciprocal translocation carriers ascertained in 3-day human embryos. European Journal of Human Genetics 10(12), 801–806 (2009)
10. Oliver-Bonet, M., Navarro, J., Carrera, M., Egozcue, J., Benet, J.: Aneuploid and unbalanced sperm in two translocation carriers: evaluation of the genetic risk. Molecular Human Reproduction 8(10), 958–963 (2002)
11. Painter, T.S.: A New Method for the Study of Chromosome Rearrangements and the Plotting of Chromosome Maps. Science 78, 585–586 (1933)
12. Sakai, Y.: A new algorithm for the characteristic string problem under loose similarity criteria. In: Asano, T., Nakano, S.-i., Okamoto, Y., Watanabe, O. (eds.) ISAAC 2011. LNCS, vol. 7074, pp. 663–672. Springer, Heidelberg (2011)
13. Schöniger, M., Waterman, M.S.: A local algorithm for DNA sequence alignment with inversions. Bulletin of Mathematical Biology 54(4), 521–536 (1992)
14. Vellozo, A.F., Alves, C.E.R., do Lago, A.P.: Alignment with non-overlapping inversions in $O(n^3)$-time. In: Bücher, P., Moret, B.M.E. (eds.) WABI 2006. LNCS (LNBI), vol. 4175, pp. 186–196. Springer, Heidelberg (2006)