Alignment of Sequences Allowing for Non-overlapping Unbalanced Translocations of Adjacent Factors

Simone Faro[1(✉)] and Arianna Pavone[2]

[1] Dipartimento di Matematica e Informatica, Università di Catania, Viale Andrea Doria 6, 95125 Catania, Italy
faro@dmi.unict.it

[2] Dipartimento di Scienze Cognitive, Università di Messina, Via Concezione 6, 98122 Messina, Italy
apavone@unime.it

Abstract. *Unbalanced translocations* take place when two unequal chromosome sub-sequences swap, resulting in an altered genetic sequence. Such large-scale gene modification are among the most frequent chromosomal alterations, accounted for 30% of all losses of heterozygosity. However, despite of their central role in genomic sequence analysis, little attention has been devoted to the problem of aligning sequences allowing for this kind of modification.

In this paper we investigate the sequence alignment problem when the edit operations are non-overlapping unbalanced translocations of adjacent factors.

Specifically, we present an alignment algorithm for the problem working in $\mathcal{O}(m^3)$-time and $\mathcal{O}(m^3)$-space, where m is the length of the involved sequences. To the best of our knowledge this is the first solution in literature for the alignment problem allowing for unbalanced translocations of factors.

Keywords: Sequence alignment · Unbalanced translocations · DNA sequence analysis · Text processing

1 Introduction

Retrieving information and teasing out the meaning of biological sequences are central problems in modern biology. Generally, basic biological information is stored in strings of nucleic acids (DNA, RNA) or amino acids (proteins).

In recent years, much work has been devoted to the development of efficient methods for aligning strings and, despite sequence alignment seems to be a well-understood problem (especially in the edit-distance model), the same cannot be said for the approximate string matching problem on biological sequences.

String alignment and *approximate string matching* are two fundamental problems in text processing. Given two input sequences x, of length m, and y,

I. Rojas et al. (Eds.): IWBBIO 2019, LNBI 11465, pp. 120–131, 2019.
https://doi.org/10.1007/978-3-030-17938-0_12

of length n, the *string alignment* problem consists in finding a set of edit operations able to transform x in y, while the *approximate string matching* problem consists in finding all approximate matches of x in y. The closeness of a match is measured in terms of the sum of the costs of the elementary edit operations necessary to convert the string into an exact match.

Most biological string matching methods are based on the *Levenshtein distance* [11], commonly referred to just as *edit distance*, or on the *Damerau distance* [8], which assume that changes between strings occur locally, i.e., only a small portion of the string is involved in the mutation event. However, evidence shows that in some cases large scale changes are possible [6,7,16] and that such mutations are crucial in DNA since they often cause genetic diseases [10,14]. For example, large pieces of DNA can be moved from one location to another (*translocations*) [6,13,17,18], or replaced by their reversed complements (*inversions*) [1–4].

Translocations can be *balanced* (when equal length pieces are swapped) or *unbalanced* (when pieces with different lengths are moved). Interestingly, unbalanced translocations are a relatively common type of mutation and a major contributor to neurodevelopmental disorders [18]. In addition, cytogenetic studies have also indicated that unbalanced translocations can be found in human genome with a de novo frequency of 1 in 2000 [17] and that it is a frequent chromosome alteration in a variety of human cancers [13]. Hence the need for practical and efficient methods for detecting and locating such kind of large scale mutations in biological sequences.

In the last three decades much work has been made for the alignment and matching problem allowing for chromosomal alteration, especially for non overlapping inversions. Concerning the alignment problem with inversions, a first solution based on dynamic programming, was proposed by Schöniger and Waterman [15], which runs in $\mathcal{O}(n^2m^2)$-time and $\mathcal{O}(n^2m^2)$-space on input sequences of length n and m. Several other papers have been devoted to the alignment problem with inversions. The best solution is due to Vellozo *et al.* [16], who proposed a $\mathcal{O}(nm^2)$-time and $\mathcal{O}(nm)$-space algorithm, within the more general framework of an edit graph.

Regarding the alignment problem with translocations, Cho *et al.* [6] presented a first solution for the case of inversions and translocations of equal length factors (i.e., balanced translocations), working in $\mathcal{O}(m^3)$-time and $\mathcal{O}(m^2)$-space. However their solution generalizes the problem to the case where edit operations can occur on both strings and assume that the input sequences have the same length, namely $|x| = |y| = m$.

In this paper we investigate the alignment problem under a string distance whose edit operations are non-overlapping unbalanced translocations of adjacent factors. To the best of our knowledge, this slightly more general problem has never been addressed in the context of alignment on biological sequences. A related result has been very recently introduced by Cantone et al. [5] who presented a $\mathcal{O}(nm^3)$-time and $\mathcal{O}(m^2)$-space algorithm for the approximate string matching problem with unbalanced translocations based on the dynamic-

programming approach, where n is the length of the text and m is the length of the pattern. They also improved their solution by making use of the Directed Acyclic Word Graph of the pattern achieving a $\mathcal{O}(n \log^2 m)$ average time complexity still maintaining the same worst case time complexity.

In this paper we present an alignment algorithm for the same problem working in $\mathcal{O}(m^3)$ worst case time and $\mathcal{O}(m^3)$-space. Given two input equal length sequences x and y, our algorithm is able to establish if x can be transformed in y by way of unbalanced translocations of adjacent factors.

The rest of the paper is organized as follows. In Sect. 2 we introduce some preliminary notions and definitions. Subsequently, in Sect. 3 we present our alignment algorithm running in $\mathcal{O}(m^3)$-time, discussing its worst case time complexity. Finally draw our conclusions in Sect. 4.

2 Basic Notions and Definitions

A string x of length $m \geq 0$, over an alphabet Σ, is represented as a finite array $x[1\,..\,m]$ of elements of Σ. We write $|x| = m$ to indicate its length. In particular, when $m = 0$ we have the empty string ε. We denote by $x[i]$ the i-th character of x, for $1 \leq i \leq m$. Likewise, the substring of x factor, contained between the i-th and the j-th characters of x is indicated with $x[i\,..\,j]$, for $1 \leq i \leq j \leq m$. We assume that $x[i\,..\,j] = \varepsilon$ when $x > y$.

A string $w \in \Sigma^*$ is a suffix of x (in symbols, $w \sqsupseteq x$) if $w = x[i\,..\,m]$, for some $1 \leq i \leq m$. Similarly, we say that w is a prefix of x (in symbols, $w \sqsubseteq x$) if $w = x[1\,..\,i]$, for some $1 \leq i \leq m$. Additionally, we use the symbol x_i to denote the prefix of x of length i (i.e., $x_i = x[1\,..\,i]$), for $1 \leq i \leq m$, and make the convention that x_0 denotes the empty string ε. In addition, we write $x.w$ for the concatenation of the strings x and w.

A string w is a *border* of x if both $w \sqsubseteq x$ and $w \sqsupseteq x$ hold. The set of the borders of x is denoted by $borders(p)$. For instance, given the string $x = atacgata$, we have that $at \sqsubseteq p$, $gata \sqsubseteq p$, while ata is a border of x. Moreover we have $borders(p) = \{a, ata\}$.

The border set of a string x of length m can be efficiently computed in $\mathcal{O}(m)$-time by using the *border-table* of x, introduced for the first time in the well known Morris-Pratt algorithm [9,12]. Formally the border table of x is a function $\pi : \{1, \ldots, m\} \rightarrow \{0, 1, \ldots, m-1\}$ such that $\pi(i)$ is the length of the longest proper prefix of $x[1..i]$ that is also a suffix of $x[1..i]$.

A *distance* $d : \Sigma^* \times \Sigma^* \rightarrow \mathbb{R}$ is a function which associates to any pair of strings x and y the minimal cost of any finite sequence of edit operations which transforms x into y, if such a sequence exists, ∞ otherwise.

In this paper we consider the *unbalanced translocation distance*, $utd(x, y)$, whose unique edit operation is the *translocation* of two adjacent factors of the string, with possibly different lengths. Specifically, in an *unbalanced translocation* a factor of the form zw is transformed into wz, provided that both $|z|, |w| > 0$ (it is not necessary that $|z| = |w|$). We assign a unit cost to each translocation.

Example 1. Let $x = g\bar{t}g\overline{accgt}\overline{ccag}$ and $y = gg\overline{at}\overline{cccag}\overline{cgt}$ be given two strings of length 12. Then $utd(x, y) = 2$ since x can be transformed into y by translocating the substrings $x[3..4] = ga$ and $x[2..2] = t$, and translocating the substrings $x[6..8] = cgt$ and $x[9..12] = ccag$.

When $utd(x, y) < \infty$, we say that x and y have utd-match. If x has utd-match with a suffix of y, we write $x \overset{utd}{\sqsupseteq} y$.

3 A New *utd*-Alignment Algorithm

In this section we present a new solution for the string alignment problem allowing for unbalanced translocations of adjacent factors. In the next sections we start by describing the algorithm, named Unbalanced-Translocations-Align (shown in Fig. 2), used for checking whenever an alignment exists between two equal length strings x and y.

The corresponding approximate string matching algorithm allowing for unbalanced translocations of adjacent factors can be trivially obtained by iterating the given procedure for all possible subsequences of the text of length $|x|$.

Our alignment algorithm is composed by a preprocessing and searching phase, which we describe in Sect. 3.1 and in Sect. 3.2, respectively. Then, in Sect. 3.3, we prove the correctness of the algorithm and discuss its worst case time complexity.

3.1 The Preprocessing Phase

During the preprocessing phase of procedure Unbalanced-Translocations-Align three functions are computed, in the form of tables, which will be then used during the alignment process.

We first define the *next position function* $\mu_x : \Sigma \times \{1, \ldots, m\} \to \{2, \ldots, m\}$, associated to a given pattern x of length m, as the function which returns the next position (to a given input position i) where a given character $c \in \Sigma$ occurs. Specifically $\mu_x(c, i)$ is defined as the position $j > i$ in the pattern such that $x[j] = c$. If such a position does not exist then we set $\mu(c.i) = m + 1$. More formally

$$\mu_x(c, i) := \min\Big(\{j \mid 1 \le i < j \le m \text{ and } x[j] = c\} \cup \{m + 1\}\Big)$$

The next position function μ_x can be precomputed and maintained in a table of size $m \times \sigma$ in $\mathcal{O}(m\sigma + m^2)$ time by using procedure Compute-Next-Position depicted in Fig. 1 (on the left).

Example 2. Let $x = gtgtaccgtgt$ be a string of length $m = 11$. We have $\mu_x(g, 1) = 3$, $\mu_x(g, 4) = \mu_x(g, 5) = 8$, $\mu_x(g, 8) = 10$, while $\mu_x(g, 10) = 12$.

```
COMPUTE-NEXT-POSITION
1. foreach c ∈ Σ do
2.       for i ← 1 to m do
3.             μ(c, i) ← m + 1
4. for i ← m downto 2 do
5.       for j ← i − 1 downto 1 do
6.             μ(x[i], j) ← i
7. return μ
```

```
COMPUTE-BORDER-SET
1. for i ← 1 to m do
2.       for j ← i to m do
3.             for k ← i to j do
4.                   Ψ[i, j, k] ← 0
5. for i ← 1 to m do
6.       for j ← i to m do
7.             π ← COMPUTE-BORDER-TABLE(x, i, j)
8.             for k ← 0 to j − i + 1 do
9.                   Ψ[i, j, π[k]] ← 1
10. return Ψ
```

Fig. 1. Procedure COMPUTE-NEXT-POSITION (on the left) for computing the *next position function* μ_x which returns the next position (to a given input position i) where a given character $c \in \Sigma$ occurs in x; and procedure COMPUTE-BORDER-SET (on the right) for computing the *border set function* ψ_x as the set of the lengths of all borders of a given string x.

We also define the *border set function* ψ_x of a given string x as the set of the lengths of all borders of x. Specifically we define $\psi_x(i, j)$, for each $1 \leq i < j \leq m$, as the set of the lengths of all borders of the string $x[i..j]$, so that $k \in \psi_x(i, j)$ if and only if the string $x[i..j]$ has a border of length k. Formally we have

$$\psi_x(i, j) := \{k \mid 0 < k < j - i \text{ and } x[i..i + k - 1] = x[j - k + 1..j]\}$$

Example 3. Let $x = gtgtaccgtgt$ be a string of length $m = 11$. We have

$\psi_x(1, 11) = \{2, 4, 11\}$, since the set of borders of $gtgtaccgtgt$ is $\{gt, gtgt, gtgtaccgtgt\}$;
$\psi_x(1, 4) = \{2, 4\}$, since the set of borders of $gtgt$ is$\{gt, gtgt\}$;
$\psi_x(4, 9) = \{1, 6\}$, since the set of borders of $taccgt$ is$\{t, taccgt\}$;
$\psi_x(5, 7) = \{3\}$ since the set of borders of acc is$\{acc\}$.

For each i, j with $1 \leq i \leq j \leq m$, we can represent the set $\psi_x(i, j)$ by using a vector of $(j - i + 1)$ boolean values such that its k-th entry is set iff $k \in \psi_x(i, j)$. More formally the function ψ_x can be maintained using a tridimensional bit-table Ψ_x, which we call *border set table of* x, defined as

$$\Psi_x[i, j, k] := \begin{cases} 1 & \text{if } x[i..i + k - 1] = x[j - k + 1..j] \\ 0 & \text{otherwise} \end{cases}$$

for $1 \leq i < j \leq m$ and $k < j - i$.

The border set table Ψ_x can be computed in $\mathcal{O}(m^3)$-time and space by using procedure COMPUTE-BORDER-SET-FUNCTION depicted in Fig. 1 (on the right), where COMPUTE-BORDER-TABLE is the $O(m)$ function used in the Knuth-Morris-Pratt algorithm [9].

Observe that using Ψ_x we can answer in costant-time to queries of the the type "*is k the length of a border of the substring $x[i..j]$?*", which translates to evaluate if $\Psi[i, j, k]$ is set.

In addition we define the *shortest border function* of a string x, as the function $\delta_x : \{1,\ldots,m\} \times \{1,\ldots,m\} \to \{1,\ldots,m\}$ which associates any nonempty substring of x to the length of its shortest border. Specifically we set $\delta_x(i,j)$ to be the length of the shortest border of the string $x[i\ldots j]$, for $1 \le i < j \le m$. More formally we have

$$\delta_x(i,j) := \min\{k \mid 0 \le k < j-i \text{ and } x[i..i+k-1] = x[j-k+1..j]\} = \min(\psi(i,j))$$

It is trivial to observe that, if we already computed the border set function ψ_x, for the pattern x, the shortest border function δ_x of x can be computed in $\mathcal{O}(m^3)$-time using $\mathcal{O}(m^2)$ space.

Example 4. Let $x = gtgtaccgtgt$ be a string of length $m = 11$. According to Example 3, we have

$\delta_x(1,11) = 2$, since gt is the shortest nonempty border of $gtgtaccgtgt$;
$\delta_x(1,4) = 2$, since gt is the shortest nonempty border of $gtgt$;
$\delta_x(4,9) = 1$, since t is the shortest nonempty border of $taccgt$;
$\delta_x(5,7) = 3$ since acc is the shortest nonempty border of acc.

In what follows we will use the symbols μ, ψ and δ, in place of μ_x, ψ_x and δ_x, respectively, when the reference to x is clear from the context.

3.2 The Searching Phase

The proposed alignment procedure finds a possible *utd*-match between two equal length strings. The pseudocode of our algorithm, Unbalanced-Translocations-Align(y,x,m), is presented in Fig. 2 and is tuned to process two strings x and y, of length m, where translocations can take place only in x.

In order to probe the details of the alignment procedure, let x and y be two strings of length m over the same alphabet Σ. The procedure sequentially reads all characters of the string y, proceeding from left to right and. While scanning it tries to evaluate all possible unbalanced translocations in x which may be involved in the alignment between the two strings.

We define a *translocation attempt* at position i of y, for $1 \le i \le m$, as a quadruple of indexes, (s_1, k_1, s_2, k_2), with all elements in $\{0,1,2,\ldots,m\} \cup \{\mathsf{null}\}$ and where, referring to the string x, s_1 and k_1 pinpoints the leftmost position and the length of the first factor (the factor moved on the left), while s_2 and k_2 pinpoints the leftmost position and the length of the second factor (the factor moved on the right). In this context we refer to s_1 and s_2 as the *key positions* of the translocation attempt. In the special case where no translocation takes place in the attempt we assume by convention that $s_1 = i$ and $s_2 = k_1 = k_2 = \mathsf{null}$[1]. During the execution of the algorithm for each translocation attempt, (s_1, k_1, s_2, k_2), at position i, the invariant given by the following lemma[2] holds.

[1] We use the value null to indicate the length of an undefined string in order to discriminate it from the length of an empty string whose value is 0 by definition.
[2] In this context we assume that $s + \mathsf{null} = s$, for any s.

```
UNBALANCED-TRANSLOCATIONS-ALIGN(y,x,m)
 1. Γ^(0) ← {(0, 0,null,null)}
 2. fo i ← 1 to m do
 3.    for each (s1, k1, s2, k2) ∈ Γ^(i−1) do
 4.       if (k2 = null) then                                          Δ Case 1
 5.          if (x[i] = y[i]) then
 6.             Γ^(i) ← Γ^(i) ∪ {(i, null, null, null)}
 7.          r ← μ(y[i], i)
 8.          while (r ≤ m) do
 9.             Γ^(i) ← Γ^(i) ∪ {(i − 1, 0, r − 1, 1)}
10.             r ← μ(y[i], r)
11.       else
12.          if (k1 =null) then                                        Δ Case 2
13.             if (x[s2 + k2 + 1] = y[i]) then
14.                Γ^(i) ← Γ^(i) ∪ {(s1, k1, s2, k2 + 1)}
15.             else k1 = 0
16.          if (k1 ≥ 0) then
17.             while (s1 + k1 < s2 and k2 > 0 and x[s1 + k1 + 1] ≠ y[i]) do   Δ Case 3.b
18.                b ← 0
19.                do b ← b + δ(s1 + 1, s2 + k2 − b)
20.                while (s1 + k1 + b < s2 and k2 − b > 0 and (k1 − s1 + 1) ∉ φ(s1 + 1, k1 + b))
21.                k2 ← k2 − b
22.                k1 ← k1 + b
23.             if (s1 + k1 ≥ s2 or k2 ≤ 0) then break
24.             if (x[s1 + k1 + 1] = y[i]) then                       Δ Case 3.a
25.                if (s1 + k1 = s2 and (i, null, null, null) ∉ Γ^(i) ) then
26.                   Γ^(i) ← Γ^(i) ∪ {(i, null, null, null)}
27.                else
28.                   Γ^(i) ← Γ^(i) ∪ {(s1, k1 + 1, s2, k2)}
29. if (Γ^(m) ≠ ∅) then
30.    return true
31. return false
```

Fig. 2. The pseudocode of the UNBALANCED-TRANSLOCATIONS-ALIGN(y,x,m) for the sequence alignment problem allowing for unbalanced translocations of adjacent factors.

Lemma 1. *Let y and x be two strings of length m over the same alphabet Σ. Let $\Gamma^{(i)}$ be the set of all translocation attempts computed by procedure* UNBALANCED-TRANSLOCATIONS-ALIGN *during the i-th iteration. If $(s_1, k_1, s_2, k_2) \in \Gamma^{(i)}$ then it holds that:*

(a) $i = s_1 + k_1 + k_2$*;*
(b) $x_i \stackrel{utd}{=} y_i$*;*
(c) if $s_2 \neq$ null *then* $x[s_1 + 1..s_1 + k_1] = y[s_2 + 1..s_2 + k_1]$*;*
(d) if $s_2 \neq$ null *then* $y[s_2 + 1..s_2 + k_2] = y[s_1 + 1..s_1 + k_2]$*;*

□

For each $1 \leq i \leq m$, we define $\Gamma^{(i)}$ as the set of all translocation attempts tried for the prefix $y[1..i]$, and set $\Gamma^{(0)} = \{(0, \text{null}, \text{null}, \text{null})\}$.

However we can prove that it is not necessary to process all possible translocation attempts. Some of them, indeed, leads to detect the same *utd*-matches so that they can be skipped.

Lemma 2. *Let y and x be two strings of length m over the same alphabet Σ. Let $s \sqsubseteq y$ and $u \sqsubseteq x$ such that $|s| = |x|$ and $s \overset{utd}{=} u$. Moreover assume that*

(i) $s.w.z \sqsubseteq y$ *and* $u.z.w \sqsubseteq x$
(ii) $s.w'.z \sqsubseteq y$ *and* $u.z.w' \sqsubseteq x$

with $|z| > 0$ and $|w'| > |w| > 0$. If we set $i = |s.w.z|$ and $j = |s.w'.z|$ then we have $x[i+1..j] \overset{utd}{=} y[i+1..j]$. □

The procedure iterates on the values of i, for $1 \leq i \leq m$, while scanning the characters of y, and during the i-th iteration it computes the set $\Gamma^{(i)}$ from $\Gamma^{(i-1)}$. For each translocation attempt $(s_1, k_1, s_2, k_2) \in \Gamma^{(i-1)}$ we distinguish the following three cases (depicted in Fig. 3):

- Case 1 ($s_2 = k_1 = k_2 = \mathsf{null}$)
  This is the case where no unbalanced translocation is taking place (line 5). Thus we simply know that $x_{i-1} \overset{utd}{=} y_{i-1}$. If $x[i] = y[i]$ the match is extended of one character by adding the attempt $(s_1 + 1, \mathsf{null}, \mathsf{null}, \mathsf{null})$ to Γ^i (line 7). Alternatively, when possible, new translocation attempts are started (lines 9–12). Specifically for each occurrence of the character $y[i]$ in x, at a position r next to s_1, a new right factor u_r is attempted starting at position r (line 10) by extending $\Gamma^{(i)}$ with the attempt $(s_1, 0, r-1, 1)$.
- Case 2 ($k_1 = 0$ and $k_2 > 0$)
  This is the case where an unbalanced translocation is taking place and the right factor u_r is currently going to be recognized (line 14). Specifically we know that $x[s_2+1..s_2+k_2] = y[i-k_2..i-1]$ and that $x[1..s_1] \overset{utd}{=} y[1..i-k_2-1]$. If $x[s_2+k_2+1] = y[i]$ the right factor u_r can be extended of one character to the right, thus Γ^i is extended by adding the attempt (s_1, k_1, s_2, k_2+1) (line 16). Otherwise, if $x[s_2+k_2+1] \neq y[i]$, the right factor u_r cannot be extended further, thus we start recognizing the left factor u_ℓ. Specifically, in this last case, we update k_1 to 0 (line 17) and move to the following Case 3.
- Case 3 ($k_1 \geq 0$)
  This is the case where an unbalanced translocation is taking place, the right factor u_r has been already recognized and we are attempting to recognize the left factor u_ℓ. Specifically we know that $x[s_1+1..s_1+k_1] = y[i-k_1..i-1]$, $x[s_2+1..s_2+k_2] = y[i-k_1-k_2..i-k_1-1]$ and that $x[1..s_1] \overset{utd}{=} y[1..i-k_1-k_2-1]$. We distinguish two sub-cases:
  - Case 3.a ($x[s_1+k_1+1] = y[i]$)
    In this case the left factor u_ℓ can be extended of one character to the right (line 24). Thus if the left factor has been completely recognized, i.e. if $s_1 + k_1 = s_2$, Γ^i is extended by adding the attempt $(s_1+k_1+k_2, \mathsf{null}, \mathsf{null}, \mathsf{null})$ (lines 27–28) which indicates that $x_i \overset{utd}{=} y_i$. Otherwise $\Gamma^{(i)}$ is extended by adding the attempt (s_1, k_1+1, s_2, k_2) (line 30).

- Case 3.b ($x[s_1 + k_1 + 1] \neq y[i]$)
  In this case the right factor u_ℓ cannot be extended. Before quitting the translocation attempt we try to find some new factors rearrangements on the same key positions, s_1 and s_2, but with different lengths, k_1 and k_2. Specifically we try to transfer a suffix w of u_r to the prefix position of u_ℓ, reducing the length k_2 and extending the length k_1 accordingly. This can be done only if we find a suffix w of u_r which is also a prefix of $x[s_1+1..s_2]$ and, in addition, we can move u_ℓ to the right of $|w|$ position along the left factor. More formally, if we assume that $|w| = b$ we must have:
  1. $|w| < |u_r|$, or rather $b < k_2$ (indicating that w is a proper suffix of u_r);
  2. $b \in \phi(s_1+1, s_2+k_2)$ (indicating that w is a border of $x[s_1+1..s_2+k_2]$);
  3. $(k_1 - s_1 + 1) \in \phi(s_1 + 1, k_1 + b)$ (indicating that u_ℓ is a border of $x[s_1 + 1..s_1 + k_1 + |w|]$);
  4. $s_1 + k_1 + |w| < s_2$ (indicating that the updated u_ℓ does not overflow onto u_r);
  5. $x[s_1 + k_1 + |w| + 1] = y[i]$ (indicating that the updated u_ℓ can be extended by $y[i]$).

3.3 Worst-Case Time and Space Analysis

In this section we discuss the worst-case time and space analysis of procedure Unbalanced-Translocations-Align presented in the previous section. In particular, we will refer to the implementation reported in Fig. 2.

Let x and y be two nonempty strings of length $m \geq 1$ over the same alphabet Σ and assume to run procedure Unbalanced-Translocations-Align(y, x, m). Regarding the space analysis, as stated in Sect. 3.1 we need $O(m\sigma)$ to maintain the next position function, $O(m^3)$ to maintain the border set function and $O(m^2)$ to maintain the shortest border function. Thus the overall space complexity of the algorithm is $O(m^3)$.

Regarding the time analysis, let $\Gamma^{(i)}$ be the set of all translocation attempts computed at iteration i, for $0 \leq i \leq m$.

First of all we observe that each $\Gamma^{(i)}$ contains at most one translocation attempt with $k_2 = \mathsf{null}$ (i.e. of the form $(s_1, \mathsf{null}, \mathsf{null}, \mathsf{null})$). We put $\Gamma^{(0)} = \{(0, \mathsf{null}, \mathsf{null}, \mathsf{null})\}$ (line 1), thus the statement holds for $i = 0$. Observe that, if $i > 0$ a translocation attempt of the form $(s_1, \mathsf{null}, \mathsf{null}, \mathsf{null})$ can be added to $\Gamma^{(i)}$ only in line 6 or in line 26. However by condition at line 25, if it is added to $\Gamma^{(i)}$ in line 6, it cannot be added again in line 26.

We now prove that the total number of translocation attempts processed during the execution of the algorithm is bounded by m^2. More formally we have

$$\sum_{i=0}^{m} |\Gamma^{(i)}| \leq m^3. \tag{1}$$

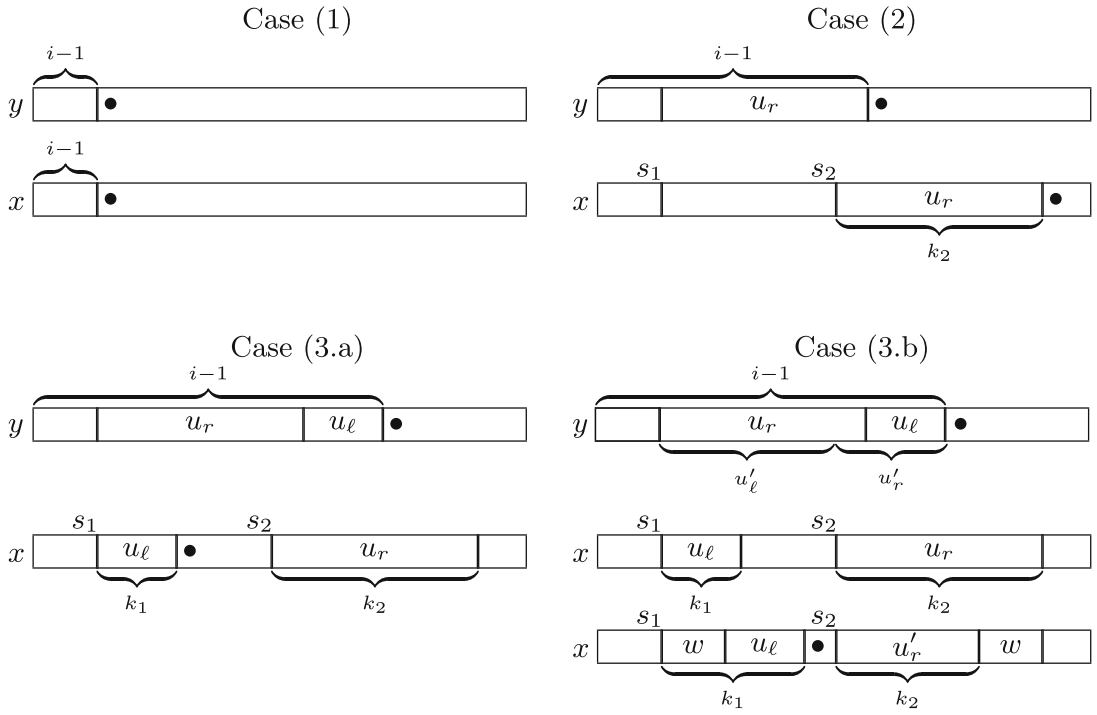

Fig. 3. Three cases of procedure UNBALANCED-TRANSLOCATIONS-ALIGN(y, x, m) while processing the translocation attempt $(s_1, k_1, s_2, k_2) \in \Gamma^{(i-1)}$ in order to extend it by charcater $y[i]$. Character $y[i]$ and its counterpart in x are depicted by a bullet symbol. Case (1): $x_{i-1} \overset{utd}{=} y_{i-1}$ and $x[i] = y[i]$, then the match is extended of one character; Case (2): the right factor u_r is currently going to be recognized and $x[s_2+k_2+1] = y[i]$, then the right factor u_r can be extended of one character; Case (3.a): the left factor u_ℓ can be extended of one character to the right; Case (3.b): the right factor u_ℓ cannot be extended, then we try to transfer a suffix w of u_r to the prefix position of u_ℓ, reducing the length k_2 and extending the length k_1 accordingly (w is a suffix of u_r and also a prefix of $x[s_1+1..s_2]$ and, in addition, we can move u_ℓ to the right of $|w|$ position along the left factor).

To prove that Eq. (1) holds observe that new translocation attempts are added to $\Gamma^{(i)}$ only when we are in Case 1. When we are in Case 2 or in Case 3 a translocation attempt is simply rearranged by extending the right factor (Case 2) or the left factor (Case 3). As observed above only one translocation attempt in $\Gamma^{(i)}$ is in Case 1 and the while cycle of line 8 can add at most $m - i$ new translocation attempts to $\Gamma^{(i+1)}$. In the worst case each translocation attempt added to $\Gamma^{(i+1)}$ will be closed only at iteration m, thus it will be extended along the sets $\Gamma^{(j)}$, for $j > i$. Thus the overall contribute of each translocation attempt added to Γ^{i+1} is $m - i$.

Thus the total number of translocation attempts processed during the execution of the algorithm is bounded by

$$\begin{aligned}\sum_{i=0}^{m} |\Gamma^{(i)}| &\leq 1 + \textstyle\sum_{i=1}^{m} (m-i)^2 \\ &= 1 + \sum_{i=1}^{m} m^2 - \sum_{i=1}^{m} 2im + \sum_{i=1}^{m} i^2 \\ &= 1 + m^3 + \frac{m(m+1)(2m+1)}{6} - \frac{m(m+1)}{2} \\ &= \frac{1}{3}m^3 - \frac{1}{2}m^2 + \frac{1}{3} \leq m^3\end{aligned}$$

Finally we observe that each translocation attempt in Case 2 and Case 3.a is processed in constant time, during the execution of procedure Unbalanced-Translocations-Align. A translocation attempt in Case 1 my be processed in $O(m-i)$ worst case time. However the overall contribution of the while cycle at line 9 is at most $O(m^2)$ since, as observer above, there is a single translocation attempt in Case 1 for each $\Gamma^{(i)}$.

For a translocation attempt in Case 3, observe that at each execution of line 19 the value of b is increased of at most 1. Then in line 21 we decrease k_2 by b. Since the value of k_2 is increased only in line 16, this implies that overall number of times the while cycle of line 22 is executed is bounded by k_2, which is at most m. Thus the overall contribution given by the while cycle of line 19 is $O(m^3)$.

We can conclude that the overall time complexity of procedure Unbalanced-Translocations-Align is $\mathcal{O}(m^3)$.

4 Conclusions and Future Works

We presented a first solution for the alignment problem allowing for unbalanced translocations of adjacent factors working in $\mathcal{O}(m^3)$ worst case time using $\mathcal{O}(m^3)$-space. As suggested in [5], an alternative solution working in $\mathcal{O}(m^3)$ worst case time can be obtained for the same problem by using a standard dynamic programming approach. However our algorithm uses a constructive approach could be more efficient in practice and which can be easily optimized. It turns out, indeed, by our preliminary experimental results (not included in this paper) that our solution has a sub-quadratic behaviour in practical cases. This suggests us to focus our future works on an accurate analysis of the algorithm's complexity in the average case.

In addition we are also planning to modify the given algorithm in order to compute the minimum number of translocations needed to transform x in y and an additional procedure able to retrieve the correct alignment of the two strings. Finally, we wonder if the problem can be solved in sub-cubical worst-case time complexity by extending the result obtained in Lemma 2 with additional restrictions.

References

1. Cantone, D., Cristofaro, S., Faro, S.: Efficient matching of biological sequences allowing for non-overlapping inversions. In: Giancarlo, R., Manzini, G. (eds.) CPM 2011. LNCS, vol. 6661, pp. 364–375. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21458-5_31
2. Cantone, D., Cristofaro, S., Faro, S.: Efficient string-matching allowing for non-overlapping inversions. Theor. Comput. Sci. **483**, 85–95 (2013)
3. Cantone, D., Faro, S., Giaquinta, E.: Approximate string matching allowing for inversions and translocations. In: Proceedings of the Prague Stringology Conference, pp. 37–51 (2010)
4. Cantone, D., Faro, S., Giaquinta, E.: Text searching allowing for inversions and translocations of factors. Discrete Appl. Math. **163**, 247–257 (2014)
5. Cantone, D., Faro, S., Pavone, A.: Sequence searching allowing for non-overlapping adjacent unbalanced translocations. Report arXiv:1812.00421. Cornell University Library (2018). https://arxiv.org/abs/1812.00421
6. Cho, D.-J., Han, Y.-S., Kim, H.: Alignment with non-overlapping inversions and translocations on two strings. Theor. Comput. Sci. **575**, 90–101 (2015)
7. Cull, P., Hsu, T.: Recent advances in the walking tree method for biological sequence alignment. In: Moreno-Díaz, R., Pichler, F. (eds.) EUROCAST 2003. LNCS, vol. 2809, pp. 349–359. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45210-2_32
8. Damerau, F.: A technique for computer detection and correction of spelling errors. Commun. ACM **7**(3), 171–176 (1964)
9. Knuth, D.E., Morris Jr., J.H., Pratt, V.R.: Fast pattern matching in strings. SIAM J. Comput. **6**(1), 323–350 (1977)
10. Lupski, J.R.: Genomic disorders: structural features of the genome can lead to DNA rearrangements and human disease traits. Trends Genet. **14**(10), 417–422 (1998)
11. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions and reversals. Sov. Phys. Dokl. **10**, 707–710 (1966)
12. Morris, J.H., Pratt, V.R.: A linear pattern-matching algorithm. Technical report 40. University of California, Berkeley (1970)
13. Ogiwara, H., Kohno, T., Nakanishi, H., Nagayama, K., Sato, M., Yokota, J.: Unbalanced translocation, a major chromosome alteration causing loss of heterozygosity in human lung cancer. Oncogene **27**, 4788–4797 (2008)
14. Oliver-Bonet, M., Navarro, J., Carrera, M., Egozcue, J., Benet, J.: Aneuploid and unbalanced sperm in two translocation carriers: evaluation of the genetic risk. Mol. Hum. Reprod. **8**(10), 958–963 (2002)
15. Schöniger, M., Waterman, M.: A local algorithm for DNA sequence alignment with inversions. Bull. Math. Biol. **54**, 521–536 (1992)
16. 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, vol. 4175, pp. 186–196. Springer, Heidelberg (2006). https://doi.org/10.1007/11851561_18
17. Warburton, D.: De novo balanced chromosome rearrangements and extra marker chromosomes identified at prenatal diagnosis: clinical significance and distribution of breakpoints. Am. J. Hum. Genet. **49**, 995–1013 (1991)
18. Weckselblatt, B., Hermetz, K.E., Rudd, M.K.: Unbalanced translocations arise from diverse mutational mechanisms including chromothripsis. Genome Res. **25**(7), 937–947 (2015)