# Efficient Matching of Biological Sequences Allowing for Non-Overlapping Inversions *

Domenico Cantone, Salvatore Cristofaro, and Simone Faro

Università degli Studi di Catania, Dipartimento di Matematica e Informatica
Viale Andrea Doria 6, I-95125, Catania, Italy
{cantone, cristofaro, faro} @dmi.unict.it

**Abstract.** Inversions are a class of chromosomal mutations, widely regarded as one of the major mechanisms for reorganizing the genome. In this paper we present a new algorithm for the approximate string matching problem allowing for non-overlapping inversions which runs in $\mathcal{O}(nm)$ worst-case time and $\mathcal{O}(m^2)$-space, for a character sequence of size $n$ and pattern of size $m$. This improves upon a previous $\mathcal{O}(nm^2)$-time algorithm.

## 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). Aligning sequences helps in revealing their shared characteristics, while matching sequences can infer useful information from them. With the availability of large amounts of DNA data, matching of nucleotide sequences has become an important application and there is an increasing demand for fast computer methods for data analysis and retrieval.

*Approximate string matching* is a fundamental problem in text processing. It consists in finding approximate matches of a pattern in a text. The precision of a match is measured in terms of the sum of the costs of the edit operations necessary to convert the string into an exact match.

Most classical models, as for instance the Levenshtein or Damerau edit distance, assume that changes between strings occur only locally (for an in-depth survey on approximate string matching, see [7]). However, evidence shows that large scale changes, like duplications, translocations, and inversions, are common events in genetic evolution [4]. For instance, chromosomal inversions are rearrangements in which a segment of a chromosome is reversed end to end. Notice that inversions do not involve any loss of genetic information, but simply rearrange the linear gene sequence.

In this paper we are interested in the approximate string matching problem allowing for non-overlapping inversions. Much work has been made for the closely

* This work has been partly supported by G.N.C.S., Istituto Nazionale di Alta Matematica “Francesco Severi”.

related sequence alignment problem with inversions. Although the latter problem does not have a known polynomial algorithm in its full generality, when restricted to non-overlapping inversions it admits polynomial solutions. A first solution was proposed by Schöniger and Waterman [8]. Their algorithm, based on dynamic programming, runs in $\mathcal{O}(n^2m^2)$-time and $\mathcal{O}(n^2m^2)$-space on input sequences of length $n$ and $m$. Later, Gao *et al.* [2] developed a space-efficient variant which requires only $\mathcal{O}(nm)$-space (and still $\mathcal{O}(n^2m^2)$-time). More recently, Vellozo *et al.* [9] proposed a $\mathcal{O}(nm^2)$-time and $\mathcal{O}(nm)$-space algorithm, within the more general framework of an edit graph.

Although proposed for the sequence alignment problem, the algorithm by Vellozo *et al.* could also be adapted to the approximate string matching problem with non-overlapping inversions, yielding a $\mathcal{O}(nm^3)$-time and $\mathcal{O}(m^2)$-space solution to the latter problem. A more efficient solution, which runs in $\mathcal{O}(nm^2)$-time and $\mathcal{O}(m^2)$-space, was presented by Cantone *et al.* [1]. They actually addressed a slightly more general problem, allowing also for translocations of equal length adjacent factors besides non-overlapping inversions. A very recent algorithm by Grabowski *et al.* [5] solves the same matching problem, i.e., when translocations and non-overlapping inversions are allowed, in $\mathcal{O}(nm^2)$-time and $\mathcal{O}(m)$-space, obtaining better performances in practical cases.

In this paper we present an algorithm for the approximate string matching problem with non-overlapping inversions which runs in $\mathcal{O}(nm)$ worst-case time and $\mathcal{O}(m^2)$-space.

The paper is organized as follows. In Section 2 we provide the basic terminology and definitions. Next, in Section 3 we present a general $\mathcal{O}(nm^2)$-time and $\mathcal{O}(m^2)$-space algorithm for the approximate matching problem with non-overlapping inversions, based on the dynamic programming approach. Such algorithm will then be refined in Section 4, yielding a $\mathcal{O}(nm)$-time and $\mathcal{O}(m^2)$-space algorithm which constitutes the main result of the paper. Finally we draw our conclusions in Section 5.

## 2 Basic notions and properties

A string $p$ of length $m \geq 0$ is represented as a finite array $p[0\,..\,m-1]$. In such a case we also write $|p| = m$. In particular, for $m = 0$ we obtain the empty string, denoted by $\varepsilon$. The concatenation of strings $p$ and $q$ is denoted as $p.q$ or, more simply, as $pq$. We denote with $p^{\mathsf{R}}$ the reversal of $p$, i.e., string $p$ written in reverse order. Notice that $|p| = |p^{\mathsf{R}}|$ and $(p^{\mathsf{R}})^{\mathsf{R}} = p$. Moreover, for any two strings $p$ and $q$, we have that $(p.q)^{\mathsf{R}} = (q^{\mathsf{R}}.p^{\mathsf{R}})$.

Given a nonempty string $p$ and an integer $i$, we denote by $p[i]$ the $(i+1)$st symbol of $p$ from left to right, if $0 \leq i < |p|$, otherwise we consider $p[i]$ as undefined.[1] Likewise, we denote with $p[i\,..\,j]$ the substring of $p$ contained between the $(i+1)$st and the $(j+1)$st symbol of $p$ (both inclusive), for $0 \leq i \leq j < |p|$. Moreover, we put $p_j = p[0\,..\,j]$, for $0 \leq j < |p|$.

[1] When $p[i]$ is undefined, the condition $p[i] = c$, for any character symbol $c$, will be regarded as false, whereas the condition $p[i] \neq c$ will be regarded as true.

We say that $p$ is a *prefix* (resp., *suffix*) of $q$, and write $p \sqsubseteq q$ (resp., $p \sqsupseteq q$), if there is a string $s$ such that $q = p.s$ (resp., $q = s.p$). A string $p$ is a *border* of $q$ if both $p \sqsubseteq q$ and $p \sqsupseteq q$ hold. The set of the borders of $p$ is denoted by $\mathit{borders}(p)$.

For a set $S$ of strings, we denote by $\|S\|$ the collection of the lengths of the strings belonging to $S$, i.e, $\|S\| = \{|p| : p \in S\}$.

For strings $p$ and $q$, we denote by $\langle p, q\rangle$ the set of all suffixes $s$ of $p$ such that $s^{\mathsf{R}}$ is a suffix of $q$, i.e., $\langle p, q\rangle = \{s : s \sqsupseteq p \text{ and } s^{\mathsf{R}} \sqsupseteq q\}$.

The following lemma states useful properties of the set of borders of two strings.

**Lemma 1.** *For all strings $p$, $q$, $v$, $w$, and $z$, and every alphabet symbol $c$, the following facts hold:*

*(a) if $v, w \in \langle p, q\rangle$, then either $v \in \mathit{borders}(w)$ or $w \in \mathit{borders}(v)$;*
*(b) if $v, w \in \langle p, q\rangle$ and $|v| \geq |w|$, then $w \in \mathit{borders}(v)$;*
*(c) if $v \in \langle p, q\rangle$ and $w \in \mathit{borders}(v)$, then $w \in \langle p, q\rangle$;*
*(d) if $z$ is the longest string belonging to $\langle p, q\rangle$, then $\langle p, q\rangle = \mathit{borders}(z)$;*
*(e) $\langle p, q.c\rangle = \{c.s : s \in \langle p, q\rangle \text{ and } c.s \sqsupseteq p\} \cup \{\varepsilon\}$;*
*(f) $\|\langle p, q.c\rangle\| = \{\ell + 1 : \ell \in \|\langle p, q\rangle\| \text{ and } p[|p| - 1 - \ell] = c\} \cup \{0\}$.*

*Proof.* First of all we notice that (b) and (f) are immediate consequences of (a) and (e), respectively; similarly, (d) follows plainly from (b) and (c). Thus, we only need to prove (a), (c), and (e).

We begin with (a). Let $v, w \in \langle p, q\rangle$. By the very definition of $\langle p, q\rangle$ we have

$$v \sqsupseteq p\,,\ v^{\mathsf{R}} \sqsupseteq q\,,\ w \sqsupseteq p\,,\ \text{and } w^{\mathsf{R}} \sqsupseteq q\,.$$

Without loss of generality, let us assume that $|v| \leq |w|$. Then, from $v \sqsupseteq p$ and $w \sqsupseteq p$ we have $v \sqsupseteq w$; likewise, from $v^{\mathsf{R}} \sqsupseteq q$ and $w^{\mathsf{R}} \sqsupseteq q$ we have $v^{\mathsf{R}} \sqsupseteq w^{\mathsf{R}}$. The latter implies $v \sqsubseteq w$, which, together the previously established relation $v \sqsupseteq w$, yields $v \in \mathit{borders}(w)$, proving (a).

Concerning (c), let $v \in \langle p, q\rangle$ and $w \in \mathit{borders}(v)$. Then we have $v \sqsupseteq p$ and $w \sqsupseteq v$, so that $w \sqsupseteq p$. Likewise, we have $v^{\mathsf{R}} \sqsupseteq q$ and $w \sqsubseteq v$. The latter is equivalent to $w^{\mathsf{R}} \sqsupseteq v^{\mathsf{R}}$, so that $w^{\mathsf{R}} \sqsupseteq q$. From $w \sqsupseteq p$ and $w^{\mathsf{R}} \sqsupseteq q$ it follows that $w \in \langle p, q\rangle$, proving (c).

Finally, we turn to the proof of (e). Let $v \in \langle p, q.c\rangle$, where $c$ is a character. Then $v \sqsupseteq p$ and $v^{\mathsf{R}} \sqsupseteq q.c$. If $v \neq \varepsilon$, then $v = c.s$, for a string $s$ such that $s \sqsubseteq q^{\mathsf{R}}$. But then $s \sqsupseteq (q^{\mathsf{R}})^{\mathsf{R}} = q$, which, together with $s \sqsupseteq p$, implies

$$\langle p, q.c\rangle \subseteq \{c.s : s \in \langle p, q\rangle \text{ and } c.s \sqsupseteq p\} \cup \{\varepsilon\}\,.$$

To show the converse inclusion, we observe preliminarily that $\varepsilon \in \langle p, q.c\rangle$. Let $s \in \langle p, q\rangle$ such that $c.s \sqsupseteq p$. Then $s^{\mathsf{R}} \sqsupseteq q$, which implies $(c.s)^{\mathsf{R}} = s^{\mathsf{R}}.c \sqsupseteq q.c$. The latter, together with $c.s \sqsupseteq p$, implies $c.s \in \langle p, q.c\rangle$. Thus

$$\{c.s : s \in \langle p, q\rangle \text{ and } c.s \sqsupseteq p\} \cup \{\varepsilon\} \subseteq \langle p, q.c\rangle\,,$$

which together with the previously established inclusion proves (e). $\square$

Given two strings $p$ and $q$ of the same length $m$, an *inverted decomposition* of $p$ and $q$ is a sequence $(\ell_1, \ell_2, \ldots, \ell_k)$ of lengths such that:

(*a*) $1 \leq \ell_i \leq m$, for $1 \leq i \leq k$;
(*b*) $\sum_{i=1}^{k} \ell_i = m$;
(*c*) $p[L_j \,..\, L_{j+1}] = (q[L_j \,..\, L_{j+1}])^{\mathsf{R}}$, for $0 \leq j < k$, and where $L_j = \sum_{i=1}^{j} \ell_i$ (so that $L_0 = 0$).

When $p$ and $q$ admit an inverted decomposition, we write $p \bowtie q$.

Observe that an inverted decomposition $(\ell_1, \ell_2, \ldots, \ell_k)$ of $p$ and $q$ induces a sequence of strings $(s_1, s_2, \ldots, s_k)$ such that $s_1 s_2 \cdots s_k = p$ and $s_1^{\mathsf{R}} s_2^{\mathsf{R}} \cdots s_k^{\mathsf{R}} = q$, and conversely. Thus, we plainly have that $p \bowtie q$ iff $q \bowtie p$. Additionally, the following property can be easily proved.

**Lemma 2.** *For all strings $p$ and $q$, we have that $p \bowtie q$ holds iff (exactly) one of the following two conditions holds:*

*(a)* $p = q = \varepsilon$, *or*
*(b)* $p = v.z$ *and* $q = w.z^{\mathsf{R}}$, *for a string* $z \neq \varepsilon$ *and strings* $v$ *and* $w$ *such that* $v \bowtie w$. □

Given a text $t$ of length $n$, a pattern $p$ of length $m$ is said to *match with non-overlapping inversions* (or to have an *occurrence with non-overlapping inversions*) at location $i$ of $t$ if $p \bowtie t[i \,..\, i+m-1]$, i.e., if there exists an inverted decomposition of $p$ and $t[i \,..\, i+m-1]$.

The *approximate matching problem with non-overlapping inversions* is to find all locations $i$ in a given text $t$ at which a given pattern $p$ matches with non-overlapping inversions.

For the sake of simplicity, in the rest of the paper we will refer to non-overlapping inversions simply as *inversions*, since this will generate no confusion.

## 3 A general dynamic programming approach

In this section we present a general dynamic programming algorithm for the pattern matching problem with inversions. Our algorithm, which will be named DPInversionMatcher, is characterized by a $\mathcal{O}(nm^2)$-time and a $\mathcal{O}(m^2)$-space complexity, where $m$ and $n$ are the length of the pattern and text, respectively. In the next section we will then show how it can be refined so as to improve its time complexity to $\mathcal{O}(nm)$.

As above, let $t$ be a text of length $n$ and $p$ a pattern of length $m$. The algorithm DPInversionMatcher solves the matching problem with inversions by computing the occurrences of all prefixes of the pattern in continuously increasing prefixes of the text using a dynamic programming approach. That is, during its $(i+1)$st iteration, for $i = 0, 1, \ldots, n-m$, our algorithm establishes whether $p_j \bowtie t[i \,..\, i+j]$, for each $j = 0, 1, \ldots, m-1$, exploiting information gathered during previous iterations.

To begin with, we denote by $M(j,i)$ the set of all integral values $k$, with $0 \leq k \leq j$, such that the prefix $p_k$ has an occurrence with inversions at location $i$ of the text, or more formally

$$M(j,i) = \begin{cases} \{0 \leq k \leq j : p_k \bowtie t[i \,..\, i+k]\} \cup \{-1\} & \text{if } i \geq 0 \text{ and } j \geq 0 \\ \{-1\} & \text{otherwise,} \end{cases}$$

for $-m \leq i \leq n-m$ and $0 \leq j < m$.

Then notice that $p_j \bowtie t[i \,..\, i+j]$ iff $j \in M(j,i)$ and hence $p \bowtie t[i \,..\, i+m-1]$ iff $m-1 \in M(m-1,i)$. Thus the matching problem with inversions can be solved by computing the sets $M(m-1,i)$, for increasing values of $i$.

We also define $R(j,i)$ as the set of the lengths of all strings $s$ such that $s \sqsupseteq p_j$ and $s^{\mathsf{R}} \sqsupseteq t_{i+j}$, or more formally

$$R(j,i) = \begin{cases} \|\langle p_j, t_{i+j} \rangle\| & \text{if } 0 \leq j < m \text{ and } 0 \leq i \leq m-n \\ \{0\} & \text{otherwise.} \end{cases}$$

By Lemma 2, we obtain the following recursive relation

$$M(j,i) = \begin{cases} M(j-1,i) \cup \{j\} & \text{if } j-\ell \in M(j-1,i), \text{ for some } \ell \in R(j,i) \\ M(j-1,i) & \text{otherwise}\,, \end{cases}$$

where $0 \leq j < m$ and $-m \leq i \leq n-m$, which allows to reduce the computation of the set $M(j,i)$ to that of the sets $M(j-1,i)$ and $R(j,i)$.

Likewise, the sets $R(j,i)$ can be computed by the recursive relation

$$R(j,i) = \{0\} \cup \{\ell+1 : \ell \in R(j,i-1) \text{ and } p[j-\ell] = t[i+j]\}\,,$$

with $0 \leq j < m$ and $0 < i \leq n-m$, which follows from Lemma 1(f).

The above considerations translate directly into the algorithm $\mathsf{DPInversionMatcher}$ in Fig. 1. Sets $R(j,i)$ are maintained by an array $\mathsf{R}$ of dimension $m$; more precisely, just after iteration $i$ of the $\mathsf{for\text{-}loop}$ at line 3, we have that $\mathsf{R}[j] = R(j,i)$. Similarly, sets $M(j,i)$ are maintained by a single set(-variable) $\mathsf{M}$, which is initialized to $\{-1\}$ at the beginning of iteration $i$ of the $\mathsf{for\text{-}loop}$ at line 3 (this corresponds to the set $M(-1,i)$). Then, during the execution of the subsequent $\mathsf{for\text{-}loop}$ at line 5, the set $\mathsf{M}$ is expanded so as to take in sequence the relevant elements $M(0,i), M(1,i), \ldots, M(m-1,i)$; more precisely, just after the execution of iteration $j$ of the $\mathsf{for\text{-}loop}$ at line 5 we have that $\mathsf{M} = M(j,i)$.

The set $\mathsf{M}$ can be implemented as a linear array $\mathcal{A}$ of length $m+1$ of Boolean values such that

$$\mathcal{A}[j] = \begin{cases} \mathsf{true}, & \text{if } j-1 \in \mathsf{M} \\ \mathsf{false}, & \text{otherwise}\,, \end{cases}$$

for $0 \leq j \leq m$. Likewise, each set $\mathsf{R}[j]$ can be implemented as a linked list (or possibly as an array of length $m+1$ of Boolean values, as well). Then it follows easily that the algorithm $\mathsf{DPInversionMatcher}$ in Fig. 1 has a $\mathcal{O}(m^2)$-space complexity and a $\mathcal{O}(nm^2)$-time complexity. Indeed, the computation of the set $\mathsf{R}[j]$ at line 6 and the conditional test at line 7 require $\mathcal{O}(j)$-time, for $0 \leq j < m$.

```
DPInversionMatcher(p, m, t, n)
 1.  for j := 0 to m − 1 do
 2.     R[j] := {0}
 3.  for i := −m + 1 to n − m do
 4.     M := {−1}
 5.     for j = max(−i, 0) to m − 1 do
 6.        R[j] := {0} ∪ {ℓ + 1 : ℓ ∈ R[j] \ {j + 1} AND p[j − ℓ] = t[i + j]}
 7.        if (∃ℓ ∈ R[j] : j − ℓ ∈ M) then
 8.           M := M ∪ {j}
 9.     if (m − 1 ∈ M) then
10.        output(i)
```

**Fig. 1.** The algorithm DPInversionMatcher for the matching problem with inversions.

## 4 The algorithm InversionSampling

In this section we present a refinement of the algorithm DPInversionMatcher presented before. The new algorithm, named InversionSampling, achieves a $\mathcal{O}(nm)$ worst-case time complexity and, as before, requires $\mathcal{O}(m^2)$ additional space.

The main idea upon which the new algorithm is based is that we do not need to maintain explicitly the whole set $R(j,i)$ to evaluate the conditional test at line 7. In particular we show that by efficiently computing the values in the set $R(j,i)$, each conditional test at line 7 can be performed in amortized $\mathcal{O}(1)$-time.

Specifically, as will be proved in Lemma 5 below, during each iteration of the algorithm DPInversionMatcher, just before the execution of the conditional test at line 7, the following condition holds

$$\begin{aligned} \text{either} \quad & \{\ell \in R(j,i) : j-\ell \in M(j-1,i)\} = \emptyset \quad \text{(when the test is false)}, \\ \text{or} \quad & \{\ell \in R(j,i) : j-\ell \in M(j-1,i)\} = R(j,i) \setminus \{0\}\,. \end{aligned} \tag{1}$$

Thus it follows that, for each $0 < j < m$ and $-m \leq i \leq n-m$, if $\max(R(j,i)) \in M(j-1,i)$ then $\{\ell \in R(j,i) : j-\ell \in M(j-1,i)\} = R(j,i) \setminus \{0\}$.

Since just before the execution of the conditional test at line 7 of the algorithm DPInversionMatcher we have that $j \notin M(j-1,i)$, the condition '$(\exists \ell \in \mathsf{R}[j] : j-\ell \in \mathsf{M})$' at line 7 can be replaced by the condition '$j - e(\mathsf{R}[j]) \in \mathsf{M}$', for any function $e(\cdot)$ such that $e(\mathsf{R}[j]) \in (\mathsf{R}[j] \setminus \{0\}) \cup \{\max(\mathsf{R}[j])\}$ holds, without affecting the correctness of the algorithm. In particular, we choose $e(\cdot) \equiv \max(\cdot)$ and describe an efficient way to compute the value $\max(R(j,i))$, which allows to reduce the time complexity of the searching-phase of the algorithm to $\mathcal{O}(nm)$.

Recalling that $R(j,i) = \|\langle p_j, t_{i+j}\rangle\|$, it turns out that the maximum of the set $R(j,i)$, for $-m < i \leq n-m$ and $0 \leq j < m$, can be computed from the maximum of the set $R(j,i-1)$, without any need to compute explicitly the whole

```
procMPT(p, m)
for k := 0 do m − 1 do
   i := 0
   j := W[k, k] := −1
   while (i < (m − k)) do
      while ((j > −1) and
            (p[i + k] ≠ p[j + k])) do
         j := W[k, k + j]
      i := i + 1
      j := j + 1
      W[k, k + i] := j
return(W)
```

```
InversionSampling(p, m, t, n)
W := procMPT(p, m)
for j := 0 to m − 1 do
   K[j] := 0
for i = −m + 1 to n − m do
   M = {−1}
   for j := max(−i, 0) to m − 1 do
      while ((K[j] > 0)
            and (p[j − K[j]] ≠ t[i + j])) do
         K[j] := W[j + 1 − K[j], j + 1]
      if p[j − K[j]] = t[i + j] then
         K[j] := K[j] + 1
      if (j − K[j] ∈ M) then
         M := M ∪ {j}
   if (m − 1) ∈ M then
      output(i)
```

**Fig. 2.** (On the left) the procedure for computing the table $\mathsf{W}$, and (on the right) the variant InversionSampling of the algorithm DPInversionMatcher.

set $R(j, i)$. This can be done by using the following relation:

$$\max(\|\langle p_j, t_{i+j}\rangle\|) = \max\{\ell + 1 : \ell \in \langle p_j, t_{i+j-1}\rangle \text{ and } p[j - \ell] = t[i + j]\}\,, \quad (2)$$

which will be proved in Lemma 6 below.

Let $\|\langle p_j, t_{i+j-1}\rangle\|$ be the set $\{\ell_1, \ell_2, \ldots, \ell_k\}$, with $\ell_i > \ell_{i+1}$, for all $0 < i < k$, and $\ell_k = 0$. For the computation of the set $\max(\|\langle p_j, t_{i+j}\rangle\|)$ we start from the value $\ell_1 = \max(\|\langle p_j, t_{i+j-1}\rangle\|)$, and examine in sequence the items $\ell_1$, $\ell_2$, $\ldots, \ell_k$ until we find a value $\ell_i$ such that $p[j - \ell_i] = t[i + m - 1]$ or we reach $\ell_k = 0$. If $\ell$ is the value obtained by such scanning process, we check whether $p[j - \ell] = t[i + j]$ and, in this case, we conclude that $\max(\|\langle p_j, t_{i+j}\rangle\|) = \ell + 1$; otherwise we conclude that $\max(\|\langle p_j, t_{i+j}\rangle\|) = 0$.

The above procedure requires to know in advance the set $\|\langle p_j, t_{i+j-1}\rangle\|$. To this purpose let us put

$$\pi(p_j, h) = \begin{cases} \max(\|\mathit{borders}(p[h\,..\,j]) \setminus \{p[h\,..\,j]\}\|) & \text{if } 0 \le h \le j \\ -1 & \text{otherwise.} \end{cases}$$

For $i = 1, \ldots, k$, let us also put $v_i = p[j + 1 - \ell_i\,..\,j]$. Then, since $v_{i+1}$ is a border of $v_i$ (by Lemma 1(b)), we have that $\ell_{i+1} = \pi(p_j, j + 1 - \ell_i)$, for $0 < i < k$.

Such values can be precomputed and collected into a table $\mathsf{W}$ of dimensions $(m + 1) \times (m + 1)$, where $\mathsf{W}[0, 0] = -1$ and $\mathsf{W}[h, k] = \pi(p_{k-1}, h)$, for $0 < k \le m$ and $0 \le h \le k$ (the values of the remaining entries of $\mathsf{W}$ are not relevant).

Table $\mathsf{W}$ can be computed in $\mathcal{O}(m^2)$-time and space by means of the procedure procMPT in Fig. 2, which is a generalization of the procedure used by the

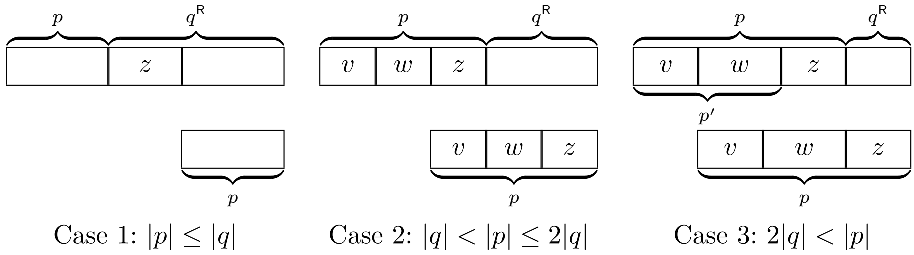


**Fig. 3.** The three cases considered in Lemma 3.

Morris-Pratt algorithm [6] for computing the length of the longest proper border of $s[0\,..\,j]$, for a given string $s$ with $0 \le j < |s|$ (see also [3], where this function is called the *prefix function* of the pattern).

The resulting algorithm, $\mathsf{InversionSampling}$, is presented in Fig. 2. Notice that the part of the code from line 7 up to line 10 implements the assignment $\mathsf{K}[j] := \max(R(j,i))$.

### 4.1 Correctness issues

In this section we prove the validity of (1) and (2), upon which the correctness of the algorithm $\mathsf{InversionSampling}$ is based. In particular, they will be direct consequences of Lemmas 5 and 6, respectively.

We first state and prove two useful properties related to the suffixes of inverted strings, which will be used in our main results.

**Lemma 3.** *Let $p$ and $q$ be strings such that $p \sqsupseteq p.q^{\mathsf{R}}$. Then there exist two strings $q_1$ and $q_2$ such that (a) $q = q_1.q_2$ and (b) $p.q^{\mathsf{R}} = q_1^{\mathsf{R}}.q_2^{\mathsf{R}}.p$.* □

*Proof.* Let $p \sqsupseteq p.q^{\mathsf{R}}$. To begin with, notice that if $|q| = 0$, the lemma follows trivially. So, let us suppose that $|q| > 0$ and assume inductively that the lemma holds for any pair $p'$, $q'$ of strings such that $|p'| < |p|$ and $p' \sqsupseteq p'.{q'}^{\mathsf{R}}$.

We distinguish the following three cases (see Fig. 3 for a pictorial illustration).

**Case 1:** $|p| \le |q|$**.** From $p \sqsupseteq p.q^{\mathsf{R}}$ and $|p| \le |q|$, it follows that $p \sqsupseteq q^{\mathsf{R}}$, so that $q^{\mathsf{R}} = z.p$, for some string $z$. Putting $q_1 = p^{\mathsf{R}}$ and $q_2 = z^{\mathsf{R}}$, we have then $q_1.q_2 = p^{\mathsf{R}}.z^{\mathsf{R}} = (z.p)^{\mathsf{R}} = (q^{\mathsf{R}})^{\mathsf{R}} = q$, and $p.q^{\mathsf{R}} = p.z.p = (p^{\mathsf{R}})^{\mathsf{R}}.(z^{\mathsf{R}})^{\mathsf{R}}.p = q_1^{\mathsf{R}}.q_2^{\mathsf{R}}.p$ and therefore (a) and (b) are both satisfied in the present case.

**Case 2:** $|q| < |p| \le 2|q|$**.** Let $z$ be the suffix of $p$ such that $|z| = |p| - |q| \le |q|$. Observe that $2|z| \le |z| + |q| = |p|$, so that $|z| \le \lfloor |p|/2 \rfloor$. Therefore $p$ can be decomposed as $p = v.w.z$, with $|v| = |z|$ and $|w.z| = |q|$. But since $p \sqsupseteq p.q^{\mathsf{R}}$, we have $v = z$ and $q^{\mathsf{R}} = w.z$. If we put $q_1 = z^{\mathsf{R}}$ and $q_2 = w^{\mathsf{R}}$, so that $z = q_1^{\mathsf{R}}$ and $w = q_2^{\mathsf{R}}$, we have $q = (q^{\mathsf{R}})^{\mathsf{R}} = (w.z)^{\mathsf{R}} = (q_2^{\mathsf{R}}.q_1^{\mathsf{R}})^{\mathsf{R}} = (q_1^{\mathsf{R}})^{\mathsf{R}}.(q_2^{\mathsf{R}})^{\mathsf{R}} = q_1.q_2$ and $p.q^{\mathsf{R}} = (z.w.z).(w.z) = (z.w).(z.w.z) = z.w.p = q_1^{\mathsf{R}}.q_2^{\mathsf{R}}.p$, proving (a) and (b) in the present case.

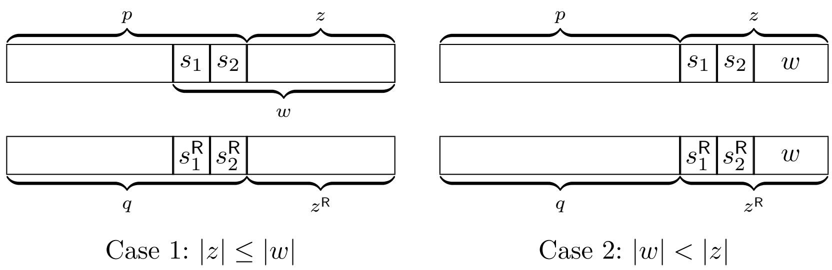


**Fig. 4.** The two cases considered in Lemma 4.

**Case 3:** $2|q| < |p|$**.** Let $v$ and $z$ be, respectively, the prefix and the suffix of $p$ such that $|v| = |z| = |q|$. Plainly, $z = q^{\mathsf{R}}$, as $p \sqsupseteq p.q^{\mathsf{R}}$. In addition, since $|v| + |z| = 2|q| < |p|$, it follows that $p = v.w.z$, for a nonempty string $w$. Let us put $p' = v.w$, so that $p = p'.z$. Observe that $|p'| = |p| - |q| < |p|$, since $|q| > 0$. We have also $p' \sqsupseteq p'.z$, so by induction $z^{\mathsf{R}} = q_1.q_2$ (i.e., $q = q_1.q_2$) and $p'.z = q_1^{\mathsf{R}}.q_2^{\mathsf{R}}.p'$, for some strings $q_1$ and $q_2$. Hence, $p.q^{\mathsf{R}} = (v.w.z).q^{\mathsf{R}} = (p'.z).q^{\mathsf{R}} = (q_1^{\mathsf{R}}.q_2^{\mathsf{R}}.p').q^{\mathsf{R}} = q_1^{\mathsf{R}}.q_2^{\mathsf{R}}.(p'.q^{\mathsf{R}}) = q_1^{\mathsf{R}}.q_2^{\mathsf{R}}.(p'.z) = q_1^{\mathsf{R}}.q_2^{\mathsf{R}}.p$, so that (a) and (b) hold in this last case too, completing the proof of the lemma. $\square$

**Lemma 4.** *Let $p$ and $q$ be strings of the same length. Then we have $p.z \bowtie q.z^{\mathsf{R}}$ if and only if $p \bowtie q$, for every string $z$.*

*Proof.* To begin with, notice that if $p \bowtie q$ then, plainly, $p.z \bowtie q.z^{\mathsf{R}}$. Thus, it is enough to prove the converse implication, namely that $p.z \bowtie q.z^{\mathsf{R}}$ implies $p \bowtie q$. So, let $p$, $q$, and $z$ be nonempty strings such that $p.z \bowtie q.z^{\mathsf{R}}$ and assume inductively that the lemma is true for all triplets $p'$, $q'$, $z'$, with $|z'| < |z|$, such that $p'.z' \bowtie q'.z'^{\mathsf{R}}$. By Lemma 2, there are strings $u$, $v$, and $w \neq \varepsilon$ such that $u \bowtie v$, $p.z = u.w$, and $q.z^{\mathsf{R}} = v.w^{\mathsf{R}}$.

We consider first the case in which $|z| \leq |w|$ (this is illustrated in Fig. 4, on the left). Since $z \sqsupseteq u.w$ and $z^{\mathsf{R}} \sqsupseteq v.w^{\mathsf{R}}$, we have that $z \sqsupseteq w$ and $z^{\mathsf{R}} \sqsupseteq w^{\mathsf{R}}$. Let $s$ be the string such that $w = s.z$ . Then, $z^{\mathsf{R}} \sqsupseteq w^{\mathsf{R}} = (s.z)^{\mathsf{R}} = z^{\mathsf{R}}.s^{\mathsf{R}}$ and hence, by Lemma 3, there are strings $s_1$ and $s_2$ such that $s_1.s_2 = s$ (which implies that $w = s_1.s_2.z$) and $z^{\mathsf{R}}.s^{\mathsf{R}} = s_1^{\mathsf{R}}.s_2^{\mathsf{R}}.z^{\mathsf{R}}$, i.e., $w^{\mathsf{R}} = s_1^{\mathsf{R}}.s_2^{\mathsf{R}}.z^{\mathsf{R}}$ . Therefore, we have that $p.z = u.s_1.s_2.z$ and $q.z^{\mathsf{R}} = v.s_1^{\mathsf{R}}.s_2^{\mathsf{R}}.z^{\mathsf{R}}$ . These equalities imply, respectively, that $p = u.s_1.s_2$ and $q = v.s_1^{\mathsf{R}}.s_2^{\mathsf{R}}$, and hence, as $u \bowtie v$, by a double application of Lemma 2, we get $p \bowtie q$.

Let us consider next the case in which $|w| < |z|$ (this is illustrated in Fig. 4, on the right). Since $w \sqsupseteq p.z$ and $w^{\mathsf{R}} \sqsupseteq q.z^{\mathsf{R}}$, in this case we have that $w \sqsupseteq z$ and $w^{\mathsf{R}} \sqsupseteq z^{\mathsf{R}}$. Let $s$ be the string such that $z = s.w$. Then, $w^{\mathsf{R}} \sqsupseteq z^{\mathsf{R}} = (s.w)^{\mathsf{R}} = w^{\mathsf{R}}.s^{\mathsf{R}}$, and hence, by Lemma 3, there are strings $s_1$ and $s_2$ such that $s_1.s_2 = s$ (which implies that $z = s_1.s_2.w$) and $w^{\mathsf{R}}.s^{\mathsf{R}} = s_1^{\mathsf{R}}.s_2^{\mathsf{R}}.w^{\mathsf{R}}$, i.e., $z^{\mathsf{R}} = s_1^{\mathsf{R}}.s_2^{\mathsf{R}}.w^{\mathsf{R}}$. Therefore, we have that $p.z = p.s_1.s_2.w$ and $q.z^{\mathsf{R}} = q.s_1^{\mathsf{R}}.s_2^{\mathsf{R}}.w^{\mathsf{R}}$ , which imply, respectively, that $u.w = p.s_1.s_2.w$ and $v.w^{\mathsf{R}} = q.s_1^{\mathsf{R}}.s_2^{\mathsf{R}}.w^{\mathsf{R}}$ , so that $u = p.s_1.s_2$ and $v = q.s_1^{\mathsf{R}}.s_2^{\mathsf{R}}$. Since $|s_2| < |z|$, by induction we deduce $p.s_1 \bowtie q.s_1^{\mathsf{R}}$ from

$p.s_1.s_2 = u \bowtie v = q.s_1^{\mathsf{R}}.s_2^{\mathsf{R}}$. Likewise, since $|s_1| < |z|$, again by induction we deduce $p \bowtie q$ from $p.s_1 \bowtie q.s_1^{\mathsf{R}}$. Thus, $p \bowtie q$ holds even when $|w| < |z|$, concluding the proof of the lemma. □

Correctness of (1) is a direct consequence of the following lemma.

**Lemma 5.** *Let $D(j,i) = \{\ell \in R(j,i) : j - \ell \in M(j-1,i)\}$, for $0 < j < m$ and $-m \leq i \leq n-m$. Then we have either $D(j,i) = \emptyset$ or $D(j,i) = R(j,i) \setminus \{0\}$.*

*Proof.* First of all, note that if $D(j,i) \neq \emptyset$ then $j \in M(j,i)$, i.e., $p_j \bowtie t[i \mathinner{..} i+j]$, so that by Lemma 4, we must have $p_{j-k} \bowtie t[i \mathinner{..} i+j-k]$ for all $k \in R(j,i) \setminus \{0\}$. But $p_{j-k} \bowtie t[i \mathinner{..} i+j-k]$, with $k \neq 0$, implies that $j-k \in M(j-1,i)$, and thus $R(j,i) \setminus \{0\} \subseteq D(j,i)$. The converse implication, i.e., $D(j,i) \subseteq R(j,i) \setminus \{0\}$, holds trivially, since $j \notin M(j-1,i)$. □

Finally, relation (2), which allows to compute the maximum of the set $R(j,i)$ from the maximum of the set $R(j,i-1)$, is established in the following lemma.

**Lemma 6.** *Given two strings $p$ and $q$, with $|p| = m$, and a character $c$, we have*

$$\max(\|\langle p, q.c\rangle\|) = \max\{|v|+1 : v \in \langle p,q\rangle \text{ and } p[m-1-|v|] = c\}\,.$$

*Proof.* Let $z$ be the longest string belonging to $\langle p,q\rangle$, so that $|z| = \max(\|\langle p,q\rangle\|)$, and let $v_1, v_2, \ldots, v_k$ be the borders of $z$, ordered by their decreasing lengths. Observe that if $v, w \in \langle p,q\rangle$, then $v \sqsupseteq p$ and $w \sqsupseteq p$, so that if $v$ and $w$ have the same length they must coincide. Hence, the set $\langle p,q\rangle$ cannot contain any two distinct strings of the same length. It also follows that the longest string belonging to $\langle p,q\rangle$ is well (and uniquely) defined. Also, note that a string $z$ cannot have two distinct borders of the same length. Thus we have

$$|v_1| > |v_2| > \cdots > |v_k|\,,$$

with $v_1 = z$ and $v_k = \varepsilon$. Then, from Lemma 1(d) it follows that $\langle p,q\rangle = \{v_1, v_2, \ldots, v_k\}$ which, by Lemma 1(d), yields

$$\|\langle p, q.c\rangle\| = \{|v|+1 : v \in \{v_1, \ldots, v_k\} \text{ and } p[m-1-|v|] = c\} \cup \{0\}\,,$$

completing the proof of the lemma. □

### 4.2 Worst-case time analysis

We show now that the worst-case time complexity $T(n,m)$ of the algorithm InversionSampling reported in Fig. 2 is $\mathcal{O}(nm)$, for an input text $t$ of length $n$ and pattern $p$ of length $m$.

To begin with, we observe that the preprocessing phase of the algorithm requires $\mathcal{O}(m^2)$-time (and space), due to the computation of the table W and the initialization at line 2. Next we evaluate the complexity of the searching phase, namely of the for-loop at line 4. Let us denote by $A$ the set of pairs

$\{-m+1,\ldots,n-m\} \times \{0\ldots,m-1\}$. For each pair $(i,j) \in A$, we let $\mathtt{C1}(i,j)$ be the number of times that the while-loop at line 7 is executed during iteration $i$ of the for-loop at line 4, and we let $K(i,j)$ be the value contained in $\mathsf{K}[j]$ just after the termination of such iteration; in addition, we put $\mathtt{C2}(i,j) = 1$, if the assignment instruction at line 10 is executed during iteration $i$, otherwise we put $\mathtt{C2}(i,j) = 0$. Plainly, we have that

$$T(n,m) = \mathcal{O}\left(\sum_{i=-m+1}^{n-m} \sum_{j=0}^{m-1} \big(\mathtt{C1}(i,j)+1\big)\right), \tag{3}$$

and therefore it is enough to prove that the double summation in (3) is asymptotically bounded above by the product $nm$.

Since $\mathtt{C2}(i,j) \leq 1$ for each $(i,j) \in A$, we have that, for $0 \leq j < m$,

$$\sum_{i=-m+1}^{n-m} \mathtt{C2}(i,j) \leq n\,. \tag{4}$$

On the other hand, we have also that

$$K(i+1,j) - \mathtt{C2}(i+1,j) \leq K(i,j) - \mathtt{C1}(i+1,j)\,, \tag{5}$$

for all $(i,j) \in A$ such that $i < n-m$. Indeed, during iteration $i$, the value contained in $\mathsf{K}[j]$ just after the execution of the while-loop at line 7 (i.e., $K(i+1,j) - \mathtt{C2}(i+1,j)$) can never exceed the value contained in $\mathsf{K}[j]$ just before this execution minus the number of times that the while-loop iterates (i.e., $K(i,j) - \mathtt{C1}(i+1,j)$), since $\mathsf{K}[j]$ is decremented at least by one unit during each iteration of the while-loop at line 7. Thus it follows that

$$0 \leq K(h,j) \leq \sum_{i=-m+1}^{h} \mathtt{C2}(i,j) - \sum_{i=-m+1}^{h} \mathtt{C1}(i,j)\,, \tag{6}$$

for all $(h,j) \in A$, as can be verified by induction on $h$, using (5). From (6) it follows that

$$\sum_{j=0}^{m-1} \sum_{i=-m+1}^{n-m} \big(\mathtt{C1}(i,j)+1\big) \leq \sum_{j=0}^{m-1} \sum_{i=-m+1}^{n-m} \big(\mathtt{C2}(i,j)+1\big)\,,$$

and thus, using (4), we finally obtain that

$$\sum_{j=0}^{m-1} \sum_{i=-m+1}^{n-m} \big(\mathtt{C1}(i,j)+1\big) \leq (n+1)m\,,$$

which in turn, by (3), yields $T(n,m) = \mathcal{O}(nm)$.

## 5 Conclusions and future work

In this paper we have presented an algorithm to solve the pattern matching problem under a string distance which allows inversions of non-overlapping factors. The algorithm, named InversionSampling, has worst case $\mathcal{O}(nm)$-time and $\mathcal{O}(m^2)$-space complexity, where $m$ and $n$ are the length of the pattern and the length of the text, respectively. We are currently working on an efficient variant of the present algorithm with a linear average time complexity.


## Acknowledgements

We thank the anonymous reviewers for their helpful comments.


## References


1. D. Cantone, S. Faro, and E. Giaquinta. Approximate string matching allowing for inversions and translocations. In Jan Holub and Jan Zdárek, editors, *Proceedings of the Prague Stringology Conference 2010*, pages 37–51, Czech Technical University, Prague, Czech Republic, 2010.
2. Z. Chen, Y. Gao, G. Lin, R. Niewiadomski, Y. Wang, and J. Wu. A space-efficient algorithm for sequence alignment with inversions and reversals. *Theor. Comput. Sci.*, 325(3):361–372, 2004.
3. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.
4. Katrien M. Devos, M. D. Atkinson, C. N. Chinoy, H. A. Francis, R. L. Harcourt, R. M. D. Koebner, C. J. Liu, P. Masoj, D. X. Xie, and M. D. Gale. Chromosomal rearrangements in the rye genome relative to that of wheat. *TAG Theoretical and Applied Genetics*, 85:673–680, 1993.
5. Szymon Grabowski, Simone Faro, and Emanuele Giaquinta. String matching with inversions and translocations in linear average time (most of the time). *Information Processing Letters*, 111(11):516 – 520, 2011.
6. J. H. Morris, Jr and V. R. Pratt. A linear pattern-matching algorithm. Report 40, University of California, Berkeley, 1970.
7. G. Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, 2001.
8. M. Schniger and M. Waterman. A local algorithm for dna sequence alignment with inversions. *Bulletin of Mathematical Biology*, 54:521–536, 1992.
9. A. F. Vellozo, C. E. R. Alves, and A. Pereira do Lago. Alignment with non-overlapping inversions in $o(n^3)$-time. In *WABI*, pages 186–196, 2006.