ARTICLE IN PRESS

Theoretical Computer Science ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

# Efficient string-matching allowing for non-overlapping inversions

Domenico Cantone, Salvatore Cristofaro, Simone Faro *

*Università di Catania, Dipartimento di Matematica e Informatica, Viale Andrea Doria 6, I-95125 Catania, Italy*

ARTICLE INFO

*Keywords:*
Approximate string matching
Average complexity analysis
Text processing
Information retrieval

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.

In addition we present a variant of our algorithm with the same complexity in the worst case, but with a $\mathcal{O}(n)$ time complexity in the average case.



## 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, 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 [10]). However, evidence shows that large scale changes, like duplications, translocations, and inversions, are common events in genetic evolution [5]. 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.

When an inversion takes place, a segment of DNA is replaced with its reverse complement, meaning that a segment reverses its orientation and each nucleotide is complemented (where nucleotide C (A) is the complement of G (T)).

In this paper we are interested in the approximate string matching problem allowing for non-overlapping inversions. Much work has been done on the closely related sequence alignment problem with inversions. Although no polynomial algorithm is known for the latter problem in its full generality, in the restricted case of non-overlapping inversions polynomial solutions have been proposed. A first solution was given by Schöniger and Waterman [11]. 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, Chen et al. [3] 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. [12] 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

* Corresponding author. Tel.: +39 095 7383053.
*E-mail address:* faro@dmi.unict.it (S. Faro).

0304-3975/$ – see front matter © 2012 Published by Elsevier B.V.
doi:10.1016/j.tcs.2012.06.009

Please cite this article in press as: D. Cantone, et al., Efficient string-matching allowing for non-overlapping inversions, Theoretical Computer Science (2012), doi:10.1016/j.tcs.2012.06.009

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. [2]. 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. [6] 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 performance 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. Additionally, we also provide a variant of our algorithm which has the same complexity in the worst case, but in the average case has a $\mathcal{O}(n)$ time complexity, for $\sigma \geq \frac{5c}{c-1}$ and sufficiently large $m > \sigma^c$, for any constant $c > 1$, where $\sigma$ is the size of the common alphabet.

For the sake of simplicity, throughout the paper we will consider standard match between characters, even when substrings are involved in inversions. The case in which matches of complemented characters must be taken into account could be solved by using a different match table for inverted substrings.

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. A further refinement of our algorithm, which has the same worst-case complexity but is linear on average, is presented in Section 5. Finally we draw our conclusions in Section 6.

## 2. Basic notions and properties

A string $p$ of length $m \geq 0$ is represented as a finite array $p[0 \ldots m-1]$. In such a case we also write $|p| = m$, thus overloading the absolute-value operator $|\cdot|$. 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 by $p^{\mathrm{R}}$ the reversal of $p$, i.e., string $p$ written in reverse order. Notice that $|p| = |p^{\mathrm{R}}|$ and $(p^{\mathrm{R}})^{\mathrm{R}} = p$. Moreover, for any two strings $p$ and $q$, we have that $(p.q)^{\mathrm{R}} = (q^{\mathrm{R}}.p^{\mathrm{R}})$.

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

For a string $p$ and character $c$, we write $occ_p(c)$ to denote the number of occurrences of $c$ in $p$. Occasionally, we will write simply $occ(c)$, to mean $occ_p(c)$, when the string $p$ is clear from the context.

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 $borders(p)$. For instance, given the string $p = abacdaba$, we have that $ab \sqsubseteq p$, $daba \sqsupseteq p$, while $aba$ is a border of $p$. Moreover we have $borders(p) = \{a, aba\}$.

For a set $S$ of strings, we denote by $\|S\|$ the collection of the lengths of the strings belonging to $S$, i.e. $\|S\| =_{\mathrm{Def}} \{|p| : p \in S\}$. For example, given the set $S = \{aba, a, abacd\}$ we have $\|S\| = \{3, 1, 5\}$.

For strings $p$ and $q$, we denote by $\langle p, q\rangle$ the set of all suffixes $s$ of $p$ such that $s^{\mathrm{R}}$ is a suffix of $q$, i.e., $\langle p, q\rangle =_{\mathrm{Def}} \{s : s \sqsupseteq p \text{ and } s^{\mathrm{R}} \sqsupseteq q\}$. For example if $p = abcabab$ and $q = ccababa$ then $\langle p, q\rangle = \{ab, abab\}$.

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 $v \in borders(w)$ or $w \in borders(v)$;*
(b) *if $v, w \in \langle p, q\rangle$ and $|v| \geq |w|$, then $w \in borders(v)$;*
(c) *if $v \in \langle p, q\rangle$ and $w \in borders(v)$, then $w \in \langle p, q\rangle$;*
(d) *if $z$ is the longest string belonging to $\langle p, q\rangle$, then $\langle p, q\rangle = borders(z)$;*
(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

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

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 $v^{\mathrm{R}} \sqsupseteq q$ and $w^{\mathrm{R}} \sqsupseteq q$ we have $v^{\mathrm{R}} \sqsupseteq w^{\mathrm{R}}$. The latter implies $v \sqsubseteq w$, which, together the previously established relation $v \sqsupseteq w$, yields $v \in borders(w)$, proving (a).

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

[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.

Please cite this article in press as: D. Cantone, et al., Efficient string-matching allowing for non-overlapping inversions, Theoretical Computer Science (2012), doi:10.1016/j.tcs.2012.06.009

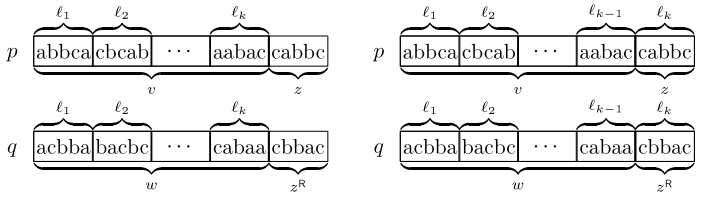


**Fig. 1.** Two cases considered in Lemma 2.

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). □

**Definition 1.** 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, with $k \geq 0$, such that:

(a) $1 \leq \ell_i \leq m$, for $1 \leq i \leq k$;
(b) $\sum_{i=1}^{k} \ell_i = m$;
(b) $p[L_j \ldots L_{j+1} - 1] = (q[L_j \ldots L_{j+1} - 1])^{\mathsf{R}}$, 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 $v \bowtie w$.*

**Proof.** We first show that if either one of (a) and (b) holds, then $p \bowtie q$ follows. If (a) holds, then the empty sequence of lengths is an inverted decomposition of $p$ and $q$, so that we have $p \bowtie q$ in this case. On the other hand, if (b) holds, let $z \neq \varepsilon$ be such that $p = v.z$ and $q = w.z^{\mathsf{R}}$, and let $(\ell_1, \ell_2, \ldots, \ell_k)$ be an inverted decomposition of $v$ and $w$. Plainly, $|p| = |q|$. Let $\ell_{k+1} =_{\text{Def}} |z|$. We show that $(\ell_1, \ell_2, \ldots, \ell_k, \ell_{k+1})$ is an inverted decomposition of $p$ and $q$. Trivially, we have $1 \leq \ell_{k+1} \leq |p|$ and $\sum_{i=1}^{k+1} \ell_i = |p|$. In addition, we have

$$p[|v|, \ldots |v| + \ell_{k+1} - 1] = z = (z^{\mathsf{R}})^{\mathsf{R}} = (q[|w|, \ldots |w| + \ell_{k+1} - 1])^{\mathsf{R}},$$

which, together with the fact that $v \bowtie w$, yields $p \bowtie q$ (see Case 1 in Fig. 1).

Next, suppose that $p \bowtie q$ and that $|p| = |q| > 0$. We show that (b) must hold. Let $(\ell_1, \ell_2, \ldots, \ell_k)$ be an inverted decomposition of $p$ and $q$, with $k \geq 1$, and let $z =_{\text{Def}} p[L_{k-1} \ldots L_k - 1]$, where $L_j =_{\text{Def}} \sum_{i=1}^{j} \ell_i$, for $0 \leq j \leq k$. Then $z^{\mathsf{R}} = q[L_{k-1} \ldots L_k - 1]$, so that $p = v.z$ and $q = w.z^{\mathsf{R}}$, where $v =_{\text{Def}} p[0 \ldots L_{k-1} - 1]$ and $w =_{\text{Def}} q[0 \ldots L_{k-1} - 1]$. To complete the proof of the present case, it is now enough to observe that $z \neq \varepsilon$ (since $|z| = \ell_k \geq 1$) and that $v \bowtie w$ (in fact, it is an easy matter to verify that $(\ell_1, \ell_2, \ldots, \ell_{k-1})$ is an inverted decomposition of $v$ and $w$); see Case 2 in Fig. 1. □

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

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.

Please cite this article in press as: D. Cantone, et al., Efficient string-matching allowing for non-overlapping inversions, Theoretical Computer Science (2012), doi:10.1016/j.tcs.2012.06.009

```
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. 2.** The algorithm DPInversionMatcher for the matching problem with inversions.

## 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 $(i+1)$th iteration, for $i = 0, 1, \ldots, n-m$, our algorithm establishes whether $p_j \bowtie t[i \ldots i+j]$, for each $j = 0, 1, \ldots, m-1$, exploiting information gathered during previous iterations.

**Definition 2.** We denote by $M(j, i)$ the set of all integral values $k$, with $0 \le k \le j$, such that the prefix $p_k$ has an occurrence with inversions at location $i$ of the text, or more formally

$$M(j,i) =_{\text{Def}} \begin{cases} \{0 \le k \le j : p_k \bowtie t[i \ldots i+k]\} \cup \{-1\} & \text{if } i \ge 0 \text{ and } j \ge 0 \\ \{-1\} & \text{otherwise,} \end{cases}$$

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

Observe that we have $M(j, i) = \{-1\}$ if $i < 0$ or $j < 0$ (note that the string $p_k$, with $k < 0$, represents the empty prefix of $p$).

Notice also that $p_j \bowtie t[i \ldots i+j]$ iff $j \in M(j, i)$ and hence $p \bowtie t[i \ldots i+m-1]$ 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 $i$.

**Definition 3.** We 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

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

We obtain the following lemma which easily follows from Lemma 2.

**Lemma 3.** *For each $0 \le j < m$ and $-m \le i \le n-m$, the computation of the set $M(j, i)$ can be reduced to the computation of the sets $M(j-1, i)$ and $R(j, i)$, by using the following recursive relation*

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

Likewise the following lemma follows from Lemma 1(f).

**Lemma 4.** *For each $0 \le j < m$ and $0 < i \le n-m$, the sets $R(j, i)$ can be computed by the recursive relation*

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

The above considerations translate directly into the algorithm DPInversionMatcher in Fig. 2. Sets $R(j, i)$ are maintained by an array R of dimension $m$; more precisely, just after iteration $i$ of the for-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) M, which is initialized to $\{-1\}$ at the beginning of iteration $i$ of the for-loop at line 3 (this corresponds to the set $M(-1, i)$). Then, during the execution of the subsequent for-loop at line 5, the set 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 for-loop at line 5, we have that $\mathsf{M} = M(j, i)$.

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

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

for $0 \le j \le m$. Likewise, each set 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 DPInversionMatcher in Fig. 2 has a $\mathcal{O}(m^2)$ space complexity and a

Please cite this article in press as: D. Cantone, et al., Efficient string-matching allowing for non-overlapping inversions, Theoretical Computer Science (2012), doi:10.1016/j.tcs.2012.06.009

$\mathcal{O}(nm^2)$ time complexity. Indeed, the computation of the set R[$j$] at line 6 and the conditional test at line 7 require $\mathcal{O}(j)$ time, for $0 \leq j < m$.

## 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 7 below, during each iteration of the algorithm DPInversionMatcher, just before the execution of the conditional test at line 7, the following condition holds

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

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

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

which will be proved in Lemma 8 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 a 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 that one knows in advance the set $\|\langle p_j, t_{i+j-1}\rangle\|$. To this purpose let us put

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

For $i = 1, \ldots, k$, let us also put $v_i = p[j+1-\ell_i \ldots j]$.

In words, $\pi(p_j, h)$ is the length of the longest border of a proper suffix of the string $p[h \ldots j]$. For instance if $p[h \ldots j] = acabbaabb$ then the proper suffix of the string, with the longest border is $abbaabb$ with a border of length 3.

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 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 $0 < k \leq m$ and $0 \leq h \leq k$ (the values of the remaining entries of W are not relevant).

Table W can be computed in $\mathcal{O}(m^2)$ time and space by means of the procedure procMPT in Fig. 3, which is a generalization of the procedure used by the Morris–Pratt algorithm [9] for computing the length of the longest proper border of $s[0 \ldots j]$, for a given string $s$ with $0 \leq j < |s|$ (see also [4], where this function is called the *prefix function* of the pattern).

The resulting algorithm, InversionSampling, is presented in Fig. 3. The algorithm makes use of a vector K such that $\mathsf{K}[j] := \max(R(j,i))$. The part of the code from line 7 up to line 10 computes the value of K[$j$].

### 4.1. Correctness issues

In this section we prove the validity of (1) and (2), upon which the correctness of the algorithm InversionSampling is based. In particular, it will turn out that they are direct consequences of Lemmas 7 and 8, 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 5.** *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) $p.q^{\mathsf{R}} = q_1^{\mathsf{R}}.q_2^{\mathsf{R}}.p$.

Please cite this article in press as: D. Cantone, et al., Efficient string-matching allowing for non-overlapping inversions, Theoretical Computer Science (2012), doi:10.1016/j.tcs.2012.06.009

```
procMPT(p, m)
1.  for k := 0 do m − 1 do
2.    i := 0
3.    j := W[k, k] := −1
4.    while (i < (m − k)) do
5.      while ((j > −1) and
          (p[i + k] ≠ p[j + k])) do
6.        j := W[k, k + j]
7.      i := i + 1
8.      j := j + 1
9.      W[k, k + i] := j
10.   return(W)
```

```
InversionSampling(p, m, t, n)
1.  W := procMPT(p, m)
2.  for j := 0 to m − 1 do
3.    K[j] := 0
4.  for i = −m + 1 to n − m do
5.    M = {−1}
6.    for j := max(−i, 0) to m − 1 do
7.      while ((K[j] > 0)
            and (p[j − K[j]] ≠ t[i + j])) do
8.        K[j] := W[j + 1 − K[j], j + 1]
9.      if p[j − K[j]] = t[i + j] then
10.       K[j] := K[j] + 1
11.     if (j − K[j] ∈ M) then
12.       M := M ∪ {j}
13.   if (m − 1) ∈ M then
14.     output(i)
```

**Fig. 3.** (On the left) the procedure for computing the table W, and (on the right) the variant InversionSampling of the algorithm DPInversionMatcher.

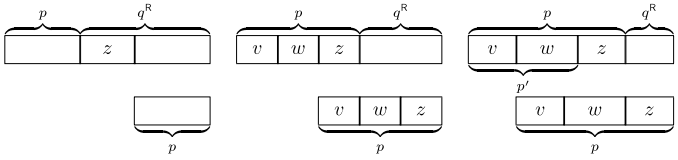


**Fig. 4.** The three cases considered in Lemma 5.

**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 depicted in Fig. 4.

**Case 1:** $|p| \leq |q|$**.** From $p \sqsupseteq p.q^{\mathsf{R}}$ and $|p| \leq |q|$, 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 then have

$$q_1.q_2 = p^{\mathsf{R}}.z^{\mathsf{R}} = (z.p)^{\mathsf{R}} = (q^{\mathsf{R}})^{\mathsf{R}} = q$$

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:** $|q| < |p| \leq 2|q|$**.** Let $z$ be the suffix of $p$ such that $|z| = |p| - |q| \leq |q|$. Observe that $2|z| \leq |z| + |q| = |p|$, so that $|z| \leq \lfloor |p|/2 \rfloor$. 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.

**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,

$$\begin{aligned} p.q^{\mathsf{R}} &= (v.w.z).q^{\mathsf{R}} = (p'.z).q^{\mathsf{R}} = (q_1^{\mathsf{R}}.q_2^{\mathsf{R}}.p').q^{\mathsf{R}} \\ &= q_1^{\mathsf{R}}.q_2^{\mathsf{R}}.(p'.q^{\mathsf{R}}) = q_1^{\mathsf{R}}.q_2^{\mathsf{R}}.(p'.z) = q_1^{\mathsf{R}}.q_2^{\mathsf{R}}.p, \end{aligned}$$

so that (a) and (b) hold in this last case too, completing the proof of the lemma. □

**Lemma 6.** *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 $z$.*

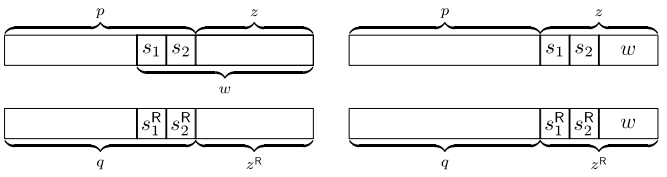


**Fig. 5.** The two cases considered in Lemma 6.

**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. 5, 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 5, 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. 5, 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 5, 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 7.** *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., $p_j \bowtie t[i \ldots i + j]$, so that by Lemma 6, we must have $p_{j-k} \bowtie t[i \ldots i + j - k]$ for all $k \in R(j, i) \setminus \{0\}$. But $p_{j-k} \bowtie t[i \ldots i + j - k]$, 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 us 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 8.** *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. 3 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 $\mathsf{C}_1(i, j)$ 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 $\mathtt{C}_2(i,j) = 1$, if the assignment instruction at line 10 is executed during iteration $i$, otherwise we put $\mathtt{C}_2(i,j) = 0$. Plainly, we have that

$$T(n,m) = \mathcal{O}\left(\sum_{i=-m+1}^{n-m}\sum_{j=0}^{m-1}\big(\mathtt{C}_1(i,j)+1\big)\right), \tag{3}$$

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

Since $\mathtt{C}_2(i,j) \le 1$ for each $(i,j) \in A$, we have that, for $0 \le j < m$,

$$\sum_{i=-m+1}^{n-m} \mathtt{C}_2(i,j) \le n. \tag{4}$$

On the other hand, we have also that

$$K(i+1,j) - \mathtt{C}_2(i+1,j) \le K(i,j) - \mathtt{C}_1(i+1,j), \tag{5}$$

for all $(i,j) \in A$ such that $i < n-m$. Indeed, during iteration $i$, the value contained in $\mathtt{K}[j]$ just after the execution of the while-loop at line 7 (i.e., $K(i+1,j) - \mathtt{C}_2(i+1,j)$) can never exceed the value contained in $\mathtt{K}[j]$ just before this execution minus the number of times that the while-loop iterates (i.e., $K(i,j) - \mathtt{C}_1(i+1,j)$), since $\mathtt{K}[j]$ is decremented at least by one unit during each iteration of the while-loop at line 7. Thus it follows that

$$0 \le K(h,j) \le \sum_{i=-m+1}^{h} \mathtt{C}_2(i,j) - \sum_{i=-m+1}^{h} \mathtt{C}_1(i,j), \tag{6}$$

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

$$\sum_{j=0}^{m-1}\sum_{i=-m+1}^{n-m}\big(\mathtt{C}_1(i,j)+1\big) \le \sum_{j=0}^{m-1}\sum_{i=-m+1}^{n-m}\big(\mathtt{C}_2(i,j)+1\big),$$

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

$$\sum_{j=0}^{m-1}\sum_{i=-m+1}^{n-m}\big(\mathtt{C}_1(i,j)+1\big) \le (n+1)m,$$

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

In the following section we present an efficient variant of our algorithm InversionSampling that has the same $\mathcal{O}(nm)$ worst-case time complexity, but exhibits a linear-time complexity on the average case.

## 5. The InversionFilter&Sample algorithm

The new algorithm, named InversionFilter&Sample algorithm, improves the searching strategy introduced in the InversionSampling algorithm by making use of an efficient filter method recently used in [6] by Grabowsky et al. in the case of the string matching problem allowing for inversions and translocations. Such a filter technique, usually referred to as the *counting filter*, is known in the literature [7,8,1] and has been used for the $k$-mismatches and $k$-differences string matching problem.

The idea behind this filtering technique is straightforward and is based upon the observation that (in our problem) if a pattern $p$ has an occurrence (possibly involving inversions) starting at position $s$ of a text $t$, then the $|p|$-substring $t[s \ldots s+|p|-1]$ of the text is a permutation of the pattern $p$.

Here we follow much the same line of arguments as presented in [6].

As above, we assume that $p$ and $t$ are strings of length $m$ and $n$, respectively, over a common alphabet $\Sigma = \{c_0, \ldots, c_{\sigma-1}\}$. Additionally, we assume that $\sigma = \mathcal{O}(n)$.

The InversionFilter&Sample algorithm firstly identifies the set $\Gamma_{p,t}$ of all candidate positions $s$ in the text such that the substring $t[s \ldots s+m-1]$ is a permutation of the characters in $p$ and, subsequently, for each such candidate position $s \in \Gamma_{p,t}$, it executes a verification procedure to check whether $p$ and $t[s \ldots s+m-1]$ match, up to non-overlapping inversions.

Specifically, for each position $0 \le s \le n-m$, we define the function $G_s : \Sigma \to Z$, by putting

$$G_s(c) =_{\text{Def}} occ_p(c) - occ_{t(s,m)}(c),$$

for $c \in \Sigma$ and where $t(s,m)$ denotes the substring $t[s \ldots s+m-1]$.

We also define, for each position $s$, the distance value $\delta_s$ as follows

$$\delta_s =_{\text{Def}} \delta(p, t_s) = \sum_{c\in\Sigma} |occ_p(c) - occ_{t(s,m)}(c)| = \sum_{c\in\Sigma} |G_s(c)|.$$

Please cite this article in press as: D. Cantone, et al., Efficient string-matching allowing for non-overlapping inversions, Theoretical Computer Science (2012), doi:10.1016/j.tcs.2012.06.009

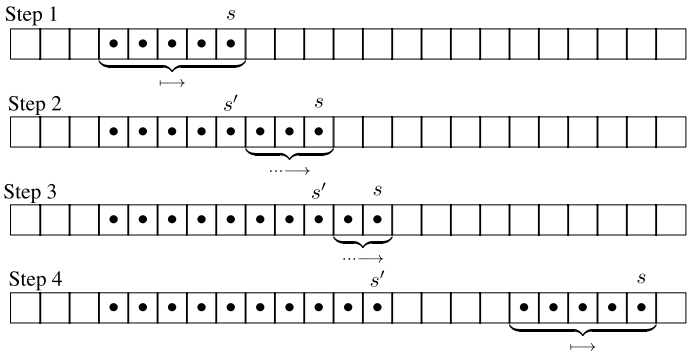


**Fig. 6.** Four working steps of the InversionFilter&Sample algorithm for a pattern of length $m = 5$. Sampled positions are identified by bullets. Steps 1 and 4: a candidate position $s$ is found and a new sampling is started from position $s - m + 1$. Steps 2 and 3: a new candidate position $s$ is found and the sampling is continued from the previous candidate position $s'$.

Then it is easy to see that the set $\Gamma_{p,t}$ of all candidate positions in the text can be defined as

$$\Gamma_{p,t} =_{\text{Def}} \{s : 0 \le s \le n - m \text{ and } \delta_s = 0\}.$$

Observe that values $\delta_{s+1}$ and $\delta_s$ can differ only in the number of occurrences of characters $t[s]$ and $t[s + m]$. Thanks to this property, $G_{s+1}(c)$ can be computed in constant time from $G_s(c)$ and, similarly, $\delta_{s+1}$ can be computed from $\delta_s$ in constant time. It follows that $\mathcal{O}(n)$ time is needed for computing all values $\delta_s$ for $s = 0, \ldots, n - m$.

When a candidate position $s$ (such that $\delta_s = 0$) is found, a verification procedure is run. The verification procedure used by the InversionFilter&Sample algorithm is named Sample and is based on the InversionSampling algorithm.

Roughly speaking the Sample procedure, when called for a particular position $s$ of the text, executes the InversionSampling algorithm in order to search for occurrences of the pattern starting from position $s - m + 1$ to position $s$ of the text, provided that none of such positions have been already sampled in a previous verification phase (it consists in searching $t[s - m + 1 \ldots s + m - 1]$ for occurrences of $p$).

When a verification phase stops, the algorithm remembers the last sampled position $s'$ of the text ($s'$ is initialized to $-m$) and stores the last values of the vector $\mathsf{K}[j]$, for $j = 0, \ldots, m - 1$. When starting a new verification phase at position $s$ of the text, the algorithm checks whether position $s'$ is less than $s - m$. If this is the case, the algorithm restarts a new sample phase from position $s - m + 1$, so that positions from $s'$ to $s - m$ are not verified (steps 1 and 4 of Fig. 6). Otherwise the previous sampling is continued from position $s' + 1$ (steps 2 and 3 of Fig. 6). In both cases, the sample phase stops at position $s$ of the text. Then the filter phase restarts from position $s + 1$.

Fig. 7 shows the code of the InversionFilter&Sample algorithm and its verification phase.

Approximate occurrences of the pattern are reported only during the sample phase of the algorithm.

A single run of the Sample procedure takes at most $\mathcal{O}(m^2)$ time, as for a pattern $p$ of length $m$ and a text $t$ of length $n$, the InversionSampling algorithm has a $\mathcal{O}(mn)$ worst-case time complexity. However, since each text position is sampled at most once by the Sample procedure, in the worst case the verification phase of the InversionFilter&Sample algorithm performs exactly the same amount of work of the DPInversionMatcher algorithm. Thus the overall complexity due to the verification phase is at most $\mathcal{O}(nm)$. Moreover, as remarked above, the filtering phase takes at most $\mathcal{O}(n)$ time for scanning the whole text. We conclude that the total worst-case time complexity of the InversionFilter&Sample algorithm is $\mathcal{O}(nm) + \mathcal{O}(n) = \mathcal{O}(nm)$.

In the next section we show that, despite its quadratic worst-case time complexity, the InversionFilter&Sample algorithm shows a linear behavior on average.

### 5.1. Average-case time analysis

In the following analysis we assume the uniform distribution and independence of characters.

The verification procedure takes at most $\mathcal{O}(m^2)$ (worst-case) time per location. Thus, to obtain a linear average-time bound, it is enough to bound with $\mathcal{O}(1/m^2)$ the probability of finding permuted subsequences of length $m$.

In [6], Grabowsky et al. proved this bound for $m = \omega(\sigma^{\mathcal{O}(1)})$ and $\sigma = \Omega(\log m / \log\log^{1-\varepsilon} m)$. Much the same analysis could be carried out also in our case. However, in the following we prove a somewhat slightly stronger result. More precisely, we derive a linear average-time bound for our algorithm, for sufficiently large $m > \sigma^c$ and *any* alphabet size $\sigma \ge \frac{5c}{c-1}$, for any given constant $c > 1$.

Let $\Pr\{s \in \Gamma_{p,t}\}$ be the probability that a given position $s$ is a candidate position to be verified. Since the preprocessing and a single verification phase of the algorithm take $\mathcal{O}(m^2)$ time, the average time complexity of the InversionFilter&Sample algorithm can be expressed as

$$T(n, m, \sigma) = \mathcal{O}(m^2) + \sum_{s=0}^{n-m} \Pr\{s \in \Gamma_{p,t}\} \cdot \mathcal{O}(m^2). \tag{7}$$

Please cite this article in press as: D. Cantone, et al., Efficient string-matching allowing for non-overlapping inversions, Theoretical Computer Science (2012), doi:10.1016/j.tcs.2012.06.009

```
InversionFilter&Sample (p, m, t, n)
 1.  W := procMPT(p, m)
 2.  for c ∈ Σ do G[c] ← 0
 3.  for s ← 0 to m − 1 do
 4.    G[p[s]] ← G[p[s]] + 1
 5.    G[t[s]] ← G[t[s]] − 1
 6.  δ ← 0
 7.  for c ∈ Σ do δ ← δ + |G[c]|
 8.  s' := −m
 9.  for s ← 0 to n − m do
10.    if δ = 0 then
11.      K := Sample(p, m, t, n, s', s, K, W)
12.      s' := s
13.    a ← t[s]
14.    b ← t[s + m]
15.    δ ← δ − |G[a]| − |G[b]|
16.    G[a] ← G[a] + 1
17.    G[b] ← G[b] − 1
18.    δ ← δ + |G[a]| + |G[b]|
19.  if δ = 0 then
20.    K := Sample(p, m, t, n, s', n − m, K, W)

Sample(p, m, t, n, s', s, K, W)
 1.  if (s' < s − m + 1) then
 2.    for j := 0 to m − 1 do K[j] := 0
 3.    s' := s − m
 4.  for i := s' + 1 to s do
 5.    M = {−1}
 6.    for j := max(−i, 0) to m − 1 do
 7.      while ((K[j] > 0) and (p[j − K[j]] ≠ t[i + j])) do
 8.        K[j] := W[j + 1 − K[j], j + 1]
 9.      if p[j − K[j]] = t[i + j] then
10.        K[j] := K[j] + 1
11.      if (j − K[j] ∈ M) then
12.        M := M ∪ {j}
13.    if (m − 1) ∈ M then
14.      output(i)
```

**Fig. 7.** Top: the InversionFilter&Sample algorithm for the approximate string matching problem with inversions. Bottom: the verification procedure based on the InversionSampling algorithm.

Thus, let $m > \sigma^c$, for some constant $c > 1$. Without loss of generality, we assume that $\sigma$ divides $m$ and we put $\alpha =_{\text{Def}} m/\sigma$. For each text position $s$, with $0 \le s \le n - m$, the probability that the $m$-substring of the text, beginning at position $s$, is a permutation of the pattern $p$ is exactly

$$\begin{aligned}\Pr\{s \in \Gamma_{p,t}\} &= \frac{\binom{m}{occ(c_0)}\binom{m-occ(c_0)}{occ(c_1)}\binom{m-occ(c_0)-occ(c_1)}{occ(c_2)}\cdots\binom{occ(c_{\sigma-1})}{occ(c_{\sigma-1})}}{\sigma^m} \\ &= \frac{m!}{\sigma^m \cdot occ(c_0)! \cdot occ(c_1)! \cdot \cdots \cdot occ(c_{\sigma-1})!}. \end{aligned} \tag{8}$$

Expression (8) is maximized when $occ(c_i) = \alpha$, for all $i = 0, \ldots, \sigma - 1$. Thus, we have:

$$\Pr\{s \in \Gamma_{p,t}\} \le \frac{m!}{(\alpha!)^\sigma \sigma^m}.$$

We make use of Stirling's approximation for $m!$ and $\alpha!$, where we recall that $\alpha = m/\sigma$:

$$\frac{m!}{(\alpha!)^\sigma \sigma^m} = \Theta\left(\frac{\sqrt{2\pi m}(m/e)^m}{(\sqrt{2\pi m/\sigma}(m/(e\sigma))^{m/\sigma})^\sigma \sigma^m}\right) = \Theta\left(\frac{\sqrt{2\pi m}}{\left(\sqrt{2\pi(m/\sigma)}\right)^\sigma}\right).$$

Since $\sqrt{2\pi}/(\sqrt{2\pi})^\sigma \le 1$, we have:

$$\Theta\left(\frac{\sqrt{2\pi m}}{\left(\sqrt{2\pi}\,(m/\sigma)\right)^{\sigma}}\right) = \Theta\left(\frac{\sqrt{m}}{\left(\sqrt{m/\sigma}\right)^{\sigma}}\right) = \Theta\left(\frac{\sigma^{\sigma/2}}{m^{(\sigma-1)/2}}\right).$$

If we take $\sigma \geq \frac{5c}{c-1}$, then $\sigma \leq c(\sigma-5)$, so that

$$\sigma^{\sigma} \leq \sigma^{c(\sigma-5)} \leq m^{\sigma-5}.$$

Therefore we get

$$\frac{\sigma^{\sigma/2}}{m^{(\sigma-1)/2}} \leq \frac{1}{m^2},$$

since, plainly,

$$m^{(\sigma-5)/2} = \frac{m^{(\sigma-1)/2}}{m^2}.$$

Thus, we have

$$\Pr\{s \in \Gamma_{p,t}\} = \mathcal{O}(1/m^2), \tag{9}$$

for any $\sigma \geq \frac{5c}{c-1}$ and sufficiently large $m > \sigma^c$, with $c > 1$.

The overall average-time complexity of the InversionFilter&Sample algorithm given in (7), assuming $\sigma \geq \frac{5c}{c-1}$, is equal to

$$\begin{aligned} T(n,m,\sigma) &= \mathcal{O}(m^2) + \sum_{s=0}^{n-m} \Pr\{s \in \Gamma_{p,t}\} \cdot \mathcal{O}(m^2) \\ &= \mathcal{O}(m^2) + (n-m+1) \cdot \mathcal{O}(1/m^2) \cdot \mathcal{O}(m^2) = \mathcal{O}(m^2+n), \end{aligned}$$

which, in view of (9), yields $T(n, m, \sigma) = \mathcal{O}(m^2 + n)$.

## 6. Conclusions and future work

In this paper we have presented two efficient algorithms to solve the pattern matching problem under a string distance which allows inversions of non-overlapping factors. The first 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. The second algorithm, named InversionFilter&Sample, has the same worst-case complexity as the InversionSampling algorithm, but shows a $\mathcal{O}(n)$ average-case time complexity for $\sigma \geq \frac{5c}{c-1}$ and sufficiently large $m > \sigma^c$, with $c > 1$.

We are currently working on an efficient variant of the present algorithms with a sublinear average time complexity.

## Acknowledgments

The authors would like to thank two anonymous referees for their helpful comments.

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

## References

[1] R.A. Baeza-Yates, G. Navarro, New and faster filters for multiple approximate string matching, Random Structures Algorithms 20 (1) (2002) 23–49.
[2] D. Cantone, S. Faro, E. Giaquinta, Approximate string matching allowing for inversions and translocations, in: Jan Holub, Jan Zdárek (Eds.), Proceedings of the Prague Stringology Conference 2010, Czech Technical University, Prague, Czech Republic, 2010, pp. 37–51.
[3] Z. Chen, Y. Gao, G. Lin, R. Niewiadomski, Y. Wang, J. Wu, A space-efficient algorithm for sequence alignment with inversions and reversals, Theoret. Comput. Sci. 325 (3) (2004) 361–372.
[4] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, second edn, MIT Press, 2001.
[5] K.M. Devos, M.D. Atkinson, C.N. Chinoy, H.A. Francis, R.L. Harcourt, R.M.D. Koebner, C.J. Liu, P. Masojć, D.X. Xie, M.D. Gale, Chromosomal rearrangements in the rye genome relative to that of wheat, TAG Theoretical and Applied Genetics 85 (1993) 673–680.
[6] S. Grabowski, S. Faro, E. Giaquinta, String matching with inversions and translocations in linear average time (most of the time), Inform. Process. Lett. 111 (11) (2011) 516–520.
[7] R. Grossi, F. Luccio, Simple and efficient string matching with $k$ mismatches, Inf. Process. Lett. 33 (3) (1989) 113–120.
[8] P. Jokinen, J. Tarhio, E. Ukkonen, A comparison of approximate string matching algorithms, Softw. Pract. Exp. 26 (12) (1996) 1439–1458.
[9] J.H. Morris Jr., V.R. Pratt, A linear pattern-matching algorithm, Report 40, University of California, Berkeley, 1970.
[10] G. Navarro, A guided tour to approximate string matching, ACM Comput. Surv. 33 (1) (2001) 31–88.
[11] M. Schöniger, M. Waterman, A local algorithm for DNA sequence alignment with inversions, Bull. Math. Biol. 54 (1992) 521–536.
[12] A.F. Vellozo, C.E.R. Alves, A. Pereira do Lago, Alignment with non-overlapping inversions in $\mathcal{O}(n^3)$-time, in: P. Bucher, B.M.E. Moret (Eds.), Proceedings of the 6th International Workshop in Algorithms in Bioinformatics, WABI 2006, in: Lecture Notes in Computer Science, vol. 4175, Springer, 2006, pp. 186–196.

Please cite this article in press as: D. Cantone, et al., Efficient string-matching allowing for non-overlapping inversions, Theoretical Computer Science (2012), doi:10.1016/j.tcs.2012.06.009