Maximal-exponent factors in strings☆

Golnaz Badkobeh[a], Maxime Crochemore[a,b,*]

[a] *King's College London, UK*
[b] *Université Paris-Est, France*

Abstract

The exponent of a string is the quotient of its length over its smallest period. The exponent and the period of a string can be computed in time proportional to the string length. We design an algorithm to compute the maximal exponent of all factors of an overlap-free string. Our algorithm runs in linear-time on a fixed-size alphabet, while a naive solution of the question would run in cubic time. The solution for non overlap-free strings derives from algorithms to compute all maximal repetitions, also called runs, occurring in the string.

We also show there is a linear number of occurrences of maximal-exponent factors in an overlap-free string. Their maximal number lies between $0.66n$ and $2.25n$ in a string of length n. The algorithm can additionally locate all of them in linear time.

Keywords: Word, string, repetition, power, repeat, periodicity, string exponent, return word, algorithm, automaton.
2000 MSC: 68W40, 68R15, 68Q45

1. Introduction

We consider the question of computing the maximal exponent of factors (substrings) of a given string. The exponent of a word is the quotient of

☆An extended abstract of a preliminary version of the article was presented at SPIRE'2012 [14].

*Corresponding author
Email address: `maxime.crochemore@kcl.ac.uk` (Maxime Crochemore)
URL: `http://www.inf.kcl.ac.uk/pg/badkobeh/` (Golnaz Badkobeh), `http://www.inf.kcl.ac.uk/staff/mac/` (Maxime Crochemore)

the string length over the string smallest period. For example `alfalfa` has period 3 and exponent 7/3, and `restore` has period 5 and exponent 7/5. A string with exponent e is also called an e-power. The exponent indicates better than the period the degree of repetitiveness of factors occurring in a string.

Factors considered in this article are of exponent at most 2. They refer to strings of the form uvu where u is the longest border (both a prefix and a suffix) of uvu. In other words, the factor u repeats at two distant positions. The study of repeats in a string is relevant to long-distance interactions between separated occurrences of the same segment (the u part) in the string. Although occurrences may be far away from each other, they may interact when the string is folded as it is the case for genomic sequences. A very close problem to considering those repeats is that of computing maximal pairs (positions of the two occurrences of u) with gaps constraints as described by Gusfield [1] and later improved by Brodal et al. [2].

From a combinatorial point of view, the question is related to return words: z is a return word associated with u if u is a prefix of zu and u has no internal occurrence in zu. For instance, if u has no internal occurrence in uvu then uv is a return word for u. The word then links two consecutive occurrences of u. The analysis of return words provide characterisations for word morphisms and infinite words. For example, a binary infinite Sturmian word, generalisation of Fibonacci word, is characterised by the fact that every factor (occurring infinitely many times) admits exactly two return words (see [3] and references therein).

The notion of maximal exponent is central to questions related to the avoidability of powers in infinite words. An infinite word is said to avoid e-powers (resp. e^+-powers) if the exponents of its finite factors are smaller than e (resp. no more than e). Dejean [4] introduced the repetitive threshold $\mathrm{RT}(a)$ of an a-letter alphabet: the smallest rational number for which there exists an infinite word on a letters whose finite factors have exponent at most $\mathrm{RT}(a)$. In other words, the maximal exponent of factors of such a word is $\mathrm{RT}(a)$, the minimum possible. The word is also said to be $\mathrm{RT}(a)^+$-power free. It is known from Thue [5] that $r(2) = 2$, Dejean [4] proved that $r(3) = 7/4$ and stated the exact values of $\mathrm{RT}(a)$ for every alphabet size $a > 3$. Dejean's conjecture was eventually proved in 2009 after partial proofs given by several authors (see [6, 7] and references therein).

The exponent of a string can be calculated in linear time using basic string matching that computes the smallest period associated with the longest border of the string (see [8]). A straightforward consequence provides a $O(n^3)$-time solution to compute the maximal exponent of all factors of a string of length n since there are potentially of the order of n^2 factors. However, a quadratic time solution is also a simple application of basic string matching. By contrast, our solution runs in linear time on a fixed-size alphabet.

When a string contains runs, that is, maximal periodicities of exponent at least 2, computing their maximal exponent can be done in linear time by adapting the algorithm of Kolpakov and Kucherov [9] that computes all the runs occurring in the string. Their result relies on the fact there exists a linear number of runs in a string [9] (see [10, 11] for precise bounds). Nevertheless, this does not apply to square-free strings, which we are considering here.

Our solution works indeed on overlap-free strings, not only on square-free strings, that is, on strings whose maximal exponent of factors is at most 2. Thus, we are looking for factors w of the form uvu, where u is the longest border of w. In order to accomplish this goal, we exploit two main tools: the Suffix Automaton of some factors and a specific factorisation of the whole string.

The Suffix Automaton (see [8]) is used to search for maximal-exponent factors in a product of two strings due to its ability to locate occurrences of all factors of a pattern. Here, we enhance the automaton to report the right-most occurrences of those factors. Using it solely in a balanced divide-and-conquer manner produces a $O(n \log n)$-time algorithm.

To remove the log factor we additionally exploit a string factorisation, namely the f-factorisation (see [8]), a type of LZ77 factorisation (see [12]) fit for string algorithms. It has now become common to use it to derive efficient or even optimal algorithms. The f-factorisation, allows to skip larger and larger parts of the strings during an online computation. For our purpose, it is composed of factors occurring before their current position with no overlap. The factorisation can be computed in $O(n \log a)$-time using a Suffix Tree or a Suffix Automaton, and in linear time on an integer alphabet using a Suffix Array [13].

The running time of the proposed algorithm depends additionally on the repetitive threshold of the underlying alphabet size of the string. The threshold restricts the context of the search for a second occurrence of u associated with a factor uvu.

We show a very surprising property of factors whose exponent is maximal in an overlap-free string: there are no more than a linear number of occurrences of them, although the number of occurrences of maximal (i.e.

non-extensible) factors can be quadratic.

We show a lower bound of $0.66n$ and an upper bound of $2.25n$ on their maximal number for a string of length n. They improve on the bounds given in a preliminary version [14] of the article. The lower bound is based on a result of Pansiot [15] on the repetitive threshold of four-letter alphabets.

As a consequence, the algorithm can be modified to output all occurrences of maximal-exponent factors of an overlap-free string in linear time.

The question would have a simple solution by computing MinGap on each internal node of the Suffix Tree of the input string, as is discussed in the conclusion. MinGap of a node is the smallest difference between the positions assigned to leaves of the subtree rooted at the node. Unfortunately, the best algorithms for MinGap computation, equivalent to MaxGap computation, run in time $O(n \log n)$ (see [16, 17, 18]) and the discussion in [19]).

A remaining question to the present study is to unify the algorithmic approaches for locating runs in non overlap-free strings and maximal-exponent factors in overlap-free strings.

The plan of the article is as follows. After defining the problem in the next section we present the general scheme of the algorithm that relies on the f-factorisation of the input string in Section 3. The sub-function operating a Suffix Automaton is described in Section 4 and the complexity of the complete algorithm is studied in Section 5. In Section 6 we prove lower and upper bounds on the number of occurrences of maximal-exponent factors. A conclusion follows.

2. Maximal-exponent factors

We consider strings (words) on a finite alphabet A of size a. If x is a string of length $|x| = m$, $x[i]$ denotes its letter at position i, $0 \le i < m$. A factor of x is of the form $x[i]x[i+1]\dots x[j]$ for two positions i and j and is denoted by $x[i\,..\,j]$ (it is the empty word if $j < i$). It is a prefix of x if $i = 0$ and a suffix of x if $j = m - 1$.

The string x has period p, $0 < p \le m$, if $x[i] = x[i + p]$ whenever both sides of the equality are defined. The period of x, $\text{period}(x)$, is its smallest period and its exponent is $\exp(x) = m/\text{period}(x)$. For example, $\exp(\texttt{restore}) = 7/5$, $\exp(\texttt{mama}) = 2$ and $\exp(\texttt{alfalfa}) = 7/3$. An overlap-free string contains no factor of exponent larger then 2, that is, no factor of the form $bwbwb$ for a letter b and a string w.

We consider a fixed overlap-free string y of length n and deal with its factors having the maximal exponent among all factor exponents. They are called **maximal-exponent factor** or MEF for short. They have exponent at most 2 since y is overlap-free.

A MEF w in y is of the form uvu, where u is its longest border (longest factor that is both a prefix and a suffix of w). Then $\text{period}(w) = |uv|$ and $\exp(w) = |uvu|/|uv| = 1 + |u|/\text{period}(w)$. By convention, in the following we allow a border-free factor to be considered as a MEF of exponent 1, though it contains no repeat in the common sense since the repeating element u is empty and it can appear only if the length of y is smaller than the alphabet size.

First note that a MEF uvu contains no occurrence of u since this would produce a factor with a larger exponent. Second, any occurrence of the MEF uvu is maximal in the sense that it cannot be extended with the same period. That is, the two occurrences of u are followed by two distinct letters and preceded by two distinct letters. These remarks are stated in Lemmas 3 and 2 respectively.

The maximality of occurrences of repetitions in non overlap-free strings implies their linear number but unfortunately this property does not hold for factors occurrences.

3. Computing the maximal exponent of factors

The core result of the article is an algorithm, MaxExpFac, that computes the maximal exponent of factors of the overlap-free string y. The algorithm has to look for factors of the form uvu, for two strings u and v, u being the longest border of uvu. The aim of this algorithm is accomplished with the help of Algorithm MaxExp, designed in the next section, which detects those factors occurring within the concatenation of two strings.

Algorithm MaxExpFac relies on the f-factorisation of y (see [8]), a type of LZ77 factorisation [12] defined as follows. It is a sequence of non-empty strings, $z_1, z_2, \ldots, z_k$, called phrases and satisfying $y = z_1 z_2 \cdots z_k$ where z_i is the longest prefix of $z_i z_{i+1} \cdots z_k$ occurring in $z_1 z_2 \cdots z_{i-1}$. When this longest prefix is empty, z_i is the first letter of $z_i z_{i+1} \cdots z_k$, thus it is a letter that does not occur previously in y. We adapt the factorisation to the purpose of our problem by defining z_1 as the longest prefix of y in which no letter occurs more than once. Then, $|z_1| \le a$ and $\text{MaxExpFac}(z_1) = 1$. Note that $\text{MaxExpFac}(z_1 z_2) > 1$ if $z_1 \neq y$.

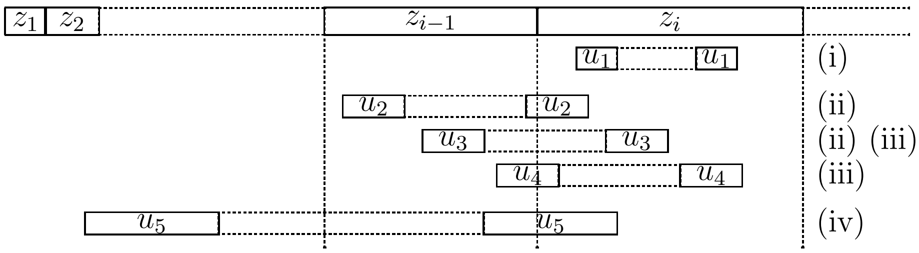


Figure 1: The only four possible locations of a factor uvu involving phrase z_i of the factorisation of the string: (i) internal to z_i; (ii) the first occurrence of u is internal to z_{i-1}; (iii) the second occurrence of u is internal to z_i; (iv) the second occurrence of u is internal to $z_{i-1}z_i$.

When the factorisation of y is computed, Algorithm MaxExpFac processes the phrases sequentially, from z_2 to z_k. After z_1, z_2, ..., z_{i-1} have been processed, the variable e stores the maximal exponent of factors of $z_1 z_2 \cdots z_{i-1}$. Then, the next factors to be considered are those involving phrase z_i. Such a factor uvu can either be internal to z_i or involve other phrases. However, the crucial property of the factorisation is that the second occurrence of u is only to be searched for in $z_{i-1}z_i$ because it cannot contain a phrase as this would contradict the definition of the factorisation.

We further distinguish four possible cases according to the position of the factor uvu as follows (see Figure 1):

(i) The two occurrences of u are contained in z_i.
(ii) The first occurrence of u is contained in z_{i-1} and the second ends in z_i.
(iii) The first occurrence of u starts in z_{i-1} and the second is contained in z_i.
(iv) The first occurrence of u starts in $z_1 \cdots z_{i-2}$ and the second is contained in $z_{i-1}z_i$.

Case (i) needs no action and other cases are dealt with calls to Algorithm MaxExp as described in the code below where $\widetilde{x}$ denotes the reverse of string x. For any two strings z and w and a positive rational number e, MaxExp(z, w, e) is the maximal exponent of factors in zw whose occurrences start in z and end in w, and whose exponent is at least e; it is e itself if there is no such factor.

MaxExpFac(y)

```
1  (z_1, z_2, ..., z_k) ← f-factorisation of y
2  ▷ z_1 is the longest prefix of y in which no letter repeats
3  e ← 1
4  for i ← 2 to k do
5       e ← max{e, MaxExp(z_{i-1}, z_i, e)}
6       e ← max{e, MaxExp(~z_i, ~z_{i-1}, e)}
7       if i > 2 then
8            e ← max{e, MaxExp(~(z_{i-1} z_i), ~(z_1 ··· z_{i-2}), e)}
9  return e
```

Note that variable e can be initialised to the repetitive threshold $\mathrm{RT}(a)$ of the alphabet of string y if the string is long enough. The maximal length of words containing no factor of exponent at least $\mathrm{RT}(a)$ is 3 for $a = 2$, 38 for $a = 3$, 121 for $a = 4$, and $a + 1$ for $a \geq 5$ (see [4]).

Another technical remark is that the instruction at line 6 can be tuned to deal only with type (iii) factors of the form $u_4 v u_4$ (see Figure 1), i.e. factors for which the first occurrence of the border starts in z_{i-1} and ends in z_i, because line 5 finds those of the form $u_3 v u_3$. However, this has no influence on the asymptotic running time.

Theorem 1. *For any overlap-free string input,* MaxExpFac *computes the maximal exponent of factors occurring in the string.*

Proof. We consider a run of MaxExpFac(y). Let $e_1, e_2, \ldots, e_k$ be the successive values of the variable e, where e_i is the value of e just after the execution of lines 5–8 for index i. The initial value $e_1 = 1$ is the maximal exponent of factors in z_1 as a consequence of its definition. We show that e_i is the maximal exponent of factors occurring in $z_1 z_2 \cdots z_i$ if e_{i-1} is that of $z_1 z_2 \cdots z_{i-1}$, for $2 \leq i \leq k$.

To do so, since e_i is at least e_{i-1} (use of max at lines 5–8), all factors occurring in $z_1 z_2 \cdots z_{i-1}$ are taken into account and we only have to consider factors coming from the concatenation of $z_1 z_2 \cdots z_{i-1}$ with z_i, that is, factors of the form uvu where the second occurrence of u ends in z_i. As discussed above and illustrated in Figure 1, only four cases are to be considered because the second occurrence of u cannot start in $z_1 z_2 \cdots z_{i-2}$ without contradicting the definition of z_{i-1}.

Line 5 deals with Case (ii) by the definition of MaxExp. Similarly, line 6 is for Case (iii), and line 8 for Case (iv).

If a factor occurs entirely in z_i, Case (i), by the definition of z_i it occurs also in $z_1 z_2 \cdots z_{i-1}$, which is reported by e_{i-1}.

Therefore, all relevant factors are considered in the computation of e_i, which is then the maximal exponent of factors occurring in $z_1 z_2 \cdots z_i$. This implies that e_k, returned by the algorithm, is that of $z_1 z_2 \cdots z_k = y$ as stated.
■

4. Locating repeats in a product

In this section, we describe Algorithm MaxExp applied to (z, w, e) for computing the maximal exponent of factors in zw that end in w, whose left border occurs in z, and whose exponent is at least e. MaxExp is called in the main algorithm of previous section.

To locate factors under consideration, the algorithm examines positions j on w and for each computes the longest potential border of a factor, a longest suffix u of $zw[0\,..\,j]$ occurring in z. The algorithm is built upon an algorithm that finds all of them using the Suffix Automaton of string z and described in [8, Section 6.6]. After u is found, some of its suffixes may have an exponent higher than e, but the next lemmas show we can discard many of them.

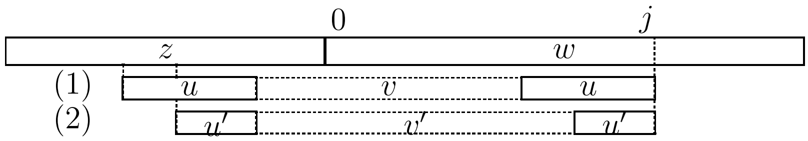


Figure 2: When u and its suffix u' end at the same rightmost position on z, factor (1) has a larger exponent than factor (2).

Figure 2 illustrates the proof of the following lemma.

Lemma 2. *Let u' be a suffix of u. If they are both associated with the same state of $\mathcal{S}(z)$ the maximal exponent of a $u'v'u'$ is not greater than the maximal exponent of its associated uvu factor.*

Proof. The hypothesis implies that u and u' ends at the same positions in z, therefore they end at the same rightmost position (see Figure 2). Then, $u'v'u'$ and uvu have the same period but since u' is not longer than u, the exponent of $u'v'u'$ is not greater than that of uvu. ■

Note that a suffix u' of u may have an internal occurrence in uvu, which would lead to a factor having a larger exponent. For example, let $z = \texttt{abadba}$ and $w = \texttt{cdaba}$. The factor $\texttt{abadbacdaba}$ with border $\texttt{aba}$ has exponent $11/8$ while the suffix $\texttt{ba}$ of $\texttt{aba}$ infers the factor $\texttt{bacdaba}$ of greater exponent $7/5$.

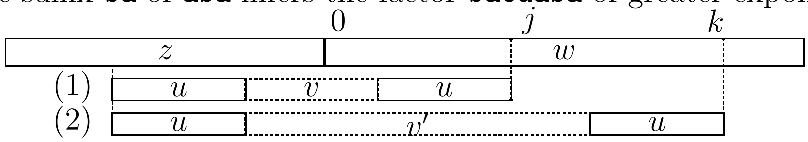


Figure 3: Factor (1) ending at position j has a larger exponent than factor (2) ending at position $k > j$.

The proof of the following lemma can be deduced from the remark in Figure 3.

Lemma 3. *If u occurs at end positions j and k on w with $k > j$, the factor $uv'u$ ending at k cannot be a maximal-exponent factor.*

Proof. To have a maximal exponent the first occurrence of u in $uv'u$ should end at the right-most position on z. But then there is a factor sharing the same first occurrence of u and with a closer second occurrence of u (see Figure 3). Therefore $1 + |u|/|uv| > 1 + |u|/|uv'|$, which proves the statement. ■

The above properties are used by Algorithm MaxExp to avoid some exponent calculations as follows. Let uvu be a factor ending at j on $zw[0\,..\,j]$ and for which u is the longest string associated with state $q = \text{goto}(\text{initial}(\mathcal{S}), u)$. Then next occurrences of u and of any of its suffixes cannot produce factors with an exponent larger than that of uvu. State q is then marked to inform the next steps of the algorithm that it has been visited.

We use the Suffix Automaton of z (minimal automaton that recognises the set of all suffixes of z), denoted $\mathcal{S}(z)$, to locate borders of factors. An example is given in Figure 4. The data structure contains the failure link F_z and the length function L_z both defined on the set of states. The link is defined as follows: let $p = \text{goto}(\text{initial}(\mathcal{S}(z)), x)$ for $x \in A^+$; then $F_z(p) = \text{goto}(\text{initial}(\mathcal{S}(z)), x')$, where x' is the longest suffix of x for which this latter state is not p. As for the length function, $L_z(p)$ is the maximal length of strings x for which $p = \text{goto}(\text{initial}(\mathcal{S}(z)), x)$.

We need another function, sc_z, defined on states of $\mathcal{S}(z)$ as follows: $sc_z(p)$ is the minimal length of paths from p to a terminal state; in other terms, if

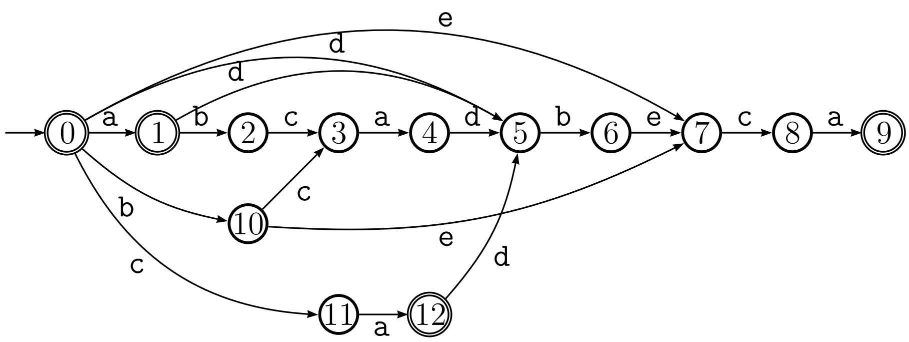


Figure 4: Suffix Automaton of `abcadbeca`. Suffix links: $F[1] = 0$, $F[2] = 10$, $F[3] = 11$, $F[4] = 1$, $F[5] = 0$, $F[6] = 10$, $F[7] = 0$, $F[8] = 11$, $F[9] = 12$, $F[10] = 0$, $F[11] = 0$, $F[12] = 1$. Maximal incoming string lengths: $L[0] = 0$, $L[1] = 1$, $L[2] = 2$, $L[3] = 3$, $L[4] = 4$, $L[5] = 5$, $L[6] = 6$, $L[7] = 7$, $L[8] = 8$, $L[9] = 9$, $L[10] = 1$, $L[11] = 1$, $L[12] = 2$. Minimal extension lengths: $sc[0] = 0$, $sc[1] = 0$, $sc[2] = 7$, $sc[3] = 6$, $sc[4] = 5$, $sc[5] = 4$, $sc[6] = 3$, $sc[7] = 2$, $sc[8] = 1$, $sc[9] = 0$, $sc[10] = 3$, $sc[11] = 1$, $sc[12] = 0$.

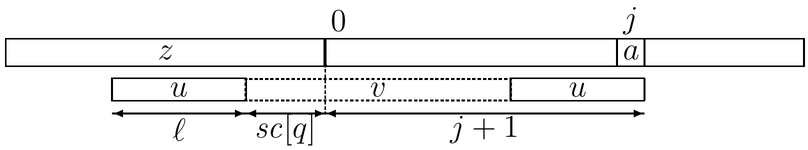


Figure 5: The maximal exponent of all factors in question bordered by u, longest factor of z ending at j, is $(\ell + sc[q] + j + 1)/(sc[q] + j + 1)$.

j		0	1	2	3	4	5	6	7	8	9
$w[j]$		`d`	`e`	`c`	`a`	`d`	`b`	`e`	`c`	`a`	`d`
q	12	5	7	8	9	5	6	7	8	9	5
ℓ	2	3	1	2	3	3	4	5	6	7	3
exp		8/5	5/4	3/2	7/4	4/3	13/9	14/9	5/3	16/9	17/14
				5/4			10/9				

Figure 6: Computing exponents when searching zw for factors uvu. The first occurrence of u is in z and the second ends in zw. The Suffix Automaton of $z =$ `abcadbeca` with function sc is in Figure 4. The search is done by parsing $w =$ `decadbecad` with the automaton. Exponents of factors are given by the expression $(\ell + sc[q] + j + 1)/(sc[q] + j + 1)$. The last line is for exponents corresponding to suffixes of u. The maximal exponent of all factors is 7/4.

$p = \text{goto}(\text{initial}(\mathcal{S}(z)), x)$, then $sc_z(p) = |x'|$ where x' is the shortest string for which xx' is a suffix of z. With this precomputed extra element, computing an exponent is a mere division (see Figure 5).

MaxExp(z, w, e)
$\mathcal{S} \leftarrow$ *Suffix Automaton of* z
mark initial$(\mathcal{S})$
$(q, \ell) \leftarrow (F[\text{last}(\mathcal{S})], L[F[\text{last}(\mathcal{S})]])$
for $j \leftarrow 0$ **to** $\min\{\lfloor |z|/(e-1) - 1 \rfloor, |w| - 1\}$ **do**
    while $\text{goto}(q, w[j]) = \text{NIL}$ **and** $q \neq \text{initial}(\mathcal{S})$ **do**
        $(q, \ell) \leftarrow (F[q], L[F[q]])$
    if $\text{goto}(q, w[j]) \neq \text{NIL}$ **then**
        $(q, \ell) \leftarrow (\text{goto}(q, w[j]), \ell + 1)$
        $(q', \ell') \leftarrow (q, \ell)$
        while q' unmarked **do**
            $e \leftarrow \max\{e, (\ell' + sc[q'] + j + 1)/(sc[q'] + j + 1)\}$
            if $\ell' = L[q']$ **then**
                mark q'
            $(q', \ell') \leftarrow (F[q'], L[F[q']])$
return e

Figure 6 illustrates a computation done by the algorithm using the Suffix Automaton of Figure 4.

Note the potential overflow when computing $\lfloor |z|/(e-1) - 1 \rfloor$ can easily be fixed in the algorithm implementation.

Theorem 4. *Algorithm* MaxExp*, applied to strings z and w and to the rational number e, produces the maximal exponent of factors in zw that end in w, whose left border occurs in z and whose exponent is at least e.*

Proof. In the algorithm, position j on w stands for a potential ending position of a relevant factor. First, we show that the algorithm does not require to examine more values of j but those specified at line 4. The exponent of a factor uvu is uvu/vu. Since we are looking for factors satisfying $uvu/vu \geq e$, the longest possible such factor has period $j+1$ and border z. Then $(z+j+1)/(j+1) > e$ implies $j < z/(e-1)-1$ (which is $+\infty$ if $e=1$). Since j is a position on w, $j < w$, which completes the first statement.

Second, given a position j on w, we show that the algorithm examines all the possible concerned factors having an exponent at least e and ending at j. The following property related to variables q, state of $\mathcal{S}$, and ℓ is known from [8, Section 6.6]: let u be the longest suffix of $zw[0\,..\,j]$ that is a factor of z, then $q = \text{goto}(\text{initial}(\mathcal{S}), u)$ and $\ell = |u|$. The property is also true just after execution of line 3 for z alone due to the initialisation of the two variables.

Then, string u is the border of a factor ending in w and whose left border occurs in z. Lines 9 to 14 check the exponents associated with u and its suffixes. If q' is unmarked, the exponent is computed as explained before (see Figure 5). If the condition at line 11 is met, which means that u is the longest string satisfying $q' = \text{goto}(\text{initial}(\mathcal{S}), u)$, due to Lemma 3 the algorithm does not need to check the exponent associated with next occurrences of u, nor with the suffixes of u since they have been checked before. Due to Lemma 2, suffixes of u ending at the same rightmost position on z do not have a larger exponent. Therefore the next suffix whose associated exponent has to be checked is the longest suffix leading to a different state of $\mathcal{S}$: it is $F(q')$ and the length of the suffix is $L(F(q'))$ by definition of F and L.

Finally note the initial state of $\mathcal{S}$ is marked because it corresponds to an empty string u, that is a factor of exponent 1, which is not larger than the values of e.

This proves the algorithm runs through all possible relevant factors, which ends the proof. ■

5. Complexity analysis

In this section we analyse the running time and memory usage of our algorithms.

Proposition 5. *Applied to strings z and w and to the rational number e, Algorithm* MaxExp *requires $O(|z|)$ space in addition to inputs and runs in total time $O(|z| + \min\{\lfloor |z|/(e-1) - 1\rfloor, |w| - 1\})$ on a fixed size alphabet. It performs less than $2|z| + \min\{\lfloor |z|/(e-1) - 1\rfloor, |w| - 1\}$ exponent computations.*

Proof. The space is used mostly for storing the automaton, which is known to have no more $2|z|$ states and $3|z|$ edges (see [8]). It can be stored in linear space if edges are implemented by successor lists, which adds a multiplicative $\log a$ factor on transition time.

It is known from [8, Section 6.6] that the algorithm runs in linear time on a fixed alphabet, including the automaton construction with elements F, L and sc, if we exclude the time for executing lines 9 to 14.

So, let us count the number of times line 11 is executed. It is done once for each position j associated with an unmarked state. If it is done more than once for a given position, then the second value of q' comes from the failure link. A crucial observation is that condition at line 12 holds for such a state. Therefore, since $\mathcal{S}(z)$ has no more than $2|z|$ states, the total number of extra executions of line 11 is at most $2|z|$. Which gives the stated result. ■

The proof of the linear running time of Algorithm MaxExpFac additionally relies on a combinatorial property of strings. It is Dejean's statement [4] proved in [6**?**] that gives for each alphabet size a, its repetitive threshold $\mathrm{RT}(a)$, i.e. the maximal exponent unavoidable in infinite strings over the alphabet. Thresholds are: $\mathrm{RT}(2) = 2$, $\mathrm{RT}(3) = 7/4$, $\mathrm{RT}(4) = 7/5$, and $\mathrm{RT}(a) = a/(a-1)$ for $a \geq 5$. Thus, if the string y is long enough the maximal exponent of its factors is at least $\mathrm{RT}(a)$ where a is its alphabet size (see the note following Algorithm `MaxRepFac`).

Theorem 6. *Applied to any overlap-free string of length n on a fixed-size alphabet, Algorithm* MaxExpFac *runs in time $O(n)$ and requires $O(n)$ extra space.*

Proof. Computing the f-factorisation $(z_1, z_2, \ldots, z_k)$ of the input takes time and space $O(n)$ on a fixed-size alphabet using any suffix data structure. (It can even be done in time $O(n)$ on an integer alphabet, see [13].)

Next instructions execute in linear extra space from Proposition 5. Line 5 takes time $O(|z| + \min\{\lfloor |z_{i-1}|/(e-1) - 1\rfloor, |z_i| - 1\})$, which is bounded by $O(|z_{i-1}| + |z_{i-1}|/(e-1) - 1)$, for $i = 2, \ldots, k$. For a long enough input, e is eventually at least $\mathrm{RT}(a)$ where a is the input alphabet. The time is then bounded by $O(|z_{i-1}| + |z_{i-1}|/(\mathrm{RT}(a)-1)-1)$, thus $O(|z_{i-1}|)$ because $\mathrm{RT}(a) > 1$. The contribution of Line 5 to the total runtime is then $O(\Sigma_{i=2}^{k}|z_{i-1}|)$.

Similarly it is $O(\Sigma_{i=2}^{k}|z_i|)$ for Line 6 and $O(\Sigma_{i=2}^{k}|z_{i-1}z_i|)$ for Line 8. Thus the overall runtime is bounded by $O(\Sigma_{i=1}^{k}|z_i|)$, which is $O(n)$ as expected. ■

6. Counting maximal-exponent factors

This section is devoted to the combinatorial aspects of maximal-exponent factors (MEF). We exhibit upper and lower bounds on their maximal number of occurrences in an overlap-free string. Note that bounds on runs (maximal periodicities of exponent at least 2) in strings are given in [11, 20] and in references therein.

The upper bound shows there is no more than a linear number of MEF occurrences in a string according to its length. And the lower bound proves that this is optimal up to a multiplicative factor that remains to be discovered.

Note that on the alphabet $\{\mathtt{a}, \mathtt{a}_1, \ldots, \mathtt{a}_n\}$ the string $\mathtt{a}\mathtt{a}_1\mathtt{a}\mathtt{a}_2\mathtt{a}\ldots\mathtt{a}\mathtt{a}_n\mathtt{a}$ of length $2n+1$ has a quadratic number of maximal factors. Indeed all occurrences of factors of the form $\mathtt{a}w\mathtt{a}$ for a non-empty word w are non extensible. But only the n factors of the form $\mathtt{a}c\mathtt{a}$ for a letter c have the maximal exponent $3/2$.

6.1. Upper bound

Before giving an upper bound, we start with a simple property of MEFs, which does not lead to their linear number, but is used later to tune the upper bound.

Lemma 7. *Consider two occurrences of MEFs with the same border length b starting at respective i and j on y, $i < j$. Then, $j - i > b$.*

Proof. The two MEFs having the same border length, since they have the same exponent, they have also the same period and the same length. Let b their border length and p their period.

Assume ab absurdo $j-i \le b$. The word $y[i \,..\, i+b-1] = y[i+p \,..\, i+p+b-1]$ is the border of the first MEF. The assumption implies that $y[i+b] = y[i+p+b]$ because these letters belong to the border of the second MEF. It means the first MEF can be extended with the same period, a contradiction because it has the largest exponent. Therefore, $j-i > b$ as stated. ■

If we count the occurrences of MEFs by their border lengths after Lemma 7 we get an initial part of the harmonic series, quantity that is not linear with respect to the length y.

To get a linear upper bound on the number of occurrences of MEFs we introduce the notion of δ-MEFs, for a positive real number δ, as follows. A MEF uvu is a δ-MEF if its border length $b = |u| = |uvu| - \text{period}(uvu)$ satisfies $2\delta < b \le 4\delta$. Then any MEF is a δ-MEF for some $\delta \in \Delta$, where $\Delta = \{1/4, 1/2, 1, 2, 2^2, 2^3, \ldots\}$. This is the technique used for example in [10, 11] to count runs in strings.

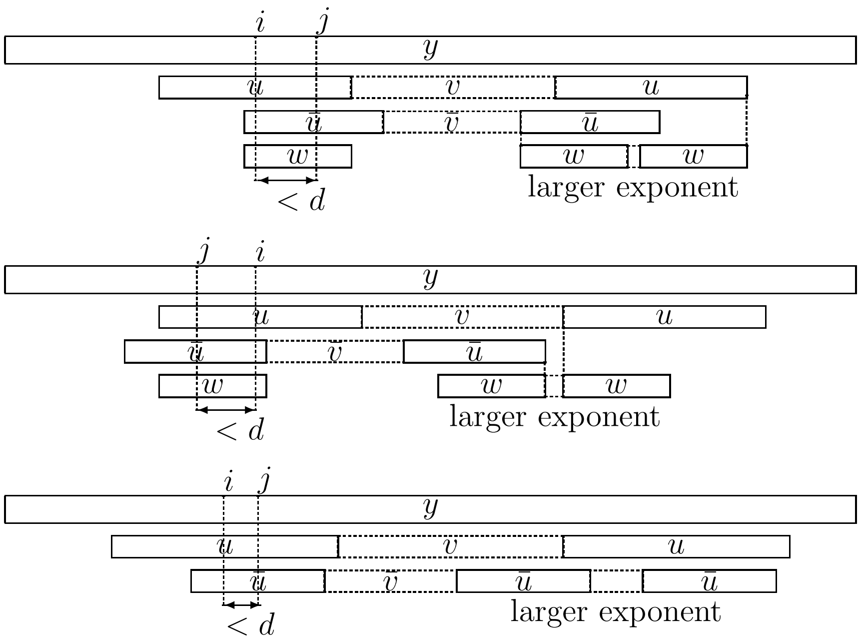


Figure 7: Two δ-MEFs, uvu and $\bar{u}\bar{v}\bar{u}$, having mid-positions of their left borders at close positions induces a factor with a larger exponent, a contradiction.

The proof of the next lemma is illustrated by Figure 7.

Lemma 8. *Let uvu and $\bar{u}\bar{v}\bar{u}$ be two occurrences of δ-MEFs in y whose left borders mid-positions are at respective positions i and j on y. Then, $|j-i| \geq \delta$.*

Proof. We consider w.l.o.g. $|u| \geq |\bar{u}|$. Assume ab absurdo $|j-i| < \delta$ (see Figure 7).

Since both $|u| > 2\delta$ and $|\bar{u}| > 2\delta$, the two occurrences of left borders overlap. Let w be the overlap. It can be a suffix of u and a prefix of $\bar{u}$, or it can be a suffix of $\bar{u}$ and a prefix of u, or w can be $\bar{u}$ itself, the shorter of two borders, when it occurs inside u The three cases are displayed in this order on Figure 7.

Let $p = |uv|$ be the period of uvu and $p' = |\bar{u}\bar{v}|$ be that of $\bar{u}\bar{v}\bar{u}$. The exponent of the two factors is $e = 1 + |u|/p = 1 + |\bar{u}|/p'$, which implies $p - p' = (|u| - |\bar{u}|)/(e-1)$.

Note w, the overlap of the two left borders occur at two other positions. For example, in the first case, it occurs as a suffix of the right border of u and as a prefix of the right border of $\bar{u}$. Due to the periodicity of the two factors, uvu and $\bar{u}\bar{v}\bar{u}$, the last two occurrences of w are $p - p'$ positions apart. Therefore the factor z starting with one occurrence and ending with the other has exponent at least (it can be longer is w if it is not the longest border of z):

$$1 + \frac{|w|}{p - p'} = 1 + \frac{|w|(e-1)}{(|u| - |\bar{u}|)}.$$

Now, from inequalities $2\delta < |\bar{u}| \leq |u| \leq 4\delta$ and the definition of w, we have both $|w| > |u|/2$ and $|u| - |\bar{u}| < |u|/2$. Then $|w| > |u| - |\bar{u}|$ and since $e - 1 > 0$ the exponent of z is then larger than e, a contradiction. Therefore $|j - i| \geq \delta$ as stated. ■

A direct consequence of the previous lemma is the linear number of MEF occurrences.

Theorem 9. *There is a constant α for which the number of occurrences of maximal-exponent factors in a string of length n is less than αn.*

Proof. Lemma 8 implies the number of δ-MEF occurrences in y is no more than n/δ. Since values of δ in Δ cover all border lengths, the total number of occurrences of MEFs is bounded by

$$\sum_{\delta \in \Delta} \frac{n}{\delta} = n \left(4 + 2 + 1 + \frac{1}{2} + \left(\frac{1}{2}\right)^2 + \dots\right) < 8\,n.$$

■

The next statement refines the upper bound given in the proof of the previous theorem by combining results of Lemmas 7 and 8.

Corollary 10. *There are less than* $2.25\,n$ *occurrences of maximal-exponent factors in a string of length* n*.*

Proof. According to Lemma 7 there are less than

$$\sum_{b=1}^{b=5} \frac{n}{b+1} = 1.45\,n$$

occurrences of MEFs with border length at most 5.

We then apply Lemma 8 with values of $\delta \in \Gamma$ that cover all remaining border lengths of MEFs: $\Gamma = \{(5/2), 5, 10, 20, \ldots\}$. It gives the upper bound

$$\sum_{\delta\in\Gamma} \frac{n}{\delta} = \frac{1}{5}\left(2 + 1 + \frac{1}{2} + \left(\frac{1}{2}\right)^2 + \ldots\right) n = \frac{4}{5}\,n$$

for the number of occurrences of MEFs with border length at least 6.

Thus the global upper bound we obtain is $2.25\,n$. ■

Note that the border length 5 minimises the expression

$$\left(\sum_{b=1}^{b=k} \frac{n}{b+1}\right) + \frac{1}{k}\left(2 + 1 + \frac{1}{2} + \left(\frac{1}{2}\right)^2 + \ldots\right) n = \left(\sum_{b=1}^{b=k} \frac{n}{b+1}\right) + \frac{4\,n}{k}$$

with respect to k, which means the technique is unlikely to produce a smaller bound. By contrast, experiments show that the number of occurrences of MEFs is smaller than n and not even close to n, at least for small values of n. The following table displays the maximal number of MEFs for overlap-free string lengths $n = 5, 6, \ldots, 20$ and for alphabet sizes 2, 3 and 4. It also displays (second element of pairs) the associated maximal exponent. In the binary case we already know that it is 2 since squares are unavoidable in strings whose length is greater than 3.

n	5	6	7	8	9	10	11	12
binary	2	3	4	5	5	6	6	8
ternary	$(2, 1.5)$	$(3, 1.5)$	$(4, 2)$	$(5, 2)$	$(5, 2)$	$(6, 1.5)$	$(6, 2)$	$(8, 2)$
$4-$ary	$(2, 1.5)$	$(3, 1.5)$	$(4, 2)$	$(5, 2)$	$(5, 2)$	$(6, 1.5)$	$(7, 1.5)$	$(8, 2)$

13	14	15	16	17	18	19	20
8	9	9	11	11	12	12	14
$(8, 2)$	$(9, 2)$	$(9, 2)$	$(11, 2)$	$(11, 2)$	$(12, 2)$	$(12, 2)$	$(14, 2)$
$(8, 1.5)$	$(9, 1.5)$	$(10, 1.5)$	$(11, 2)$	$(12, 1.5)$	$(12, 1.5)$	$(13, 1.5)$	$(14, 1.5)$

6.2. Lower bound

We now deal with a lower bound on the maximal number of occurrences of maximal-exponent factors. We first consider an infinite word whose factors have maximal exponent 3/2 and then show that its prefixes contain a linear number of occurrences of these factors.

The infinite word is on the four-letter alphabet $A_4 = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$ and the maximal exponent of its factors is 7/5. The existence of such a word was proved by Pansiot [15] and it is easy to see that the exponent value cannot be smaller for an infinite word on A_4. Indeed, the result is part of the conjecture of Dejean [4] who stated the repetitive threshold for all alphabet sizes; the proof of this conjecture was eventually completed by Rao [6] and by Currie and Rampersad [7]. Here is an example of such a word given by Pansiot [15]:

$$\mathbf{p} = \mathtt{abcdabcdabcdabcdabcdabcdabcd}\ldots$$

From the word $\mathbf{p}$ we define $\mathbf{q}$ on the alphabet $A_5 = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}\}$ by inserting letter $\mathtt{e}$ in between any two consecutive letters. That is, for each integer $i \geq 0$,

$$\begin{array}{rcl} \mathbf{q}[2i] & = & \mathtt{e} \\ \mathbf{q}[2i+1] & = & \mathbf{p}[i] \end{array}$$

or in other words $\mathbf{q} = g(\mathbf{p})$, where g is the morphism defined by $f(a) = \mathtt{e}a$, for any letter $a \in A_4$. The word $\mathbf{q}$ is:

$$\mathbf{q} = \mathtt{eaebecedeaebecedeaebecedeaebecedeaebecedeaebecedeaebeced}\ldots$$

Let uvu be a factor of $\mathbf{p}$, where u is its longest border and then $|uv|$ is its smallest period. By the choice of $\mathbf{p}$, we have $\exp(uvu) = |uvu|/|uv| \leq 7/5$. In addition, we know that the period length of all 7/5-powers in $\mathbf{p}$ is at least 10 (see [21]). Thus the induced factor $f(uvu)\mathtt{e}$ in $\mathbf{q}$ has exponent $(2|uvu| + 1)/2|uv|$, which is 29/20 when uvu is a 7/5-power. This value is less than 3/2.

As another examples, consider the factor $\mathtt{abcda}$ of $\mathbf{p}$. It has exponent 5/4 and its induced factor in $\mathbf{q}$, $f(\mathtt{abcda})\mathtt{e} = \mathtt{eaebecedeae}$, has exponent 11/8, which is less than $3/2$ again. By contrast, the factor $\mathtt{abca}$ of $\mathbf{p}$ has exponent $4/3$ and its induced factor in $\mathbf{q}$, $\mathtt{eaebeceae}$ has exponent $9/6 = 3/2$.

The next lemma shows that very few factors of $\mathbf{q}$ have exponent $3/2$ the maximal value.

Lemma 11. *Let w be a factor of $\boldsymbol{q}$, then* $\exp(w) \leq 3/2$*. Additionally* $\exp(w) = 3/2$ *when* $w = f(uvu)\mathtt{e}$ *with either* $uvu = v = \mathtt{a}$ *or* $u = \mathtt{a}$ *and* $v = \mathtt{bc}$ *up to a permutation of letters.*

Proof. Let w be a factor with maximal exponent among the factors of $\mathbf{q}$. Its first letter is $\mathtt{e}$ because otherwise its length could be increased by one unit without changing the period, which would increase the exponent. Similarly, its last letter is $\mathtt{e}$. Then, w is of the form $f(uvu)\mathtt{e}$ for a factor uvu of $\mathbf{p}$ whose longest border is u.

Assume that $\exp(w) \geq 3/2$. Then

$$\frac{2|uvu| + 1}{2|uv|} \geq 3/2\,,$$

which gives

$$2|u| + 1 \geq |uv|\,.$$

Also, since uvu is a factor of $\mathbf{p}$, it satisfies

$$|uvu|/|uv| \leq 7/5\,,$$

which implies

$$\frac{5}{2}|u| \leq |uv|\,.$$

Therefore

$$\frac{5}{2}|u| \leq 2|u| + 1\,,$$

which is only possible for $|u| = 0, 1,$ or 2.

If $|u| = 0$, $|v| = |uv| = 1$, and the induced factor in $\mathbf{q}$ is of the form $\mathtt{e}a\mathtt{e}$, for a letter $a \in A_4$, and has exponent $3/2$.

If $|u| = 1$, $|uv| = 3$, and then uvu is of the form $\mathtt{abca}$ up to a permutation of letters, inducing a factor of exponent $3/2$ in $\mathbf{q}$.

Finally, if $|u| = 2$, $|uv| = 5$ and $\exp(uvu) = 7/5$. But as recalled above, no factor of $\mathbf{p}$ with that exponent has period 5. This case is impossible, which concludes the proof. ∎

The conclusion of previous lemma is that the maximal exponent of factors is $3/2$. The lower bound on the occurrence number of $3/2$-powers in $\mathbf{q}$ requires another property of $\mathbf{p}$, which is used in the proof of following corollary.

Corollary 12. *The number of occurrences of maximal-exponent factors in prefixes of* $\mathbf{q}$ *tends to* $2n/3$ *with the prefix length* n.

Proof. From the previous lemma, maximal-exponent factors in $\mathbf{q}$ are induced by factors of the form `a` or `abca`, up to a permutation of the four letter of A_4, in $\mathbf{p}$.

It is clear from the definition of $\mathbf{q}$ that at every two of its positions occur one of the factors `eae`, `ebe`, `ece`, `ede`. Their occurrence number then tends to $n/2$.

Turning to the other factors of exponent $3/2$, it is known that the six factors of the form *abca* appear at every three positions in $\mathbf{p}$. Indeed, an occurrence of `abca`, can extend to `abcad` and `abcadb` but not to `abcadbc` whose suffix `bcadbc` has exponent $6/4 = 3/2 > 7/5$. Therefore, the induced factors of exponent $3/2$ occur at every six positions in $\mathbf{q}$, leading to a limit of $n/6$.

Summing up the two limits, the occurrence numbers of $3/2$-powers in prefixes of $\mathbf{q}$ tend to $n/2 + n/6 = 2n/3$ as stated. ■

7. Conclusion

The result of Section 6 implies that Algorithm MaxExpFac can be modified to output all the MEFs occurring in the input string in the same asymptotic time. Indeed, the only occurrences of MEFs that are skipped by the algorithm when computing the maximal exponent are those occurring inside a phrase of the f-factorisation (Case (i) of Section 3). However storing the previous occurrences of MEFs and listing them can be done in time proportional to their number, which does not affect the asymptotic running time of the algorithm and yields the next statement.

Corollary 13. *All the occurrences of maximal-exponent factors of a string can be listed in linear time with respect to its length.*

The present work triggers the study of a uniform solution to compute both repetitions (of exponent at least 2) and repeats. However, exponent 2 seems to reflect a transition phase in the combinatorics of these studied objects. For instance, the number of repetitions in a string can be of the order of $n \log n$ and the number of maximal periodicities (runs) is linear, while the number of maximal occurrences of factor uvu can be quadratic.

An interesting question is to select factors related to repeats and that occur only a linear number of times or slightly more. An attempt has been achieved in [22] where it is shown that the number of maximal repetitions of any exponent more than $1+\epsilon$ is bounded by $\frac{1}{\epsilon} n \ln n$. See also the discussions at the end of [9] and of [23].

Other interesting problems are the exact evaluation of the maximal number occurrences of MEF and the calculation of the maximal number of (distinct) MEFs occurring in a string.

8. Acknowledgements

We warmly thank G. Kucherov and R. Kolpakov for interesting discussions on repetitions, runs and repeats in strings.

References

[1] D. Gusfield, Algorithms on strings, trees and sequences: computer science and computational biology, Cambridge University Press, Cambridge, 1997.

[2] G. S. Brodal, R. B. Lyngsø, C. N. S. Pedersen, J. Stoye, Finding maximal pairs with bounded gap, in: M. Crochemore, M. Paterson (Eds.), Combinatorial Pattern Matching, Vol. 1645 of Lecture Notes in Computer Science, Springer, 1999, pp. 134–149.

[3] L. Vuillon, A characterization of sturmian words by return words, Eur. J. Comb. 22 (2) (2001) 263–275.

[4] F. Dejean, Sur un théorème de Thue, J. Comb. Theory, Ser. A 13 (1) (1972) 90–99.

[5] A. Thue, Über unendliche Zeichenreihen, Norske Vid. Selsk. Skr. I Math-Nat. Kl. 7 (1906) 1–22.

[6] M. Rao, Last cases of Dejean's conjecture, Theor. Comput. Sci. 412 (27) (2011) 3010–3018.

[7] J. D. Currie, N. Rampersad, A proof of Dejean's conjecture, Math. Comput. 80 (274) (2011) 1063–1070.

[8] M. Crochemore, C. Hancart, T. Lecroq, Algorithms on Strings, Cambridge University Press, 2007, 392 pages.

[9] R. Kolpakov, G. Kucherov, On maximal repetitions in words, J. Discret. Algorithms 1 (1) (2000) 159–186.

[10] W. Rytter, The number of runs in a string, Inf. Comput. 205 (9) (2007) 1459–1469.

[11] M. Crochemore, L. Ilie, L. Tinta, The "runs" conjecture, Theoretical Computer Science 412 (27) (2011) 2931–2941.

[12] J. Ziv, A. Lempel, A universal algorithm for sequential data compression 23 (1977) 337–343.

[13] M. Crochemore, G. Tischler, Computing longest previous non-overlapping factors, Information Processing Letters 111 (2011) 291–295.

[14] G. Badkobeh, M. Crochemore, C. Toopsuwan, Computing the maximal-exponent repeats of an overlap-free string in linear time, in: L. Calderon-Benavides, C. Gonzlez-Caro, E. Chvez, N. Ziviani (Eds.), Symposium on String Processing and Information Retrieval, no. 7608 in LNCS, Springer, 2012, pp. 61–72.

[15] J.-J. Pansiot, A propos d'une conjecture de F. Dejean sur les répétitions dans les mots, in: J. Díaz (Ed.), Automata, Languages and Programming, 10th Colloquium, Barcelona, Spain, July 18-22, 1983, Proceedings, Vol. 154 of Lecture Notes in Computer Science, Springer, 1983, pp. 585–596.

[16] O. Berkman, C. S. Iliopoulos, K. Park, The subtree max gap problem with application to parallel string covering, Inf. Comput. 123 (1) (1995) 127–137.

[17] C. S. Iliopoulos, D. W. G. Moore, K. Park, Covering a string, Algorithmica 16 (3) (1996) 288–297.

[18] G. S. Brodal, C. N. S. Pedersen, Finding maximal quasiperiodicities in strings, in: R. Giancarlo, D. Sankoff (Eds.), Combinatorial Pattern Matching, no. 1848 in LNCS, Springer, 2000, pp. 397–411.

[19] M. Christou, M. Crochemore, C. S. Iliopoulos, M. Kubica, S. P. Pissis, J. Radoszewski, W. Rytter, B. Szreder, T. Walen, Efficient seeds computation revisited, in: R. Giancarlo, G. Manzini (Eds.), Combinatorial Pattern Matching, no. 6661 in LNCS, Springer, Berlin, 2011, pp. 350–363.

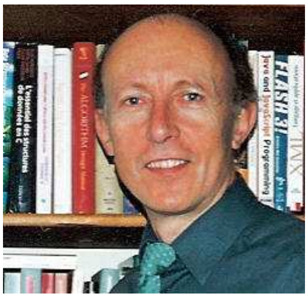
[20] H. Bannai, M. Giraud, K. Kusano, W. Matsubara, A. Shinohara, J. Simpson, The number of runs in a ternary word, in: J. Holub, J. Zdárek (Eds.), Stringology, Prague Stringology Club, Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, 2010, pp. 178–181.

[21] G. Badkobeh, M. Crochemore, M. Rao, Finite-repetition-threshold for large alphabets, RAIRO - Theoretical Informatics and ApplicationsTo appear.

[22] R. Kolpakov, G. Kucherov, P. Ochem, On maximal repetitions of arbitrary exponent, Inf. Process. Lett. 110 (7) (2010) 252–256.

[23] M. Crochemore, L. Ilie, Maximal repetitions in strings, Journal of Computer and System Sciences 74 (2008) 796–807, dOI: 10.1016/j.jcss.2007.09.003.

Dr. Golnaz Badkobeh completed her PhD in Computer Science at King's College London in 2013 after obtained a BSc in Mathematics at University of Bristol in 2008. She has been working in the field of Combinatorics on Words and string processing algorithms, areas in which she has published eight scholarly research articles in her academic career. She currently has three more articles under review.



Maxime Crochemore is presently Professor at King's College London and Emeritus Professor of Université Paris-Est. His research interests are in the design and analysis of algorithms. His major achievements are on string algorithms, which includes pattern matching, text indexing, coding, and text compression. He also works on the combinatorial background of these subjects and on their applications to bio-informatics. He has co-authored several textbooks on algorithms and published more than 200 articles. He participated in a large number of international projects on algorithms and supervised to completion more than twenty PhD students.