Finding Maximal Repetitions in a Word in Linear Time*

Roman Kolpakov
French-Russian Institute for Informatics and Applied Mathematics, Moscow University
119899 Moscow, Russia
e-mail: roman@vertex.inria.msu.ru

Gregory Kucherov
LORIA/INRIA-Lorraine
615, rue du Jardin Botanique
B.P. 101, 54602 Villers-lès-Nancy, France
e-mail: kucherov@loria.fr

Abstract

A repetition in a word w is a subword with the period of at most half of the subword length. We study maximal repetitions occurring in w, that is those for which any extended subword of w has a bigger period. The set of such repetitions represents in a compact way all repetitions in w. We first prove a combinatorial result asserting that the sum of exponents of all maximal repetitions of a word of length n is bounded by a linear function in n. This implies, in particular, that there is only a linear number of maximal repetitions in a word. This allows us to construct a linear-time algorithm for finding all maximal repetitions. Some consequences and applications of these results are discussed, as well as related works.

1. Introduction

Repetitions (periodicities) in words are fundamental objects, due to their primary importance in word combinatorics [14] as well as in various applications, such as string matching algorithms [8, 5], molecular biology [9], or text compression [20].

Several notions of repetitions have been studied, and to make it precise, we start with basic definitions. Recall that *the period* of a word $w = a_1 \dots a_n$ is the smallest positive integer p such that $a_i = a_{i+p}$ for all i, provided $1 \leq i, i + p \leq n$. The rational n/p is called *the exponent* of w. If the exponent is an integer number $k > 1$, w can be simply written as $u^k = \underbrace{uu \dots u}_{k}$ and is called an *integer power* (or k-power or *tandem arrays*). A word of exponent 2 is called a *square* (or *tandem repeat*). A word which is not an integer power is called *primitive*. In general, any word w of period p and exponent e can be written as $u^k v$, where u is a primitive word, $|u| = p$, v is a proper prefix of u and $e = k + \frac{|v|}{|u|}$.

We call *a repetition* a word of exponent 2 or more (equivalently, with the period of at most half the word length). The problem addressed in this paper is to construct an efficient algorithm for identifying all subwords in a word which are repetitions. Note that in this paper we are interested in characterizing all *occurrences* of repetitions in the word, and not in all *syntactically distinct* repetitions (cf [19, 22]).

Clearly, a word may contain a quadratic number of repetitions (e.g. a^n). To represent them in a compact way, we introduce the notion of maximal repetition. A *maximal repetition*[1] in a word is a repetition such that its extension by one letter to the right or to the left yields a word with a bigger period. For example, the subword 10101 in the word $w = 1011010110110$ is a maximal repetition (with period 2), while the subword 1010 is not. Another maximal repetitions of w are prefix 10110101101 (period 5), suffix 10110110 (period 3), prefix 101101 (period 3), and the three occurrences of 11 (period 1). Maximal repetitions encode, in a most compact way, all repetitions in the word, hence their importance.

Let us now survey the known algorithmic results on searching for repetitions in a word, which is a classical string matching problem (see [4]). In early 80s, Slisenko [19] proposed a linear (real-time) algorithm for finding all *syntactically distinct* maximal repetitions in a word. Independently, Crochemore [3] described a simple and elegant linear algorithm for finding a square in a word (and thus checking if a word is repetition-free). The algorithm

*Part of this work has been done during the first author's visit of LORIA/INRIA-Lorraine supported by a grant from the French Ministry of Public Education and Research. The first author has been in part supported by the Russian Foundation of Fundamental Research, under grant 96–01–01068, and by the Russian Federal Program "Integration", under grant 473. The work has been done within a joint project of the French-Russian A.M.Liapunov Institut of Applied Mathematics and Informatics at Moscow University

[1]called *run* in [10] and *maximal periodicity* in [17]

is based on a special factorization of the word, called s-factorization (f-factorization in [4], or Lempel-Ziv decomposition [9]). Another linear algorithm checking whether a given word contains a square was proposed in [16].

However, it is known that there may be up to $\Omega(n \log n)$ square occurrences in a word, even if only primitively-rooted squares are considered [2] (an integer power u^k is primitively-rooted if u is a primitive word). An example is provided by Fibonacci words, that contain $\Theta(n \log n)$ squares all of which are primitively-rooted (an exact formula is given in [7]). This implies that there is no hope to construct a linear algorithm to explicitly find all squares in a word as their number is super-linear.

There are several different $O(n \log n)$ algorithms finding all occurrences of repetitions in a string. Note however that each of these algorithms uses its own notion of repetition. In 1981, Crochemore [2] proposed an $O(n \log n)$ algorithm for finding all occurrences of non-extendable primitively-rooted integer powers in a word (i.e. those primitively-rooted integer powers u^k which are not followed or preceded by another occurrence of u). This is an asymptotically optimal bound, as the number of such powers can be $\Omega(n \log n)$. Using a suffix tree technique, Apostolico and Preparata [1] described an $O(n \log n)$ algorithm for finding all *right-maximal* repetitions, which are repetitions that cannot be extended *to the right* without increasing the period. Main and Lorentz [15] proposed another algorithm which actually finds all maximal repetitions in $O(n \log n)$ time. They also point out the optimality of this bound under the assumption of unbounded alphabet and under the restriction that the algorithm is based only on symbol comparisons. In 1989, using s-factorization, Main [17] proposed a *linear-time* algorithm which finds all *leftmost* occurrences of distinct maximal repetitions in a word.

As far as other related works are concerned, Kosaraju [13] describes an $O(n)$ algorithm which, given a word, finds for each position the shortest square starting at this position. He also claims a generalization which finds all primitively-rooted squares in time $O(n + S)$ where S is the number of such squares. In [21], Stoye and Gusfield proposed several algorithms that are based on a unified suffix tree framework. Their results are based on an algorithm which finds in time $O(n \log n)$ all "branching tandem repeats". In our terminology, branching tandem repeats are (not necessarily primitively-rooted) square suffixes of maximal repetitions. In a recent paper, Stoye and Gusfield [22] proposed a different approach, combining s-factorization and suffix tree techniques. The goal achieved is to find, in linear time, a representative of each *syntactically distinct* square. The feasibility of this task is supported by the result of [6] asserting that there is a linear number (actually, no more than $2n$) distinct squares in words of length n over an arbitrary alphabet. The approach allows also to solve some other problems, e.g. to achieve the results claimed in [13].

However, so far it has been an open question whether a *linear* algorithm for finding *all* maximal repetitions exists. In the concluding section of [17], Main speculates that such an algorithm might exist. The same question is raised in [10]. However, there has been no evidence in support of this conjecture as the number of maximal repetitions has not been known to be linear. This paper provides this argument.

In the first part of the paper (Section 3) we prove a combinatorial result asserting that the sum of exponents of all maximal repetitions in a word is linearly bounded. Obviously, this implies that the number of maximal repetitions in a word is linear, which contrasts to the $O(n \log n)$ bounds for the number of primitively-rooted squares or integer powers. This also explains a trade-off between the number of repetitions in a word and their exponents; Fibonacci words, for example, have a linear number of maximal repetitions that are all of small exponent (smaller than 4). Based on the linearity result, we show in Section 4 that all maximal repetitions in a word can be found in linear time. Since maximal repetitions characterize completely the repetitive structure of the word, this allows to solve other related problems, e.g. to output all squares in a word in time $O(n + S)$, where S is the output size [13, 22], or to find in linear time all "branching tandem repeats" [21], or to determine, in linear time, the number of repetitions of a given exponent starting at every position in the word. We believe that other applications of these results are still to be discovered.

2. Further definitions and basic results

For a word $w = a_1 \dots a_n$, $w[i..j]$ denotes its subword $a_i \dots a_j$. A position in a word $w = a_1 \dots a_n$ is an integer between 0 and n. Each position π in w defines a factorization $w = w_1 w_2$ where $|w_1| = \pi$. The position of letter a_i in w is $i-1$. If $v = w[i..j]$, we denote $initpos(v) = i - 1$ and $endpos(v) = j$. We say that subword $v = w[i..j]$ *crosses* a position π in w, if $i \leq \pi < j$.

If w is a subword of u^n for some natural n, $|u|$ is called a *period* of w, and word u is a *root* of w. Clearly, p is a period of $w = a_1 \dots a_n$ iff $a_i = a_{i+p}$ whenever $1 \leq i, i + p \leq n$. Another equivalent definition is (see [14]): p is a period of $w = a_1 \dots a_n$ iff $w[1..n-p] = w[p+1..n]$. Each word w has the minimal period that we will denote $p(w)$ and call *the* period of w. The ratio $\frac{|w|}{p(w)}$ is called the *exponent* of w and denoted $e(w)$. Clearly, a root u of w such that $|u| = p(w)$, is *primitive*, that is u cannot be written as v^k for $k \geq 2$.

Consider $w = a_1 \dots a_n$. A *repetition* in w is any subword occurrence $r = w[i..j]$ with $e(r) \geq 2$. A *maximal repetition* in w is a repetition $r = w[i..j]$ such that $p(w[i..j]) < p(w[i-1..j])$ whenever $i > 1$, and $p(w[i..j]) < p(w[i..j+1])$ whenever $j < n$. In other words, a maximal repetition is a repetition $r = w[i..j]$ such that no subword of w which contains r as a proper subword has the same minimal period as r. Note that any repetition in a word can be extended to a unique maximal repetition.

A basic result about periods is the Fine and Wilf's theorem (see [14]):

Theorem 1 (Fine and Wilf) *If w has periods p_1, p_2, and $|w| \geq p_1 + p_2 - gcd(p_1, p_2)$, then $gcd(p_1, p_2)$ is also a period of w.*

The following Lemma states some useful facts about maximal repetitions that will be used in the sequel.

Lemma 1 *(i) Two distinct maximal repetitions with the same period p cannot have an overlap of length greater than or equal to p,*

(ii) Two maximal repetitions with periods p_1, p_2, $p_1 \neq p_2$, cannot have an overlap of length greater than or equal to $(p_1 + p_2 - gcd(p_1, p_2)) \leq 2\max\{p_1, p_2\}$.

A repetition r is said to have a period in some subword of w if r overlaps with this subword on at least $p(r)$ letters. Also, we say that a repetition r has a period on the right (respectively on the left) of a position π with the meaning that $w[\pi + 1..\pi + p(r)]$ (respectively $w[\pi - p(r) + 1..\pi]$) is a subword of r.

$\#S$ denotes the cardinality of a set S. All logarithms are binary unless the base is indicated.

3 Estimating the total size of exponents of maximal repetitions

In this section we prove our main result asserting that the sum of exponents of all maximal repetitions in a word over an arbitrary alphabet is bounded by a linear function on the length of the word. Formally, let $R(w)$ be the set of all maximal repetitions in a word w, and let $Sexp(w) = \sum_{r \in R(w)} e(r)$, $Sexp(n) = \max_{|w|=n} Sexp(w)$.

Before proceeding to the general case, let us look at Fibonacci words which have numerous interesting combinatorial properties and often provide a good example to test conjectures and analyze algorithms on words (cf [10]). Fibonacci words are binary words defined recursively by $f_0 = 0$, $f_1 = 1$, $f_n = f_{n-1}f_{n-2}$ for $n \geq 2$.

As it was noted in Introduction, Fibonacci word f_n contains $\Theta(|f_n| \log |f_n|)$ squares all of which are primitively-rooted. In [7], the exact number of squares in Fibonacci words has been obtained, which is asymptotically $\frac{2}{5}(3 - \phi)n|f_n| + O(|f_n|)$ ($\phi \approx 1.618$ is the golden ratio). Since general words of length n contain $O(n \log n)$ primitively-rooted squares [5], Fibonacci words contain asymptotically maximal number of them. In [11, 12] we computed the exact number $\#R(f_n)$ of maximal repetitions in Fibonacci words, which turned out to be $2|f_{n-2}| - 3$ (curiously enough, this number is one less than the number of *distinct* squares, computed in [7]). For $Sexp(f_n)$, we obtained the approximate formula $Sexp(f_n) = C \cdot |f_n| + o(1)$, where $1.922 \leq C \leq 1.926$. Thus, the total sum of exponents of maximal repetitions in Fibonacci words is linear in the length, which suggests that this might hold for general words too.

Let us now turn to the general case and state the main result.

Theorem 2 $Sexp(n) = O(n)$.

The proof is based on the following Lemma.

Lemma 2 *Let $w = w_1 \ldots w_k$, and let CR_i be the set of repetitions of $R(w)$ crossing the frontier between w_i and w_{i+1}, $i = 1, \ldots k-1$. Then*

$$Sexp(w) < \sum_{i=1}^{k} Sexp(w_i) + 4 \cdot \sum_{i=1}^{k-1} \#CR_i.$$

Proof: By induction, it is sufficient to prove the Lemma for $k = 2$. Let $w = w_1w_2$. For every repetition $r \in CR$, denote r_1 its intersection with w_1, and r_2 its intersection with w_2. It is easy to see that the difference $Sexp(w) - (Sexp(w_1) + Sexp(w_2))$ is

$$\sum_{\substack{r \in CR_1 \\ |r_1| < 2p(r)}} \frac{|r_1|}{p(r)} + \sum_{\substack{r \in CR_1 \\ |r_2| < 2p(r)}} \frac{|r_2|}{p(r)} < 4 \cdot \#CR_1.$$

$\square$

To prove Theorem 2, we prove the following stronger statement.

Theorem 3 *There exist absolute positive constants C_1, C_2 such that*

$$Sexp(n) \leq C_1 n - C_2 \sqrt{n} \log n. \qquad (1)$$

The full-detailed proof of Theorem 2 is rather technical, and is presented in [12]. Here we give a high-level description of the proof omitting tedious details and presenting some typical arguments.

We assume, without loss of generality, that C_1 is sufficiently larger than C_2, say $C_1 \geq 2C_2$, so that function $C_1 x - C_2 \sqrt{x} \log x$ is monotonically increasing for all $x \geq 1$. We use induction over n.

Take a word $w = a_1 \ldots a_n$ of length n. We split the proof into two major cases depending on whether or not w contains a maximal repetition of exponent $\geq \sqrt{n}$.

Case 1: Assume all maximal repetitions in w are of exponent smaller than $\sqrt{n}$. Write $w = w_1 w_2$, where $|w_1| = |w_2| = \frac{n}{2}$ (n even for simplicity). Then by Lemma 2,

$$Sexp(w) < Sexp(w_1) + Sexp(w_2) + 4 \cdot \#CR(w), \quad (2)$$

where $CR(w)$ is the set of repetitions of $R(w)$ crossing the frontier between w_1 and w_2. By induction,

$$Sexp(w_1) + Sexp(w_2) \leq 2 \cdot Sexp(\frac{n}{2}) \leq C_1 n - C_2\sqrt{2n}\log\frac{n}{2}. \quad (3)$$

We now prove that $\#CR(w) = O(\sqrt{n}\log n)$. Let us concentrate on those repetitions r of $CR(w)$ which overlap with w_1 by at least $p(r)$ letters, and with w_2 by at least $p(r)/2$ letters. By Lemma 1(i), no two such repetitions have the same period.

Assume r_1, r_2 are two such repetitions with periods $p(r_1)$, $p(r_2)$ respectively. Assume that $p(r_1) > p(r_2)$, and let $\Delta = p(r_1) - p(r_2)$. Consider the (non-empty) word $v = w[\pi_b + 1..\pi_e]$, where $\pi_b = \max\{initpos(r_1) + p(r_1), initpos(r_2) + p(r_2)\}$ and $\pi_e = \min\{endpos(r_1), endpos(r_2)\}$. Observe that v is a subword of both r_1 and r_2 which occurs, in each of them, at least one period away from the beginning. Then v has two other occurrences at positions $\pi_b - p(r_1)$ and $\pi_b - p(r_2)$. Consider word $v' = w[\pi_b - p(r_1) + 1..\pi_b - p(r_2) + |v|]$. Observe that $|v'| = |v| + \Delta$, and v' has a period Δ, as v occurs both as a prefix and a suffix of v'. Since w does not have maximal repetitions of exponent $\sqrt{n}$ or more, we can bound $\frac{|v'|}{\Delta} = \frac{|v|}{\Delta} + 1 \leq \lceil\sqrt{n}\rceil$. Since v contains the subword $w[\lceil\frac{n}{2}\rceil + 1..\pi_e]$ of length at least $p(r_2)/2$, we have $|v| \geq p(r_2)/2$. We then have $\frac{p(r_2)}{2\Delta} \leq \sqrt{n}$ which implies $\frac{p(r_2)}{p(r_1)} \leq 1 - \frac{1}{2\sqrt{n}+1}$. Turning to logarithms, $\log p(r_2) - \log p(r_1) \leq \log(1 - \frac{1}{2\sqrt{n}+1}) \leq -\frac{1}{2\sqrt{n}+1}$, as $\log(1-x) \leq -x$ for $0 \leq x < 1$. Therefore, $\log p(r_1) - \log p(r_2) \geq \frac{1}{2\sqrt{n}+1}$. Recall that each repetition r under consideration has a distinct period $p(r)$ and hence a distinct value $\log p(r)$. On the other hand, $\log p(r)$ can vary from 0 to $(\log n - 1)$. Therefore, there are at most $(\log n - 1)(2\sqrt{n} + 1) + 1 = O(\sqrt{n}\log n)$ distinct values $\log p(r)$, and therefore that many repetitions considered.

For those repetitions which overlap with w_1 by at least $p(r)$ letters, and with w_2 by less than $p(r)/2$ letters, the proof is similar except that here v contains the subword $w[\max\{endpos(r_1) - p(r_1), endpos(r_2) - p(r_2)\} + 1..\lceil\frac{n}{2}\rceil]$ of length at least $p(r_2)/2$ which implies that $|v| \geq p(r_2)/2$. Thus, there are at most $O(\sqrt{n}\log n)$ such repetitions too.

The case of the repetitions which overlap with w_1 by more than $p(r)$ letters, is symmetrical. We conclude that there are $O(\sqrt{n}\log n)$ maximal repetitions in $CR(w)$. By (2),(3), it remains to show that $C_2\sqrt{2n}\log\frac{n}{2} - O(\sqrt{n}\log n) \geq C_2\sqrt{n}\log n$. This can be always achieved by picking a sufficiently large constant C_2. The proof of Case 1 is completed.

Case 2: Let us now turn to the case where w does contain a maximal repetition r of exponent $\geq \sqrt{n}$. Write $w = w_1 r w_2$, and denote $p_r = p(r)$ and $e_r = e(r)$. Note that $p_r = \frac{|r|}{e_r} \leq \sqrt{n}$ as $e_r \geq \sqrt{n}$. Denote $\pi_{init} = initpos(r)$, $\pi_{end} = endpos(r)$. We now split r into three approximately equal parts. (We assume that n is big enough so that each of these parts is at least $3p_r$ long.) Formally, we find positions $\pi_{left} = \pi_{init} + \lfloor\frac{|r|}{3}\rfloor$, $\pi_{right} = \pi_{end} - \lfloor\frac{|r|}{3}\rfloor$. Denote by $w_l = w[1..\pi_{left}]$, $w_r = w[\pi_{right} + 1..n]$, and $r_0 = w[\pi_{left} + 1..\pi_{right}]$. By Lemma 2,

$$Sexp(w) < Sexp(w_l) + Sexp(w_r) + Sexp(r_0) + 4 \cdot \#LR(w) + 4 \cdot \#RR(w), \quad (4)$$

where

$LR(w)$ are the repetitions of $R(w)$, crossing position π_{left},

$RR(w)$ are the repetitions of $R(w)$, crossing position π_{right}.

Part 2.1 We first estimate the number of repetitions in $LR(w)$, $RR(w)$ being analyzed similarly. Our goal is to prove that $\#LR(w) = O(e_r)$.

The general idea of this part is to split $LR(w)$ down to subclasses such that all repetitions in a subclass have distinct periods (typically according to Lemma 1(i)). Then, an upper bound on the number of possible periods implies an upper bound on the number of repetitions in the subclass. Below we illustrate this idea.

Split $LR(w)$ into subset $SLR(w)$ of repetitions with a period smaller or equal to p_r, and subset $BLR(w)$ of repetitions with a period larger than p_r.

If two repetitions from $SLR(w)$ have a period on the right (on the left) of π_{left}, then by Lemma 1(i), they cannot have the same period length. Therefore, each of these two subsets cannot have more than p_r distinct elements and there are no more than $2p_r$ overall maximal repetitions crossing π_{left}. So $\#SLR(w) \leq 2p_r \leq 2e_r = O(e_r)$.

Let us turn to $BLR(w)$. The first observation is that repetitions of $BLR(w)$ cannot lie entirely inside r as this would contradict Lemma 1(ii). Thus, any repetition of $BLR(w)$ contains at least one of the letters $a_{\pi_{init}}, a_{\pi_{end}+1}$. We further split $BLR(w)$ according to different possibilities:

$BLR0(w)$= $\{u \in BLR(w) | initpos(u) < \pi_{init}$ and $endpos(w) > \pi_{end}\}$,

$BLR1(w)$= $\{u \in BCR(w) | initpos(u) \geq \pi_{init}$ and $endpos(w) > \pi_{end}\}$,

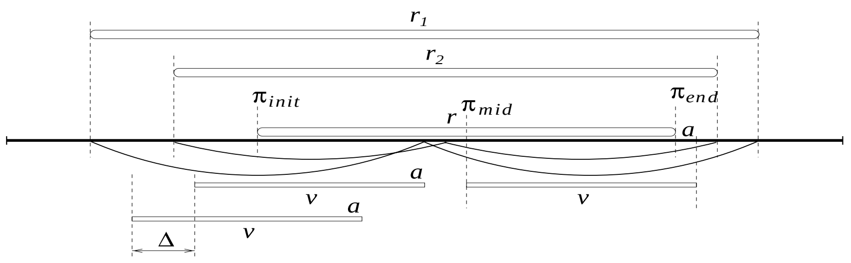

Figure 1. Illustration to Part 2.1

$BLR2(w)= \{u \in BLR(w) | initpos(u) < \pi_{init}$ and $endpos(w) \leq \pi_{end}\}$.

Then $\#BLR(w) = \#BLR0(w) + \#BLR1(w) + \#BLR2(w)$. Below we prove that $\#BLR0(w) = O(e_r)$. The proofs for $BLR1(w)$, $BLR2(w)$ are somewhat similar, and we refer the reader to [12].

Let us pick the position $\pi_{mid} = \pi_{init} + \lfloor |r|/2 \rfloor$ in the middle of r. Consider those repetitions of $BLR0(w)$ which have at least p_r letters on the left of π_{mid} (the other case is symmetrical). Consider two such repetitions r_1, r_2. By Lemma 1(i), $p(r_1) \neq p(r_2)$. Assume $p(r_1) > p(r_2)$. Consider the word $v = w[\pi_{mid}{+}1..\pi_{end}{+}1]$. Note that $a_{\pi_{end}+1}$ is the letter right after the end of repetition r, which implies that $a_{\pi_{end}+1} \neq a_{\pi_{end}+1-p_r}$. Note also that any proper prefix of v is a part of r and then has a period p_r. Word v belongs to both r_1 and r_2 and starts, in each of them, at least one period away from the beginning. Then v has two other occurrences starting at positions $\pi_{mid} - p(r_1)$ and $\pi_{mid} - p(r_2)$ (see Figure 1). The shift between these occurrences is $\Delta = p(r_1) - p(r_2)$ and we claim that $\Delta \geq |v| - p_r$. Otherwise, if $\Delta < |v| - p_r$, then the two occurrences of v have an overlap of length at least $p_r + 1$. Since this overlap is a prefix of the occurrence of v starting at $\pi_{mid} - p(r_2)$, it has a period p_r. Since the overlap is also a suffix of the occurrence of v starting at $\pi_{mid} - p(r_1)$ (see Figure 1), we have that $a_{\pi_{end}+1} = a_{\pi_{end}+1-p_r}$ which is a contradiction.

Thus, $p(r_1)$ and $p(r_2)$ differ by at least $|v| - p_r \geq \frac{|r|}{2} - p_r$. As the periods of considered repetitions are all distinct, and belong to the interval $(p_r, \frac{n}{2})$, there are at most $\frac{n/2-p_r}{|r|/2-p_r} + 1$ of them and therefore as many considered repetitions. Finally, $\frac{n/2-p_r}{|r|/2-p_r} + 1 \leq \frac{n}{p_r(e_r-2)} + 1 \leq \frac{e_r^2}{e_r-2} + 1 = O(e_r)$, and we conclude that there are $O(e_r)$ repetitions in $BLR0(w)$.

After proving that both $LR(w)$ and $RR(w)$ contain $O(e_r)$ maximal repetitions, (4) is rewritten into

$$Sexp(w) < Sexp(w_l) + Sexp(w_r) + Sexp(r_0) + O(e_r). \quad (5)$$

Part 2.2 The next step is to estimate $Sexp(r_0)$ which has to be done by induction. A direct induction argument does not work however, which leads to a more subtle analysis.

We split r_0 into $|r_0|/\Delta$ consecutive blocks of length Δ that will be defined later. (For simplicity, we assume that Δ divides $|r_0|$ evenly; a possible remainder block does not affect the analysis [12].) Then by Lemma 2,

$$Sexp(r_0) < \frac{|r_0|}{\Delta} Sexp(\Delta) + 4 \cdot \sum_{i=1,\ldots,\frac{|r_0|}{\Delta}} \#CR_i, \quad (6)$$

where CR_i is the set of all repetitions in r_0 which cross the boundary between blocks i and $i+1$, $i = 1, \ldots \frac{|r_0|}{\Delta}$. Since r_0 is a repetition with period p_r, by Lemma 1(ii), there is no maximal repetition inside r_0 with a period larger than p_r. Therefore, for any fixed position π in r_0, there are at most $2p_r$ repetitions in r_0 crossing this position (by the same argument as that for $SLR(w)$ in Part 2.1). Thus, $\#CR_i \leq 2p_r$ for any $i = 1, \ldots, \frac{|r_0|}{\Delta}$, and

$$Sexp(r_0) < \frac{|r_0|}{\Delta} Sexp(\Delta) + 8\frac{|r_0|}{\Delta} p_r \leq C_1|r_0| - C_2\frac{|r_0|\log\Delta}{\sqrt{\Delta}} + 8\frac{|r_0|}{\Delta}p_r. \quad (7)$$

The second inequality has been obtained by the induction hypothesis $Sexp(\Delta) \leq C_1\Delta - C_2\sqrt{\Delta}\log\Delta$.

Denote $e_{mid} = |r_0|/p_r$ (the exponent of r_0). For technical reasons we now assume that $p_r \geq 8$ (the case $p_r < 8$ is considered separately [12]), and we choose $\Delta = \lfloor \frac{p_r{}^2}{4} \rfloor$. With this choice of Δ, inequation (7) can be transformed as follows. First, for the chosen Δ, $\frac{\log\Delta}{\sqrt{\Delta}} \geq \frac{\log\frac{p_r^2}{4}}{\sqrt{\frac{p_r^2}{4}}} = \frac{4\log p_r - 4}{p_r}$. The term $8\frac{|r_0|}{\Delta}p_r$ in (7) is $O(e_{mid})$, using the fact that $\frac{|r_0|}{\Delta} = O(\frac{e_{mid}}{p_r})$. We then rewrite (7) as

$$Sexp(r_0) \leq C_1|r_0| - 4C_2 e_{mid}(\log p_r - 1) + O(e_{mid}). \quad (8)$$

Part 2.3 We now count together $Sexp(w_l)$, $Sexp(w_r)$, $Sexp(r_0)$. Recall that according to (5), our goal is to prove

$$Sexp(w_l) + Sexp(w_r) + Sexp(r_0) + O(e_r) \leq$$

$$C_1 n - C_2\sqrt{n}\log n. \qquad (9)$$

which would conclude the induction argument.

Estimating $Sexp(w_l)$, $Sexp(w_r)$ by induction, we have

$$\begin{aligned} Sexp(w_l) &\leq C_1|w_l| - C_2\sqrt{|w_l|}\log|w_l|, & (10)\\ Sexp(w_r) &\leq C_1|w_r| - C_2\sqrt{|w_r|}\log|w_r|. & (11)\end{aligned}$$

Substituting (8), (10), (11) into (9), we are left with the inequation

$$C_2(\sqrt{n}\log n - \sqrt{|w_l|}\log|w_l| - \sqrt{|w_r|}\log|w_r|) + O(e_{mid}) \leq 4C_2 e_{mid}(\log p_r - 1) \quad (12)$$

($O(e_r)$ in (9) has been replaced by $O(e_{mid})$ as $e_r \leq 3e_{mid}$).

The next step is to estimate the expression in parentheses. Using elementary calculus considerations, the following estimation can be proved [12].

$$\sqrt{n}\log n - \sqrt{|w_l|}\log|w_l| - \sqrt{|w_r|}\log|w_r| \leq e_{mid}(2\log p_r + 1). \quad (13)$$

To prove (12), it then suffices to prove

$$C_2 e_{mid}(2\log p_r + 1) + O(e_{mid}) \leq 4C_2 e_{mid}(\log p_r - 1). \quad (14)$$

Recalling that $\log p_r \geq 3$ and $e_{mid} \geq 3$, inequation (14) can be satisfied by choosing a sufficiently large constant C_2. This completes the proof of Theorem 3. Theorem 2 follows.

An important corollary of Theorem 2 is that the maximal number of maximal repetitions in words of length n is linearly-bounded on n. We state this in the following Theorem.

Theorem 4 $\max_{|w|=n} \#R(w) = O(n)$

4. Finding all maximal repetitions in a word

In this section we show how Theorems 2,4 allow to obtain linear-time algorithms for several string matching problems. First, we present a linear-time algorithm for finding all maximal repetitions in a word together with their periods. The algorithm is a modification of Main's algorithm [17] for finding all *leftmost* occurrences of distinct maximal repetitions, which is in turn based on the idea of s-factorization [2], or Lempel-Ziv decomposition [9]. We first describe Main's algorithm.

Definition 1 ([2, 17]) *Let w be an arbitrary word. The* s-factorization *of w is the factorization $w = u_1u_2\ldots u_k$, where u_i's are defined inductively as follows:*

- *If letter a occurring in w immediately after $u_1u_2\ldots u_{i-1}$ does not occur in $u_1u_2\ldots u_{i-1}$, then $u_i = a$.*
- *Otherwise, u_i is the longest word such that $u_1u_2\ldots u_{i-1}u_i$ is a prefix of w and u_i has at least two (possibly overlapping) occurrences in $u_1u_2\ldots u_{i-1}u_i$.*

As an example, the s-factorization of the word 1011010110110 is 1|0|1|101|01101|10. If $w = u_1u_2\ldots u_k$ is the s-factorization, we call u_i's *s-factors*.

The usefulness of s-factorization is explained by the following theorem, which is a slight reformulation of Theorem 3.4 from [17].

Theorem 5 *Let $w = u_1u_2\ldots u_k$ be the s-factorization of w, and let r be a maximal repetition in w such that $initpos(r) \leq initpos(u_i)$ and $initpos(u_i) \leq endpos(r) < endpos(u_i)$. Then $initpos(u_i) - initpos(r) \leq |u_i| + 2|u_{i-1}|$.*

Theorem 5 suggests a partition of all maximal repetitions of w into two classes:

1. repetitions r such that $initpos(r) \leq initpos(u_i)$ and $initpos(u_i) \leq endpos(r) < endpos(u_i)$ for some s-factor u_i,
2. repetitions r such that $initpos(u_i) < initpos(r) < endpos(r) < endpos(u_i)$ for some s-factor u_i.

The above classification does not cover repetitions r which are suffixes of w, but we make this set empty by appending a new symbol \$ at the end of w. This also ensures that the last s-factor u_k consists of one letter. Maximal repetitions verifying conditions 1 and 2 will be called repetitions of type 1 and 2 respectively.

As follows from the definition of s-factorization, every repetition of type 2 has another occurrence on the left. Therefore, finding all repetitions of type 1 guarantees finding all *distinct* maximal repetitions, and in particular all *leftmost* occurrences of distinct maximal repetitions.

Let us describe now how repetitions of type 1 are found by Main's algorithm. Assume we are given the s-factorization $w = u_1\ldots u_k$. By Theorem 5, we have to find, for each $2 \leq i \leq k$, the maximal repetitions in the word t_iu_i, which start in t_i and end in u_i, where t_i is the suffix of $u_1\ldots u_{k-1}$ of length $|u_{i-1}| + 2|u_i|$ (t is the whole word $u_1\ldots u_{i-1}$ in case its length is smaller than $2|u_{i-1}| + |u_i|$). Let us show how to find, in general, all maximal repetition in a word tu that start in t and end in u.

Assume that $t = t[1..m]$, $u = u[1..n]$, and we want to find all maximal repetitions r in the word $v = tu = v[1..m+n]$ such that $initpos(r) \leq m$ and $endpos(r) \geq m$. Every such repetition belongs (non-exclusively) to one of the two classes: the repetitions which have a period in u and those which have a period in t. Note that by Lemma 1(i), for every $1 \leq j \leq n$, there is at most one maximal repetition of period j starting in t, ending in u, and having a period in u. This shows, in particular, that the number of such repetitions is linear in $|u|$. Similarly, the number of such repetitions having a period in t is linear in $|t|$, and thus, the number of maximal repetitions in $v = tu$ which start in $|t|$ and end in u is linear in $|v|$.

Let us focus on maximal repetitions r which have a period in u. The repetitions which have a period in t are found symmetrically. We need two auxiliary functions:

- $LP(i)$, $2 \leq i \leq n+1$ defined by $LP(i) = \max\{j|u[1..j] = u[i..i+j-1]\}$ for $2 \leq i \leq n$, and $LP(n+1) = 0$,
- $LS(i)$, $1 \leq i \leq n$ defined by $LS(i) = \max\{j|t[m-j+1..m] = v[m+i-j+1..m+i]\}$.

Informally, $LP(i)$ is the length of the longest prefix of u which is also a prefix of $u[i..n]$, and $LS(i)$ is the length of the longest suffix of t which is also a suffix of $tu[1..i]$. The following theorem holds.

Theorem 6 ([17]) *For $1 \leq j \leq n$, there exists a maximal repetition of period j in $v = tu$ which starts in t, ends in u and has a period in u iff $LS(j) + LP(j+1) \geq j$. If the inequality holds, this repetition is $v[m - LS(j) + 1..m + j + LP(j+1)]$.*

Function LP can be computed in time linear in $|u|$ and LS in time linear in $|v|$ using the Knuth-Morris-Pratt algorithm (see [17, 4]). Therefore, all maximal repetitions in $v = tu$ which start in t and end in u can be computed in $O(|v|)$ time.

To find all repetitions of type 1 in a word w, the Main's algorithm proceeds as follows. First compute the s-factorization $w = u_1u_2\ldots u_k$. This computation can be done in time $O(|w|)$ using suffix tree construction [18, 23]. Then for each i from 2 to k compute, using the above method, the maximal repetitions in word tu, where u is u_i and t is the suffix of $u_1\ldots u_{i-1}$ of length $2|u_{i-1}| + |u_i|$. Each such computation takes time $O(|u_{i-1}| + |u_i|)$, and therefore finding all maximal repetitions of type 1 takes $O(|w|)$ time.

Note that according to the definition of type 1, at each step we need only those repetitions which end strictly before the end of u_i. The reason for this requirement is that if a repetition is a suffix of $u_1\ldots u_i$, it may not be a maximal repetition, as it may extend in w to the right to a longer repetition. On the other hand, if it is a maximal repetition, it will be found at the next step of the algorithm, and thus will not be missed. Note also that the algorithm may still output the same maximal repetition many times (even unboundedly many times). However, the essential feature is that the algorithm is linear-time and finds all repetitions of type 1.

To find *all* maximal repetitions, we have to find, in addition, all repetitions of type 2. We now show how it can be done. The task is greatly simplified by the fact that every repetition of type 2 occurs entirely inside some s-factor u_i, and each u_i has an earlier occurrence in w.

During the computation of s-factorization we store, for each s-factor u_i, a pointer to an earlier occurrence of u_i in w. This can be easily done using the suffix tree construction, so that the computation of s-factorization remains linear-time. Let v_i be this earlier occurrence of u_i, and let $\Delta_i = initpos(u_i) - initpos(v_i)$. Obviously, each repetition of type 2 occurring inside u_i is a copy of a maximal repetition occurring inside v_i shifted by Δ_i to the right.

We now proceed as follows. First, we compute all maximal repetitions of type 1 with the Main's algorithm. Then we sort them, using basket sort, into n lists, such that list j contains the repetitions with end position j. (Note that during the sort we can eliminate the duplicates.) Then we process all the lists in the increasing order and sort the repetitions again, using basket sort, into n lists according to their initial position. After this double sort, the repetitions with the same initial position j are sorted inside the list j in the increasing order of their end positions. As there is a linear number of repetitions of type 1, both sort procedures take a linear time.

Now we find the repetitions of type 2. We will store them in the same data structure. For each u_i, $i = 1..k$, and for each internal position j inside u_i, we have to find the maximal repetitions of w starting at this position and ending inside u_i. We then have to find the maximal repetitions starting at position $j - \Delta_i$ in v_i which end inside v_i, and then shift them by Δ_i to the right. Note that these repetitions may be either of type 1, or previously found repetitions of type 2. We look through the list $j - \Delta_i$ and retrieve its prefix consisting of those repetitions which end inside v_i. Then we shift each of these repetitions by Δ_i and append a modified copy of this prefix to the head of the list j. Note that the data structure is preserved, as all appended repetitions have their end position inside u_i, and those which have been previously stored in the list j are of type 1 and then have their end position outside u_i. Since we process u_i's from left to right, no repetition can be missed. Thus, we recover all repetitions of type 2 and after all u_i's have been processed, the data structure contains all repetitions of both types.

Note that when we retrieve a prefix of the list corresponding to some position in v_i, each repetition in this prefix results in a new repetition of type 2 in u_i. This shows that the time spent to processing the lists is proportional to the number of newly found repetitions. Theorem 4 from the previous section states that the number of all maximal repetitions is linear in the length of the word. This proves that the whole algorithm takes linear time.

The set of all maximal repetitions, found by the above algorithm in linear time, provides exhaustive information about the repetitive structure of the word. It allows easily to extract all repetitions of other types, such as (primitively- or non-primitively-rooted) squares, cubes, or integer powers. Thus, all these tasks can be done in time $O(n+T)$ where T is the output size (these bounds have been also obtained in [13, 22] with more sophisticated algorithms). Another example is the set of *branching tandem repeats*, notion studied in [21]. In our terminology, branching tandem repeats are (not necessarily primitively-rooted) square suffixes of maximal repetitions. In [21], the authors conjecture that the maximal number of branching tandem repeats in a word is linearly-bounded in the length. Our Theorem 2 confirms that conjecture, since each maximal repetition r contains $\lfloor e(r)/2 \rfloor$ branching tandem repeats, and therefore their total number is $O(n)$. Clearly, the set of maximal repetitions found by the above algorithm, allows to extract all branching tandem repeats. Since their number is linear, finding all branching tandem takes linear time.

As another application, the set of maximal repetitions allows to determine, in linear time, the number of (primitively-rooted) integer powers of a given exponent k, starting at each position of the word. Here is how this can be done. For each position $i \in 1..|w|$, we create two counters $c^b(i)$ and $c^e(i)$, initially set to 0. For each repetition $r = w[m..l]$, we increment $c^b(m)$ and $c^e(l-kp(r)+1)$ by 1 ($[m..l-kp(r)+1]$ is the interval, where primitively-rooted k-powers induced by repetition r start). By Theorem 4, the number of updates is linear. To compute the numbers $d^k(i)$ of k-powers starting at each character i, we scan all characters from left to right applying the following iterative procedure: $d^k(1) = c^b(1)$, $d^k(i+1) = d^k(i) + c^b(i) - c^e(i-1)$, $i = 2..|w|$. Note that the algorithm can be extended to all (not necessarily primitively-rooted) k-powers. In this case, we increment $c^b(m)$ by $\lfloor e(r)/k \rfloor$, and we increment by 1 each $c^e(j)$, for $j = l-kp(r)+1, l-2kp(r)+1, \ldots, l-\lfloor e(r)/k \rfloor kp(r)+1$. Here, Theorem 2 guarantees that the number of updates is linear. Finally, note that the procedure can be easily modified in order to count arbitrary (non-integer) repetitions of given exponent, as well as repetitions ending (or centered) at each position.

5. Concluding remarks

The main drawback of our proof of Theorem 2 is that it does not allow to extract a "reasonable" constant factor in the linear bound. It seems however that this constant factor is quite small. Computer experiments suggest that the number of maximal repetitions is actually smaller than n and the sum of their exponents smaller than $2n$, at least for the binary alphabet. It would be interesting to find a simpler proof of Theorems 2,4 implying a small multiplicative constant in the linear bound.

An experimental implementation of the algorithm described in Section 4 has been recently made by Mathieu Giraud at LORIA/INRIA-Lorraine. The program has been tested on biological sequences, and some interesting long repetitions have been discovered. As expected, the algorithm turned out to be very fast. The "bottleneck" seems to be the memory occupied by a suffix tree-like construction needed for computing the s-factorization. However, strings of 20000 characters could be easily processed.

References

[1] A. Apostolico and F. Preparata. Optimal off-line detection of repetitions in a string. *Theoretical Computer Science*, 22(3):297–315, 1983.

[2] M. Crochemore. An optimal algorithm for computing the repetitions in a word. *Information Processing Letters*, 12:244–250, 1981.

[3] M. Crochemore. Recherche linéaire d'un carré dans un mot. *Comptes Rendus Acad. Sci. Paris Sér. I Math.*, 296:781–784, 1983.

[4] M. Crochemore and W. Rytter. *Text algorithms*. Oxford University Press, 1994.

[5] M. Crochemore and W. Rytter. Squares, cubes, and time-space efficient string searching. *Algorithmica*, 13:405–425, 1995.

[6] A. Fraenkel and J. Simpson. How many squares can a string contain? *J. Combinatorial Theory (Ser. A)*, 82:112–120, 1998.

[7] A. Fraenkel and J. Simpson. The exact number of squares in Fibonacci words. *Theoretical Computer Science*, 218(1):83–94, 1999.

[8] Z. Galil and J. Seiferas. Time-space optimal string matching. *Journal of Computer and System Sciences*, 26(3):280–294, 1983.

[9] D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997.

[10] C. Iliopoulos, D. Moore, and W. Smyth. A characterization of the squares in a Fibonacci string. *Theoretical Computer Science*, 172:281–291, 1997.

[11] R. Kolpakov and G. Kucherov. Maximal repetitions in words or how to find all squares in linear time. Rapport Interne 98-R-227, Laboratoire Lorrain de Recherche en Informatique et ses Applications, 1998. available from http://www.loria.fr/~kucherov/res_activ.html.

[12] R. Kolpakov and G. Kucherov. On the sum of exponents of maximal repetitions in a word. Rapport Interne 99-R-034, Laboratoire Lorrain de Recherche en Informatique et ses Applications, 1999. available from http://www.loria.fr/~kucherov/res_activ.html.

[13] S. R. Kosaraju. Computation of squares in string. In M. Crochemore and D. Gusfield, editors, *Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching*, number 807 in Lecture Notes in Computer Science, pages 146–150. Springer Verlag, 1994.

[14] M. Lothaire. *Combinatorics on Words*, volume 17 of *Encyclopedia of Mathematics and Its Applications*. Addison Wesley, 1983.

[15] M. Main and R. Lorentz. An $O(n \log n)$ algorithm for finding all repetitions in a string. *Journal of Algorithms*, 5(3):422–432, 1984.

[16] M. Main and R. Lorentz. Linear time recognition of square free strings. In A. Apostolico and Z. Galil, editors, *Combinatorial Algorithms on Words*, volume 12 of *NATO Advanced Science Institutes, Series F*, pages 272–278. Springer Verlag, 1985.

[17] M. G. Main. Detecting leftmost maximal periodicities. *Discrete Applied Mathematics*, 25:145–153, 1989.

[18] E. M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM*, 23(2):262–272, 1976.

[19] A. Slisenko. Detection of periodicities and string matching in real time. *Journal of Soviet Mathematics*, 22:1316–1386, 1983.

[20] J. Storer. *Data compression: methods and theory*. Computer Science Press, Rockville, MD, 1988.

[21] J. Stoye and D. Gusfield. Simple and flexible detection of contiguous repeats using a suffix tree. In M. Farach-Colton, editor, *Proceedings of the 9th Annual Symposium on Combinatorial Pattern Matching*, number 1448 in Lecture Notes in Computer Science, pages 140–152. Springer Verlag, 1998.

[22] J. Stoye and D. Gusfield. Linear time algorithms for finding and representing all the tandem repeats in a string. Technical Report CSE-98-4, Computer Science Department, University of California, Davis, 1998.

[23] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.