The Maximum Number of Runs in a String*

František Franěk[1], R. J. Simpson[2], and W. F. Smyth[1,3]

[1] Algorithms Research Group, Department of Computing & Software
McMaster University, Hamilton, Ontario, Canada L8S 4K1
{franek,smyth}@mcmaster.ca
www.cas.mcmaster.ca/cas/research/groups.shtml

[2] Department of Mathematics & Statistics, Curtin University
GPO Box U1987, Perth WA 6845, Australia
simpson@math.curtin.edu.au

[3] Department of Computing, Curtin University, GPO Box U1987
Perth WA 6845, Australia

June 17, 2003

Abstract. A ***run*** (***maximal periodicity***) in a string $\boldsymbol{x}$ is a nonempty substring $\boldsymbol{x}[i..j] = \boldsymbol{u}^k\boldsymbol{u}'$ of minimum period $|\boldsymbol{u}|$, $k \geq 2$, that is "nonextendible" (neither $\boldsymbol{x}[i-1..j]$ nor $\boldsymbol{x}[i..j+1]$ is a run of period $|\boldsymbol{u}|$). Runs provide a basis for computing ***repetitions*** (adjacent repeating substrings) in $\boldsymbol{x}$, and a recent paper presents an algorithm that computes all the runs in $\boldsymbol{x}$ in time linear in $|\boldsymbol{x}|$. If $\rho(n)$ denotes the maximum number of runs that can occur in any string of length n, the same paper also shows that $\rho(n) < kn$, but provides no information about the magnitude of the constant k. In this paper we first identify an infinite family of strings of increasing lengths $n_1, n_2, ...$ such that

$$\lim_{i\to\infty} r(n_i)/n_i = \frac{3}{1+\sqrt{5}},$$

where $r(n_i)$ is the number of runs in the string of length n_i. We provide evidence to support the conjecture that this limit is a maximum over all infinite families of strings. Finally, we establish a restriction on the frequency of occurrence of letters of the alphabet in strings that contain $\rho(n)$ runs.

1 Introduction

The study of repetitions in strings is as old as the study of strings themselves: the paper that is generally considered to have founded "stringology" [12] raised and solved problems about the existence/construction of strings of infinite length that, for a given integer $r \geq 2$, contain no repetitions of exponent r. Much later,

* The work of the first and third authors was supported in part by grants from the Natural Sciences & Engineering Research Council of Canada.

with the invention of digital computers, it became clear that the identification of the repetitions in given finite strings was important in a variety of contexts: computational biology, data compression, cryptology, coding theory, and others. Thus in the early 1980s three quite different repetitions algorithms were proposed [2, 1, 10], all of them executing in $O(n \log n)$ time in the worst case. The efficiency of these algorithms depends critically upon the definition of "repetition" [2], given below.

We use the standard representation of a string as an array, and the concatenation of k occurrences of the same string $\boldsymbol{u}$ is written $\boldsymbol{u}^k$. Then, given a string $\boldsymbol{x} = \boldsymbol{x}[1..n]$, a triple (i, p, r) of positive integers is said to be a ***repetition*** in $\boldsymbol{x}$ if and only if $r \geq 2$ is the largest integer such that $\boldsymbol{x}[i..i{+}rp{-}1] = \boldsymbol{x}[i..i{+}p{-}1]^r$, where $\boldsymbol{x}[i..i{+}p{-}1]$ is not itself a repetition. The integers i, p and r are called the ***position***, the ***period*** and the ***exponent***, respectively, of the repetition. The substring $\boldsymbol{x}[i..i{+}p{-}1]$ is called the ***generator***.

Thus only maximal repetitions of minimum period need to be reported: in $\boldsymbol{y} = 00001010101$, for example, the repetitions are completely specified by the outputs $(1, 1, 4)$, $(4, 2, 4)$ and $(5, 2, 3)$. It was shown in [2] that the number of repetitions in a Fibonacci string $\boldsymbol{f_n}$ ($\boldsymbol{f_0} = 0$; $\boldsymbol{f_1} = 1$; $\forall n \geq 2, \boldsymbol{f_n} = \boldsymbol{f_{n-1}}\boldsymbol{f_{n-2}}$) is $\Theta(|\boldsymbol{f_n}| \log |\boldsymbol{f_n}|)$; it follows that all three of the repetitions algorithms cited above have asymptotically optimal time complexity.

In [9] Main extended the idea of a repetition somewhat: he realized that in some cases output could be reduced because the generators of overlapping repetitions were simply rotations (cyclic shifts) of one another. For example, in the string $\boldsymbol{y}$ shown above, the repetition $(5, 2, 3) = (10)^3$ could be easily inferred from the repetition $(4, 2, 4) = (01)^4$. Formally: a ***run*** (***maximal periodicity***) in a string $\boldsymbol{x}$ is a 4-tuple (i, p, r, t), $t \in 0..p{-}1$, where

$$(i, p, r),\ (i{+}1, p, r),\ \ldots,\ (i{+}t, p, r)$$

and, for $r \geq 3$,

$$(i{+}t{+}1, p, r{-}1),\ (i{+}t{+}2, p, r{-}1),\ \ldots,\ (i{+}p{-}1, p, r{-}1)$$

are all repetitions, but neither $(i{-}1, p, r)$ nor $(i{+}t{+}1, p, r)$ is a repetition. The integer t is the ***tail*** of the run.

Thus a run is ***nonextendible***: it cannot be extended either to left or right. And every run corresponds to $t{+}1$ repetitions of exponent r plus, for $r \geq 3$, an additional $p{-}t{-}1$ repetitions of exponent $r{-}1$. Observe that if $\boldsymbol{u} = \boldsymbol{x}[i..i{+}p{-}1]$ is the generator of a run (i, p, r, t), we may write

$$(i, p, r, t) = \boldsymbol{u}^r \boldsymbol{u}[1..t].$$

In the above example, the runs in $\boldsymbol{y}$ are $(1, 1, 4, 0) = 0^4$ and $(4, 2, 4, 0) = (01)^4$.

In [9] the idea of a run was used to reduce output and so to compute all the "leftmost" runs in $\boldsymbol{x} = \boldsymbol{x}[1..n]$ in $\Theta(n)$ time, assuming that an s-factorization [8, 13], hence a suffix tree, of $\boldsymbol{x}$ had already been computed. It has since been shown [3] that on an ***indexed*** alphabet (that is, equivalent to integers $1..\alpha$

where $\alpha \in O(n)$), a suffix tree of $\boldsymbol{x}$ can be computed in $\Theta(n)$ time; since the s-factorization is also computable from the suffix tree in $\Theta(n)$ time, it follows that Main's algorithm computes the "leftmost" runs in $\boldsymbol{x}$ in linear time.

In [5, 4] it was shown that runs in certain special strings could be calculated in linear time. But then in [6] Kolpakov & Kucherov completed Main's work, showing in general how the "rightmost" runs also could be calculated. In addition they proved that the maximum number $\rho(n)$ of runs that could exist in any string $\boldsymbol{x}[1..n]$ satisfied

$$\rho(n) \leq k_1 n - k_2 \sqrt{n} \log_2 n, \tag{1}$$

for some positive constants k_1 and k_2. Thus, in principle, the calculation of all the runs in $c[1..n]$ can be completed in $\Theta(n)$ time.

This remarkable achievement is not problem-free, however. First, it is not clear that Farach's linear-time suffix tree algorithm [3] is practical for long strings. More generally, the components (suffix tree construction, s-factorization) of the linear-time all-runs algorithm seem to be unnecessarily sophisticated: one would hope to be able to find a more direct approach, based on a more precise and focussed understanding of periodicity in strings. Finally, the very technical and lengthy proof of (1) is not constructive: it provides no information about the size of the constants k_1 and k_2.

At the same time, [6] includes a table, based on exhaustive calculation, that gives $\rho(n)$ for $n = 5, 6, \ldots, 31$, and specifies corresponding ***run-maximal*** strings. On the basis of these results, it seems very likely that the following propositions are true for all $n > 1$ [11]:

(1) $\rho(n) < n$;
(2) $\rho(n) \leq \rho(n-1)+2$ (of course $\rho(n) \geq \rho(n-1)$);
(3) $\rho(n)$ is attained by a cube-free string on $\{0, 1\}$.

To date, however, to our knowledge, none of these simple statements about periodicity has been proved.

In this paper we take a first step toward establishing propositions (1) and (3). In Section 2 we describe constructions suggesting that

$$\lim_{n\to\infty} \rho(n)/n = \frac{3}{1+\sqrt{5}} = 0.92705\cdots.$$

Then in Section 3 we prove a result that partially characterizes the run-maximal strings. Our expectation is that this line of research will eventually enhance our understanding of periodicity in strings so that a simpler and more natural linear-time algorithm for the computation of runs will emerge.

2 Constructing Strings with Many Runs

In this section we produce a sequence of strings in which the ratio of the number of runs to the string length approaches $3/(1+\sqrt{5})$.

We begin by defining an operator $\circ$ that ***composes*** two given strings $\boldsymbol{x}\lambda$, $\mu\boldsymbol{y}$ according to the following rule:

$$\boldsymbol{x}\lambda \circ \mu\boldsymbol{y} = \begin{cases} \boldsymbol{x}\lambda\boldsymbol{y} \text{ if } \lambda = \mu; \\ \boldsymbol{xy} \quad \text{if } \lambda \neq \mu. \end{cases} \tag{2}$$

Thus the ***composition*** of two strings has length *less* by one or two than the sum of the lengths of its components. In our search for run-rich strings, we try therefore to identify strings whose composition contains *more* runs than the sum of the runs in its two components.

Consider the strings

$$\boldsymbol{w_0} = 01\boldsymbol{v_0}10, \quad \boldsymbol{w_1} = 10\boldsymbol{v_1}01,$$

both of length ℓ. Then, for example,

$$\boldsymbol{w_0} \circ \boldsymbol{w_0} = 01\boldsymbol{v_0}101\boldsymbol{v_0}10, \quad \boldsymbol{w_0} \circ \boldsymbol{w_1} = 01\boldsymbol{v_0}10\boldsymbol{v_1}01,$$

and

$$|\boldsymbol{w_0} \circ \boldsymbol{w_0}| = |\boldsymbol{w_1} \circ \boldsymbol{w_1}| = 2\ell - 1, \quad |\boldsymbol{w_0} \circ \boldsymbol{w_1}| = |\boldsymbol{w_1} \circ \boldsymbol{w_0}| = 2\ell - 2. \tag{3}$$

Note that for every $i \in \{0,1\}$, $j \in \{0,1\}$, $\boldsymbol{w_i} \circ \boldsymbol{w_j}$ contains $\boldsymbol{w_i}$ as a prefix and $\boldsymbol{w_j}$ as a suffix. Let $r(\boldsymbol{x})$ denote the number of runs in a string $\boldsymbol{x}$, and suppose that $r(\boldsymbol{w_0}) = r(\boldsymbol{w_1}) = k$.

Now we define mappings

$$f(0) = \boldsymbol{w_0}, \quad f(1) = \boldsymbol{w_1}, \tag{4}$$

and, for any string $\boldsymbol{x} = \boldsymbol{x}[1..n]$ on $\{0,1\}$,

$$g(\boldsymbol{x}) = f(\boldsymbol{x}[1]) \circ f(\boldsymbol{x}[2]) \circ \cdots \circ f(\boldsymbol{x}[n]). \tag{5}$$

Next suppose that $\boldsymbol{v_0}$ and $\boldsymbol{v_1}$ are chosen to be the shortest strings that satisfy the following condition:

> Every possible composition of $\boldsymbol{w_0}$ and $\boldsymbol{w_1}$ (that is, $\boldsymbol{w_0} \circ \boldsymbol{w_0}$, $\boldsymbol{w_0} \circ \boldsymbol{w_1}$, $\boldsymbol{w_1} \circ \boldsymbol{w_0}$, $\boldsymbol{w_1} \circ \boldsymbol{w_1}$) contains the $2k$ runs of its components together with one additional run.

Thus we suppose that

$$r(\boldsymbol{w_0} \circ \boldsymbol{w_1}) = r(\boldsymbol{w_1} \circ \boldsymbol{w_0}) = 2k+1 \tag{6}$$

and, since $\boldsymbol{w_0} \circ \boldsymbol{w_0}$ and $\boldsymbol{w_1} \circ \boldsymbol{w_1}$ are themselves runs,

$$r(\boldsymbol{w_0} \circ \boldsymbol{w_0}) = r(\boldsymbol{w_1} \circ \boldsymbol{w_1}) = 2k+2. \tag{7}$$

Based on these assumptions, we can compute $|g(\boldsymbol{x})|$ and $r\big(g(\boldsymbol{x})\big)$ for any string $\boldsymbol{x} = \boldsymbol{x}[1..n]$:

– Let q denote the number of occurrences of either 00 or 11 in $\boldsymbol{x}$. Then, using (3)–(5),

$$\begin{aligned}|g(\boldsymbol{x})| &= \ell n - q - 2(n-q-1)\\ &= (\ell-2)n + (q+2).\end{aligned} \tag{8}$$

– Since every run in $\boldsymbol{x}$ becomes a run in $g(\boldsymbol{x})$,

$$\begin{aligned}r\big(g(\boldsymbol{x})\big) &= r(\boldsymbol{x}) + kn + (n-1)\\ &= r(\boldsymbol{x}) + (k+1)n - 1.\end{aligned} \tag{9}$$

Now we consider iterating the compositions of $\boldsymbol{w_0}$ and $\boldsymbol{w_1}$ beginning with some string $\boldsymbol{x}$:

$$g^0(\boldsymbol{x}) = \boldsymbol{x};\quad g^i(\boldsymbol{x}) = g\big(g^{i-1}(\boldsymbol{x})\big), \forall i \geq 1. \tag{10}$$

For example, if $\boldsymbol{x} = 0$, then $g^0 = 0$, $g^1 = \boldsymbol{w_0}$, $g^2 = \boldsymbol{w_0 w_1} g(\boldsymbol{v_0}) \boldsymbol{w_1 w_0}$, and so on. We in fact choose $\boldsymbol{x} = 0$ and set $\boldsymbol{x_i} = g^i(0)$, $i = 0, 1, \ldots$. If $n_i = |\boldsymbol{x_i}|$, $m_i = r(\boldsymbol{x_i})$, we can rewrite (8) and (9) for every $i \geq 0$ as follows:

$$n_{i+1} = (\ell-2)n_i + (q_i+2), \tag{11}$$

$$m_{i+1} = m_i + (k+1)n_i - 1, \tag{12}$$

where q_i is defined to be the number of occurrences of 00/11 in $g^i(0)$. Since $\boldsymbol{x} = 0$, $q_0 = 0$; for $i > 0$, we suppose that $\boldsymbol{w_0}$ and $\boldsymbol{w_1}$ both contain q occurrences of 00/11, so that therefore $q_i = qn_{i-1}$. Thus for $i \geq 1$, we can use (11) to compute

$$\frac{n_i}{n_{i+1}} = \frac{1}{(\ell-2) + q\left(\frac{n_{i-1}}{n_i}\right) + \frac{2}{n_i}},$$

and then, setting $A = \lim_{i\to\infty} \frac{n_i}{n_{i+1}}$, we find

$$A = \frac{1}{(\ell-2) + qA},$$

a quadratic equation in A whose positive solution is

$$A = \frac{\sqrt{(\ell-2)^2 + 4q} - (\ell-2)}{2q}. \tag{13}$$

Similarly, using both (11) and (12), we can write

$$\frac{m_{i+1}}{n_{i+1}} = \frac{\frac{m_i}{n_i} + (k+1) - \frac{1}{n_i}}{(\ell-2) + q\frac{n_{i-1}}{n_i} + \frac{2}{n_i}},$$

from which, setting $B = \lim_{i\to\infty} \frac{m_{i+1}}{n_{i+1}}$, we find

$$B = \frac{k+1}{(\ell-3) + qA}. \tag{14}$$

Recall that in order to get the maximum number of runs from our construction, $\boldsymbol{v_0}$ and $\boldsymbol{v_1}$ were defined to be the *shortest* strings, both of length ℓ–4, satisfying (6) and (7). It is easy to verify that no choice for $\ell = 5$ can satisfy these conditions. However, for $\ell = 6$, we can choose

$$\boldsymbol{v_0} = 00, \ \boldsymbol{v_1} = 11, \tag{15}$$

satisfying (6) and (7) with $k = 2$, $q = 1$. In fact, no other choice for $\ell = 6$ improves on (15). Making appropriate substitutions in (13) and (14), we find $A = -2+\sqrt{5}$ and

$$\lim_{i\to\infty} \frac{m_i}{n_i} = \frac{3}{1+\sqrt{5}} = \frac{3}{2\phi} = 0.92705\cdots, \tag{16}$$

where ϕ is the ***golden mean***.

We remark that the recurrences (11) and (12) can also be solved directly for $\ell = 6, k = 2, q = 1$, yielding

$$n_i = \frac{5+\sqrt{5}}{20}(2+\sqrt{5})^{i+1} + \frac{5-\sqrt{5}}{20}(2-\sqrt{5})^{i+1} - \frac{1}{2},$$
$$m_i = \frac{3\sqrt{5}}{20}\big((2+\sqrt{5})^{i+1} - (2-\sqrt{5})^{i+1}\big) - \frac{5i+3}{2}.$$

Here are some values of $m(i)$ and $n(i)$:

i	m_i	n_i	m_i/n_i
0	0	1	0.000
1	2	6	0.333
2	19	27	0.704
3	99	116	0.853
4	446	493	0.905
5	1924	2090	0.921

We note that the initial values of this construction do not produce run-optimal strings: $\rho(6) = 3 > m_1$ and $\rho(27) = 21 > m_2$. Nevertheless we state the following

Conjecture 1. $\lim_{n\to\infty} \rho(n)/n = 3/2\phi$.

Note also that other infinite sequences of strings can be constructed by making alternate choices of $\boldsymbol{x}$, $\boldsymbol{w_0}$, $\boldsymbol{w_1}$ in (10).

3 The Nature of Run-Maximal Strings

From the available evidence [6, 7] it seems that for every n there exists a run-maximal string on the alphabet $\{0, 1\}$. In this section we provide some support for this conjecture.

Theorem 1. *Let $\boldsymbol{x} = \boldsymbol{x}[1..n]$ be a run-maximal string that contains $\alpha \geq 3$ distinct letters. Suppose that one of these letters λ occurs fewer than three times. Then there exists a run-maximal string of length n that contains $\alpha - 1$ distinct letters.*

Proof. First suppose that $\boldsymbol{x}$ contains exactly one occurrence of λ. Then $\boldsymbol{x}$ takes the form $\boldsymbol{u}\lambda\boldsymbol{v}$. Observe that if λ is either a prefix or a suffix of $\boldsymbol{x}$, we can simply replace it by any one of the other letters and so satisfy the statement of the theorem. We suppose therefore that the strings $\boldsymbol{u}$ and $\boldsymbol{v}$ are nonempty.

Consider now the removal of λ from $\boldsymbol{x} = \boldsymbol{x_0}$. The number of runs cannot increase as a result and in fact may be reduced by the coalescence of one or more runs that are suffixes of $\boldsymbol{u}$ and prefixes of $\boldsymbol{v}$. If there is no such reduction, we can simply move λ to the right end of $\boldsymbol{x_0}$ in order to form $\boldsymbol{x_1}$, and so as discussed above satisfy the theorem. If however there is coalescence, $\boldsymbol{u}$ must terminate with a nonempty square, $\boldsymbol{w_1}^2$ say, that is also a prefix of $\boldsymbol{v}$. We may suppose that $\boldsymbol{w_1}$ is the generator of a run and therefore not itself a repetition. We suppose further that $\boldsymbol{w_1}^2$ is the longest such square and so write $\boldsymbol{x_0} = \boldsymbol{u'}\boldsymbol{w_1}^2\lambda\boldsymbol{w_1}^2\boldsymbol{v'}$, where $\boldsymbol{u} = \boldsymbol{u'}\boldsymbol{w_1}^2$, $\boldsymbol{v} = \boldsymbol{w_1}^2\boldsymbol{v'}$. Now consider

$$\boldsymbol{x_1} = \boldsymbol{u'}\boldsymbol{w_1}^2\boldsymbol{v'}\lambda\boldsymbol{w_1}^2. \tag{17}$$

It is clear that $\boldsymbol{x_1}$ cannot contain more runs than $\boldsymbol{x_0}$. Suppose then that $\boldsymbol{x_1}$ contains fewer runs than $\boldsymbol{x_0}$. It follows that there must exist some run in $\boldsymbol{u'}\boldsymbol{w_1}^2$ that coalesces with a run in $\boldsymbol{w_1}^2\boldsymbol{v'}$. In order for this coalescence to take place, a generator $\boldsymbol{z} \neq \boldsymbol{w_1}$ of the run must coincide with the beginning of $\boldsymbol{w_1}^2$ and in fact it must be true that $\boldsymbol{w_1}^2 = \boldsymbol{z}^s\boldsymbol{z'}$ for some prefix $\boldsymbol{z'}$ of $\boldsymbol{z}$.

Since we chose $\boldsymbol{w_1}^2$ to be the longest square, it follows that $|\boldsymbol{w_1}| > |\boldsymbol{z}|$, hence that $s \geq 2$. Since $\boldsymbol{w_1}$ is not a repetition, $|\boldsymbol{z}|$ cannot divide $|\boldsymbol{w_1}|$. Consequently both $\boldsymbol{z}$ and some nontrivial rotation (cyclic shift) of $\boldsymbol{z}$ are simultaneously a prefix of $\boldsymbol{w_1}$, hence equal. But since $\boldsymbol{z}$ is a generator of a run and therefore not a repetition, this is impossible. We conclude that $\boldsymbol{x_1}$ must contain the same number of runs as $\boldsymbol{x_0}$.

Two cases now arise: if $\boldsymbol{v'} \neq \varepsilon$, λ occurs at a position in $\boldsymbol{x_1}$ that lies to the right of its position in $\boldsymbol{x_0}$; if however $\boldsymbol{v'} = \varepsilon$, then we use an isomorphism of the set of letters of $\boldsymbol{w_1}$ into itself (of course excluding λ) to transform the suffix $\boldsymbol{v}$ of $\boldsymbol{x_0}$ into $\widehat{\boldsymbol{w_1}}^2$. This transformation leaves the number of runs in $\boldsymbol{x}$ unchanged, but now the longest square suffix of $\boldsymbol{u}$ that is also a prefix of $\boldsymbol{v}$ must be a *proper* prefix of $\boldsymbol{v}$. Thus we can always determine a string (17) that shifts λ to the right.

We can continue, applying the same transformation to $\boldsymbol{x_1}$ to compute a new string $\boldsymbol{x_2}$ that achieves $\rho(n)$ runs either without λ or with a suffix $\lambda\boldsymbol{w_2}^2$, $0 < |\boldsymbol{w_2}| < |\boldsymbol{w_1}|$. Continuing this process eventually determines a string $\boldsymbol{x_r}$ of length $n-1$ that achieves $\rho(n)$ runs but does not contain λ, as required.

Suppose then that $\boldsymbol{x} = \boldsymbol{u}\lambda\boldsymbol{v}\lambda\boldsymbol{w}$ contains exactly two occurrences of λ, and observe that these occurrences can participate in at most one run. It follows therefore that we can use the transformation just described to form first $\boldsymbol{y}\lambda$ from $\boldsymbol{v}\lambda\boldsymbol{w}$, then $\boldsymbol{z}\lambda$ from $\boldsymbol{u}\lambda\boldsymbol{y}$, so that $\boldsymbol{x}$ is transformed into $\boldsymbol{z}\lambda\lambda$, where $\boldsymbol{z}$ contains at least $\rho(n)-1$ runs. In fact, since $\lambda\lambda$ is a run, we see that $\boldsymbol{z}$ contains exactly $\rho(n)-1$ runs, hence that $\boldsymbol{z}\lambda\lambda$ contains $\rho(n)$ runs. But then we can replace λ by any letter of $\boldsymbol{z}$ that is not a suffix of $\boldsymbol{z}$ and so achieve $\rho(n)$ runs without λ. $\Box$

Remark: A run-maximal string of length $n \geq 2$ can contain exactly one occurrence of a letter λ if and only if $\rho(n) = \rho(n-1)$. Note more generally that $\rho(n+1) \geq \rho(n-1) + 1$ in all cases.

References

[1] Alberto Apostolico & Franco P. Preparata, **Optimal off-line detection of repetitions in a string**, *TCS 22* (1983) 297–315.

[2] Maxime Crochemore, **An optimal algorithm for computing the repetitions in a word**, *IPL 12–5* (1981) 244–250.

[3] Martin Farach, **Optimal suffix tree construction with large alphabets**, *Proc. 38th Annual IEEE Symp. FOCS* (1997) 137–143.

[4] František Franěk, Ayşe Karaman & W. F. Smyth, **Repetitions in Sturmian strings**, *TCS 249–2* (2000) 289–303.

[5] Costas S. Iliopoulos, Dennis Moore & W. F. Smyth, **A characterization of the squares in a Fibonacci string**, *TCS 172* (1997) 281–291.

[6] Roman Kolpakov & Gregory Kucherov, **On maximal repetitions in words**, *J. Discrete Algorithms 1* (2000) 159–186.

[7] Roman Kolpakov & Gregory Kucherov, private communication (2001).

[8] Abraham Lempel & Jacob Ziv, **On the complexity of finite sequences**, *IEEE Trans. Information Theory 22* (1976) 75–81.

[9] Michael G. Main, **Detecting leftmost maximal periodicities**, *Discrete Applied Maths. 25* (1989) 145–153.

[10] Michael G. Main & Richard J. Lorentz, **An $O(n \log n)$ algorithm for finding all repetitions in a string**, *J. Algs. 5* (1984) 422–432.

[11] W. F. Smyth, **Repetitive perhaps, but certainly not boring**, *TCS 249–2* (2000) 289–303.

[12] Axel Thue, **Über unendliche zeichenreihen**, *Norske Vid. Selsk. Skr. I. Mat. Nat. Kl. Christiana 7* (1906) 1–22.

[13] Jacob Ziv & Abraham Lempel, **A universal algorithm for sequential data compression**, *IEEE Trans. Information Theory 23* (1977) 337–343.