Computing Periodicities in Strings — A New Approach*

W. F. Smyth[1,2]

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

[2] Department of Computing, Curtin University, GPO Box U1987
Perth WA 6845, Australia
smyth@computing.edu.au

November 14, 2007

Abstract. The most efficient methods currently available for the computation of repetitions or repeats in a string $\boldsymbol{x} = \boldsymbol{x}[1..n]$ all depend on the prior computation of a suffix tree/array $\mathrm{ST}\boldsymbol{x}/\mathrm{SA}\boldsymbol{x}$. Although these data structures can be computed in asymptotic $\Theta(n)$ time, nevertheless in practice they involve significant overhead, both in time and space. Since the number of repetitions/repeats in $\boldsymbol{x}$ can be reported in a way that is at most linear in string length, it therefore seems that it should be possible to devise less roundabout means of computing repetitions/repeats that take advantage of their infrequent occurrence. This survey paper provides background for these ideas and explores the possibilities for more efficient computation of periodicities in strings.

1 Introduction

The mathematical study of ***strings*** (***words***) began with an investigation of their periodicity properties [25], a focus that has been sustained over the intervening 100 years. But in addition, periodicity has turned out to be important from a computational/algorithmic point of view: in bioinformatics (repeating subsequences of DNA or protein), for data compression, in cryptanalysis, and in various other contexts. Thus algorithms to compute or recognize periodicities in strings were among the earliest to be developed when the importance of string processing was first recognized about 35–40 years ago.

* This research was supported in part by grants from the Natural Sciences & Engineering Research Council of Canada.

This survey paper begins in Section 2 with an overview of the most important kinds of periodicity and the algorithms that compute them. Then in Section 3 a start is made on a "theory of periodicities" that may provide clues about their more efficient computation. Finally, Section 4 briefly discusses future directions for research.

2 Periodicities

This section defines the most common and important kinds of periodicity in strings and characterizes the algorithms that compute them. Essentially, we discover that periodicities are in some sense infrequent (at most linear in string length), but that the algorithms that compute them, though asymptotically efficient, are also rather complex, perhaps more complex than they need to be. Terminology and notation generally follow [24].

Suffix Structures

Since existing algorithms almost always use suffix structures to achieve effectiveness and efficiency, we briefly define them here.

A ***suffix tree*** $\mathrm{ST}_{\boldsymbol{x}}$ of a given string $\boldsymbol{x}$ is a ***compacted trie*** on the nonempty suffixes of $\boldsymbol{x}$ [26, 22], in which every terminal node identifies a suffix and every internal node I identifies the ***least common prefix*** (LCP) of all the terminal nodes of the subtree rooted at I. Terminal nodes are labelled i, identifying suffix $\boldsymbol{x}[i..n]$, and internal nodes are labelled $\mathrm{lcp} = |\mathrm{LCP}|$, the length of the LCP of the terminal nodes in that subtree. Thus the terminal nodes of each subtree identify suffixes whose prefixes of length lcp are repeating substrings in $\boldsymbol{x}$. Figure 1 shows the suffix tree of $\boldsymbol{x} = abaababa$. As is usual with suffix trees, the terminal nodes of $\mathrm{ST}_{abaababa}$ taken in left-to-right order give the suffixes in lexicographic order; however, for the calculation of periodicities, this property is not necessary.

The ***suffix array*** $\mathrm{SA}_{\boldsymbol{x}}$ is just the array of the suffixes i of $\boldsymbol{x}$ read off from $\mathrm{ST}_{\boldsymbol{x}}$ in a preorder (left-to-right) traversal. Sometimes also used in conjunction with $\mathrm{ST}_{\boldsymbol{x}}$ is an lcp array $\mathrm{lcp}_{\boldsymbol{x}}$ in which, for every $j \in 2..n$,

$$\mathrm{lcp}_{\boldsymbol{x}}[j] = \mathrm{lcp}\big(\mathrm{SA}_{\boldsymbol{x}}[j-1], \mathrm{SA}_{\boldsymbol{x}}[j]\big).$$

The lcp array can also be read off from $\mathrm{ST}_{\boldsymbol{x}}$ in a preorder traversal. For example:

$$\begin{array}{rl} & \scriptstyle 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8 \\ \boldsymbol{x} = & a\ b\ a\ a\ b\ a\ b\ a \\ \mathrm{SA}_{\boldsymbol{x}} = & 8\ 3\ 6\ 1\ 4\ 7\ 2\ 5 \\ \mathrm{lcp}_{\boldsymbol{x}} = & -\ 1\ 1\ 3\ 3\ 0\ 2\ 2 \end{array}$$

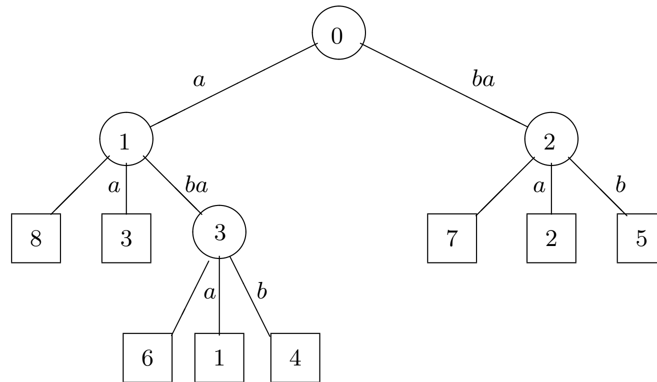

Fig. 1. $ST_{abaababa}$

Repetitions

A ***repetition*** in a string $\boldsymbol{x} = \boldsymbol{x}[1..n]$ is a substring $\boldsymbol{x}[i..i{+}pe{-}1] = \boldsymbol{u}^e$, where $|\boldsymbol{u}| = p$ and $e \geq 2$. If moreover $\boldsymbol{u}$ itself is not a repetition, then $\boldsymbol{u}^e$ is said to be ***irreducible***; and if neither $\boldsymbol{x}[i{-}p..i{-}1]$ nor $\boldsymbol{x}[i{+}pe..i{+}p(e{+}1){-}1]$ equals $\boldsymbol{u}$, then $\boldsymbol{u}^e$ is said to be ***maximal***. Unless specifically stated otherwise, all repetitions referred to in this paper are both irreducible and maximal.

We call $\boldsymbol{u}$ the ***generator***, p the ***period***, and e the ***exponent*** of the repetition $\boldsymbol{u}^e$. Note that a repetition is completely specified by the triple (i, p, e). In the string

$$\begin{array}{rl} & \scriptstyle 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8 \\ \boldsymbol{x} = & a\ b\ a\ a\ b\ a\ b\ a, \end{array}$$

the repetitions are $(1, 3, 2) = (aba)^2$, $(3, 1, 2) = a^2$, $(4, 2, 2) = (ab)^2$, and $(5, 2, 2) = (ba)^2$. Since in each case $e = 2$, all of these are ***squares***.

About a quarter-century ago, three algorithms were discovered [4, 2, 21] that employed widely different approaches to compute all the repetitions in a given string $\boldsymbol{x}[1..n]$ in $O(n \log n)$ time; of these algorithms, two were based on a form of suffix tree calculation ([2] explicitly, [4] implicitly), while the third used a divide-and-conquer technique. In [4] it was moreover shown that the Fibonacci string $\boldsymbol{f_K}$:

$$\boldsymbol{f_0} = b,\ \boldsymbol{f_1} = a;\quad \boldsymbol{f_k} = \boldsymbol{f_{k-1}} \boldsymbol{f_{k-2}},\ k = 2, 3, \ldots, K$$

actually contains $\Theta\big(|\boldsymbol{f_K}| \log |\boldsymbol{f_K}|\big)$ repetitions. Hence these algorithms were regarded as asymptotically optimal, a concept that as we shall see depends heavily on what is accepted as a sufficient specification of a repetition.

Runs

It was mentioned above that the maximum number of repetitions in a string $\boldsymbol{x} = \boldsymbol{x}[1..n]$ is $\Theta(n \log n)$. But this is a count of repetitions that are both maximal and irreducible. If instead we were asked to output the distinct squares $\boldsymbol{u}^2$ without these restrictions, we would find that $\boldsymbol{x} = a^n$, for example, would require $\lceil n^2/4 \rceil$ — that is, $\Theta(n^2)$ — outputs to specify squares $\boldsymbol{x}[1..2], \boldsymbol{x}[2..3], \ldots, \boldsymbol{x}[n-1..n], \boldsymbol{x}[1..4], \boldsymbol{x}[2..5], \ldots, \boldsymbol{x}[n-3..n]$, and so on. Thus in restricting the output to maximal irreducible repetitions, we ***encode*** the output, by tacit agreement with the user, so as to reduce its quantity, hence the asymptotic complexity of the algorithm. For $\boldsymbol{x} = a^n$, this encoding dramatically reduces the output to a single repetition $(1, 1, n)$.

We now describe another encoding of repetitions that further reduces the quantity of output required to $\Theta(n)$. We say that a repetition $(i, p, e) = \boldsymbol{u}^e$ is ***left-extendible*** (LE) if there exists a repetition at position $i-1$ of $\boldsymbol{x}$ that is also of period p. If no such repetition exists, we say that (i, p, e) is NLE. Given an NLE repetition (i, p, e), denote by t the greatest integer such that, for every $j \in 0..t$, $(i+j, p, e)$ is a repetition. Note that since (i, p, e) is maximal, therefore $t \in 0..p-1$. We call t the ***tail*** of (i, p, e). Then a ***run*** (***maximal periodicity***) is a 4-tuple (i, p, e, t), where (i, p, e) is an NLE repetition of tail t.

The idea of a run was first introduced by Main in [20], where also an algorithm was proposed to compute the leftmost occurrence of every distinct run in $\boldsymbol{x}[1..n]$. Given the suffix tree $\mathrm{ST}_{\boldsymbol{x}}$ and the Lempel-Ziv decomposition [18] of $\boldsymbol{x}$ (computable in linear time from $\mathrm{ST}_{\boldsymbol{x}}$), Main's algorithm computes all the leftmost runs in $\Theta(n)$ time. In [17] Kolpakov

& Kucherov showed that the maximum number $\rho(n)$ of runs in any string of length n satisfies

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

for some pair of universal positive constants k_1 and k_2. They also extended Main's algorithm to compute *all* the runs in $\boldsymbol{x}$ in time proportional to their number; thus by (1), given $\mathrm{ST}_{\boldsymbol{x}}$, all the runs in $\boldsymbol{x}$, and so in effect all the repetitions, could be computed in $\Theta(n)$ time.

Repeats

A ***repeat*** in $\boldsymbol{x}$ is a tuple

$$M_{\boldsymbol{x},\boldsymbol{u}} = (p; i_1, i_2, \ldots, i_e),$$

where $e \geq 2$, $1 \leq i_1 < i_2 < \cdots < i_e \leq n$, and

$$\boldsymbol{u} = \boldsymbol{x}[i_1..i_1{+}p{-}1] = \boldsymbol{x}[i_2..i_2{+}p{-}1] = \cdots = \boldsymbol{x}[i_e..i_e{+}p{-}1].$$

Note that it may happen, for some $j \in 1..e{-}1$, that $i_{j+1}{-}i_j = p$ or that $i_{j+1}{-}i_j < p$ — that is, the substrings of a repeat may form a repetition or even overlap. Analogous to a repetition, we call $\boldsymbol{u}$ the ***generator***, p the ***period***, and e the ***exponent*** of $M_{\boldsymbol{x},\boldsymbol{u}}$; also, $M_{\boldsymbol{x},\boldsymbol{u}}$ is called a ***square*** if $e = 2$. We say that $M_{\boldsymbol{x},\boldsymbol{u}}$ is ***maximal*** if for every

$$i \in 1..n \text{ and } i \notin \{i_1, i_2, \ldots, i_e\},$$

we are assured that $\boldsymbol{x}[i..i{+}p{-}1] \neq \boldsymbol{u}$. Again analogous to runs, we say that $M_{\boldsymbol{x},\boldsymbol{u}}$ is ***left-extendible*** (LE) if

$$(p; i_1{-}1, i_2{-}1, \ldots, i_e{-}1)$$

is a repeat; in this case, $(p{+}1; i_1{-}1, i_2{-}1, \ldots, i_e{-}1)$ is a repeat whose suffixes of length p are specified by $M_{\boldsymbol{x},\boldsymbol{u}}$. Similarly, $M_{\boldsymbol{x},\boldsymbol{u}}$ is ***right-extendible*** (RE) if

$$(p; i_1{+}1, i_2{+}1, \ldots, i_e{+}1)$$

is a repeat; in this case, $(p{+}1; i_1, i_2, \ldots, i_e)$ is a repeat whose prefixes of length p are specified by $M_{\boldsymbol{x},\boldsymbol{u}}$. If $M_{\boldsymbol{x},\boldsymbol{u}}$ is neither LE nor RE, we say that it is ***nonextendible*** (NE). Unless explicitly stated otherwise, all repeats discussed in this paper are both maximal and NE.

In $\boldsymbol{x} = abaababa$, the repeats are

$$M_{\boldsymbol{x},a} = (1; 1, 3, 4, 6, 8) \text{ and } M_{\boldsymbol{x},aba} = (3; 1, 4, 6);$$

since every occurrence of b is both preceded and followed by a, there are no others.

Of particular interest are repeating substrings $\boldsymbol{u}$ such that $M_{\boldsymbol{x},\boldsymbol{v_1}\boldsymbol{u}\boldsymbol{v_2}}$ is a repeat if and only if $\boldsymbol{v_1} = \boldsymbol{v_2} = \boldsymbol{\varepsilon}$, the empty string — in other words, $\boldsymbol{u}$ is not a proper substring of any other repeating substring. We call such repeats ***supernonextendible*** (SNE). The repeating substring $\boldsymbol{u}$ in an SNE repeat $M_{\boldsymbol{x},\boldsymbol{u}}$ is in some sense the longest in a class of nested repeating substrings that are substrings of $\boldsymbol{u}$. In the above example, $(3; 1, 4, 6)$ is the unique SNE repeat.

In [11, p. 147] an algorithm is described that, given the suffix tree $\mathrm{ST}_{\boldsymbol{x}}$ of $\boldsymbol{x}$, computes all the NE squares (whether maximal or not) in $\boldsymbol{x}$ in time $O(n+q)$, where q is the number of squares output. [3] uses similar methods to compute all NE squares $(p; i_1, i_2)$ such that $i_2 - i_1 \leq g$ for some user-defined ***gap*** g. [1] shows how to use the suffix array $\mathrm{SA}_{\boldsymbol{x}}$ of $\boldsymbol{x}$ to compute the NE squares and the SNE squares, both in time $O(n+q)$. [10] uses either the suffix trees of both $\boldsymbol{x}$ and its reversed string $\overline{\boldsymbol{x}} = \boldsymbol{x}[n]\boldsymbol{x}[n-1] \cdots \boldsymbol{x}[1]$, or alternatively the suffix arrays of both, to compute all the (maximal and NE) repeats in $\boldsymbol{x}$ in $\Theta(n)$ time. Thus the number of repeats in $\boldsymbol{x}$ is linear in string length.

There is a noteworthy distinction between the reporting of repetitions/runs on the one hand and repeats on the other. Suppose

$$\boldsymbol{x} = \cdots \overset{i_1}{abababab} \cdots \overset{i_2}{abababab} \cdots,$$

where neither a nor b occurs elsewhere in $\boldsymbol{x}$. Then two separate repetitions $(i_1, 2, 4) = (i_2, 2, 4) = (ab)^4$ will be reported in $\boldsymbol{x}$, whereas a single SNE repeat $(8; i_1, i_2) = abababab$ will be reported. The period of the repetition or run is *reduced* as much as possible, while the period of the SNE repeat is *extended* as much as possible.

Summary

We have seen in this section that the cardinality of the basic periodicities in a string (repetitions/runs and repeats) is in some sense linear in string length; further that, using suffix trees, these periodicities can be computed in linear time. Since every computation using a suffix tree $\mathrm{ST}_{\boldsymbol{x}}$ can be replaced by an equivalent computation using a suffix array $\mathrm{SA}_{\boldsymbol{x}}$ with no sacrifice in asymptotic complexity [1], and since $\mathrm{SA}_{\boldsymbol{x}}$ can be computed in guaranteed $\Theta(n)$ time [14–16], it follows that the total time requirement for the computation of periodicities is $\Theta(n)$. This is a

significant achievement, but, as we discuss in the next section, there is some reason to believe that even "better" (that is, faster and simpler) algorithms can be found.

3 A Theory of Periodicities?

It turns out that the linear-time suffix array construction algorithms (SACAs) are not the fastest in practice! As shown in [23], there are several other SACAs with supralinear worst-case behaviour that, over a wide range of frequently-occurring strings, require less than half of both the time and the space used by their linear-time cousins. Even so, the space requirements in particular are substantial: at least 5 bytes per symbol. This means that for a large string ($n \geq 10^9$, say), most computers would need to use secondary storage for an additional $5n$ bytes, a requirement that would have a disastrous effect on the speed of the algorithm. Thus it becomes of interest to seek ways of avoiding suffix array construction altogether.

Since as we have seen runs and repeats can both be reported in linear time, and are therefore in some sense infrequent in strings, an alternate approach to their calculation is not *a priori* absurd. For repeats no such approach as to my knowledge been studied, but for runs theoretical questions arise whose resolution could plausibly lead to new methods, as we now explain.

The result (1) is flawed in a certain sense: its proof is nonconstructive, and so the magnitude of the constants k_1 and k_2 is completely unknown. At the same time, [17] provides convincing experimental evidence that the following statements are true:

(a) $\rho(n) < n$;
(b) $\rho(n+1) - \rho(n) \leq 2$;
(c) there exists a cube-free string $\boldsymbol{x}[1..n]$ on $\{a, b\}$ that contains $\rho(n)$ runs.

These conjectures are fundamental ones about periodicity in strings, yet they have scarcely been studied.

Conjecture (a) relates to the well-known conjecture that

$$\sigma(n) < n, \tag{2}$$

where $\sigma(n)$ is the maximum number of *distinct* squares that can occur in any string of length n. Since every run must begin with a square, it follows that $\sigma(n) \leq \rho(n)$, so that a proof of (a) would establish (2). In

fact, it has been shown that $\sigma(n) \leq 2n-2$ [8,12], more recently that $\sigma(n) \leq 2n-\Theta(\log n)$ [13].

[9] identifies an infinite family of strings $\boldsymbol{x}[1..n]$ containing a number $r(n)$ of runs, where

$$\lim_{n\to\infty} \frac{r(n)}{n} = \frac{3}{2\phi}$$

and $\phi = (1+\sqrt{5})/2$ is the ***golden mean***. It is conjectured that this limit is a maximum over all infinite families of strings. In the same paper the following theorem is proved (a string is said to be ***run-maximal*** if it contains $\rho(n)$ runs):

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.* $\Box$

This result, not easy to prove, succeeds only in establishing conjecture (c) in the very special case that all letters except two occur at most twice.

In order to establish conjecture (a), one needs to somehow limit to less than one the average number of runs that begin at the positions of $\boldsymbol{x}$. This requirement draws attention to positions i where two or more runs begin: one hopes to be able to show that at positions "neighbouring" to i, no runs can begin. To date, the most famous and important theorem restricting periodicity is the "periodicity lemma" [7]:

Theorem 2 *Let p and q be two periods of $\boldsymbol{x} = \boldsymbol{x}[1..n]$ and let $d = \gcd(p,q)$. If $p+q \leq n+d$, then d is also a period of $\boldsymbol{x}$.* $\Box$

Unfortunately this theorem is silent on the case of most interest to us: two squares occurring at the same position. A more relevant result is the "three squares lemma" [5,19] which we now state using the convention that the length of string $\boldsymbol{x}$ is denoted by x:

Theorem 3 *Suppose $\boldsymbol{u}^2$ is irreducible, and suppose $\boldsymbol{w} \neq \boldsymbol{u}^k$ for any $k \geq 1$. If $\boldsymbol{u}^2$ is a prefix of $\boldsymbol{w}^2$, in turn a proper prefix of $\boldsymbol{v}^2$, then $w \leq v-u$.* $\Box$

This result tells us that, given two squares $\boldsymbol{u}^2$ and $\boldsymbol{v}^2$ of periods $u < v$ occurring at some position i in $\boldsymbol{x}$, any third square $\boldsymbol{w}^2$ whose period $w \in u+1..v-1$ must satisfy $w \leq v-u$. Thus the existence of two squares at i imposes restrictions on the period of any third square at i. Since as already observed a run begins with a square, this imposes corresponding restrictions on runs. A recent result, that we call the "new periodicity

lemma" [6,23] substantially generalizes Theorem 3. Let us say that a square $\boldsymbol{u}^2$ is ***regular*** if no prefix of $\boldsymbol{u}$ is a square.

Theorem 4 *Suppose that $\boldsymbol{x}$ has regular prefix $\boldsymbol{u}^2$ and irreducible prefix $\boldsymbol{v}^2$, $u < v$.*

(a) $v > \max\{u+1, 3u/2\}$;
(b) *if $v < 2u$, then for every $w \in u{+}1..v{-}1$ and for every $k \in 0..v{-}u{-}1$, $\boldsymbol{x}[k{+}1..k{+}2w]$ is not a square;*
(c) *if $v < 2u$, then for every $w \in v-u+1..u-1$ and for every $k \in 2u{-}v..v{-}u{-}1$, $\boldsymbol{x}[k{+}1..k{+}2w]$ is not a square.* □

This result tells us that if there exists "small" $v < 2u$, then strict restrictions apply to the period w of any other square that might occur close to the start (at position $k+1$, in fact) of $\boldsymbol{x}$. Theorem 3 actually states the special case $k = 0$ of Theorem 4(b). Figure 2 shows the excluded values of w corresponding to the overlapping zones specified by k in Theorem 4(b)-(c).

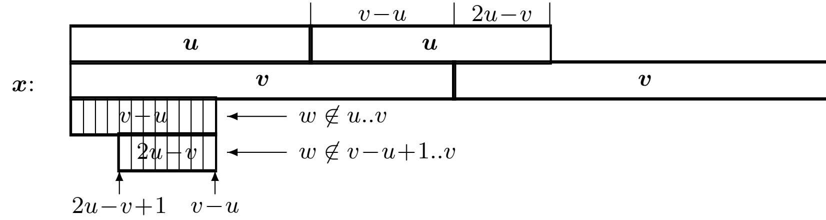

Fig. 2. Exclusion zones for $\boldsymbol{u}^2$ and $\boldsymbol{v}^2$

It should be remarked that Theorem 4 holds only trivially for $u < 3$, since for $u \leq 2$, we must have $v > 2u$:

- for $u = 1$, $\boldsymbol{u}^2 = \lambda^2$ for some letter λ, and the minimum $v = 3$ corresponds to $\boldsymbol{v}^2 = (\lambda^2\mu)^2$, $\mu \neq \lambda$;
- for $u = 2$, $\boldsymbol{u}^2 = (\lambda\mu)^2$, and the minimum $v = 5$ corresponds to $\boldsymbol{v}^2 = \big((\lambda\mu)^2\nu\big)^2$.

Thus the occurrence of small regular squares $\boldsymbol{u}^2$ can be analyzed as special cases giving rise to "large" v, while for $u \geq 3$, squares $\boldsymbol{v}^2$ satisfying $v < 2u$ ("small" v) can always be formed and are covered by Theorem 4.

4 Future Work

In this paper I have described three important kinds of periodicity in strings — repetitions, runs, repeats — and the algorithms currently available for computing them. Generally speaking, these algorithms depend on the time- and space-consuming construction of suffix arrays/trees.

I have put forward conjectures and theoretical results that may facilitate — at least for the calculation of runs — a more direct approach.

This is very much work-in-progress. For example, it is not yet clear whether Theorem 4 is optimal: it is possible that the values of k and w for which $\boldsymbol{x}[k{+}1..k{+}w]^2$ is impossible can be extended. It is also not clear how Theorem 4 can be used to prove conjecture (a), still less how it might be used algorithmically, so as to avoid testing for the existence of certain squares in a certain neighbourhood of a position where two squares are already known to exist.

Proof of conjecture (c) is also an important and largely untouched objective: if (c) holds, then $\rho(n)$ must depend also on alphabet size α, and so the properties of a function $\rho(n, \alpha)$ become an interesting object of research, both from theoretical and algorithmic points of view.

References

1. Mohamed Ibrahim Abouelhoda, Stefan Kurtz & Enno Ohlebusch, **The enhanced suffix array and its applications to genome analysis**, *Proc. Second Workshop on Algorithms in Bioinformatics*, LNCS 2452 (2002) 449–463.
2. Alberto Apostolico & Franco P. Preparata, **Optimal off-line detection of repetitions in a string**, *Theoret. Comput. Sci. 22* (1983) 297–315.
3. Gerth S. Brodal, Rune B. Lyngso, Christian N. S. Pederesen & Jens Stoye, **Finding maximal pairs with bounded gap**, *J. Discrete Algorithms 1* (2000) 77–103.
4. Maxime Crochemore, **An optimal algorithm for computing the repetitions in a word**, *Inform. Process. Lett. 12–5* (1981) 244–250.
5. Maxime Crochemore & Wojciech Rytter, **Squares, cubes and time-space efficient strings searching**, *Algorithmica 13* (1995 405–425.
6. Kangmin Fan, R. J. Simpson & W. F. Smyth, **A new periodicity lemma**, *Proc.* 16th *Annual Symp. Combin. Pattern Matching*, Alberto Apostolico, Maxime Crochemore & Kunsoo Park (eds.), LNCS 3537, Springer-Verlag (2005) 257–265.
7. N. J. Fine & Herbert S. Wilf, **Uniqueness theorems for periodic functions**, *Proc. Amer. Math. Soc. 16* (1965) 109–114.
8. Aviezri Fraenkel & R. J. Simpson, **How many squares can a string contain?**, *J. Combin. Theory Ser. A 82* (1998) 112–120.
9. Frantisek Franek, R. J. Simpson & W. F. Smyth, **The maximum number of runs in a string**, *Proc.* 14th *Australasian Workshop on Combin. Algorithms*, Mirka Miller & Kunsoo Park (eds.) (2003) 26–35.
10. Frantisek Franek, W. F. Smyth & Yudong Tang, **Computing all repeats using suffix arrays**, *JALC 8–4* (2003) 579–591.

11. Dan Gusfield, *Algorithms on Strings, Trees & Sequences*, Cambridge University Press (1997) 534 pp.
12. Lucian Ilie, **A simple proof that a word of length n has at most $2n$ distinct squares**, *J. Combin. Theory Ser. A* (2005) to appear.
13. Lucian Ilie, **A note on the number of distinct squares in a word** (2005) manuscript.
14. Juha Kärkkäinen & Peter Sanders, **Simple linear work suffix array construction**, *Proc.* 30$^{\text{th}}$ *Internat. Colloq. Automata, Languages & Programming* (2003) 943–955.
15. Dong Kyue Kim, Jeong Seop Sim, Heejin Park & Kunsoo Park, **Linear-time construction of suffix arrays**, *Proc.* 14$^{\text{th}}$ *Annual Symp. Combin. Pattern Matching*, Ricardo Baeza-Yates, Edgar Chávez & Maxime Crochemore (eds.), LNCS 2676, Springer-Verlag (2003) 186–199.
16. Pang Ko & Srinivas Aluru, **Space efficient linear time construction of suffix arrays**, *Proc.* 14$^{\text{th}}$ *Annual Symp. Combin. Pattern Matching*, Ricardo Baeza-Yates, Edgar Chávez & Maxime Crochemore (eds.), LNCS 2676, Springer-Verlag (2003) 200–210.
17. Roman Kolpakov & Gregory Kucherov, **On maximal repetitions in words**, *J. Discrete Algorithms 1* (2000) 159–186.
18. Abraham Lempel & Jacob Ziv, **On the complexity of finite sequences**, *IEEE Trans. Information Theory 22* (1976) 75–81.
19. M. Lothaire, *Algebraic Combinatorics on Words*, Cambridge University Press (2002) 504 pp.
20. Michael G. Main, **Detecting leftmost maximal periodicities**, *Discrete Applied Maths. 25* (1989) 145–153.
21. Michael G. Main & Richard J. Lorentz, **An $O(n \log n)$ algorithm for finding all repetitions in a string**, *J. Algorithms 5* (1984) 422–432.
22. Edward M. McCreight, **A space-economical suffix tree construction algorithm**, *J. Assoc. Comput. Mach. 23–2* (1976) 262–272.
23. Simon J. Puglisi, W. F. Smyth & Andrew Turpin, **A taxonomy of suffix array construction algorithms**, *Proc. Prague Stringology Conf. '05* (2005) to appear.
24. Bill Smyth, *Computing Patterns in Strings*, Pearson Addison-Wesley (2003) 423 pp.
25. Axel Thue, **Über unendliche zeichenreihen**, *Norske Vid. Selsk. Skr. I. Mat. Nat. Kl. Christiana 7* (1906) 1-22.
26. Peter Weiner, **Linear pattern matching algorithms**, *Proc. 14th Annual IEEE Symp. Switching & Automata Theory* (1973) 1–11.