# REPETITIVE PERHAPS, BUT CERTAINLY NOT BORING

**W. F. Smyth**

*Department of Computer Science & Systems*
*McMaster University*
**e-mail:** `smyth@mcmaster.ca`

*School of Computing*
*Curtin University of Technology*
**e-mail:** `smyth@cs.curtin.edu.au`

## ABSTRACT

In this paper some of the work done on repetitions in strings is surveyed, especially that of an algorithmic nature. Several open problems are described and conjectures formulated about some of them.

**KEYWORDS:** string, word, repetition, repeat, cover

## 1 INTRODUCTION

Repetitions in strings are usually thought of as adjacent or "tandem"; that is, the string $uvu$ is counted as a repetition of $u$ if and only if $v = \epsilon$, the empty string. However, in certain contexts — for example, DNA sequence analysis [S98], data compression [IS98], analysis of musical texts [CIR96] — this definition may be too narrow. Here therefore we take a wider view and regard $uvu$ as a repetition of a nonempty string $u$ for *any* finite string $v$. Even more generally, we also count as repetitions cases where the string $u$ overlaps itself; for example, $abaabaab$ is accepted as a repetition of $abaab$.

In order to make sure that these ideas are clear, we express them more formally. Throughout this paper $x$ will denote a string of length $n = |x| > 0$ defined on an alphabet $A$ of size $\alpha = |A|$. $A' \subseteq A$ will denote the subset of letters that actually occur in $x$; we let $\alpha' = |A'|$ and observe that $\alpha' \leq \alpha$ and also $\alpha' \leq n$. It will usually be convenient to represent $x$ as an array $x[1..n]$. Then the nonempty string $u$ is called a *substring* of $x$ if and only if $u = x[i..j]$ for integers $i$ and $j$ satisfying $1 \leq i \leq j \leq n$. (Equivalently, $x$ is called a *superstring* of $u$.) If $i = 1$, $u$ is said to be a *prefix* of $x$, a *proper prefix* if in addition $j < n$; similarly, if $j = n$, $u$ is said to be a *suffix* of $x$, a *proper suffix* if in addition $i > 1$. The empty string $\epsilon$ is counted both as a proper prefix and a proper suffix of every $x$. For nonempty substrings $u = x[i..j]$, we say that $u$ *occurs* at position $i$ of $x$ and that $i$ is an *occurrence* of $u$ in $x$.

Now let $S = S_{u,x}$ denote a tuple

$$(p; i_1, i_2, \ldots, i_r),$$

where $p = |u|$ and $i_1, i_2, \ldots, i_r$ is a monotone increasing sequence of positive integers in which every $i_j$, $1 \leq j \leq r$, is an occurrence of $u$ in $x$. If $r > 1$, $S$ is called a *repetition* of $u$ in $x$, and if moreover $S$ includes *every* occurrence of $u$ in $x$, it is called a *complete repetition* of $u$ in $x$; otherwise, $S$ is said to be *incomplete*. The substring $u = x[i_1..i_1 + p - 1]$ is then called the *generator* of the repetition, the integers $p$ and $r$ its *period* and its *exponent*, respectively. For example, if

$$\begin{array}{ccccccccccccccc} & & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \\ x & = & a & b & a & a & b & a & b & a & a & b & a & a & b, \end{array}$$

then $S_{aab,x} = (3; 3, 11)$ is a repetition; $S_{aab,x} = (3; 3, 8, 11)$, $S_{aba,x} = (3; 1, 4, 6, 9)$ and $S_{abaab,x} = (5; 1, 6, 9)$ are complete repetitions; while $S_{bb,x} = (2)$ and $S_{aab,x} = (3; 8)$ are not repetitions at all.

We will now see how to classify repetitions in a useful way. Let $S = S_{u,x} = (p; i_1, i_2, \ldots, i_r)$ denote a repetition and consider the *gaps* $g_j = i_{j+1} - i_j$, $1 \leq j \leq r - 1$. We say that $S$ is a *tandem* (respectively, *split*, *overlapping*) repetition if and only if every gap $g_j$ is equal to (respectively, greater than, less than) $p$. Further, if every $g_j \leq p$, $S$ is said to be a *cover* of the substring $x\big[i_1..i_r + p - 1\big]$, and this substring is accordingly said to be *coverable*. Thus in the preceding example, $(5;1,6)$ is tandem, $(5;1,9)$ is split and $(5;6,9)$ is overlapping, while $(3;1,4,6,9)$ and $(5;1,6,9)$ are covers of $x[1..11]$ and $x = x[1..13]$, respectively. Observe then that a complete repetition such as $(5;1,6,9)$ may be decomposed in various ways; for instance, into one tandem repetition $(5;1,6)$ and one overlapping repetition $(5;6,9)$, or alternatively into one split repetition $(5;1,9)$ together with $(5;6,9)$. Similarly, even though the complete repetition $S_{a,x} = (1;1,3,4,6,8,9,11,12)$ is not classified (since it combines both tandem and split pairs), it can nevertheless be broken down into incomplete repetitions, some of them tandem, such as $(1;3,4)$ and $(1;11,12)$; others split — for example, $(1;1,4,6)$ and $(1;8,12)$. The notation $u^r$ is often used to denote a tandem repetition of $u$ of exponent $r$; for $r=2$, $u^r$ is referred to as a *square*; for $r=3$, a *cube*.

The taxonomy given here may seem a little awkward, but as we shall see it does have the merit of permitting a unified view of repetitions and problems related to them. It should be noted that the repetitions here defined as "split" have generally been referred to in the literature as "nontandem"; the nomenclature has been changed to reflect the fact that there are really three kinds of repetition (tandem, split, overlapping) rather than two (tandem, nontandem).

Historically, interest in repetitions focussed almost entirely on tandem repetitions. The mathematician Axel Thue set the tone early in the $20^{\text{th}}$ century by showing [T06] that an infinitely long string free of tandem repetitions could be constructed on an alphabet of only three letters. This problem has since been generalized to what may be called the $(\alpha, r)$*-avoidance problem*:

> *Construct an infinite string on an alphabet of size $\alpha$ that contains no tandem repetition of exponent $r$ (but that does contain tandem repetitions of exponents $2,3,\ldots,r-1$).*

Thue also described a construction to solve the $(2,3)$-avoidance problem — infinite strings on a binary alphabet with squares but no cubes — and in [K83] it was shown that the infinite Fibonacci string (defined in Section 3) solves the $(2,4)$-avoidance problem — cubes but no fourth powers. Recently this latter result has been generalized [FKS98] to show that there exists an infinite Sturmian string (also defined in Section 3) to solve the $(2,r)$-avoidance problem for every $r \geq 4$. Thus solutions exist to the $(\alpha,r)$-avoidance problem for all $\alpha \geq 2$ and all $r \geq 3$ as well as for all $\alpha \geq 3$ and $r=2$. Avoidance problems have been generalized in various ways [BEM79,JC93,KK97].

The rise of computer science in the 1950s led naturally to an interest in strings and their applications, but the emphasis shifted gradually toward algorithms operating on finite strings. In the 1960s a fundamental theoretical result, the "periodicity lemma", was published [FW65]. Then in the early 1980s three papers were published [C81,AP83,ML84] describing three quite different algorithms for computing all the tandem repetitions in a given string $x$ in time $\Theta(n \log n)$. In [ML84] it was shown in addition that this was best possible, in the sense that any deterministic method based on letter comparisons that recognized whether or not $x$ was free of tandem repetitions was shown to require $\Omega(n \log n)$ time. Recently a new $\Theta(n \log n)$ algorithm for all tandem repetions has appeared [SG98] that makes simple and effective use of suffix trees.

It is noteworthy here that the number of tandem repetitions in $x$ cannot exceed the time required to compute them. Thus the number of tandem repetitions is $O(n \log n)$. But this claim is apparently contradicted by examples such as $x = a^6$ containing $a^2$ (five times), $(a^2)^2$ (three times), and $(a^3)^2$ (once). In general, it is easily seen that $x = a^n$ contains $\lfloor n^2/4 \rfloor$ such tandem squares. The difficulty is resolved by agreeing that tandem repetitions of substrings that are themselves tandem repetitions do not need to be reported, and also by reporting tandem repetitions that are maximal. In [C81] the explicit notation $(i,p,r)$ was introduced for reporting tandem repetitions, where the generator $u = x[i..i+p-1]$ is not a repetition and $r$ is a maximum. Thus the repetitions in $x = a^n$ are compactly reported by the single triple $(1,1,n)$; it turns out however [C81] that the tandem repetitions in Fibonacci strings require $\Theta(n \log n)$ space for output using the $(i,p,r)$ encoding.

For $r=2$ the idea of a tandem square was generalized in [E61] to an *Abelian square*: two adjacent substrings (for example, *abba*) that are permutations of each other. The corresponding $(\alpha,2)$-avoidance problem on Abelian squares was shown in [P70] to be solvable for $\alpha = 5$, then in [K92] for $\alpha = 4$ — that is, infinitely long strings without Abelian squares can be constructed on an alphabet of only four letters. Related problems were also considered in [D79]. In [CS97] it was shown that a lower bound on recognizing whether or not a given string $x$ is free of Abelian squares is also $\Omega(n \log n)$, but the only published algorithm [CS97] for computing all the Abelian squares in $x$ requires $\Theta(n^2)$ time. Thus the open problem:

> *What is the exact time complexity for the computation of all the Abelian squares in a given string $x$?*

In the 1990s problems on tandem repetitions were generalized in another way — to covers. The problem of computing all the tandem repetitions in $x$ became the problem of computing all the coverable substrings of $x$; that is, maximal repetitions in $x$ with gaps at most $p$. In [AE93] an $O(n \log^2 n)$ algorithm for this problem was presented, a result recently improved to $\Theta(n \log n)$ in [IM98]. This latter bound is clearly best possible, since it matches the optimal time bound for the included problem of computing all the tandem repetitions in $x$.

But for covers a new problem arose that for tandem repetitions was trivial: determine whether or not the string $x$ itself has a cover, as it does with $S_{abaab,x} = (5; 1, 6, 9)$ in the above example. (To determine whether or not $x$ is a tandem repetition is a straightforward application of the calculation of the maximum border, or failure function, of $x$. Related ideas are discussed in more detail in both Sections 2 and 3.) In [AFI91] the concept of a cover was introduced for the first time, as well as that of *shortest cover* — the cover of least period $p$. The authors went on to describe a linear time algorithm to compute the shortest cover of $x$. Then a sequence of algorithms followed, all of them linear-time, that improved on this result: [B92] gave an on-line algorithm for the shortest cover (thus computing the shortest cover of every prefix of $x$); [MS94,MS95] described an algorithm to compute all the covers of $x$; and finally [LS98] presented an on-line algorithm for all the covers of every prefix of $x$. Interesting parallel algorithms were also developed for this problem: in [IP94] an $O(n \log \log n)$ time PRAM algorithm for the shortest cover problem was given, shown in [B92] to be best possible. In [IS98] the idea of a $k$-cover was introduced — that is, a minimum cardinality set of strings of length $k$ that together cover $x$ — and an $O\big(n^2(n-k)\big)$-time algorithm described to compute it. Underlying many of the above algorithms is the idea of "gap", introduced above in a simple form and discussed extensively in [BBIP95].

The idea of cover has recently been further generalized in [IMP96]: a *seed* of a given string $x$ is defined to be a substring of $x$ that is a cover of some superstring of $x$. Then, corresponding to the all-covers problem discussed above, there arises the *all-seeds problem*: find all the seeds of a given string $x$. An $O(n \log n)$ time sequential algorithm was given for this problem in [IMP96] and a PRAM algorithm requiring $O(\log n)$ time and $O(n \log n)$ work in [BBIP95]. Considerable effort has been expended to try to improve on these algorithms, so far without success. Thus another open problem:

> *What is the exact time complexity for the computation of all the seeds of a given string $x$?*

Apart from covers and seeds, it has only been in the last two or three years that a more comprehensive approach to repetitions has crept, rather hesitantly, into the literature [G97,S98]. In Section 2 we try to encourage this incipient trend by discussing the computation of all the repetitions in a given string $x$. Then in Section 3 we return to a discussion of tandem repetitions, especially the complexity of reporting the tandem repetitions in $x$. Section 4 briefly discusses a very new area of research: approximate periodicity. Finally, Section 5 draws attention to the importance of output encoding in string problems.

## 2 COMPUTING ALL THE REPETITIONS

Armed with a little terminology, we will not find it difficult to see [D98] that all the repetitions in $x$ can be reported in $O(n^2)$ time. A string $u$ that is both a proper prefix and a suffix of $x$ is called a *border* of $x$. For example, $x = abaabaab$ has borders $u = abaab, ab, \epsilon$. A useful data structure for many string algorithms is the *border array* $\beta = \beta[1..n]$, defined by the property that, for every integer $i \in 1..n$, $\beta[i]$ is the length of the longest border of $x[1..i]$. The border array of the string used as an example in Section 1 is as follows:

$$\begin{array}{ccccccccccccccc} & & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \\ x & = & a & b & a & a & b & a & b & a & a & b & a & a & b \end{array}$$

$$\beta = 0\ 0\ 1\ 1\ 2\ 3\ 2\ 3\ 4\ 5\ 6\ 4\ 5$$

It is well known that $\beta$ specifies *all* the borders of every prefix of $x$; for example, $x[1..11]$ has nonempty borders of lengths $\beta[11] = 6$, $\beta[\beta[11]] = 3$ and $\beta\Big[\beta[\beta[11]]\Big] = 1$. It is also well known [AHU74] that $\beta$ can be computed in $\Theta(n)$ time.

Suppose now that for given $x$ the border array is computed for every nonempty suffix $x[i..n]$ of $x$, $i = 1, 2, \ldots, n$, and imagine that these border arrays are arranged in an upper triangular matrix $B$. In the context of our example, this would yield

$$B = \begin{pmatrix}
0 & 0 & 1 & 1 & 2 & 3 & 2 & 3 & 4 & 5 & 6 & 4 & 5 \\
 & 0 & 0 & 0 & 1 & 2 & 1 & 2 & 3 & 4 & 5 & 3 & 4 \\
 & & 0 & 1 & 0 & 1 & 0 & 1 & 2 & 3 & 4 & 2 & 3 \\
 & & & 0 & 0 & 1 & 2 & 3 & 1 & 2 & 3 & 1 & 2 \\
 & & & & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 \\
 & & & & & 0 & 0 & 1 & 1 & 2 & 3 & 4 & 5 \\
 & & & & & & 0 & 0 & 0 & 1 & 2 & 3 & 4 \\
 & & & & & & & 0 & 1 & 0 & 1 & 2 & 3 \\
 & & & & & & & & 0 & 0 & 1 & 1 & 2 \\
 & & & & & & & & & 0 & 0 & 0 & 1 \\
 & & & & & & & & & & 0 & 0 & 1 \\
 & & & & & & & & & & & 0 & 0 \\
 & & & & & & & & & & & & 0
\end{pmatrix}.$$

Here $B[i,j] = k > 0$ if and only if $x$ contains the repetition

$$x[i..i+k-1] = x[j-k+1..j]$$

of exponent $r = 2$. (As we have seen, tandem repetitions of exponent 2 are called *squares*; here we extend this usage also to split and overlapping repetitions of exponent 2.) It is easy to see that in fact $B$ specifies *all* the squares in $x$, from which we conclude that the number of squares, hence the number of repetitions, in $x$ is $O(n^2)$. Since $B$ is the result of $n$ border array calculations, it follows that the repetitions in $x$ can, as stated above, be computed in $O(n^2)$ time. The question is: can they be computed more quickly?

It is instructive to consider next the corresponding recognition problem: that of recognizing whether or not $x$ is repetition-free — in other words, square-free. Clearly $x$ will be square-free if and only if it contains no letter twice — that is, when $\alpha' = n$. To determine efficiently whether or not $\alpha' = n$, we try to insert each $x[i]$, $i = 1, 2, \ldots, n$, into a search tree (for example, AVL tree or 2-3 tree) of logarithmic height. If we make the usual assumption that a comparison of two letters of the alphabet requires constant time, the total time requirement for these insertions is $O(n \log n)$, and $\alpha' = n$ if and only if each insertion succeeds (no $x[i]$ is duplicated). On the other hand, if in fact there are $n$ distinct letters in the alphabet, each comparison of two letters may require comparing $\log n$ bits and so may consume $O(\log n)$ time; in this case the time required becomes $O(n \log^2 n)$. Thus, for an alphabet on which a total order relation holds (the only case that arises in digital computers), a repetition-free string can be recognized in $O(n \log^2 n)$ time. In special cases where the letters of the alphabet can be indexed or hashed into an array of length $O(n)$ ($\log n$ bits for each array element), such a string could be recognized in $O(n \log n)$ time. In any of these cases, we find a gap between the recognition problem and the computation problem similar to that noted in Section 1 for Abelian squares.

It therefore becomes interesting to look more carefully at the number of repetitions in $x$. In the above example, observe that since $B[1,11] = 6$, it follows that there exists the (overlapping) square

$$x[1..6] = x[6..11] = abaaba.$$

Then, with the reporting of this square, it becomes unnecessary and redundant to report the squares

$$x[i..j] = x[i+5..j+5]$$

for every $i, j \in 1..6$ satisfying $0 \leq j - i \leq 4$. This observation gives rise to the following definitions. (To accomodate beginning and end conditions in these definitions, we suppose that $x$ is preceded and followed by two distinct and unique letters $x[0] = \$_1$ and $x[n+1] = \$_2$, respectively.)

Let $S_{u,x} = (p; i_1, i_2)$ denote a square in $x$. $S_{u,x}$ is said to be *left-extendible* if and only if $x[i_1 - 1] = x[i_2 - 1]$, *right-extendible* if and only if $x[i_1 + p] = x[i_2 + p]$, and *extendible* if and only if it is either left-extendible or right-extendible or both. If $S_{u,x}$ is neither left- nor right-extendible, it is said to be *nonextendible.* Thus in our example, $(1; 5, 7)$ is both left- and right-extendible, $(2; 5, 7)$ is left-extendible but not right-extendible, $(2; 4, 6)$ is right-extendible but not left-extendible, and $(3; 4, 6)$ is nonextendible.

More generally, a repetition $(p; i_1, i_2, \ldots, i_r)$ is said to be *left-extendible* (respectively, *right-extendible*) if and only if every square $(p; i_j, i_{j+1})$, $1 \leq j < r$, is left-extendible (respectively, right-extendible). Then as above a repetition is *extendible* if and only if it is either left-extendible or right-extendible or both. In our example, the complete repetition $(1; 2, 5, 7, 10, 13)$ is left-extendible, the repetition $(2; 1, 4, 6, 9)$ is right-extendible, and the complete repetition $(3; 2, 7, 10)$ is both left-extendible and right-extendible. The repetition $(3; 1, 4, 6, 9)$ is nonextendible.

Now consider a nonextendible square $(p; i_1, i_2)$. This square implies the existence of the following $\binom{p+1}{2} - 1$ extendible squares, that therefore (if the user agrees) do not need to be reported:

$$\begin{array}{l} (p-1; i_1, i_2),\ (p-1; i_1+1, i_2+1); \\ (p-2; i_1, i_2),\ (p-2; i_1+1, i_2+1),\ (p-2; i_1+2, i_2+2); \\ \qquad\qquad\qquad\qquad\vdots \\ (1; i_1, i_2),\ (1; i_1+1, i_2+1),\ \ldots,\ (1; i_1+p-1, i_2+p-1). \end{array}$$

More generally, a nonextendible repetition $(p; i_1, i_2, \ldots, i_r)$ implies the existence of $\binom{p+1}{2} - 1$ extendible repetitions

$$(p-j; i_1, i_2, \ldots, i_r),\ (p-j; i_1+1, i_2+1, \ldots, i_r+1),\ \ldots,\ (p-j; i_1+j, i_2+j, \ldots, i_r+j),$$

where $j = 1, 2, \ldots, p-1$. Observe [D98] that these collections of repetitions correspond to equilateral triangles of side $p$ in the border matrix $B$, each of which contains $\binom{p+1}{2}$ elements; however, a single output corresponding to each triangle suffices to describe all the repetitions. For example, the nonextendible repetition $(3; 1, 4, 6, 9)$ describes all the repetitions specified by the $r = 4$ triangles

$$\begin{array}{ccc} 0 & 0 & 1 \\ & 0 & 0 \\ & & 0 \end{array} \qquad \begin{array}{ccc} 1 & 2 & 3 \\ & 1 & 2 \\ & & 1 \end{array} \qquad \begin{array}{ccc} 3 & 2 & 3 \\ & 1 & 2 \\ & & 1 \end{array} \qquad \begin{array}{ccc} 4 & 5 & 6 \\ & 4 & 5 \\ & & 4 \end{array}$$

Thus it suffices that an algorithm report only nonextendible repetitions, in fact nonextendible complete repetitions, each of which implies up to $r\left(\binom{p+1}{2} - 1\right)$ extendible repetitions.

The nonextendible property turns out to be an important one, since in particular nonextendibility to the right implies that at least two occurrences of the generator $x[i_1..i_1+p-1]$ of the repetition must be followed by distinct letters. As observed in [G97], this implies that corresponding to each complete nonextendible repetition there exists at least one uniquely defined internal node in the compacted suffix tree $T_x$ of $x$. Since $T_x$ contains at most $n-1$ internal nodes, it follows that there can be at most $n-1$ complete nonextendible repetitions in $x$. This observation leads immediately [G97] to two algorithms, both based on suffix trees, that compute

(1) all the distinct generators of nonextendible repetitions in $x$;
(2) all the nonextendible complete repetitions $(p; i_1, i_2, \ldots, i_r)$ in $x$.

The time bound claimed for these algorithms is

$$\Theta(n + \text{output\_size}),$$

but this quantity assumes a fixed alphabet and so does not take account of the time required for suffix tree construction and use. For many applications the use of suffix trees is inappropriate, but nevertheless these results encourage speculation that there may exist an efficient algorithm to compute all the repetitions that does not depend on the use of suffix trees. Further encouragement is provided by the fact that of the three kinds of repetition identified in Section 1 (tandem, split, overlapping), two can be computed in $O(n \log n)$ time: tandem repetitions [C81,ML84] and overlapping repetitions [IM98]. We state then the first main conjecture of this section:

*The nonextendible complete repetitions in a given string $x$ are computable without the use of suffix trees in $O(n \log n)$ time?*

If this conjecture is true, it of course implies that the nonextendible complete repetitions in $x$ can be output in only $O(n \log n)$ space. But as we shall see in the next section, there is some evidence that a stronger result may hold. It has recently been shown that the tandem repetitions in $x$ can be reported in linear space, a result that suggests that the split repetitions in $x$ can also be reported in linear space. Our second main conjecture is therefore the following:

*The nonextendible complete repetitions in $x$ can be described in $O(n)$ space?*

Proving or disproving one or the other of these conjectures would contribute greatly to our understanding of the repetitive structure of strings. Observe in particular that an $O(n \log n)$ algorithm to compute all the nonextendible complete repetitions would yield optimal algorithms for all tandem repetitions and all coverable substrings as a byproduct. Such an algorithm might well also have useful application to data compression.

## 3 TANDEM REPETITIONS IN RUNS

In this section we return to the study of tandem repetitions. We apply the idea of nonextendibility to tandem repetitions in order to define nonextendible "runs" of repetitions that it turns out can be described in $O(n)$ space. This fact gives rise in turn to a number of open problems.

Suppose that a given string $x$ contains the $k \leq p$ tandem repetitions

$$\left.\begin{aligned} S_{u_0,x} &= (p; i_1, i_2, \ldots, i_r), \\ S_{u_1,x} &= (p; i_1+1, i_2+1, \ldots, i_r+1), \\ &\vdots \\ S_{u_{k-1},x} &= (p; i_1+k-1, i_2+k-1, \ldots, i_r+k-1), \end{aligned}\right\} \qquad \ldots (3.1)$$

all of period $p$, where the gap $i_j - i_{j-1} = p$ for every $j \in 2..r$ and the tuples

$$(p; i_1-1, i_2-1, \ldots, i_r-1) \text{ and } (p; i_1+k, i_2+k, \ldots, i_r+k) \qquad \ldots (3.2)$$

do *not* represent repetitions in $x$. Observe that condition (3.2) is equivalent to requiring that $S_{u_0,x}$ be not left-extendible and $S_{u_{k-1},x}$ not right-extendible; in other words, that the range $k$ of the equations (3.1) is maximum. Observe also that each substring $u_j$, $1 \leq j \leq k-1$, is necessarily a cyclic shift of $u_{j-1}$. Following [IMS97], we say that the collection (3.1) of tandem repetitions is a *run* in $x$ and we denote it by

$$R_{u_0,x} = (i_1, p, r, k).$$

As an example, consider the *Fibonacci string* $F_n$ defined recursively by

$$F_0 = b,\ F_1 = a;\ F_i = F_{i-1}F_{i-2},\ i = 2, 3, \ldots, n.$$

We discover that the example introduced in Section 1 was actually $F_6$. For $n = 7$ we have

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21

$$F_7 = a\ b\ a\ a\ b\ a\ b\ a\ a\ b\ a\ a\ b\ a\ b\ a\ a\ b\ a\ b\ a.$$

This string contains runs $(3,1,2,1)$, $(4,2,2,2)$, $(6,3,3,1)$, $(1,5,2,2)$, and so on. Note that $(6,3,2,3)$ is *not* a run because it is not maximal: $(3;9,12)$ is a repetition.

We have seen from this example that Fibonacci strings may contain tandem cubes. But we saw in Section 1 that no $F_n$ contains a tandem repetition of exponent 4. Thus runs in Fibonacci strings are constrained to be "short", of exponent $r \leq 3$. Nevertheless, despite this restriction, the repetitions in Fibonacci strings have recently [IMS97] been characterized in terms of runs; further, it was shown that the total number of runs in $F_n$ is $\Theta\big(|F_n|\big)$ — linear in the string length — and that they can all be computed in $\Theta\big(|F_n|\big)$ time. This somewhat surprising result holds even though $F_n$ actually contains $\Theta\big(|F_n|\log|F_n|\big)$ tandem repetitions [C81], the maximum possible in the $(i,p,r)$ encoding. More exact results about tandem squares in Fibonacci strings are found in [FS99,KK98].

The *infinite Fibonacci string* is the string that contains every $F_n$ except $F_0$ as a prefix. This string is a special case of a *Sturmian string*; that is, an infinite string on $\{a,b\}$ that for every integer $k \geq 1$ contains exactly $k+1$ distinct substrings of length $k$. (Of course an arbitrary string on $\{a,b\}$ may contain as many as $2^k$ distinct strings of length $k$.) A Sturmian string is then said to be of *complexity* $k+1$. Sturmian strings have been much studied [S76] and may be defined or represented in several different ways [B93,BS93,LM94,R84,S99].

In the context of computing repetitions, we of course consider finite prefixes of Sturmian strings. As we have seen, the finite prefixes $F_n$ contain the maximum number of tandem repetitions but only a linear number of runs. In general, because Sturmian strings contain "few" distinct substrings, it does not seem unreasonable to expect that they contain "many" repetitions. As shown in [FKS98], this expectation is correct: Sturmian prefixes of length $n$ have $\Theta(n\log n)$ tandem repetitions. Again it turns out [FKS98] that these repetitions can be arranged into only $\Theta(n)$ runs; moreover, they can, like tandem repetitions in Fibonacci strings, also be computed in $\Theta(n)$ time.

Since such a property holds for strings with many repetitions, it is natural to ask if it holds also for all strings. A very recent result shows that in fact it does. Let $\rho(n)$ denote the maximum number of runs that can possibly occur in any string of length $n$. Then [KK98] there exist positive constants $c_1$ and $c_2$ such that

$$\rho(n) \leq c_1 n - c_2\sqrt{n}\log n.$$

Even though the existing proof of this result yields no upper bound on $c_1$ and $c_2$, nevertheless [KK98] includes a listing of $\rho(n)$ for every $n \in 5..31$, giving for each $n$ an example of a *maximal string* $x^* = x^*[1..n]$ that contains $\rho(n)$ runs. In every case $\rho(n) < n$ and $x^*$ is a string on $a, b$ that contains no fourth powers. [KK98] also describe a "linear" time algorithm based on suffix trees to compute all the runs in a given string $x$. These results raise the following open problems/conjectures:

> *Prove that* $\rho(n) < n$.
>
> *Show that for every* $n$ *there exists a maximal string* $x^*$ *that is defined on a binary alphabet and that is cube-free.*
>
> *For given* $n$*, find an efficient way to compute a maximal string* $x^*$*; or, failing that, find an efficient way to compute* $\rho(n)$*; or, failing that, establish sharp bounds on the constants* $c_1$ *and* $c_2$*.*
>
> *Identify classes of strings for which, like Sturmian strings, all the tandem repetitions can be computed in linear time (without resorting to suffix trees).*

## 4 APPROXIMATE PERIODICITY

In the previous sections we have generally considered problems in which a generator $u$ of a repetition is invariant; except for Abelian squares, we have always required that occurrences of $u$ match each other exactly. In some applications, specifically DNA sequence analysis and data compression, it becomes interesting to relax this condition and to recognize $u'$ as an occurrence of $u$ if the *distance* $d(u,u') \leq k$ for some nonnegative integer $k$ and an appropriate definition of distance. The usual definitions of distance used are as follows:

* *Hamming Distance*

  For $|u| = |u'|$, the minimum number of letter substitutions required to transform $u'$ into $u$.

* *Edit Distance*

  The minimum number of insertions and deletions of letters required to transform $u'$ into $u$.

* *Levenshtein Distance*

  The minimum number of substitutions, insertions and deletions required to transform $u'$ into $u$.

In these definitions, substitutions of one letter by another may be weighted according to a so-called "scoring matrix".

Although there is an enormous literature dealing with approximate pattern matching according to these and other definitions of distance, very little has so far been published on approximate repetitions. In [LS93] algorithms for finding approximate tandem repetitions under Hamming and Levenshtein distance were described, while in [S98] distance in a grid graph is used as the basis of algorithms to find approximate tandem and split repetitions. Very recently, the idea of an approximate generator was introduced [SIKS99]: $u$ is said to be a *$k$-approximate generator* of $x$ if

$$x = u_1 u_2 \cdots u_r$$

for some integer $r \geq 1$, where for every $j \in 1..r$, the distance $d(u, u_j) \leq k$. Using this definition, algorithms to solve the following two problems under various distance measures were proposed [SIKS99]:

* Given strings $x$ and $u$, find the least integer $k$ such that $u$ is a $k$-approximate generator of $x$.
* Given a string $x$, find a substring $u$ of $x$ that is a $k$-approximate generator of $x$ with minimum distance $k$.

There has apparently been no work done that deals specifically with approximate covers or approximate $k$-covers.

## 5 ENCODING THE OUTPUT

We conclude by drawing attention to a prominent feature of algorithms on strings: the effect of output encoding on algorithmic complexity. We have seen several examples of this phenomenon in this paper:

* The border array $\beta$ is the same length as the string $x$. But $\beta$ actually encodes information about borders of prefixes of $x$ that may number as many as $\Theta(n \log n)$.
* The number of tandem squares in $x$, including squares of repeating substrings, may be as many as $\Theta(n^2)$. But the $(i,p,r)$ encoding reduces the length of the output to $O(n \log n)$.
* Every Sturmian string, in particular the Fibonacci string, contains $\Theta(n \log n)$ tandem repetitions in the $(i,p,r)$ encoding. But these repetitions are arranged into $\Theta(n)$ runs that can be reported in $\Theta(n)$ time using the $(i,p,r,k)$ encoding. More generally, of course, the tandem repetitions in *every* string can be reported as $O(n)$ 4-tuples representing runs.
* A nonextendible repetition may encode a large number of implied extendible repetitions that we may therefore agree do not need to be reported.

## REFERENCES

[**AHU74**] Alfred V. Aho, John E. Hopcroft & Jeffrey D. Ullman, *The Design & Analysis of Computer Algorithms*, Addison-Wesley (1974).

[**AE93**] Alberto Apostolico & Andrzej Ehrenfeucht, **Efficient detection of quasiperiodicities in strings**, *TCS* 119 (1993) 247-265.

[**AFI91**] Alberto Apostolico, Martin Farach & Costas S. Iliopoulos, **Optimal superprimitivity testing for strings**, *IPL* 39 (1991) 17-20.

[**AP83**] Alberto Apostolico & Franco P. Preparata, **Optimal off-line detection of repetitions in a string**, *TCS* 22 (1983) 297-315.

[**BEM79**] Dwight R. Bean, Andrzej Ehrenfeucht & George F. McNulty, **Avoidable patterns in strings of symbols**, *Pacific J. Math.* 85-2 (1979) 261-294.

[**BBIP95**] A. M. Ben-Amram, Omer Berkman, Costas S. Iliopoulos & Kunsoo Park, **The subtree max gap problem with application to parallel string covering**, *Information & Computation* 123-1 (1995) 127-137.

[**BS93**] Jean Berstel & Patrice Séébold, **A characterization of Sturmian morphisms**, *The Mathematical Foundations of Computer Science*, A. Borzyszkowski & S. Sokolowski (eds.), Springer-Verlag (1993) 281-290.

[**B92**] Dany Breslauer, **An on-line string superprimitivity test**, *IPL* 44 (1992) 345-347.

[**B93**] Tom C. Brown, **Descriptions of the characteristic sequence of an irrational**, *Canad. Math. Bull.* 36-1 (1993) 15-21.

[**CIR97**] Tim Crawford, Costas S. Iliopoulos & Rajeev Raman, **String matching techniques for musical similarity & melodic recognition**, preprint.

[**C81**] Maxime Crochemore, **An optimal algorithm for computing all the repetitions in a word**, *IPL* 12-5 (1981) 244-248.

[**CS97**] L. J. Cummings & W. F. Smyth, **Weak repetitions in strings**, *J. Combinatorial Math. & Combinatorial Computing* 24 (1997) 33-48.

[**C93**] James Currie, **Open problems in pattern avoidance**, *Amer. Math. Monthly* 100-8 (1993) 790-793.

[**D98**] Bandula Dahanayake, *Generalized Repetitions: A New Approach*, M. Sc. Thesis, Department of Computer Science & Systems, McMaster University (1998).

[**D79**] F. M. Dekking, **Strongly nonrepetitive sequences and progression-free sets**, *JCT (A)* 27 (1979) 181-185.

[**E61**] Pál Erdős, **Some unsolved problems**, *Hungarian Academy of Sciences Mat. Kutató Intézet Közl.* 6 (1961) 221-254.

[**FW65**] N. J. Fine & Herb S. Wilf, **Uniqueness theorems for periodic functions**, *Proc. Amer. Math. Soc.* 16 (1965) 109-114.

[**FS99**] Aviezri S. Fraenkel & Jamie Simpson, **The exact number of squares in Fibonacci words**, *TCS* special "Words" issue (1999) to appear.

[**FKS98**] František Franěk, Ayşe Karaman & W. F. Smyth, **Repetitions in Sturmian strings**, *Proc. Ninth Australasian Workshop on Combinatorial Algorithms*, School of Computing, Curtin University (1998) 107-116.

[**G97**] Dan Gusfield, *Algorithms on Strings, Trees & Sequences*, Cambridge University Press (1997) 534 pp.

[**IMP96**] Costas S. Iliopoulos, Dennis Moore & Kunsoo Park, **Covering a string**, *Algorithmica* 16 (1996) 288-297.

[**IMS97**] Costas S. Iliopoulos, Dennis Moore & W. F. Smyth, **A characterization of the squares in a Fibonacci string**, *TCS* 172 (1997) 281-291.

[**IM98**] Costas S. Iliopoulos & Laurent Mouchard, **Fast local covers**, preprint.

[**IP94**] Costas S. Iliopoulos & Kunsoo Park, **An optimal $O(n \log\log n)$-time algorithm for parallel superprimitivity testing**, *J. Korea Information Science Society* 21-8 (1994) 1400-1404.

[**IS98**] Costas S. Iliopoulos & W. F. Smyth, **On-line algorithms for $k$-covering**, *Proc. Ninth Australasian Workshop on Combinatorial Algorithms*, School of Computing, Curtin University (1998) 97-106.

[**K83**] Juhani Karhumäki, **On cube-free $\omega$-words generated by binary morphisms**, *Disc. Appl. Math.* 5 (1983) 279-297.

[**K92**] Veikko Keränen, **Abelian squares are avoidable on 4 letters**, *Lecture Notes in Computer Science* 623, Springer-Verlag (1992) 41-52.

[**KK97**] Roman Kolpakov & Gregory Kucherov, **Minimal letter frequency in $n^{\mathbf{th}}$ power-free binary words**, *Lecture Notes in Computer Science* 1295, Springer-Verlag (1997) 347-357.

[**KK98**] Roman Kolpakov & Gregory Kucherov, *Maximal Repetitions in Words or How to Find all Squares in Linear Time*, Internal Report No. LORIA 98-R-227, Laboratoire Lorrain de Recherche en Informatique et ses Applications (1998) 22 pp.

[**LS93**] Gad M. Landau & Jeanette P. Schmidt, **An algorithm for approximate tandem repeats**, *Proc. Fourth Symp. Combinatorial Pattern Matching*, *Lecture Notes in Computer Science* 648, Springer-Verlag (1993) 120-133.

[**LS98**] Yin Li & W. F. Smyth, **An optimal on-line algorithm to compute all the covers of a string**, preprint.

[**LM94**] Aldo de Luca & Filippo Mignosi, **Some combinatorial properties of Sturmian words**, *TCS* 136 (1994) 361-385.

[**ML84**] 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.

[**M89**] Filippo Mignosi, **On the number of factors of Sturmian words**, *TCS* 82 (1989) 221-242.

[**MS94**] Dennis Moore & W. F. Smyth, **An optimal algorithm to compute all the covers of a string**, *IPL* 50-5 (1994) 239-246.

[**MS95**] Dennis Moore & W. F. Smyth, **A correction to "An optimal algorithm to compute all the covers of a string"**, *IPL* 54 (1995) 101-103.

[**P70**] Peter A. B. Pleasants, **Non-repetitive sequences**, *Proc. Camb. Phil. Soc.* 68 (1970) 267-274.

[**R84**] G. Rauzy, **Mots infinis en arithmétique**, *Automata on Infinite Words*, M. Nivat & D. Perrin, eds., *Lecture Notes in Computer Science* 192, Springer-Verlag (1984) 165-171.

[**S98**] Jeanette P. Schmidt, **All highest scoring paths in weighted grid graphs and their application to finding all approximate repeats in strings**, *SIAM J. Computing* 27-4 (1998) 972-992.

[**S99**] Jeffrey Shallit, **Characteristic words as fixed points of homomorphisms**, preprint.

[**SIKS99**] Jeong Seop Sim, Costas S. Iliopoulos, Kunsoo Park & W. F. Smyth, **Approximate periods of strings**, preprint.

[**S76**] Kenneth B. Stolarsky, **Beatty sequences, continued fractions, and certain shift operators**, *Canad. Math. Bull.* 19-4 (1976) 473-482.

[**SG98**] Jens Stoye & Dan Gusfield, **Simple and flexible detection of contiguous repeats using a suffix tree**, preprint.

[**T06**] Axel Thue, **Über unendliche zeichenreihen**, *Norske Vid. Selsk. Skr. I, Mat. Nat. Kl. Christiana* 7 (1906) 1-22.

## ACKNOWLEDGEMENTS

This work was supported in part by Grant No. A8180 of the Natural Sciences & Engineering Research Council of Canada and by Grant No. GO-12278 of the Canadian Genome Analysis & Technology agency. The author is indebted to two anonymous referees for helpful comments and suggestions that have materially improved this paper.