Suffix trees and string complexity

Luke O'Connor Tim Snider

Department of Computer Science
University of Waterloo, Ontario, Canada, N2L 3G1
email: ljpoconn@watmath.uwaterloo.ca

Abstract

Let $\mathbf{s} = (s_1, s_2, \ldots, s_n)$ be a sequence of characters where $s_i \in Z_p$ for $1 \leq i \leq n$. One measure of the complexity of the sequence s is the length of the shortest feedback shift register that will generate s, which is known as the maximum order complexity of s [17, 18]. We provide a proof that the expected length of the shortest feedback register to generate a sequence of length n is less than $2\log_p n + o(1)$, and also give several other statistics of interest for distinguishing random strings. The proof is based on relating the maximum order complexity to a data structure known as a suffix tree.

1 Introduction

A common form of stream cipher are the so-called running key ciphers [4, 9] which are deterministic approximations to the one time pad. A running key cipher generates an ultimately periodic sequence $\mathbf{s} = (s_1, s_2, \ldots, s_n)$, $s_i \in Z_p$, $1 \leq i \leq n$, for a given seed or key K. Encryption is performed as with the one time pad, using s as the key stream, but perfect security is no longer guaranteed. Considerable effort has been devoted to developing algorithms for generating sequences s that are pseudorandom [11, 13, 19, 28]. The purpose of such work is to define sequences that are efficiently generated and satisfy one, or possibly several, measures of randomness for finite strings. Let Ω_p^n be the set of all sequences s of length n where $s_i \in Z_p$ for $1 \leq i \leq n$. A *statistic* is a function $\alpha : \Omega_p^n \rightarrow \mathbf{R}$ which measures some property of a sequence, such as the length of the longest gap for binary sequences, or the distribution of the binary derivative [14]. If the

R.A. Rueppel (Ed.): Advances in Cryptology - EUROCRYPT '92, LNCS 658, pp. 138-152, 1993.

distribution of α can be computed, in particular its expectation $\mathbf{E}(\alpha)$, α may be used to distinguish between random and nonrandom sequences by discarding those sequences s for which $\alpha(s)$ deviates significantly from the mean. If there are several statistics $\alpha_1, \alpha_2, \ldots, \alpha_j$ available for which the expectations are known, the more likely we are to detect nonrandom sequences. A collection of statistical tests for randomness is given in Knuth [21].

A notion attributed to Kolmogorov [22] characterizes the randomness of a sequence s as the encoded length of the smallest Turing machine program to produce s. Unfortunately, the Kolmogorov complexity of a sequence is not computable in general [23, §2.5], and consequently, the model of computation must be simplified in order to obtain a computable complexity measure. *Finite state machines* [4, 11] (FSM) are a class of automatons that consist of a finite set of states $Q = \{q_1, q_2, \ldots, q_m\}$, and a transition function $\delta : Q \rightarrow Q$. There is also an output function $\Delta : Q \rightarrow A$ which outputs a character from the alphabet A on each transition. The function δ is the 'program' associated with a FSM, and in this case, when executed will cause an infinite sequence of characters to be printed. If the state sequence of a FSM M after t transitions is $q'_1, q'_2, \ldots, q'_t$, then the output of M will be $\Delta(q'_1), \Delta(q'_2), \ldots, \Delta(q'_t)$. We will informally say that the size of an FSM M, or the length of its description, is defined as the size of the information required to identify state q_i, plus the space required to store δ, denoted as $|\delta|$.

A FSM is a special instance of a deterministic finite automaton (DFA) [15] where δ only depends on the current state, rather than also depending on a current input symbol. It is clear that DFAs with the ability to write output symbols, known as Moore machines, can mimic any FSM. Alternately, a FSM is a Moore machine which prints the same output string for every input string w.

Feedback shift registers (FSR) are a special class of FSMs which have much practical import as they can be directly implemented in hardware, and are fundamental to the design of most digital circuitry. A FSR consists of m stages, or memory cells, $x_1, x_2, \ldots, x_m$ and a feedback polynomial $f(X) \in Z_2[x_1, x_2, \ldots, x_m]$. A state transition in a FSR corresponds to a shift of the register contents ($x_i = x_{i-1}$, $2 \leq i \leq m$), and the assignment $x_1 = f(x_1, x_2, \ldots, x_m)$. The size or description of each state in an FSR is then m, the size of the shift register, and $|f(X)|$ is the cost of storing $f(X)$: it follows that the size of the machine is $m + |f(X)|$. With respect to FSRs, the complexity of a sequence s is given as the smallest FSR that generates s.

As with the class of Turing machines, we are now left with the problem of actually determining m and $f(X)$ for a given sequence s, or the shortest program which describes the sequence. A further attraction of FSRs is that when $f(X)$ is restricted to be a linear

polynomial (degree at most 1), the celebrated Massey-Berlekamp algorithm [24] can be used to determine m and $f(X)$ in time which is a polynomial function of the sequence length. Such a machine is known as a linear feedback shift register (LFSR). The number of stages required to generate a given sequence $\mathrm{s} = (s_1, s_2, \ldots, s_n)$ with a linear feedback polynomial is known as the *linear complexity* or *linear span* of a sequence, and will be denoted as $L(\mathrm{s})$. For example, if s = 010101110010 then $L(\mathrm{s}) = 5$, and the corresponding feedback polynomial is $f(X) = x_3 + x_4$. The following theorem is due to Rueppel [28].

Theorem 1.1 Let $\mathrm{s} = (s_1, s_2, \ldots, s_n)$ where $s_i \in Z_2$ for $1 \leq i \leq n$. Then assuming the uniform distribution on Ω_2^n, the expected linear span $\mathrm{E}(L(\mathrm{s}))$ of a binary sequence is

$$\mathrm{E}(L(\mathrm{s})) = \frac{n}{2} + \frac{4 + [n \text{ is odd}]}{18} + O(1). \tag{1}$$

□

Our original notion was to find the smallest program to generate a given sequence, with respect to some machine class. The actual program size of a LFRS is $m + |f(X)|$, but the linear span is typically measured as simply m, since the number of terms in $f(X)$ is bounded by m. The size of a LFSR is then at most $2m + 1$, where we have included a constant term in $f(X)$.

A natural extension of these ideas is to permit the feedback function $f(X)$ of the FSR to be a polynomial of arbitrary degree. We reiterate that the linear case is attractive as the linear span can be directly computed, but there has been little work in computing spans of higher order with the exception of Chan and Games [7]. Let $F_k(\mathrm{s})$ be the span of the sequence $\mathrm{s} = s_1, s_2, \ldots, s_n$ where $f(X)$ is a polynomial of degree at most k, $0 \leq k \leq n$. For a fixed sequence length n it follows that

$$F_n(\mathrm{s}) \leq F_{n-1}(\mathrm{s}) \leq \ldots \leq F_1(\mathrm{s}) \leq F_0(\mathrm{s}) \tag{2}$$

where $F_1(\mathrm{s}) = L(\mathrm{s})$ is the linear span of s. The inequalities in eq. (2) state that the size of the shift register may decrease as we allow the degree of $f(X)$ to increase. We may illustrate this possible reduction in memory by considering de Bruijn sequences [8]. Chan and Games [7] have studied the quadratic span of these sequences, and their results show a large difference between the linear and quadratic spans. For example, the de Bruijn sequences of length $63 = 2^6 - 1$ have an expected linear span of at least 62, while the expected quadratic span is at most 12.

There will always exist a pair (n_0, n_1) where $1 \leq n_0, n_1 \leq n$ such that $F_i(\mathrm{s}) = n_0$ for $n_1 \leq i \leq n$, which states that at some point the size of the shift register cannot

be reduced any further by permitting higher order feedback polynomials. Let $M^*(\mathbf{s})$ be the *span* of s, defined as the length of the shortest FSR generates s. The span gives a lower bound on the amount of information that must be stored in the states of a FSR to generate a given sequence. We note that for a de Bruijn sequence of length $2^n - 1$ the span is exactly n. Chan and Games [7] have commented that in general, determining the span of an arbitrary sequence s appears to be difficult given the nonlinearities involved. Surprisingly, there are several efficient nonalgebraic algorithms for determining the span of a sequence s. Later in this paper we will prove the following theorem.

Theorem 1.2 Let $\mathbf{s} = (s_1, s_2, \ldots, s_n)$ where $s_i \in Z_p$ for $1 \leq i \leq n$. Then assuming the uniform distribution on Ω_p^n, the expected span $\mathbf{E}(M^*\mathbf{s}))$ of s is then

$$\mathbf{E}(M^*(\mathbf{s})) = 2\log_p n + o(1).$$

□

Similar results to this theorem have been proven by Arratia, Gordon and Waterman [3], Apostolico and Szpankowski [2], Jansen [17] and Maurer [25]. In this paper we will prove Theorem 1.2 by showing that $M^*(\mathbf{s})$ is equivalent to the height of a data structure known as a suffix tree [26]. From this equivalence it will also follow that $M^*(\mathbf{s})$ can be computed in $O(n)$ time for sequences of length n. Also from this characterization there are several other statistics of interest that can be computed which can be used to distinguish random from nonrandom sequences.

The results of theorems 1.1 and 1.2 indicate that the length of the shift register to generate a sequence is influenced dramatically by the degree of the feedback polynomial: the expected linear span of binary sequence of length n is roughly $\frac{n}{2}$, while the expected span for the same sequence is approximately $2\log n$. For example, if s is a binary sequence of length 10^6, then the expected linear span is approximately 500,000 while the expected span is less than 50. The striking difference between the linear span and the span of a sequence can be accounted for by considering the space required to store the feedback polynomial $f(X)$. As noted for the linear case, the size of the feedback polynomial is bounded by the linear span itself. But for the case where $f(X)$ is unrestricted, the size of $f(X)$ may be an exponential function of the span. Thus the dramatic saving in memory cells for the shift register with arbitrary feedback is explained by encoding more information about the sequence into the feedback polynomial.

The paper is organized as follows. In §2.1 we review previous work on modeling sequence complexity with FSRs using nonlinear feedback. In §2.2 we introduce the suffix tree, give its relation to maximum order complexity, and present some asymptotic

properties of suffix trees. In §3 we examine the expected size of a feedback polynomial, both theoretically and experimentally, and prove that the expected size of a feedback polynomial is exponential in the length of the FSR.

2 Computing the span of a sequence

2.1 Previous work

Chan and Games [7] have presented an algorithm for computing the quadratic span of a sequence. The algorithm determines the coefficients of the feedback polynomial by repeatedly solving systems of linear equations. A feedback polynomial $f(X)$ over the m indeterminates $x_1, x_2, \ldots, x_m$ is quadratic if it can be written in the form

$$f(X) = \sum_{i=1}^{m} \sum_{j=1}^{m} a_{i,j} x_i x_j \tag{3}$$

where $a_{i,j} \in Z_2$ for $1 \leq i, j \leq m$, $i \neq j$. For example, if $\mathrm{s} = s_1, s_2, \ldots, s_8$ is a sequence of length 8, then the coefficients of the feedback polynomial for a shift register of length 3 are given as the solution to

$$\begin{pmatrix} s_1 & s_2 & s_1s_2 & s_3 & s_1s_3 & s_2s_3 \\ s_2 & s_3 & s_2s_3 & s_4 & s_2s_4 & s_3s_4 \\ s_3 & s_4 & s_3s_4 & s_5 & s_3s_5 & s_4s_5 \\ s_4 & s_5 & s_4s_5 & s_6 & s_4s_6 & s_5s_6 \\ s_5 & s_6 & s_5s_6 & s_7 & s_5s_7 & s_6s_7 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_{1,2} \\ a_3 \\ a_{1,3} \\ a_{2,3} \end{pmatrix} = \begin{pmatrix} s_4 \\ s_5 \\ s_6 \\ s_7 \\ s_8 \end{pmatrix}. \tag{4}$$

If there is no solution to the linear system in eq. (4), then the quadratic span of s necessarily exceeds 3. The Chan and Games algorithm, as well as the Massey-Berlekamp algorithm, is an *on-line* algorithm, which means that the algorithm processes the sequence one character at a time (left to right) and computes the span for the portion of the sequence that has been read. Let the span of the first k characters of s be n_k, such that $f_k(X)$ is the current feedback polynomial. If $s_{k+1} \neq f(s_{k-n_k+1}, \ldots, s_k)$, then a *discrepancy* is said to occur. When a discrepancy occurs the length of the shift register may have to be incremented. For the linear span, a change in the current span only occurs if $n_k < \frac{k}{2}$, and if this is the case, then the new linear span is given as $(k+1) - n_k$ [28]. Part of the efficiency of the Massey-Berlekamp algorithm is the knowledge of the correct increment to be made when a discrepancy occurs. For the quadratic case, the

appropriate increment to be made when a discrepancy occurs is not known, and must be found by searching an interval of possible increments. The Chan and Games algorithm is very general, and can in fact be used to compute $F_k(\mathrm{s})$ for $0 \leq k \leq n$. Perhaps for this reason, Games and Chan commented that determining the span of a sequence is a difficult problem, as their algorithm becomes less efficient for higher degree feedback polynomials.

A result similar to Theorem 1.2 was also presented in the thesis of Jansen [17]. Jansen has shown that $M^*(\mathrm{s})$ can be characterized as a property of the Directed Acyclic Word Graph (DAWG) [5] for a sequence s. The DAWG for a sequence s is a finite automaton that recognizes all substrings or subwords of s, or accepts the language $L_{\mathrm{s}} = \{u \mid vux = \mathrm{s}\}$. For this reason, the DAWG is also called the subword automaton. It can be shown that if the longest path in the DAWG for s from the start vertex to a vertex of outdegree at least 2 is k, then $M^*(\mathrm{s}) = k + 1$ [17]. While we may determine the span of s from the DAWG for s, there is no obvious way to determine any analytical information concerning the span from considering the DAWG since enumerating automatons is difficult in general. Jansen proved several of his results using statistical and combinatorial arguments which did not refer to the DAWG (see §3.4 of the thesis). Some properties of the DAWG are presented in [6, 12].

2.2 Pattern-matching algorithms

Pattern-matching algorithms are concerned with methods for finding and/or retrieving a pattern or substring w from a given piece of text Y [1]. If there are to be repeated searches on the text Y, then Y may be preprocessed and stored in a particular data structure to make searches of Y more efficient. Aho *et al.* [1] present a data structure which is useful to solve the following three pattern-matching problems: (a) given text Y and pattern w, find all occurrences of w in Y ; (b) given text Y, determine the longest repeated substring of Y; (c) given two texts Y_1 and Y_2, determine the longest string that is a substring of both Y_1 and Y_2. For this paper, it will be the solution of problem (b) that is of interest.

The data structure they presented is called a *position tree*, or later, a *suffix tree* [26]. Let $\mathrm{w} = w_1, w_2, \ldots, w_n, \$$ be a string of length $n + 1$ such that \$ is a unique character that only appears in position $n + 1$. Then every position in w is uniquely identified by at least one substring of w, namely $w_i, w_{i+1}, \ldots, w_n, \$$, which are known as the *suffixes* of w. Let $v(i)$ be the shortest substring of w that uniquely identifies position i of w. Consider inserting $v(1), v(2), \ldots, v(n)$ into a tree $T(\mathrm{w})$, which is called the *suffix tree* for

w. The tree $T(\mathrm{w})$ will have n leaves, corresponding to the first n positions of w, and the edges of the tree are labeled so that the path from the root to the leaf representing position i is $v(i)$ for $1 \leq i \leq n$. The height of a suffix tree $h(T(\mathrm{w}))$ is the length of the longest path from the root to some leaf in the tree.

Example 2.1 The suffix tree for s = 0101011100010$ is given in Figure 1. □

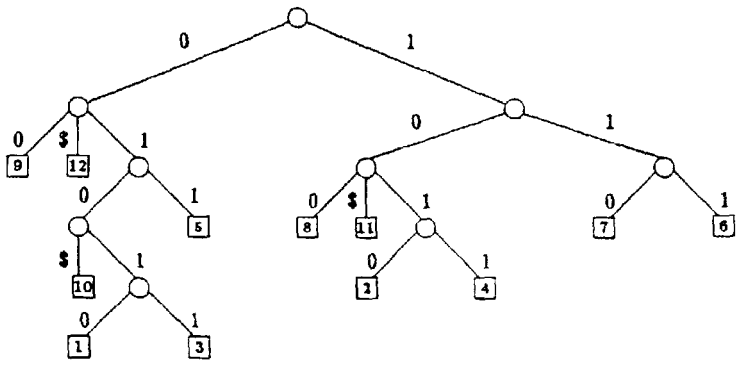

Figure 1: Suffix tree for 0101011100010$

The span of a sequence s is greater than k when there exist two substrings $s^1 = s_i, s_{i+1}, \ldots, s_{i+k-1}$ and $s^2 = s_j, s_{j+1}, \ldots, s_{j+k-1}$ such that $s^1_{i+h} = s^2_{j+h}$, $0 \leq h < k$, and $s_{i+k} \neq s_{j+k}$. We will say that the substring $s' = s_i, s_{i+1}, \ldots, s_{i+k-1}$ occurs in s with *two different successor characters*. Thus no function $f(x_1, x_2, \ldots, x_k)$ of k variables can generate the sequence s since

$$f(s_i, s_{i+1}, \ldots, s_{i+k-1}) = s_{i+k} \neq s_{j+k} = f(s_i, s_{i+1}, \ldots, s_{i+k-1}). \tag{5}$$

For the string w, let LRS(w) be the longest repeated substring of w, and let |LRS(w)| be the length of the LRS of w. Then observe that $|\mathrm{LRS}(\mathrm{w})| \leq h(T(\mathrm{w})) - 1$. From Figure 1, for s = 0101011100010$, LRS(s) = 0101 and |LRS(s)| = 4. In this case $|\mathrm{LRS}(\mathrm{w})| = h(T(\mathrm{w})) - 1$ which is not true in general. In fact, the difference between the span of a finite sequence and the length of its longest common substring can differ dramatically. Consider the $(n+1)$-bit string

$$\mathrm{s} = \underbrace{1\,0, 0, \ldots, 0}_{n-1}\$ \tag{6}$$

for which $|\mathrm{LRS(s)}| = n - 2$, while the span $M(s) = 1$ (where we assume that the shift register does not have to generate \$ as it is only appended to terminate the suffixes). The span of a sequence is then the longest string s' that occurs in s with two different successor characters β_1, β_2 such that $\beta_1, \beta_2 \notin \{\$\}$. This relation is formally proved next.

Theorem 2.1 The span of s is equal to the longest path from the root of $T(\mathrm{s})$ to a leaf u, where the parent of u has at least two children whose edges are not labeled by \$.

Proof. Let the length of the path from the root to u be k, such that the path represents the substring $s_i, s_{i+1}, \ldots, s_{i+k-1}$ from s. Since the parent of u has two children whose edges are not labeled by the \$ character, then the substring $s_i, s_{i+1}, \ldots, s_{i+k-1}$ occurs twice in s with different successor characters, both distinct from \$. Further, for any j, $1 \leq j \leq k-1$, there is a substring of length j, namely $s_{i+j}, s_{i+j+1}, \ldots, s_{i+k-1}$, with different successor characters, distinct from \$. Thus no FSR of length shorter than k can generate s, and it follows that $M(\mathrm{s}) \geq k$.

It remains to prove that $M(\mathrm{s}) \leq k$. Let u' be any leaf in $T(\mathrm{s})$ of maximum depth $k' > k$. Since $u' \neq u$, then the parent of u' must be an internal node of degree 2, with one child whose edge is labeled as \$. W.l.o.g, let u' be the child that has the edge labeled \$. If u' occurs in position i of s then $v(i) = s_i, s_{i+1}, \ldots, s_n, \$$. Thus if the character \$ is deleted from s, then the string $v'(i) = s_i, s_{i+1}, \ldots, s_n$ occurs at least twice in s, and either has a unique successor character, or no successor character. It follows that s can be generated with a FSR of length $k' - 1$.

By repeating the above argument for all leaves at depths $k'-1, k'-2, \ldots, k+1$, it follows that s can be generated by a FSR of length less than $(k+1) - 1 = k$. Thus $M^*(\mathrm{s}) = k$ which completes the proof. □

In general we may assume that the height of the suffix tree is an upper bound on the maximum order complexity. The experiments of Jansen [17] suggest that this bound is tight in the expected case. The possible discrepancy between $|\mathrm{LRS(s)}|$ and $M^*(\mathrm{s})$, as shown in eq. (6) and formalized in Theorem 2.1, is due to the fact that s is finite. A string s is said to be a semi-infinite string, or a *sistring* [12], if $\mathrm{s} = s_1, s_2, s_3, \ldots$ and thus extends infinitely to the right. The *ith* suffix $v(i)$ of s is defined as the substring $v(i) = s_i, s_{i+1}, s_{i+2}, \ldots$, for $i \geq 1$, and a suffix tree $T(\mathrm{s}^n)$ can be built from the first $n \geq 1$ suffixes of s. The analysis of suffix trees by Apostolico and Szpankowski [2] is done using sistrings but is directly adapted to the case of finite strings.

Recall that for a sequence s, the suffix tree $T(\mathrm{s})$ for s is a tree containing strings $v(1), v(2), \ldots, v(n)$, where $v(i)$ is the smallest string that identifies position i in s. The

strings $v(1), v(2), \ldots, v(n)$ are not *independent* as they may overlap. The (search) tree that results from inserting n independent strings $v'(1), v'(2), \ldots, v'(n)$ is a digital search tree called a *trie*. The properties of digital search trees have been extensively studied [10, 20, 27, 30], and in particular, the expected height of a random trie built with n strings drawn from an alphabet with p symbols is $2\log_p n + o(\log n)$. The properties of $T(\mathbf{s}^n)$ are similar (asymptotically equivalent) to the properties of a suffix tree built using a string of length n [16].

The structure of a suffix tree depends upon the overlaps that exist between the given suffixes of a string. For suffixes $v(i)$ and $v(j)$, $1 \leq i \neq j \leq n$, let V_{ij} be defined as the length of the longest prefix that is common to both $v(i)$ and $v(j)$. Then we may define the following statistics:

$$H_n = \max_{1 \leq i,j \leq n} V_{ij} \tag{7}$$

$$h_n = \min_{1 \leq i \leq n} \max_{1 \leq j \leq n} V_{ij} \tag{8}$$

$$D_n = \sum_{i=1}^{n} \frac{\max_{1 \leq j \leq n} V_{ij}}{n}. \tag{9}$$

Then H_n is called the nth-height, which corresponds to the height of a suffix tree built on n suffixes; h_n is called the nth-shallowness, and D_n is the average depth of the suffix tree. We note that each of these quantities can be computed directly as the suffix tree for a sequence can be constructed in $O(n)$ time using linear $O(n)$ space [26]. Further assume that the characters of s are drawn randomly and independently from Z_p, such that the jth character of s is p_i with probability q_i for $0 \leq i \leq p-1$. If $q_i = p^{-1}$ then the strings occur with uniform probability, which is known as the Bernoulli model [2].

Recall that $f(n) \sim g(n)$ if and only if $f(n) = g(n) + o(1)$.

Theorem 2.2 (Apostolico and Szpankowski [2]) Assuming the Bernoulli model, for large n, $\mathrm{E}(H_n) \sim 2\log_p n, \mathrm{E}(h_n) \sim \log_p n, \mathrm{E}(D_n) \sim \log_p n$. □

For binary sequences Jansen [17] has computed the span of all sequences up to length 22, and the empirical average was close to $2\log_2 n$, which agrees with $\mathrm{E}(H_n)$. Other estimates of $\mathrm{E}(H_n)$ have also been obtained, for example [3], but the derivation from the suffix tree seems the most robust. In fact, Apostolico and Szpankowski [2] are still able to compute the expected values of H_n, h_n and D_n when the Bernoulli model is not assumed. For example, when the q_i are not equal, expected value of H_n becomes

$$\mathrm{E}(H_n) = \frac{2}{\ln q_{max}^{-1}} \ln n + c \tag{10}$$

where ln is the natural logarithm, q_{max} the maximum of the q_i, and c a constant.

3 Properties of feedback polynomials

In this section we will restrict ourselves to considering binary sequences, because such sequences are of the most practical importance. Let the feedback polynomial of a FSR be $f(X)$. It then follows that $f(X) \in Z_2[x_1, x_2, \ldots, x_m]$, which is the set of all polynomial expressions involving the indeterminates $x_1, x_2, \ldots, x_m$. The polynomials of $Z_2[x_1, x_2, \ldots, x_m]$ correspond to boolean functions $f : \{0,1\}^m \rightarrow \{0,1\}$. Conversely, any m-bit boolean function f can be represented as a polynomial $Q_f(X) \in Z_2[x_1, x_2, \ldots, x_m]$, which is known as the Ring Sum Expansion (RSE) [29] or Algebraic Normal Form (ANF) [28] of f. Let $Z_2^A[x_1, x_2, \ldots, x_m]$ be the set of 2^{2^m} ANF polynomials, which are exactly those polynomials that remain in $Z_2[x_1, x_2, \ldots, x_m]$ when the elements of the ring are are simplified according to $x_i = x_i^2$, $1 \leq i \leq m$.

We are interested in two quantities associated with feedback polynomials: the size, or number of terms, and the degree. For $f(X) \in Z_2^A[x_1, x_2, \ldots, x_m]$, let $T(f(X))$ be the number of terms in $f(X)$, and let $\deg(f(X))$ be the degree of $f(X)$. Then the space required to store $f(X)$ is bounded by $n \cdot T(f(X))$.

Theorem 3.1 Assuming the uniform distribution on $Z_2^A[x_1, x_2, \ldots, x_m]$

$$\mathrm{E}(\deg(f, m)) \;=\; m - \frac{1}{2} + O\left(\frac{1}{2^m}\right)$$

Proof. Let $B(m,k) = \sum_{0 \leq j \leq k} \binom{m}{k}$ be the sum of the first $k+1$ binomial coefficients, $0 \leq k \leq m$. Then

$$\begin{aligned}
\mathrm{E}(\deg(f,m)) &= \sum_{0 \leq i \leq m} i \cdot \Pr(Q_f(X) \text{ has degree } i\ ) \\
&= \sum_{0 \leq i \leq m} i \cdot \frac{2^{B(m,i-1)} \cdot (2^{\binom{m}{i}} - 1)}{2^{2^m}} \\
&= 2^{-2^m} \cdot \left[\sum_{1 \leq i \leq m} i \cdot 2^{B(m,i)} - i \cdot 2^{B(m,i-1)} \right] \\
&= 2^{-2^m} \cdot \left[m \cdot 2^{2^m} - 2^{2^m - 1} - \sum_{0 \leq i \leq m-2} 2^{B(m,i)} \right] \\
&= m - \frac{1}{2} + O\left(\frac{1}{2^m}\right).
\end{aligned}$$

□

Thus we expect all taps of the FSR to be involved in the feedback polynomial. Further it follows from the binomial theorem that $E(T(f(X))) = 2^{m-1}$. Thus a random polynomial in $Z_2^A[x_1, x_2, \ldots, x_m]$ has a number of terms exponential in m, and a degree approximately m. Again consider the example of computing the linear span and span of of sequence of length $n = 10^6$. The total storage for the LFSR is approximately 750,000 bits, assuming half the coefficients in $f(X)$ are nonzero. Then from Theorem 1.2, the smallest shift register will have approximately $2 \log n$ stages, and from Theorem 3.1, assuming that the feedback polynomial is drawn at random from $Z_2^A[x_1, x_2, \ldots, x_{2\log n}]$, $f(X)$ will have $2^{2\log n - 1} > 2^{39}$ terms, which is far larger than the length of the sequence!

We note that the feedback polynomial to generate a given sequence is not unique in general. Let the span of a sequence s be m, and let $f(X)$ be a feedback polynomial. There are 2^m m-bit strings, and let $k, 1 \leq k \leq 2^m$, of these strings occur as substrings in s. It then follows that there are in fact $2^{2^m - k}$ polynomials $g(X)$ that agree with $f(X)$ on the k input strings found in s. We may still speak of *the* feedback polynomial $f(X)$ associated with every sequence s by defining $f(X)$ to be the polynomial which generates s, and for which $f(X) + 1$ has the smallest number of roots.

3.1 Experimental results

For this experiment we generated random bit sequences using the C library function *srandom*. The span for each sequence was calculated by comparing suffixes and a feedback function generated for the sequence. The feedback function was simplified to give a count of the number of terms using MAPLE™. This was applied to 100,000 32-bit sequences. The mean span was 8.65 with a standard deviation of 1.59. The following table shows the number of terms in the feedback function for different length spans.

4 Conclusion

We have shown that the suffix tree provides an alternate characterization of the maximum order complexity of a sequence. With this characterization it is then possible to accurately determine distribution of the maximum order complexity of a sequence, and also several other statistics. Also we have given some evidence to indicate that the linear complexity of a sequence is expected to be less than the maximum order complexity when both the length of the shift register and the size of the feedback polynomial are taken into account. One open problem is then to determine that order k of a feedback polynomial for which $F_k(s) + f(X)$ is minimized for sequences s of length n.

Span	# of cases	Mean # of terms	Std. Dev.
5	11	15.82	3.24
6	2871	30.30	4.84
7	20917	53.86	11.80
8	29822	92.47	11.80
9	22228	154.25	66.08
10	12485	251.68	140.59
11	6328	415.13	292.70
12	2887	666.99	599.33
13	1339	1078.58	1103.69
14	638	1713.08	2199.19
15	256	2679.77	4304.51
16	109	4335.17	9380.74
17	60	14028.02	32298.27
18	30	24725.47	64266.20
19	11	10025.73	14349.16
20	5	12698.00	11095.74
21	0	0.00	0.00
22	2	560.00	464.00
23	1	874.00	0.00

Table 1: Number of terms for different spaas

ACKNOWLEDGEMENTS

We would like to thank Alfredo Viola, Richard Games, and the Eurocrypt referees for their comments on earlier versions of the paper.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison–Wesley Publishing Company, 1974.

[2] A. Apostolico and W. Szpankowski. Self-alignments in words and their applications. Technical Report CDS-TR-732, Purdue University, 1987.

[3] R. Arratia, L. Gordon, and M. Waterman. An extreme value theory for sequence matching. *The Annals of Statistics*, 14(3):971–993, 1986.

[4] H. Beker and F. Piper. *Cipher Systems*. Wiley, 1982.

[5] A. Blumer, J. Blumer, A. Ehrenfeucht, D. Haussler, and R McConnell. Linear size finite automata for the set of all subwords of a word: outline of results. *Bulletin of the European Association of Theoretical Computer Science*, 21:12–20, 1983.

[6] A. Blumer, E. Ehrenfeucht, and D. Haussler. Average sizes of suffix trees and DAWGs. *Discrete Applied Mathemetics*, 24:37–45, 1989.

[7] A. Chan and R. Games. On the quadratic spans of periodic sequences. *IEEE Transactions on Information Theory*, IT-36(4):822–829, 1990.

[8] N. G. de Bruijn. A combinatorial problem. *Nederl. Akad. Wetensch. Proc*, 49:754–758, 1946.

[9] D. E. R. Denning. *Cryptography and Data Security*. Addison–Wesley Publishing Company, 1982.

[10] L. Devroye. A probabilistic analysis of the height of tries and of the complexity if triesort. *Acta Informatica*, 21:229–232, 1984.

[11] S. Golumb. *Shift Register Sequences*. Aegean Park Press, 1982.

[12] G. H. Gonnet and R. Baeza-Yates. *Handbook of Algorithms and Data Structures*. Addison-Wesley, Second Edition, 1991.

[13] E. J. Growth. Generation of binary sequences with controllable complexity. *IEEE Transactions on Information Theory*, 17(3):288–296, 1971.

[14] H. Gustafson, E. Dawson, and W. Caellie. Comparison of block ciphers. *Advances in Cryptology, AUSTCRYPT 90, Lecture Notes in Computer Science, vol. 453, J. Seberry and J. Piepryzk eds., Springer-Verlag*, pages 208–220, 1990.

[15] J. Hopcroft and J. Ullman. *An introduction to automata, languages and computation.* Addison–Wesley Publishing Company, 1979.

[16] P. Jacquet and W. Szpankowski. Autocorrelation on words and its applications: analysis of suffix trees by string-ruler approach. preprint, 1990.

[17] C. J. A. Jansen. *Investigations on Nonlinear Streamcipher systems: Construction and Evaluation methods.* PhD thesis, Philips, USFA BV, 1989.

[18] C. J. A. Jansen and D. Boekee. The shortest feedbck shift register that can generate a sequence. *Advances in Cryptology, CRYPTO 89, Lecture Notes in Computer Science, vol. 218, G. Brassard ed., Springer-Verlag*, pages 90–99, 1990.

[19] E. L. Key. An analysis of the structure and complexity of nonlinear binary sequence generators. *IEEE Transactions on Information Theory*, 22(6):732–736, 1976.

[20] D. E. Knuth. *The Art of Computer Programming : Volume 3, Sorting and Searching.* Addsion Wesley, 1973.

[21] D. E. Knuth. *The Art of Computer Programming : Volume 2, Seminumerical Algorithms.* Addsion Wesley, 1981.

[22] A. N. Kolmogorov. Three approaches to the quantitative definition of definition. *Problems in Information Transmission*, 1(1):1–7, 1965.

[23] M. Li and P. M. B. Vitanyi. Two decades of applied Kolmogorov complexity. Technical Report CS-R8813, Centre for Mathematics and Computer Science, April, 1988.

[24] J. L. Massey. Shift-register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, 15:122–127, 1969.

[25] U. M Maurer. Asymptotically tight bounds on the number of cycles in generalized de Bruijn-Good graphs. to appear in Discrete Applied Mathematics.

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

[27] M. Regnier. On the average height of trees in digital search and dynamic hashing. *Information Processing Letters*, 13(2):64–66, 1981.

[28] R. A. Rueppel. *Design and Analysis of Stream Ciphers.* Springer–Verlag, 1986.

[29] J. Savage. *The Complexity of Computing.* John Wiley, 1976.

[30] W. Szpankowski. On the analysis of the average height of a digital search tree: another approach. Technical Report CSD-TR-646, Purdue University, 1986.