


This item was submitted to Loughborough's Institutional Repository by the author and is made available under the following Creative Commons Licence conditions.




CC creative commons  
COMMONS DEED


**Attribution-NonCommercial-NoDerivs 2.5**


**You are free:**

- to copy, distribute, display, and perform the work

**Under the following conditions:**

 **Attribution.** You must attribute the work in the manner specified by the author or licensor.


 **Noncommercial.** You may not use this work for commercial purposes.

 **No Derivative Works.** You may not alter, transform, or build upon this work.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

**Your fair use and other rights are in no way affected by the above.**

This is a human-readable summary of the [Legal Code \(the full license\)](#).

[Disclaimer](#) 

For the full text of this licence, please go to:  
<http://creativecommons.org/licenses/by-nc-nd/2.5/>

# On the computation of the linear complexity and the $k$ -error linear complexity of binary sequences with period a power of two

Ana Sălăgean

**Abstract**—The linear Games-Chan algorithm for computing the linear complexity  $c(s)$  of a binary sequence  $s$  of period  $\ell = 2^n$  requires the knowledge of the full sequence, while the quadratic Berlekamp-Massey algorithm only requires knowledge of  $2c(s)$  terms. We show that we can modify the Games-Chan algorithm so that it computes the complexity in linear time knowing only  $2c(s)$  terms. The algorithms of Stamp-Martin and Lauder-Paterson can also be modified, without loss of efficiency, to compute analogues of the  $k$ -error linear complexity for finite binary sequences viewed as initial segments of infinite sequences with period a power of two.

We also develop an algorithm which, given a constant  $c$  and an infinite binary sequence  $s$  with period  $\ell = 2^n$ , computes the minimum number  $k$  of errors (and the associated error sequence) needed over a period of  $s$  for bringing the linear complexity of  $s$  below  $c$ . The algorithm has a time and space bit complexity of  $\mathcal{O}(\ell)$ . We apply our algorithm to decoding and encoding binary repeated-root cyclic codes of length  $\ell$  in linear,  $\mathcal{O}(\ell)$ , time and space. A previous decoding algorithm proposed by Lauder and Paterson has  $\mathcal{O}(\ell(\log \ell)^2)$  complexity.

**Index Terms**— $k$ -error linear complexity, linear complexity, repeated-root codes, stream cipher.

## I. INTRODUCTION

The linear complexity of a sequence (i.e. the length of the shortest recurrence relation, or Linear Feedback Shift Register which generates the sequence) is a fundamental parameter for virtually all applications of linearly recurrent sequences.

Computing the linear complexity  $c(s)$  of a linearly recurrent sequence  $s$  over a field needs in general quadratic time (Berlekamp-Massey algorithm, [1], [2]). For the particular case of binary sequences with period a power of two, Games and Chan devised an algorithm with linear time and space bit complexity, [3].

The  $k$ -error linear complexity of a periodic sequence  $s$  of period  $N$  is the minimum linear complexity that can be obtained for  $s$  by modifying up to  $k$  terms in one period (and modifying all other periods in the same way). This notion was defined in [4] and is closely related to previously defined notions of sphere complexity [5] and weight complexity [6].

The Games-Chan method has been extended by Stamp and Martin, [4] to computing the  $k$ -error linear complexity of a binary sequence with period a power of two, still in linear time. Further, Lauder and Paterson, [7], showed that the whole error linear complexity spectrum (i.e. the  $k$ -error complexity for each value of  $k$ ) of a binary sequence of period  $\ell = 2^n$  can be computed in  $\mathcal{O}(\ell(\log \ell)^2)$  time.

An important application of computing the linear complexity and  $k$ -error linear complexity appears in cryptography. If a sequence is used as a keystream in a stream cipher, an opponent intercepting part of the sequence will want to recover the whole sequence, thus breaking the cipher. If this is not possible, they might hope to at least determine a sequence which coincides with the correct sequence in all but a “small” number of positions.

The initial motivation of our work comes from a remark in [7], pointing to the fact that all the above mentioned efficient algorithms for binary sequences with period a power of two “suffer from the fact that they require as input an entire period of a sequence  $s$  to compute

$c(s)$ , while the Berlekamp-Massey algorithm only needs  $2c(s)$  bits. Thus, they are not applicable in realistic cryptographic situations”.

The results presented in the current paper remedy this situation. Namely, we prove in Section III that by suitably using the Games-Chan algorithm it is possible to compute the linear complexity of a binary sequence  $s$ , given only a finite segment of  $t \geq 2c(s)$  bits of the sequence, as long as we know that the period is a power of two (and we do not need to know in advance which power of two it is). Moreover, by suitably using the Stamp-Martin algorithm we can compute the linear complexity of a finite sequence of length  $t$ , viewed as an initial segment of an infinite sequence with period a power of two, even in the case when  $t$  is less than twice the complexity. Hence, for this particular type of sequences we obtain a linear (rather than quadratic) complexity algorithm with the same input and output specifications as the Berlekamp-Massey algorithm.

We cannot expect to be able to compute the  $k$ -error complexity of an infinite periodic sequence when we know less than one period of the sequence, as we do not know how many of the errors in an error pattern that minimises linear complexity will fall outside our known portion of the sequence. What we can compute instead is an analogue notion of  $k$ -error complexity for finite sequences, which we define in Section II as being the minimum complexity of any infinite sequence from a given class, whose initial segment coincides with the given finite sequence on all but possibly up to  $k$  positions. This definition fits well the cryptographic application mentioned above, and could be used for example in looking for sequences of low complexity which coincide with the correct sequence except for a certain percentage of the positions in any initial segment.

In Section III we also show that by suitably using the Stamp-Martin algorithm, the  $k$ -error linear complexity of a finite binary sequence, viewed as an initial segment of a sequence of period a power of two can be computed in  $\mathcal{O}(t)$  time, where  $t$  is the length of the finite sequence. The error linear complexity spectrum of such a finite sequence can be computed in  $\mathcal{O}(t(\log t)^2)$  time, by suitably using the Lauder-Paterson algorithm.

However, the cryptographic applications of our results are more limited than it may look at first sight. This is due to the fact that sequences with periods a power of two are relatively weak from the point of view of the number of terms needed to recover the whole sequence, see the discussion in Section IV.

In Section V we develop an algorithm which computes the minimum number of errors that need to be made in one period of a binary sequence of period  $\ell = 2^n$  in order to bring the complexity of the sequence below a given value. In other words, for a given  $c$  it computes the minimum  $k$  such that the  $k$ -error complexity of the sequence is no greater than  $c$ . It will also compute one error sequence for which this complexity is achieved. Our algorithm uses techniques similar to the Stamp-Martin Algorithm and to the so called L-pullup and B-pullup constructions of [7]. We include the explicit algorithm as Algorithm 5.1 and give its detailed bit complexity analysis, showing that the time and space complexities are  $\mathcal{O}(\ell)$ . A coarser estimation in [7] gives a  $\mathcal{O}(\ell \log \ell)$  rather than  $\mathcal{O}(\ell)$  bit complexity for the Stamp-Martin algorithm.

While our algorithm is interesting in its own right, the main motivation comes from its applicability to coding theory. Binary repeated-root cyclic codes of length  $\ell = 2^n$  were introduced in [8]. They are subcodes of Reed-Muller codes and a majority logic decoding is proposed, *loc. cit.* Lauder and Paterson, [7], apply their algorithm to decoding these codes in  $\mathcal{O}(\ell(\log \ell)^2)$  time. We improve on their result, by showing in Section VI that Algorithm 5.1 can be used for encoding and decoding binary repeated-root cyclic codes in linear time.

The material in this correspondence was included in part in a paper accepted for the International Symposium on Sequences and Their Applications, SETA'04, Seoul, South Korea, November 2004.

The author is with the Department of Computer Science, Loughborough University, UK (e-mail A.M.Salagean@lboro.ac.uk)

## II. NOTATION, DEFINITIONS AND KNOWN RESULTS

We first recall some basic facts on linearly recurrent sequences and establish the notation and definitions used in the paper.

Throughout the paper we work with binary sequences, i.e. sequences over the finite field  $\mathbb{F}_2$ . Most of the facts and definitions below work for sequences over any finite field, but we will not consider them here.

### A. Infinite sequences

We denote by  $\mathcal{S}$  the set of all (infinite) linearly recurrent sequences over  $\mathbb{F}_2$ . Note that for sequences over finite fields the following three notions are equivalent: periodic, recurrent and linearly recurrent.

Let  $s \in \mathcal{S}$ ,  $s = s_0, s_1, s_2 \dots$ . We will say that a polynomial  $f \in \mathbb{F}_2[x]$ ,  $f = x^m + a_{m-1}x^{m-1} + \dots + a_1x + a_0$  is an *annihilator polynomial* for  $s$  if  $s$  satisfies the linear recurrence given by the coefficients of  $f$  i.e.  $s_{i+m} + a_{m-1}s_{i+m-1} + \dots + a_1s_{i+1} + a_0s_i = 0$  for  $i = 0, 1, 2, \dots$ . The annihilator polynomials of  $s$  form an ideal in  $\mathbb{F}_2[x]$ , denoted  $\text{Ann}(s)$ . The monic annihilator polynomial of minimal degree is unique and is called the *characteristic polynomial* of  $s$ . We will denote it by  $\sigma(s)$ . Note that  $\sigma(s)$  generates  $\text{Ann}(s)$ . The linear complexity of  $s$  is the degree of the characteristic polynomial and will be denoted by  $c(s)$ .

Denote by  $P_N$  the set of sequences in  $\mathcal{S}$  of (not necessarily minimal) period  $N$ . If  $s \in P_N$  then  $x^N - 1 \in \text{Ann}(s)$  and  $\sigma(s) | x^N - 1$ .

In this paper we will concentrate on sequences that admit as period a power of two. We will denote by  $\mathcal{T}$  the set of binary sequences with period any power of two, i.e.  $\mathcal{T} = \bigcup_{i=0}^{\infty} P_{2^i}$ .

Using the fact that in  $\mathbb{F}_2[x]$  we have  $x^{2^n} - 1 = (x - 1)^{2^n}$  for any  $n$  we immediately obtain the following result:

**Proposition 2.1:** Let  $s \in \mathcal{T}$ . The linear complexity of  $s$  equals  $c$  if and only if the characteristic polynomial of  $s$  is  $(x - 1)^c$ .

Hence for sequences with period a power of two, knowing the linear complexity is tantamount to knowing the characteristic polynomial. Note that this is not the case for sequences of arbitrary period  $N$ ; sequences of same complexity  $c$  can have different characteristic polynomials if  $x^N - 1$  has several divisors of degree  $c$ .

The Games-Chan algorithm, [3], computes the linear complexity for any binary sequence with period a power of two, i.e. for any  $s \in P_{2^n}$  it computes  $c(s)$ . The whole sequence needs to be known, i.e. we need to know a (not necessarily minimal) period  $\ell = 2^n$  of the sequence and  $2^n$  consecutive terms of the sequence. The time (bit operations) and space complexity is linear in the period  $\ell$  of the sequence.

We now define the  $k$ -error linear complexity as in [4]. Let  $s \in P_N$  and let  $k \geq 0$  an integer. Denote by  $\text{wt}(\cdot)$  and  $d(\cdot, \cdot)$  the Hamming weight and the Hamming distance, respectively. The  $k$ -error linear complexity of  $s$  as a sequence of period  $N$ , denoted by  $c_{k,N}(s)$ , is defined as the minimum complexity that  $s$  can have after modifying  $k$  bits of a period:

$$c_{k,N}(s) = \min\{c(s + e) | e \in P_N, \text{wt}((e_0, e_1, \dots, e_{N-1})) \leq k\}. \quad (1)$$

The Stamp-Martin algorithm, [4], computes the  $k$ -error linear complexity for any binary sequence with period a power of two, i.e. for any  $s \in P_{2^n}$  it computes  $c_{k,2^n}(s)$ . As in the Games-Chan algorithm, the whole sequence needs to be known and the time (bit operations) and space complexity of the algorithm is linear in the period  $\ell = 2^n$  of the sequence. We stress the fact that the number of bit operations in the Stamp-Martin algorithm is indeed linear. The complexity estimate  $\mathcal{O}(\ell \log \ell)$  in [7] is too coarse, and the actual number of bit operations is  $\mathcal{O}(\ell)$ , see Theorem 5.4.

The error linear complexity spectrum of a sequence  $s \in P_N$  is defined as the set of pairs  $\{(k, c_{k,N}(s)) | 0 \leq k \leq \text{wt}((s_0, \dots, s_{N-1}))\}$ . The Lauder-Paterson algorithm, [7], computes the error linear complexity spectrum of any binary sequence with period  $\ell = 2^n$ . The bit complexity of the algorithm is  $\mathcal{O}(\ell(\log \ell)^2)$ . Again, one needs to know the full sequence in order to apply the algorithm.

One can also define  $k$ -linear complexity for costed sequences, following [4], [7]. For a sequence  $s \in P_N$ , a cost vector  $\text{cost} \in \mathbb{R}^N$  and a real number  $k$ , the  $k$ -error linear complexity of the costed sequence is defined as:

$$c_{k,N}(s, \text{cost}) = \min\{c(s + e) | e \in P_N, \sum_{0 \leq i < N, e_i \neq 0} \text{cost}[i] \leq k\}. \quad (2)$$

The usual  $k$ -error complexity of a (non-costed) sequence corresponds to the  $k$ -error complexity of the same sequence with an associated cost vector in which all entries equal 1.

As noted in [7], the Stamp-Martin algorithm can be adapted to compute the  $k$ -error linear complexity of costed sequences with period  $\ell = 2^n$  i.e. to compute  $c_{k,2^n}(s, \text{cost})$ . If the entries of the cost vector are bounded by a constant  $M$ , then the complexity of the algorithm will be  $\mathcal{O}(\ell \log M)$  (see Theorem 5.4 below).

### B. Finite sequences

We will define now the notions of linear complexity and  $k$ -linear complexity for finite sequences. The finite sequences will be viewed as initial segments of infinite sequences from a certain set of infinite sequences. More precisely, let  $z = (z_0, z_1, \dots, z_{t-1}) \in \mathbb{F}_2^t$  be a finite sequence of length  $t \geq 1$  and  $A \subseteq \mathcal{S}$  a set of infinite sequences. The linear complexity of  $z$  in  $A$ , denoted  $c(z, A)$ , is defined as the minimum linear complexity of all sequences in  $A$  which have  $z$  as an initial segment i.e.:

$$c(z, A) = \min\{c(s) | s \in A, s_i = z_i \text{ for } i = 0, \dots, t-1\}. \quad (3)$$

For any sequence of period  $N$ ,  $s \in P_N$  we have  $c((s_0, s_1, \dots, s_{N-1}), P_N) = c(s)$  as  $s$  is uniquely determined by its first  $N$  elements.

It is well known that one can determine the characteristic polynomial of a sequence  $s \in \mathcal{S}$  once at least  $2c(s)$  successive terms of  $s$  are known. We have therefore:

**Proposition 2.2:** If  $s \in A$  and  $t \geq 2c(s)$  then  $c((s_0, \dots, s_{t-1}), A) = c(s)$ .

The Berlekamp-Massey algorithm, [1], [2], computes the linear complexity of finite sequences, i.e. it computes  $c(z, \mathcal{S})$  for any finite sequence  $z \in \mathbb{F}_2^t$ . Equivalently, one can think of this algorithm as computing the linear complexity  $c(s)$  of an infinite sequence  $s$  knowing only the first  $2c(s)$  terms of the sequence. Note the algorithm is not restricted to sequences of a particular given period. The complexity of the Berlekamp-Massey algorithm is quadratic in the length  $t$  of the finite sequence.

Next we will extend the definition of  $k$ -error linear complexity to finite sequences. As before, let  $z = (z_0, z_1, \dots, z_{t-1}) \in \mathbb{F}_2^t$  be a finite sequence of length  $t \geq 1$ ,  $A \subseteq \mathcal{S}$  a set of infinite sequences and  $k \geq 0$ . Intuitively, there are two ways of defining the  $k$ -error linear complexity of  $z$  in  $A$ . One is to define it as the minimum linear complexity of all infinite sequences in  $A$  which coincide with  $z$  on all except up to  $k$  of the first  $t$  positions. The other is to define it as the minimum linear complexity in  $A$  of any finite sequence of the same length as  $z$  and which differs from  $z$  in at most  $k$  positions. It is easy to check that these two notions are equivalent, so we can define the  $k$ -error linear complexity of  $z$  in  $A$ , denoted  $c_k(z, A)$ , as:

$$\begin{aligned} c_k(z, A) &= \min\{c(z + e, A) | e \in \mathbb{F}_2^t, \text{wt}(e) \leq k\} \\ &= \min\{c(s) | s \in A, d((s_0, s_1, \dots, s_{t-1}), z) \leq k\}. \end{aligned} \quad (4)$$

Again, for any sequence of period  $N$ ,  $s \in P_N$  we have  $c_k((s_0, s_1, \dots, s_{N-1}), P_N) = c_{k,N}(s)$ , as  $s$  is uniquely determined by its first  $N$  elements.

Similarly one could define the  $k$ -error linear complexity of costed finite sequences, but we will not need it here.

### III. COMPUTING THE LINEAR COMPLEXITY AND $k$ -ERROR LINEAR COMPLEXITY FOR FINITE SEQUENCES

In this section our goal is to develop algorithms which compute the linear complexity and the  $k$ -error linear complexity of a finite sequence viewed as an initial segment of a binary sequence with period a power of two (we do not need to know which power though).

Note that any infinite sequence  $s \in \mathcal{T}$  will have period  $2^v$  where  $v$  is minimal such that  $c(s) \leq 2^v$ . Hence if we know at least  $2c(s)$  terms of  $s$ , we know in fact a whole period of the sequence, i.e. we know the whole sequence.

**Theorem 3.1:** Let  $z = (z_0, z_1, \dots, z_{t-1}) \in \mathbb{F}_2^t$  be a finite sequence of length  $t \geq 1$ . Define  $u = \lceil \log_2 t \rceil$  and define the infinite sequence  $s'$  of period  $2^u$  as follows:  $s'_i = z_i$  for  $i = 0, 1, \dots, t-1$  and  $s'_i = z_{i-2^{u-1}}$  for  $i = t, t+1, \dots, 2^u-1$ . Then

- (i) If  $c(z, \mathcal{T}) \leq \frac{t}{2}$  then  $c(z, \mathcal{T}) = c(s')$ .
- (ii) If  $c(z, \mathcal{T}) > \frac{t}{2}$  then  $c(s') > \frac{t}{2}$ .

*Proof:* By the definition (3) of  $c(z, \mathcal{T})$ , there is a sequence  $s \in \mathcal{T}$  such that  $s_i = z_i$  for  $i = 0, 1, \dots, t-1$  and  $c(s) = c(z, \mathcal{T})$ . By construction  $2^{u-1} < t \leq 2^u$ .

- (i) Since  $c(s) = c(z, \mathcal{T}) \leq \frac{t}{2} \leq 2^{u-1}$ ,  $(x-1)^{2^{u-1}} = x^{2^{u-1}} - 1$  is an annihilator polynomial for  $s$ . Hence  $s$  has period  $2^{u-1}$  i.e.  $s_i = s_{i-2^{u-1}}$  for all  $i \geq 2^{u-1}$  and in particular for  $i \geq t$ . It is now easy to check that  $s = s'$  so  $c(z, \mathcal{T}) = c(s')$ .
- (ii) From (3) we have  $c(s') \geq c(z, \mathcal{T})$ , so  $c(s') > \frac{t}{2}$ . ■

The theorem above can be used for computing the linear complexity of a finite sequence as follows. For a finite sequence  $z$  of length  $t$ , viewed as an initial segment of a sequence of period a power of two, we set up (in linear time) an infinite sequence  $s'$  of period  $2^{\lceil \log_2 t \rceil}$  as in Theorem 3.1. We then compute  $c(s')$  using the Games-Chan algorithm. If the result is at most  $\frac{t}{2}$  we output it as  $c(z, \mathcal{T})$ . Otherwise we output a message “complexity of  $z$  greater than half the number of terms”. This scenario may be useful when we actually want to compute the complexity of an infinite sequence  $s$  for which we know only the first  $t$  terms. We know by Proposition 2.2 that the complexity of the finite sequence is only guaranteed to give us the correct result for the infinite one if it is below half the number of terms.

We may however want to compute the exact value of  $c(z, \mathcal{T})$  even if it turns out to be higher than  $\frac{t}{2}$ . This, as well as the  $k$ -error complexity can be computed using the theorem below. The main idea is that if we expand the finite sequence to an infinite periodic costed sequence such that the new terms of the sequence are arbitrary but have zero cost, then any changes to the new terms will not count towards the  $k$  errors, only the changes in our original finite sequence will count.

**Theorem 3.2:** Let  $z = (z_0, z_1, \dots, z_{t-1}) \in \mathbb{F}_2^t$  be a finite sequence of length  $t \geq 1$ . Define  $u = \lceil \log_2 t \rceil$  and define the infinite costed sequence  $s'$  of period  $2^u$  as follows:  $s'_i = z_i$  and  $\text{cost}[i] = 1$  for  $i = 0, 1, \dots, t-1$  and  $s'_i$  have arbitrary binary values and  $\text{cost}[i] = 0$  for  $i = t, t+1, \dots, 2^u-1$ . Then  $c_k(z, \mathcal{T}) = c_{k,2^u}(s', \text{cost})$  for all  $k = 0, 1, \dots, \text{wt}(z)$ . In particular,  $c(z, \mathcal{T}) = c_{0,2^u}(s', \text{cost})$ .

The proof is straightforward.

Hence by setting up (in linear time) an infinite costed sequence  $s'$  of period  $2^{\lceil \log_2 t \rceil}$  as in Theorem 3.2 and then applying the Stamp-Martin algorithm to compute  $c_{k,2^{\lceil \log_2 t \rceil}}(s', \text{cost})$  we obtain in fact

$c_k(z, \mathcal{T})$ , the  $k$ -error linear complexity of  $z$ . In particular, for  $k = 0$  we obtain the linear complexity of  $z$ ,  $c(z, \mathcal{T})$ , regardless of whether this complexity is below half the number of terms of  $z$  or not. The resulting algorithm obviously runs in  $\mathcal{O}(t)$  time and is thus a more efficient alternative to the Berlekamp-Massey algorithm for the particular class of binary sequences with period a power of two.

By applying the Lauder-Paterson algorithm to the same costed sequence  $s'$  described above we obtain an  $\mathcal{O}(t(\log t)^2)$  algorithm for computing the full error linear complexity spectrum  $\{(k, c_k(z, \mathcal{T})) | k = 0, 1, \dots, \text{wt}(z)\}$ .

### IV. CRYPTOGRAPHIC CONSEQUENCES

Let us briefly look at the cryptographic significance of our results. The scenario we will consider here is that we have a linearly recurrent binary sequence (used in a stream cipher, for example) and a cryptanalyst is attempting to recover the whole sequence given only a “short” finite segment of the sequence.

Berlekamp-Massey algorithm allows them to do so, in quadratic time, once a segment of length equal to twice the complexity is known. For binary sequences with period a power of two, the method described in the previous section allows the cryptanalyst to achieve the same goal in linear rather than quadratic time, knowing again only a segment of length equal to twice the complexity.

However, note that if it is known that the sequence has as period a power of two, knowing a segment of length equal to the complexity (rather than twice the complexity) allows a cryptanalyst to recover the whole sequence. This is due to the fact that in this case knowing the complexity is equivalent to knowing the characteristic polynomial, see Proposition 2.1. All a cryptanalyst would do is, given a segment of  $t$  terms of the sequence  $s$ , assume that  $(x-1)^t$  is an annihilator polynomial for  $s$  and compute the rest of the sequence using the linear recurrence given by  $(x-1)^t$ . If the complexity of the sequence was indeed no greater than  $t$ , they would get the correct sequence. Hence, the class of sequences with period a power of two does not seem suitable for cryptographic applications from this point of view.

### V. COMPUTING THE MINIMUM $k$ TO ACHIEVE A GIVEN $k$ -ERROR COMPLEXITY

In this section we modify the Stamp-Martin algorithm so that for a given infinite periodic sequence of period  $\ell = 2^n$  and a given complexity  $c$  the algorithm outputs the minimum number of errors  $k$  needed so that the linear complexity of the sequence equals or falls below  $c$ . The corresponding error sequence is also computed.

The general idea is that while the Stamp-Martin algorithm starts with a number  $k$  of allowable errors and “forces” as many errors and as early in the algorithm as possible in order to obtain the lowest complexity possible, our algorithm will “force” as few errors and as late in the algorithm as possible, and only when absolutely needed in order to ensure the complexity stays no greater than the target complexity  $c$ .

Since we also want to compute the error pattern (rather than just the number of errors) which brings the complexity of  $s$  below our target  $c$ , we have to also keep track of the positions of the errors. Algorithms for computing the error pattern are described in [7], [9]. For our algorithm we will use a method similar to the so-called  $L$ -pullup and  $B$ -pullup of [7], but in a more compact and efficient form. Namely, we avoid the need of examining the cost vectors again (hence we can safely overwrite them) and we use bitwise XOR and AND for a further increase in efficiency and compactness of the formulation.

As in the Games-Chan and Stamp-Martin algorithms, at each step  $j = 0, \dots, n-1$  we work with a sequence  $a$  of period  $2^{n-j}$ . We split the sequence up into the left and the right half,  $L$  and  $R$ , and construct

a new sequence  $a$  of period  $2^{n-j-1}$  which will be processed at the subsequent step. We will use a two-dimensional array  $error$  to keep track of the errors that need to be made to the current sequence. A vector  $flag$  will contain flags such that  $flag[j]$ , for  $j = 0, \dots, n-1$  will signal if we decided to introduce or not errors in the sequence at step  $j$ . The row  $error[j]$ , containing binary values, will give the positions of the errors for step  $j$  in the case we do need to introduce errors, or in the case we do not, the positions where errors should be introduced, should it become necessary later. Only the entries  $error[j][0], \dots, error[j][2^{n-j} - 1]$  will be used for each  $j$ , so the two-dimensional array  $error$  can in fact be stored efficiently as a one-dimensional array of  $\sum_{j=0}^{n-1} 2^{n-j} = 2^{n+1} - 2$  bits. We use the two-dimensional array representation for expository purposes only.

An explicit algorithm is given below. We tried to keep as close as possible to the original formulation of the Stamp-Martin algorithm. To avoid confusion with the notation of the Stamp-Martin algorithm, here we use  $k'$  for the current number of errors and  $c'$  for the current value of the linear complexity of the sequence. To allow for extra flexibility, we work with costed sequences. If the input sequence is not a costed sequence, the cost vector is initialised by setting all entries to the value 1.

**Algorithm 5.1:** (Computing a sequence  $e$  of minimum cost such that  $c(s + e) \leq c$ )

Input:  $n, c, s, cost$   
 where  $n, c$  are positive integers,  
 $s = (s_0, s_1, \dots, s_{2^n-1})$  is a sequence of period  $2^n$  given by its first  $2^n$  terms and  $cost \in \mathbb{R}^{2^n}$  is a cost vector  
 Output:  $e$ , a sequence of period  $2^n$  given by its first  $2^n$  terms,  
 where  $e$  is of minimal cost  $\sum_{i=0}^{2^n-1} e_i cost[i]$  such that  $c(s + e) \leq c$ .

```

begin
 $a \leftarrow s; \ell \leftarrow 2^n; c' \leftarrow 0; k' \leftarrow 0,$ 
for  $j = 0$  to  $n - 1$  do
   $flag[j] \leftarrow 0$ 
  for  $i = 0$  to  $2^{n-j} - 1$  do  $error[j][i] \leftarrow 0$  endfor
endfor
for  $j = 0$  to  $n - 1$  do
   $\ell \leftarrow \ell/2$  % now  $\ell = 2^{n-j-1}$ 
   $L = a_0 a_1 \dots a_{\ell-1}; R = a_\ell a_{\ell+1} \dots a_{2\ell-1};$ 
   $b \leftarrow L + R$ 
   $T \leftarrow \sum_{i=0}^{\ell-1} b_i \min(cost[i], cost[i + \ell])$ 
  if  $T = 0$  or  $c' + \ell \geq c$  then
     $k' \leftarrow k' + T$ 
     $flag[j] \leftarrow 1$ 
    for  $i = 0$  to  $\ell - 1$  do
      if  $b_i = 1$  then
        if  $cost[i] \leq cost[i + \ell]$  then
           $a_i \leftarrow R_i; cost[i] \leftarrow cost[i + \ell] - cost[i];$ 
           $error[j][i] \leftarrow 1$ 
        else
           $a_i \leftarrow L_i; cost[i] \leftarrow cost[i] - cost[i + \ell];$ 
           $error[j][i + \ell] \leftarrow 1$ 
        endif
      endif
    else
       $a_i \leftarrow L_i; cost[i] \leftarrow cost[i] + cost[i + \ell]$ 
    endif
    endfor
  endfor
   $c' \leftarrow c' + \ell$ 
  for  $i = 0$  to  $\ell - 1$  do
     $a_i \leftarrow b_i;$ 
    if  $cost[i] \leq cost[i + \ell]$  then

```

```

       $error[j][i] \leftarrow 1$ 
    else
       $cost[i] \leftarrow cost[i + \ell]; error[j][i + \ell] \leftarrow 1$ 
    endif
  endif
endfor
 $e \leftarrow 0$ 
if  $a_0 = 1$  then
  if  $c' + 1 > c$  or  $cost[0] = 0$  then
     $k' \leftarrow k' + cost[0]$ 
     $e \leftarrow 1$ 
  else  $c' \leftarrow c' + 1$ 
  endif
endif
for  $j = n - 1$  downto  $0$  do
   $e \leftarrow duplicate(e)$ 
  if  $flag[j] = 1$  then  $e \leftarrow e \text{ XOR } error[j]$ 
  else  $e \leftarrow e \text{ AND } error[j]$ 
  endif
endfor
return( $e$ )
end

```

The function *duplicate* simply duplicates a binary string, i.e. concatenates it with a copy of itself. The XOR and AND operators are bitwise operators between binary strings of equal lengths.

**Theorem 5.2:** Algorithm 5.1 is correct.

*Proof:* As in [4], [7] it can be seen that the cost vector is updated at any step so that  $cost[i]$  reflects the cost of changes in the original sequence  $s$  in order to change the current element  $a_i$  without disturbing the results of the previous steps.

We prove by induction on  $n$  that the quantity  $k'$  computed in Algorithm 5.1 is indeed minimal such that  $c_{k', 2^n}(s, cost) \leq c$ . For  $n = 0$  this can be readily verified. We assume the algorithm works correctly for  $n - 1$  and prove that it works for  $n$ . We consider the first run of the main **for** loop, when  $j = 0$ . We denote by  $a^{(0)}$  and  $cost^{(0)}$  the values of the variables  $a$  and  $cost$  at the beginning of the first run of the **for** loop, and by  $a^{(1)}$  and  $cost^{(1)}$  their values at the end of the first run. The quantity  $T$  represents the minimal cost of making changes in the current sequence  $a^{(0)}$  such as to make its left half,  $L$  be equal to its right half  $R$ . The “**if**  $T = 0$  **or**  $c' + \ell \geq c$ ” will decide whether we make these changes or not. If  $T = 0$ , we obviously should make these changes, as they decrease the complexity of the sequence at no cost. If  $c' + \ell \geq c$ , i.e.  $2^{n-1} \geq c$  (as  $c' = 0$  and  $\ell = 2^{n-1}$  at this point), it means  $a^{(0)}$  has to be changed so that it has period  $2^{n-1}$  or less. Hence we do have to force  $L$  to be equal to  $R$ . We are left with the case when  $T > 0$  but  $2^{n-1} < c$ . Not doing changes in this case will mean that we add  $2^{n-1}$  to the current value  $c'$  of the complexity and then process the sequence  $a^{(1)} = b$ , effectively computing  $k'$  as the minimal quantity such that  $c_{k', 2^{n-1}}(a^{(1)}, cost^{(1)}) \leq c - 2^{n-1}$ . By the induction hypothesis, the algorithm computes this  $k'$  correctly. Note that  $T$  is exactly the minimum cost of changing all entries of  $a^{(1)} = b$  to 0, i.e. the minimum cost of reducing the complexity of  $a^{(1)}$  to 0. Hence  $k' \leq T$ . This means that not doing changes at this step is guaranteed to lead to a final cost no greater than the cost of doing changes at this step, while still keeping the complexity below the target  $c$ .

The correctness of the computation of the error pattern follows from the correctness proofs of the so called L-pullup and B-pullup constructions in [7]. One can show that  $duplicate(e) \text{ XOR } error[j]$  and  $duplicate(e) \text{ AND } error[j]$ , with  $error[j]$  computed as in the algorithm above, are in fact equivalent, more compact expressions for the L-pullup and the B-pullup of an error pattern  $e$ . ■

**Example 5.3:** We consider the sequence  $s$  of period 16 given by one period 1011 0111 1011 0110. We will compute the error pattern which makes the complexity of this sequence be at most 5. We apply Algorithm 5.1 to  $s$ , with  $n = 4, c = 5$  and the cost vector having all entries initialised to 1. The values of the string  $a$  during the algorithm will be:  $a = 1011\ 0110$ ,  $a = 1101$ ,  $a = 01$  and  $a = 1$ . The values of  $error[j]$  will be  $error[0] = 0000\ 0001\ 0000\ 0000$ ,  $error[1] = 1110\ 0001$ ,  $error[2] = 1000$ ,  $error[3] = 10$  and the flags will be  $flag[0] = 1$ ,  $flag[1] = 0$ ,  $flag[2] = 1$ ,  $flag[3] = 1$ . The values of  $e$  before each run of the final **for** loop will be 0, 10, 0010, 0010 0000, and finally  $e = 0010\ 0001\ 0010\ 0000$ . We also have  $k' = 3$  and  $c' = 5$  at the end of the algorithm. It can be verified that the sequence  $s + e = 1001\ 0110\ 1001\ 0110$  has characteristic polynomial  $(x - 1)^5$ , i.e. it has indeed complexity 5.

We now examine the complexity of the algorithm:

**Theorem 5.4:** Let  $s$  be an infinite binary sequence of period  $\ell = 2^n$ . The time bit complexity and the space bit complexity of Algorithm 5.1 and of the Stamp-Martin algorithm are linear,  $\mathcal{O}(\ell)$ . If the sequence is costed and the initial cost vector entries are all integers of absolute value at most  $M$ , the time bit complexity and the space bit complexity of Algorithm 5.1 and of the Stamp-Martin algorithm are  $\mathcal{O}(\ell \log M)$ .

*Proof:* We prove the Theorem for Algorithm 5.1; the proof for the Stamp-Martin algorithm is similar.

We first assume the sequence is not costed, so the cost vector is initialised with all entries equal to 1. The initialisation of  $flag$  and  $error$  have complexity equal to the size of these arrays, which is linear in  $2^n$  (see the space complexity analysis below). At each execution of the main **for** loop the values of the vector  $cost$  are at most doubled, so their bit length is increased by 1. This means that at the  $j$ -th execution of the **for** loop the bit length of the entries of the vector  $cost$  are changed from at most  $j + 1$  to at most  $j + 2$ . Inside the main **for** loop, the inner **for** loops run for  $i = 0, 1, \dots, \ell - 1$ . For each  $i$  there is one manipulation (addition, subtraction or comparison) of entries in the vector  $cost$ , so there are at most  $j + 2$  bit operations. Hence in total, there are  $(j + 2)\ell = (j + 2)2^{n-j-1}$  bit operations during the  $j$ -th execution of the main **for** loop. In total the main **for** loop performs

$$\sum_{j=0}^{n-1} (j + 2)2^{n-j-1} = 3 \cdot 2^n - n - 3 \quad (5)$$

bit operations, which is linear in  $2^n$ .

If the sequence is costed, then the entries of the cost vector have initially a bit length of  $\log_2 M$  and at the  $j$ -th execution of the **for** loop their bit length is changed from at most  $\log_2 M + j + 1$  to at most  $\log_2 M + j + 2$ . The sum (5) becomes  $\sum_{j=0}^{n-1} (\log_2 M + j + 2)2^{n-j-1} = (2^n - 1)\log_2 M + 3 \cdot 2^n - n - 3$ , which is  $\mathcal{O}(\ell \log M)$ .

As a side remark, we note that, while full details are not given in [7] regarding the  $\mathcal{O}(\ell \log \ell)$  claim for the bit complexity of the Stamp-Martin algorithm, we suspect this stems from a too coarse estimation for (5) along the lines  $\sum_{j=0}^{n-1} (j + 2)2^{n-j-1} \leq \sum_{j=0}^{n-1} (n + 1)2^{n-j-1} = (n + 1)(2^n - 1)$ , which would then suggest a  $\mathcal{O}(\ell \log \ell)$  complexity.

The last **for** loop, which computes the value of the error  $e$ , runs for  $j = n - 1$  downto 0 and for each  $j$  it performs a bitwise XOR or a bitwise AND between two bitstrings of length  $2^{n-j}$ . Hence in total this **for** loop performs  $\sum_{j=0}^{n-1} 2^{n-j} = 2^{n+1} - 2$  bit operations, which again is linear in  $2^n$ .

We now look at the space complexity. The bit arrays  $a, L, R, b$  have length at most  $2^n$  at all times. The entries of the array  $cost$  increase in size, but fewer and fewer are used. Namely, during the  $j$ -th execution of the main **for** loop, we use only  $2\ell = 2 \cdot 2^{n-j-1}$  entries,

each entry having a bit length of at most  $j + 2$  (or  $\log_2 M + j + 2$  for costed sequences). The total space taken by the vector  $cost$  is  $(j + 2)2^{n-j} \leq 2^{j+1}2^{n-j} = 2^{n+1}$ , hence linear in  $2^n$  (respectively  $(\log_2 M + j + 2)2^{n-j} \leq 2^n(\log_2 M + 2)$  hence  $\mathcal{O}(\ell \log M)$  for costed sequences). As mentioned earlier, the matrix  $error$  has  $n$  rows but in row  $j$  only  $2^{n-j}$  entries are used, with  $j = 0, \dots, n - 1$ . So we only need  $\sum_{j=0}^{n-1} 2^{n-j} = 2^{n+1} - 2$  bits. ■

## VI. DECODING REPEATED-ROOT CYCLIC CODES

Repeated-root binary codes with length a power of two have been introduced in [8]. It is shown, *loc. cit.*, that these codes are subcodes of Reed-Muller codes, and it is proposed that they be decoded by majority logic, just like the Reed-Muller codes. An alternative decoding algorithm with bit complexity  $\mathcal{O}(\ell(\log \ell)^2)$ , where  $\ell = 2^n$  is the length of the code, is proposed in [7]. In this section we develop a linear,  $\mathcal{O}(\ell)$ , decoding algorithm for these codes. We show that encoding can also be achieved in linear time.

As usual, a binary cyclic code of length  $\ell$  can be viewed as an ideal in  $\mathbb{F}_2[x]/\langle x^\ell - 1 \rangle$  and is generated by a divisor of  $x^\ell - 1$ . When  $\ell = 2^n$  we have  $x^\ell - 1 = x^{2^n} - 1 = (x - 1)^{2^n}$ , so the generator polynomial is of the form  $(x - 1)^g$  for some  $g$ . Alternatively, a codeword of length  $\ell$  can be viewed as an infinite sequence of period  $\ell$ , with the codeword being equal to one period of the sequence. If  $C$  is a code with generator polynomial  $f$ , a sequence  $s$  of period  $\ell$  represents a codeword in  $C$  iff the reciprocal of  $(x^\ell - 1)/f$  is an annihilator polynomial for  $s$ . For  $\ell = 2^n$  this means that a sequence  $s$  of period  $\ell$  represents a codeword in  $C = \langle (x - 1)^g \rangle$  iff  $(x - 1)^{\ell-g}$  is an annihilator polynomial for  $s$ , which in turn happens iff  $s$  has complexity at most  $c$ , where  $c = \ell - g$ .

Hence to decode a received vector  $r$ , viewed as a sequence of period  $\ell$ , we have to find the error pattern  $e$  of minimum weight such that  $r + e \in C$ , i.e.  $r + e$  has complexity at most  $c$ . This means we have to find the minimum  $k$  such that the  $k$ -error complexity of  $r$  is at most  $c$  i.e.  $c_{k,\ell}(r) \leq c$ . In [7] this is achieved by computing the full error linear complexity spectrum of  $r$  and picking up the smallest value  $k$  for which  $c_{k,\ell}(r) \leq c$ .

We show that we could instead use Algorithm 5.1 for decoding, and also for encoding these codes.

**Theorem 6.1:** Binary repeated-root cyclic codes of length  $\ell = 2^n$  can be encoded and decoded in linear time and space.

*Proof:* Let  $C$  be the code consisting of sequences with period  $\ell$  and complexity at most  $c$ . For decoding a received vector  $r$  obtained by transmitting the codeword  $s$  with error  $e$ , we apply Algorithm 5.1 for the inputs  $n, c, r$  and a cost vector with all entries initialised to 1. The output  $e$  gives the error, i.e.  $r + e$  is the corrected codeword.

For encoding, note that the code has dimension  $2^c$ . A message  $m \in \mathbb{F}_2^c$  can be systematically encoded as the unique sequence of period  $\ell$  whose first  $c$  symbols coincide with  $m$  and which has annihilator polynomial  $(x - 1)^c$ . To compute this sequence we could simply apply the recurrence relation given by  $(x - 1)^c$  with the initial terms given by  $m$ . However, this would yield an  $\mathcal{O}(c(\ell - c))$  algorithm i.e. a quadratic algorithm in general. Instead, we will again use Algorithm 5.1, thus encoding in linear time. Namely we initialise a sequence  $s'$  by putting  $s'_i = m_i$  and  $cost[i] = 1$  for  $i = 0, \dots, c - 1$  and  $s'_i$  having arbitrary values and  $cost[i] = 0$  for  $i = c, \dots, \ell - 1$ . We run Algorithm 5.1 on the inputs  $n, c, s', cost$  and obtain an error vector  $e$ . Note that in this case the error vector will always have zero cost. The encoding sequence for the message  $m$  will be  $s' + e$ . ■

**Example 6.2:** We consider the code  $C = \langle (x - 1)^{11} \rangle \in \mathbb{F}_2[x]/\langle x^{16} - 1 \rangle$ . This code can also be viewed as consisting of all periodic sequences of period 16 which have complexity at most  $16 - 11 = 5$ . Let us first encode a message of length 5, say  $m = 10010$ . We apply Algorithm 5.1 to the sequence  $s' = 1001\ 0000\ 0000\ 0000$ ,

$n = 4$ ,  $c = 5$  and a cost vector with the first 5 entries set to 1 and the remaining 11 entries set to 0. The values of the string  $a$  during the algorithm will be:  $a = 1001\ 0000$ ,  $a = 1001$ ,  $a = 11$  and  $a = 1$ . The values of  $error[j]$  will be  $error[0] = 0000\ 0000\ 1001\ 0000$ ,  $error[1] = 1000\ 0111$ ,  $error[2] = 0110$ ,  $error[3] = 00$  and the flags will be  $flag[0] = 1$ ,  $flag[1] = 0$ ,  $flag[2] = 1$ ,  $flag[3] = 1$ . The values of  $e$  before each run of the final **for** loop will be 0, 00, 0110, 0000 0110, and finally  $e = 0000\ 0110\ 1001\ 0110$ . We also have  $k' = 0$  and  $c' = 5$  at the end of the algorithm. It can be verified that the sequence  $s = s' + e = 1001\ 0110\ 1001\ 0110$  has indeed linear complexity 5 so it is a codeword.

Next assume the codeword  $s$  above is received as  $r = 1011\ 0111\ 1011\ 0110$ , i.e. with three errors. To decode we apply Algorithm 5.1 to  $r$ , with  $n = 4$ ,  $c = 5$  and the cost vector consisting of all 1's. As in Example 5.3, we obtain the error  $e = 0010\ 0001\ 0010\ 0000$ , and one can verify that this gives the correct decoding, i.e.  $r + e = s$ .

Lauder and Paterson note that their decoding algorithm will also be suitable for soft decoding, by setting  $s$  to be a hard decision binary version of the received word and setting each entry in the cost vector to a real value corresponding to the "reliability" of the corresponding 0/1 value in the received word. Our Algorithm 5.1 can be used for soft decoding in a similar way.

## VII. EXTENSION TO $p$ -ARY SEQUENCES

The Games-Chan and Stamp-Martin algorithms have been extended to sequences over  $\mathbb{F}_{p^m}$  with period  $\ell = p^n$ , where  $p$  is a prime in [10], [5].

It is natural to ask if the results of this paper can be extended to such sequences when  $p > 2$ . Theorem 3.1 does not hold in this setting. This can be seen from the fact that a sequence  $s$  over  $\mathbb{F}_p$  with period a power of  $p$  will have as minimal period  $p^v$  where  $v$  is minimal such that  $c(s) \leq p^v$ . Having  $2c(s)$  terms of the sequence does not necessarily mean we have a full period, as we may still have  $2c(s) < p^v$  if  $p > 2$ .

Theorem 3.2 on the other hand, does hold for arbitrary  $p$ . Hence we can use it in conjunction with the algorithms of [10], [5] to compute the complexity and  $k$ -linear complexity of finite sequences over  $\mathbb{F}_{p^m}$ , viewed as initial segments of infinite sequences with period  $\ell = p^n$ .

We expect that the algorithms of [10], [5] for  $p > 2$  could be modified along the lines of Algorithm 5.1 and then applied to encoding and decoding repeated-root cyclic codes over  $\mathbb{F}_{p^m}$  with length  $p^n$ .

## REFERENCES

- [1] E. Berlekamp, *Algebraic Coding Theory*. McGraw Hill, 1968.
- [2] J. Massey, "Shift register synthesis and BCH decoding," *IEEE Trans on Information Theory*, vol. 15, pp. 122–127, 1969.
- [3] R. Games and A. Chan, "A fast algorithm for determining the complexity of a binary sequence with period  $2^n$ ," *IEEE Trans. Information Theory*, vol. 29, pp. 144–146, 1983.
- [4] M. Stamp and C. Martin, "An algorithm for the  $k$ -error linear complexity of binary sequences of period  $2^n$ ," *IEEE Trans. Information Theory*, vol. 39, pp. 1398–1401, 1993.
- [5] C. Ding, G. Xiao, and W. Shan, *The stability Theory of Stream Ciphers*, ser. LNCS. Springer Verlag, 1991, vol. 561.
- [6] C. Ding, "Lower bounds on the weight complexities of cascaded binary sequences," in *Advances in Cryptology – AUSCRYPT'90*, ser. Lecture Notes in Computer Science, J. Seberry and J. Pieprzyk, Eds., vol. 453. Springer Verlag, 1991, pp. 39–43.
- [7] A. Lauder and K. Paterson, "Computing the error linear complexity spectrum of a binary sequence of period  $2^n$ ," *IEEE Trans. Information Theory*, vol. 49, pp. 273–280, 2003.
- [8] J. Massey, D. Costello, and J. Justesen, "Polynomial weights and code constructions," *IEEE Trans on Information Theory*, vol. 19, pp. 101–110, 1973.

- [9] T. Kaida, S. Uehara, and K. Imamura, "Computation of the  $k$ -error linear complexity of binary sequences with period  $2^n$ ," in *Concurrency and Parallelism, Programming, Networking*, ser. Lecture Notes in Computer Science, R. Yap, Ed., vol. 1179. Springer Verlag, 1996, pp. 182–191.
- [10] —, "An algorithm for the  $k$ -error linear complexity of sequences over  $GF(p^m)$  with period  $p^n$ ,  $p$  a prime," *Inform. Comput.*, vol. 151, pp. 134–147, 1999.

**Ana Sălăgean** Ana Sălăgean is currently a Lecturer in the Department of Computer Science of Loughborough University, UK. She has previously held positions at Nottingham Trent University and University of Britol, UK and at University of Bucharest, Romania. She graduated from Univeristy of Bucharest, Romania and holds a PhD from J. Kepler University, Linz, Austria. Her research interests are in coding theory, symbolic computation and cryptography.