A Genetic Algorithm for Computing the k-Error Linear Complexity of Cryptographic Sequences

A. Alecu and A. M. Salagean

***Abstract*— Some cryptographical applications use pseudorandom sequences and require that the sequences are secure in the sense that they cannot be recovered by only knowing a small amount of consecutive terms. Such sequences should therefore have a large linear complexity and also a large k-error linear complexity. Efficient algorithms for computing the k-error linear complexity of a sequence over a finite field only exist for sequences of period equal to a power of the characteristic of the field. It is therefore useful to find a general and efficient algorithm to compute a good approximation of the k-error linear complexity. In this paper we investigate the design of a genetic algorithm to approximate the k-error linear complexity of a sequence. Our preliminary experiments show that the genetic algorithm approach is suitable to the problem and that a good scheme would use a medium sized population, an elitist type of selection, a special design of the two point random crossover and a standard random mutation. The algorithm outputs an approximative value of the k-error linear complexity which is on average only 19.5% higher than the exact value. This paper intends to be a proof of concept that the genetic algorithm technique is suitable for the problem in hand and future research will further refine the choice of parameters.**

I. Introduction

The k-error linear complexity of a sequence is a generalisation of the notion of linear complexity. While the linear complexity of a sequence is defined as the length of the smallest linear recurrence relation which generates that sequence, the k-error linear complexity is the length of the smallest linear recurrence relation which generates a sequence which differs from the original sequence in at most k positions.

When designing a stream cipher, the keystream sequence has to have a large linear complexity. Using the Berlekamp-Massey Algorithm, a sequence can be efficiently recovered by knowing a number of consecutive terms equal to twice its linear complexity. Sequences with low linear complexity would therefore be vulnerable to known plaintext attacks. Similarly, sequences with low k-error linear complexity for small values of k could also be vulnerable if the corresponding linear recurrence relation was found.

An exact algorithm to compute the k-error linear complexity only exists for periodic sequences over a finite field $GF(p^m)$ and with period a power of p, p being prime and $m \geq 1$ (see Stamp and Martin [13], Lauder and Paterson [8] for $p = 2$ and Kaida, Uehara and Imamura [7] for an arbitrary p). These algorithms are based on the algorithms of Games and Chan([4]) and Ding, Xiao, Shan [3] for computing the linear complexity of such sequences, and they work only when a full period of the sequence is known, i.e. the whole sequence is known, which is not the case in cryptanalysis applications.

We propose and investigate a genetic algorithm for computing the k-error linear complexity focusing on the choice of parameters (population size, number of generations, technique of selection, crossover or mutation, mutation probability, crossover probability) some of them depending on the size of the input sequence and the number of errors k and also how to choose the evaluation function.

A. Alecu and A. M. Sălăgean are with the Department of Computer Science, Loughborough University, LE11 3TU, Loughborough, UK (contact email: {a.alecu, a.m.salagean}@lboro.ac.uk).

II. Background

We will first introduce some of the background regarding the linear complexity and k-error linear complexity of cryptographical sequences and also about the evolutionary techniques we used.

A. Linear complexity and k-error linear complexity

Stream ciphers are symmetric ciphers in which the plain text bits are encrypted one at a time by XORing them with a bit from the secret key stream. Often the key stream is generated using a certain combination of Linear Feedback Shift Registers (LFSRs) which expands a short key shared by the sender and receiver into a longer pseudorandom sequence. However, any recurrent sequence over a finite field is linearly recurrent and can therefore be generated by one single (usually much larger) LFSR.

A sequence generated by a LFSR can be defined by a linear recurrence relation or, equivalently, by a characteristic polynomial.

Definition 2.1: Given an infinite sequence $s = s_0, s_1, \ldots$ (or a finite sequence $s = s_0, s_1, \ldots, s_{t-1}$) with elements in a field K, we say that s is a linear recurrent sequence if it satisfies a relation of the form

$$s_j + c_{L-1}s_{j-1} + \ldots + c_1 s_{j-L+1} + c_0 s_{j-L} = 0 \quad (1)$$

for all $j = L, L+1, \ldots$ (or for all $j = L, L+1, \ldots t-1$, respectively), where $c_0, c_1, \ldots, c_{L-1} \in K$ are constants. The equation (1) is called a homogeneous linear *recurrence relation of order* L and we associate to it a *characteristic polynomial* $C(X) = X^L + c_{L-1}X^{L-1} + \ldots + c_1 X + c_0$. If L is minimal for the given sequence, we call L the *linear complexity* of s, denoted $L(s)$. A recurrence relation of minimal order is called a *minimal recurrence relation* and a characteristic polynomial of minimal degree is called a *minimal characteristic polynomial*.

More details about linear recurrent sequences and terminology can be found for example in Lidl and Niederreiter [9].

If a sequence has linear complexity L, the minimal linear recurrence relation that generates the sequence can be determined knowing $2L$ consecutive terms of the sequence. This can be done by solving the system of linear equations obtained by writing equation (1) for $j = L, L+1, \ldots, 2L-1$. A more efficient method is given by the Berlekamp-Massey Algorithm.

A sequence which is used as a key stream for a stream cipher needs therefore to have a high linear complexity, in order to make it hard for an intruder to be able to find the whole sequence by only intercepting a short number of consecutive terms.

The notion of linear complexity has been generalised to k-error linear complexity, which is the minimal linear complexity of the sequence in which at most k positions are changed. The concept was first outlined by Ding, Xiao, Shan ([3]) under the name of weight complexity, and defined under the name of k-error linear complexity by Stamp and Martin ([13]). Note that the 0-error linear complexity coincides with the linear complexity.

Definition 2.2: Given an infinite sequence $s = s_0, s_1, \ldots$ of period N, with elements in a field K and a fixed integer k, $0 \leq k \leq w_H((s_0, \ldots, s_{N-1}))$, the *$k$-error linear complexity* of the sequence s is defined as

$$L_k(s) = \min\{L(s+e) |\ e \text{ is a sequence of period } N \text{ over } K, \\ w_H((e_0, e_1, \ldots, e_{N-1})) \leq k\}$$

For a given finite sequence $s = s_0, s_1, \ldots, s_{t-1}$ with elements in a field K and for a fixed integer k, $0 \leq k \leq w_H(s)$, the *k-error linear complexity* of the sequence s is defined as

$$L_k(s) = \min\{L(s+e) | e \in K^t, w_H(e) \leq k\} \qquad (2)$$

The sequences e are called error sequences or error patterns. The k-error linear complexity profile of the sequence is defined as being the set of pairs $(k, L_k(s))$, for all k with $0 \leq k \leq w_H(s)$. We denote k-error linear complexity profile of order k_0, the set of pairs $(k, L_k(s))$, for all k with $0 \leq k \leq \max\{k_0, w_H(s)\}$. ($w_H(s)$ denotes the Hamming weight i.e. the number of non-zero entries of s.)

Property 2.1: Given a (finite or infinite) sequence s with elements in a finite field $GF(q)$, we have $L_i(s) \geq L_j(s)$, for all $i < j$.

If the k-error linear complexity of a sequence is very low for small values of k (e.g., k less than 10% of the length of the sequence), then that sequence is likely to be easily recovered when only knowing a short segment of the sequence. This is why it would not be secure to use it as a key stream for a stream cipher.

By extending the Games-Chan Algorithm ([4]), which computes the linear complexity of a periodic binary sequence with the period a power of 2, Stamp and Martin ([13]) have devised an algorithm to efficiently (in linear time and space) compute the k-error linear complexity of a periodic binary sequence with the period a power of 2. The Stamp-Martin Algorithm was further extended to compute the whole k-error linear complexity profile by Lauder and Paterson [8]. Algorithms for computing the linear complexity and the k-error linear complexity of a sequence, for periodic sequences which have as period a power of the characteristic of the field have been given by Ding, Xiao, Shan [3], Kaida, Uehara, Imamura [7], Kaida [6]. All these algorithms, unlike the Berlekamp-Massey Algorithm, need a whole period as input, which means that the whole sequence is already known, which would not be the case in cryptanalysis applications.

There is no general algorithm to compute the k-error linear complexity profile of an arbitrary sequence over an arbitrary finite field, other than the exhaustive search.

B. Genetic Algorithms

Evolutionary computing techniques are inspired by the natural evolution observable in species and the process which allows them to survive by continuously adapting to the changes in their environment. The main principles implemented by evolutionary computing are natural selection, or 'survival of the fittest', and inheritance ([5]).

Genetic algorithms have proven to be useful in solving a big variety of problems. They have been successfully applied on famous NP-complete problems like Travelling Salesman Problem, Knapsack Problem, Prisoner's Dilemma etc.

A genetic algorithm is a probabilistic algorithm which maintains a population of potential solutions for the problem in hand, by evolving it throughout a number of generations using genetic operators like selection and combination. At each iteration, the quality of each possible solution is measured using a fitness function and then a new population is created by selecting the most fit individuals on that basis (same individual can be duplicated in a population, the order of duplication being usually direct proportional to its fitness). Some members of the new population undergo transformations in order to create new solutions. The transformations can be unary (mutation), which create new individuals by slightly changing single solutions or of higher order (crossover), which combine a number of solutions to create a new individual. After a number of generations the algorithm converges and it is hoped that the best individual which has been found represents a reasonable solution ([11]).

It is still a challange and much research is invested into finding the optimum values for the parameters involved (population size, number of generations, selection and crossover technique, probability of crossover, mutation technique, probability of mutation) so that the algorithm is efficient (i.e. fast) and accurate (i.e. finds a good approximation of the exact solution).

C. Berlekamp-Massey Algorithm

The Berlekamp-Massey Algorithm ([1],[10]) computes the characteristic polynomial and the linear complexity of a sequence over a field. Besides being general in that it applies to a sequence over an arbitrary field, the Berlekamp-Massey Algorithm has another advantage: if the linear complexity of the sequence is L, the algorithm will determine the

characteristic polynomial and the linear complexity after processing $2L$ terms of the sequence. The algorithm runs in quadratic time.

The algorithm is iteratively taking each term of a finite sequence $s_0, s_1, \ldots, s_{t-1}$ and processes it one by one, adjusting the characteristic polynomial if necessary. At each step of the algorithm the current characteristic polynomial $C^{(n)}(X)$ generates the n sequence terms s_0, s_1, ..., s_{n-1} processed so far. Therefore, after all the terms are processed, the characteristic polynomial of the input sequence is obtained. The linear complexity is the degree of the resulting characteristic polynomial.

At each step, in addition to the current characteristic polynomial $C^{(n)}(X)$, the last characteristic polynomial $C^{(m)}(X)$ of degree strictly smaller than the degree of $C^{(n)}(X)$ is also stored. We denote $L^{(i)} = \deg(C^{(i)})$. By calculating the discrepancy $d^{(n)}$, where

$$d^{(n)} = s_n + \sum_{i=0}^{L^{(n)}-1} c_i^{(n)} s_{i+n-L^{(n)}} \tag{3}$$

which represents the difference between the term which is expected using the current polynomial and the actual term s_n which is currently processed, 3 possible cases are identified:

1. If $d^{(n)} \neq 0$ then s_n cannot be generated using $C^{(n)}(X)$
   a) If $2L^{(n)} > n$ then the new characteristic polynomial is computed as $C^{(n+1)}(X) \leftarrow C^{(n)}(X) - \frac{d^{(n)}}{d^{(m)}} \cdot X^{(m-L^{(m)})-(n-L^{(n)})} \cdot C^{(m)}(X)$ and it has the same degree as the previous one;
   b) If $2L^{(n)} \leq n$ then the new characteristic polynomial is computed as $C^{(n+1)}(X) \leftarrow X^{(n-L^{(n)})-(m-L^{(m)})} \cdot C^{(n)}(X) - \frac{d^{(n)}}{d^{(m)}} \cdot C^{(m)}(X)$ and it has a higher degree than the previous one, namely $L^{(n+1)} = n + 1 - L^{(n)}$; m is updated to n.
2. If $d^{(n)} = 0$ then s_n can be generated using $C^{(n)}(X)$, so the characteristic polynomial stays unchanged $C^{(n+1)}(X) = C^{(n)}(X)$.

For initialisation, the first non-zero term in the sequence, say s_j is detected, the characteristic polynomials are set to $C^{(i)}(X) \leftarrow 1$ for $i = 0, \ldots, j$, $C^{(j+1)}(X) \leftarrow X^{j+1}$, and $m \leftarrow j$. At the end of the algorithm, $L^{(t)}$ is the linear complexity of the sequence and $C^{(t)}(X)$ is a minimal characteristic polynomial (which is unique if $2L^{(t)} \leq t$, otherwise it may not be unique).

III. A GENETIC ALGORITHM FOR COMPUTING THE k-ERROR LINEAR COMPLEXITY OF A SEQUENCE

In this section we will describe our investigations and findings in designing a genetic algorithm for computing the k-error linear complexity of a sequence s of size t with elements in a finite field $GF(q)$, where q is a prime power. Since this is an optimisation problem with a well defined search space (see definition 2.2), a genetic algorithm is very well suited.

Algorithm 1 Genetic Algorithm for computing the k-error linear complexity - A Schematic View

Input: A finite sequence $s = s_0, s_1, \ldots, s_{t-1}$; k_0
Output: The approximate k_0-error linear complexity
Initialise population $POP(0)$ of size PS
Evaluate $POP(0)$
$gen \leftarrow 0$
while $gen < NOGEN$ **do**
  Select new $POP(gen+1)$ from $POP(gen)$
  Crossover in $POP(gen+1)$ with probability p_X
  Mutate in $POP(gen+1)$ with probability p_M
  Report statistics for the current generation
  Evaluate $POP(gen+1)$
  $gen \leftarrow gen + 1$
end while

The input of the algorithm is the sequence s and a value k_0, the order of the k-error linear complexity which is to be computed. The output of the algorithm will be an approximation of the k_0-error linear complexity. See listing 1 for a schematic view on the algorithm.

The algorithm holds a global solution which is updated whenever necessary (i.e. when an individual improves the current global solution).

In the following, we will expand on the different types of implementations and schemes that we considered.

A. Chromosomes

Since we are dealing with sequences over finite fields, it is natural to use a haploid string encoding for the chromosomes. We define a chromosome to be any possible error pattern $e \in GF(q)^t$, $e = (e_0, e_1, \ldots, e_{t-1})$ of weight at most k_0 (i.e. $w_H(e) \leq k_0$).

The best chromosomes are the error patterns which inflict smaller linear complexity on the input sequence s. The search space size depends on the size of the sequence t, the order of the finite field q and the number of errors, k_0. We denote the set $E_k = \{e | e \in GF(q)^t, w_H(e) \leq k\}$, therefore the search space size, SS, is given by the formula

$$SS = \sharp(E_{k_0}) = \sum_{i=0}^{k_0} \binom{n}{i} (q-1)^i \tag{4}$$

The initial population is randomly generated. The random number generator used is the C `rand()` linear congurential generator function. The algorithm 2 describes the method used in generating the individuals, error patterns of size t with elements in $GF(q)$ and weight less than k_0.

We denote the size of the population to be PS and we will experiment in order to find the best choice of value. Some papers show that a moderate population size is leading to fitter populations faster (e.g. [12]). Since we desire to make sure we adjust the population size and number of generations depending on the size of the search space we will use a population size of the following form and we will try to find

Algorithm 2 Generate a random sequence e of size t and weight at most k_0

for $i = 0, 1, \ldots, t-1$ **do**
  $e_i \leftarrow 0$
end for
$k' \leftarrow$ a random number less than or equal to k_0
for $i = 0, 1, \ldots, k'-1$ **do**
  $pos \leftarrow$ a random position between 0 and $t-1$
  $val \leftarrow$ a random value in $GF(q)$
  $e_{pos} \leftarrow val$
end for

the optimum choice for the coefficient c ($c > 0$).

$$PS = ck_0\lceil \ln(q^t)\rceil = ck_0\lceil t \ln q\rceil \quad (5)$$

B. The fitness function

The quality of each individual is evaluated using the fitness function. Our goal is to find the element e in E_{k_0} which minimizes the linear complexity of the sum[1] $s + e$.

Within our search space all possible error patterns, e, of Hamming weigt up to k_0 are comparable using the linear complexity of the sum $s + e$, so this is a natural choice for the fitness function. However, since traditionally genetic algorithms are maximizing and not minimizing the fitness function we will choose the fitness function to be, for each error pattern, the reverse of the linear complexity of the sequence onto which we applied that error pattern.

We define the fitness function f by

$$f : E_{k_0} \to \mathbf{Z}, \text{ where } f(e) = -L(s + e)$$

We use the Berlekamp-Massey Algorithm (section II-C) to compute the fitness function for each element of the population. The computational complexity of the evaluation is therefore at most $\mathcal{O}(PS \cdot t^2)$.

Experiments show that the search space is fragmented and there are many local minima and maxima. This is a challange for the genetic algorithm. Due to the discrete nature of the linear complexity of the sequence when summed with different error patterns from the search space we are not able to directly pinpoint the elements in the domain E_{k_0} of function f which correspond to the minimum or the maximum values.

Example 3.1: Figure 1 shows the shape of the distribution of linear complexities for a given binary sequence $s = 1011110011010110$ of size 16 when taking all the possible error patterns in the full space $GF(2)^{16}$. The x and y axis are corresponding to each possible weight from 0 to 16 and each possible linear complexity from 0 to 16 respectively. The third coordinate, z, in each point (x, y, z) represents the number of error patterns e of weight x such that $L(s+e) = y$. The figure presents a scaled version of the real distribution.

[1]We consider the term by term addition between sequences.
If $a = (a_0, \ldots, a_{t-1})$ and $b = (b_0, \ldots, b_{t-1})$ then the sum sequence $(a+b) = (a_0+b_0, \ldots, a_{t-1}+b_{t-1})$ with the additions in the field which includes the terms of a and b.

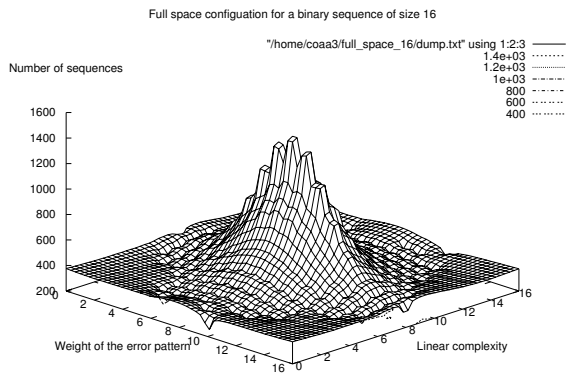


Fig. 1. Distribution of linear complexities of $s = 0110111101110101$ when combined with all poosible error sequences over $GF(2)^{16}$

C. Genetic operators

1) Selection: There are various schemes for the selection of the best individuals for further recombination. However the general idea is that each chromosome will be copied zero, one or more times according to its fitness (more times if it is more fit) making sure that the population remains varied.

We will use three alternative schemes for selection:

- Elitist selection of level $25\% \cdot PS$ (ELSEL). A set percentage of the population is chosen for survival in the order of the fitness values. It has been found that the higher the level of elitism the lower the efficiency as the GA will deal and will have to evaluate individuals which were previously processed ([2]). For this reason and since intuitively the algorithm does not benefit from the overduplication of fit individuals, but more from the population diversity, the rest of the individuals up to the population size are randomly generated using the same generation method as for the initial population. The complexity of this approach is $\mathcal{O}(PS \ln(PS))$ since it is necessary to order the individuals in the population by their fitness value. ([11])
- Roulette wheel with slots sized according to fitness (RWSEL). We first evaluate every single individual in the population, $e^{(i)}$, for their fitness value, $f(e^{(i)})$ ($i = 0, 1, \ldots, PS-1$). We also compute the total fitness of the population, TF, which represents the sum of the fitness of each individual

$$TF = \sum_{i=0}^{PS-1} f(e^{(i)})$$

This way we can compute the relative and the cumulative probability of each individual, which is $rprob$ and $cprob$ respectively

$$rprob(e^{(i)}) = \frac{f(e^{(i)})}{TF}$$

$$cprob(e^{(i)}) = \sum_{j=0}^{i} rprob(e^{(j)})$$

The selection process consists of spinning the roulette wheel PS times and select each time an existing individual such that the fitter the individual the bigger the probabillity is for it to be selected.

Formally, the following two steps are repeated PS times:

1) Generate a random value r, $r \in [0, 1]$.
2) If $r < cprob(e^{(0)})$ then select $e^{(0)}$, otherwise find j such that $cprob(e^{(j-1)}) < r \leq cprob(e^{(j)})$ and select $e^{(j)}$.

- Tournament Selection (TRSEL). In a two order tournament model, random pairs of individuals from the current population are chosen and the best one out of the two is selected to survive in the next population. Intuitively, this method is particularly suitable as the fitness values for this problem are very close which makes the Roulette Wheel selection to give close probabilities of selection to most of the individuals.

  Formally, the following two steps are repeated PS times:

  1) Generate two random values pos_1 and pos_2, such that $0 \leq pos_1 < pos_2 \leq t - 1$.
  2) If $f(e^{(pos_1)}) < f(e^{(pos_2)})$ then select $e^{(pos_2)}$, otherwise select $e^{(pos_1)}$.

2) Crossover: For each chromosome e, when calculating the linear complexity of $s + e$ with the Berlekamp-Massey Algorithm we hold all the intermediary linear complexities, therefore we obtain the whole linear complexity profile. That is, for each $e^{(i)}$, $i = 0, 1, \ldots, PS - 1$ we hold $lcp^{(i)} = (lcp_0^{(i)}, \ldots, lcp_{t-1}^{(i)})$, a vector of size t such that $lcp_j^{(i)}$ represents the linear complexity of the sequence $s+e^{(i)}$ up to term j. We also hold the intermediary discrepancies in an array $dis^{(i)}$ where $dis_j^{(i)}$ is the intermediary discrepancy calculated by the Berlekamp-Massey algorithm at step j when being run on the sequence $s + e^{(i)}$.

During the Berlekamp-Massey algorithm, only the case when the discrepancy $d^{(n)} \neq 0$ and $2L^{(n)} \leq n$ (case (1b) in Section II) yields an increase in the current complexity of the sequence. We are interested in minimizing the linear complexity. It seems therefore natural in this case to concentrate on error patterns which have that current term changed in such a way as to make the discrepancy zero and therefore make an increase in complexity unncessary.

Two of the crossover types that we consider are using the previous remark and the information given by the linear complexity profile as well as the intermediary discrepancies held against each chromosome.

We define a parameter called probability of crossover, p_X, $p_X \in [0, 1]$. The crossover involves choosing two parents PS times, therefore repeating the following steps PS times

1) Generate a random value r, $r \in [0, 1]$.
2) If $r < p_X$ and no parent yet selected then choose first parent $p^{(i)}$.
3) If $r < p_X$ and first parent has been selected then choose the second parent $p^{(j)}$ such that $p^{(i)} \neq p^{(j)}$, crossover parents $p^{(i)}$ and $p^{(j)}$ to obtain one or two children and reset parents.

After each crossover the best two individuals out of the two parents and the children, depending on their fitness value, are kept for the new generation.

For readability, in the following, we will denote the parent chromosomes $p^{(1)}$ and $p^{(2)}$ with the corresponding linear complexity profiles, $lcp^{(1)}$ and $lcp^{(2)}$ and the intermediary discrepancies $dis^{(1)}$ and $dis^{(2)}$.

Having chosen two parents $p^{(1)}$ and $p^{(2)}$, the following crossover schemes are considered.

- Single point crossover (SPX). Generate a random natural number pos, $pos \in 0, 1, \ldots, t - 1$.
  $p^{(1)} = (\underbrace{p_0^{(1)}, p_1^{(1)}, \ldots, p_{pos-1}^{(1)}}, p_{pos}^{(1)}, \ldots, p_{t-1}^{(1)})$
  $p^{(2)} = (p_0^{(2)}, p_1^{(2)}, \ldots, p_{pos-1}^{(2)}, p_{pos}^{(2)}, \ldots, p_{t-1}^{(2)})$
  The resulted offsprings are:
  $c^{(1)} = (\underbrace{p_0^{(1)}, p_1^{(1)}, \ldots, p_{pos-1}^{(1)}}, p_{pos}^{(2)}, \ldots, p_{t-1}^{(2)})$
  $c^{(2)} = (p_0^{(2)}, p_1^{(2)}, \ldots, p_{pos-1}^{(2)}, \underbrace{p_{pos}^{(1)}, \ldots, p_{t-1}^{(1)}})$
  This strategy provides some diversity without disrupting any long building blocks. [2]
- One point crossover using the linear complexity profile (LCPSPX). Generate a random natural number pos, $pos \in \{0, 1, \ldots, t - 1\}$. Find in parent $p^{(1)}$ the first position after pos where there is an increase in the complexity of $p^{(1)} + s$. Otherwise stated, find first position i in the first parent $p^{(1)}$ such that $pos < i$, $lcp_i^{(1)} < lcp_{i+1}^{(1)}$ and $p_{i+1}^{(1)} \neq p_{i+1}^{(2)}$. That means that if we apply the following recombination then it is likely that in some of the cases the linear complexity of the first child to be reduced.
  $p^{(1)} = (\underbrace{p_0^{(1)}, p_1^{(1)}, \ldots, p_{pos}^{(1)}, \ldots, p_i^{(1)}}, p_{i+1}^{(1)}, \ldots, p_{t-1}^{(1)})$
  $p^{(2)} = (p_0^{(2)}, p_1^{(2)}, \ldots, p_{pos}^{(2)}, \ldots, p_i^{(2)}, p_{i+1}^{(2)}, \ldots, p_{t-1}^{(2)})$
  The resulted offsprings are:
  $c^{(1)} = (\underbrace{p_0^{(1)}, p_1^{(1)}, \ldots, p_{pos}^{(1)}, \ldots, p_i^{(1)}}, p_{i+1}^{(2)}, \ldots, p_{t-1}^{(2)})$
  $c^{(2)} = (p_0^{(2)}, p_1^{(2)}, \ldots, p_{pos}^{(2)}, \ldots, p_i^{(2)}, \underbrace{p_{i+1}^{(1)}, \ldots, p_{t-1}^{(1)}})$
  This strategy provides good diversity without disrupting any long building blocks. Experimentally we noticed that especially on fields with a lower order (for example binary field) we can take out the third condition when finding the crossover point i ($p_{i+1}^{(1)} \neq p_{i+1}^{(2)}$) as this is eliminating a considerable amount of possible crossover positions without providing consistent advantages.
- Two point crossover (TPX). Generate two random natural numbers pos_1 and pos_2, such that $0 \leq pos_1 < pos_2 \leq t - 2$.
  $p^{(1)} = (\underbrace{p_0^{(1)}, \ldots, p_{pos_1}^{(1)}, \ldots, p_{pos_2}^{(1)}}, \ldots, p_{t-1}^{(1)})$

[2] Building blocks are short sequences of good genes which appear in the chromosomes. It is desired not to disrupt them if possible in order to promote them to the following generations.

$p^{(2)} = (p_0^{(2)}, \ldots, p_{pos_1}^{(2)}, \ldots, p_{pos_2}^{(2)}, \ldots, p_{t-1}^{(2)})$
The resulted offsprings are:
$c^{(1)} = (\underbrace{p_0^{(1)}, \ldots}, \underline{p_{pos_1}^{(2)}, \ldots, p_{pos_2}^{(2)}}, \underbrace{\ldots, p_{t-1}^{(1)}})$
$c^{(2)} = (\underline{p_0^{(2)}, \ldots}, \underbrace{p_{pos_1}^{(1)}, \ldots, p_{pos_2}^{(1)}}, \underline{\ldots, p_{t-1}^{(2)}})$

- Two point crossover using the linear complexity profile (LCPTPX). One thing which can be improved on the LCPSPX crossover presented above, is to use the linear complexity information for the second parent as well.
  Generate two random natural numbers pos_1 and pos_2, such that $0 \leq pos_1 < pos_2 \leq t-2$. Find the first position i in the first parent $p^{(1)}$ such that $pos_1 < i$, $lcp_i^{(1)} < lcp_{i+1}^{(1)}$ and $p_{i+1}^{(1)} \neq p_{i+1}^{(2)}$. Also find the first position j in the second parent $p^{(2)}$ such that $pos_2 < j$, $lcp_j^{(2)} < lcp_{j+1}^{(2)}$ and $p_{j+1}^{(2)} \neq p_{j+1}^{(1)}$.
  That means that if we apply the following recombination it is likely that in some of the cases the linear complexity of one or both children to be reduced improving their fitness. Note that we assume to have $i < j$. If it does not happen $p^{(1)}$ and $p^{(2)}$ as well as i and j can be interchanged to fulfill this requirement.
  $p^{(1)} = (\underbrace{p_0^{(1)}, \ldots, p_i^{(1)}, p_{i+1}^{(1)}, \ldots, \ldots, \ldots, \ldots, p_{t-1}^{(1)}})$
  $p^{(2)} = (\underline{p_0^{(2)}, \ldots, \ldots, \ldots, \ldots, p_j^{(2)}, p_{j+1}^{(2)}, \ldots, p_{t-1}^{(2)}})$
  The resulted offsprings are:
  $c^{(1)} = (\underbrace{p_0^{(1)}, \ldots, p_i^{(1)}}, \underline{p_{i+1}^{(2)}, \ldots, \ldots, \ldots, \ldots, p_{t-1}^{(2)}})$
  $c^{(2)} = (\underline{p_0^{(2)}, \ldots, \ldots, \ldots, \ldots, p_j^{(2)}}, \underbrace{p_{j+1}^{(1)}, \ldots, p_{t-1}^{(1)}})$
  This strategy provides a higher diversity than LCPSPX. It is likely for disruption of long building blocks to appear but the ones at the beginning of the sequence remain untouched.
- Uniform random crossover (URX). Recent studies show that the use of the uniform random crossover operator is superior in most cases, see [14]. This crossover technique obtains one child only, c from the two parents $p^{(1)}$ and $p^{(2)}$.
  $p^{(1)} = (p_0^{(1)}, p_1^{(1)}, \ldots, p_{t-1}^{(1)})$
  $p^{(1)} = (p_0^{(2)}, p_1^{(2)}, \ldots, p_{t-1}^{(2)})$
  Generate t random real numbers, $r_i \in [0,1]$, $i = 0, 1, \ldots, t-1$. For each i, if $r_i < 0.5$ then $c_i = p_i^{(1)}$, otherwise $c_i = p_i^{(2)}$.

3) Mutation: Whereas selection and crossover are the evolutionary operators which are implementing the need to promote good patterns from one generation to the next one, the mutation is an operator which introduces variety and implements the need to throw the individuals away from any potential local optimum that they are converging to.

We consider two types of mutation, the standard one and one which uses the linear complexity information similarly with the crossover types LCPSPX and LCPTPX. We define a parameter called probability of mutation, p_M, $p_M \in [0,1]$.

- Simple random mutation (SRM). This type of mutation loops through all PS individuals in the population and for each of them through all t terms, it generates a random value r, $r \in [0,1]$ and if $r < p_M$ then it adds a random value from the field to the current term.
  Formally, for each i, $i = 0, 1, \ldots, PS-1$ and for each j, $j = 0, 1, \ldots, t-1$, generate a random value r, $r \in [0,1]$. If $r < p_M$ then generate another random value val, $val \in GF(q)$ and $e_j^{(i)} = e_j^{(i)} + val$.
- Random mutation using the linear complexity profile (RMLCP). This mutation process tries to obtain individuals with a higher fitness by using the linear complexity similarly with the crossover types LCPSPX and LCPTPX. Aditionally it uses the discrepancy information for a better chance to enhance the fitness of the new individual.
  Formally, for each i, $i = 0, 1, \ldots, PS$ and each $j = 0, \ldots, t-1$ generate a random value r_j, $r_j \in [0,1]$. If $r_j < p_M$ then generate a random position $pos \in \{0, 1, \ldots, t-1\}$ and find the first position m in $e^{(i)}$ such that $pos < m$ and $lcp_m^{(i)} < lcp_{m+1}^{(i)}$. Once m found make $e_{m+1}^{(i)} = e_{m+1}^{(i)} - dis_{m+1}^{(i)}$. We remind that $dis_{m+1}^{(i)}$ represents the discrepancy at step $m+1$ in Berlekamp-Massey algorithm applied to the sequence $s + e^{(i)}$.

The fitness value of the mutated individual is evaluated and the global solution updated if necessary. For the sake of diversity the initial individual is discarded and the mutated one is kept for the next population disregarding the value of its fitness.

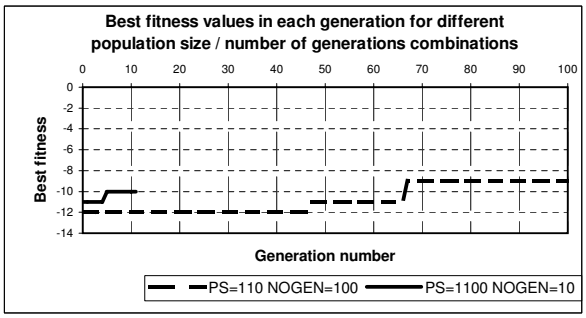
For both crossover and mutation an additional postprocessing is needed in order to check if the resulted offsprings have a higher weight than k_0 and if so, randomly switch to 0 some of the non zero terms until the weight is at most k_0. The fitness values of the children are evaluated and the global solution updated.

Since the genetic algorithm depends on the choice of quite a few parameters we will refer to the genetic algorithm as: $kGA(t, k, s, PS, NOGEN, ST, XT, MT, p_X, p_M)$ where t, k, s are the input values and

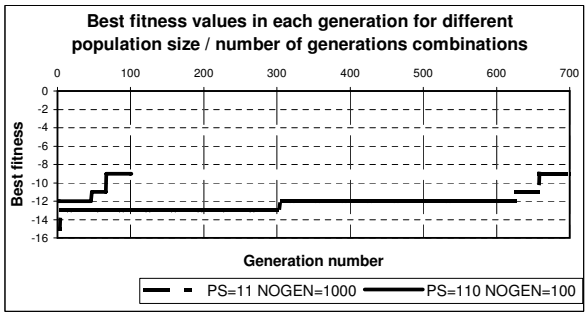
- PS is an integer representing the population size,
- $NOGEN$ is an integer representing the number of generations,
- ST is the selection scheme used, it can be $ELSEL$, $RWSEL$, $TRSEL$ (see section III-C.1),
- XT is the crossover scheme used, it can be SPX, $LCPSPX$, TPX, $LCPTPX$, URX (see section III-C.2),
- MT is the mutation scheme used, which can be SRM or $LCPRM$ (see section III-C.3),
- p_X is a value in the range $[0,1]$ representing the probability of crossover,
- p_M is a value in the range $[0,1]$ representing the probability of mutation.

IV. Tests and Results

In order to assess the accuracy of the algorithm and to establish which is the best combination of parameters to choose for the genetic algorithm we have set up a series of tests as presented in the following.



(a) Speed of convergence for large population size and small number of generations (PS=1100 and NOGEN=10) opposed to moderate population size and number of generations (PS=110 and NOGEN=100)



(b) Speed of convergence for moderate population size and number of generations (PS=110 and NOGEN=100) opposed to small population size and large number of generations (PS=110 and NOGEN=1000)

Fig. 2. Best fitness value in each generation when the population size and number of generation vary

We considered 10 randomly chosen sequences of length 32 and $k_0 = 5$ (each bit of the sequences is generated with the C `rand()` linear congruential generator function). The algorithms have been ran with different combination of parameters, but on the same input the same seeds were used for each algorithm so that the initial population is the same.

The search space size in this case is $SS = \sum_{i=0}^{5} \binom{32}{i} = 243.001$ (see equation (4)). This is a relatively small search space however we are only testing that the concept is fit for the purpose. Also, another benefit is the fact that for this length we can use the exhaustive search as a benchmarking algorithm to evaluate the solutions of the genetic algorithm.

In the following tests, the evaluation of the best algorithm is done by calculating the ratio between the k_0-error linear complexity value obtained by the genetic algorithm and the exact k_0-linear complexity value given by the exhaustive search. We call this value the accuracy of the genetic algorithm. We then average the accuracy over the same 10 sequences for each test.

1) Population size and number of generations: First test decides the best choice for the population size PS. If we put $q = 2, t = 32$ and $k_0 = 5$ in (5) we obtain $PS = 110c$. We take $c = 0.1$, $c = 1$ and $c = 10$, therefore population sizes 11, 110 and 1100. In order to have similar durations for the tests, we choose the number of generations $NOGEN$ to be inverse proportional with the population size, namely 1000, 100 and 10, respectively. We ran $kGA(t, k, s, PS, NOGEN, ST, XT, MT, p_X, p_M)$ for each of these combinations using selection type $ST = ELSEL$, crossover type $XT = URX$ with probability of crossover $p_X = 0.66$ and mutation type $MT = SRM$ with probability of mutation $p_M = 0.33$. We denote the best choice for the population size, PS^*, and the number of generations, $NOGEN^*$. The average accuracies obtained for each combination of parameters $(PS, NOGEN)$, $(11, 1000)$, $(110, 100)$ and $(1100, 10)$ are $1.29861, 1.24682$ and 1.27222, respectively. Therefore we chose the combination of parameters $PS^* = 110$ and $NOGEN^* = 100$.

The figure 2 shows the convergence speed for each of the $(PS, NOGEN)$ combinations when the genetic algorithm is applied to the binary sequence $11110110010011101000100101010100$ to compute the 5-error linear complexity. The exact value is 8, therefore the fittest error sequence would have the fitness -8. When using small or moderate population size ($(PS, NOGEN)$ is $(11, 1000)$ or $(110, 100)$) the result is 9, whereas when using a large population size the result is 10. In the figure, the best fitness value is shown against each of the generations.

The algorithm with large population size reaches its optimum, -10, in generation number 5, after processing 5500 individuals. However it does not manage to improve the solution any further by the end of the 10 generations.

From a convergence point of view the other two configurations are similar. When using a small population size and large number of generations the optimum, -9, is reached in generation number 659, after processing 7249 individuals. The algorithm ran with moderate population size and number of generations arrives at its optimum, -9, in generation number 67, after processing 7370 individuals. We are therefore confident in choosing the moderate population size and number of generation since the convergence to the solution is smooth and the average accuracy over all the test sequences is the best.

2) Selection: Having the population size PS^* and number of generations $NOGEN^*$ we compare the three types of selection $ELSEL$, $RWSEL$ and $TRSEL$ by running the algorithm $kGA(t, k, s, PS^*, NOGEN^*, ST, XT, MT, p_X, p_M)$ for each of the three selection types ST. Crossover type is $XT = URX$ with probability of crossover $p_X = 0.66$ and mutation type is $MT = SRM$ with probability of mutation $p_M = 0.33$. We denote the best selection scheme, ST^*. The average accuracies for the different types of selection $ELSEL$, $RWSEL$ and $TRSEL$ are $1.24682, 1.40579$ and 1.38396. Therefore we choose the elitist selection type of level 25% for further experiments, $ST^* = ELSEL$.

3) Crossover: Having the population size PS^*, the number of generations $NOGEN^*$ and the

selection type ST^* we compare now the different types of crossover (SPX, $LCPSPX$, TPX, $LCPTPX$ and URX). We ran the algorithm $kGA(t, k, s, PS^*, NOGEN^*, ST^*, XT, MT, p_X, p_M)$ for each of the crossover types XT with probability of crossover $p_X = 0.66$ and for mutation type $MT = SRM$ and probability of mutation $p_M = 0.33$. The average accuracies for the different types of crossover (SPX, $LCPSPX$, TPX, $LCPTPX$ and URX) are $1.26250, 1.19464, 1.32043, 1.19464$ and 1.24682. The difference between the accuracy of $LCPSPX$ and $LCPTPX$ is of the order 10^{-17}. Therefore we can choose either the one or the two point crossover which uses the linear complexity profile information ($XT^* = LCPSPX$ or $XT^* = LCPTPX$) for further experiments.

4) Mutation: Having the population size PS^*, the number of generations $NOGEN^*$, the selection type ST^* and the crossover type XT^* we finally compare the two proposed types of mutation (SRM and $LCPRM$). Since the accuracy is so close for crossover $LCPSPX$ and $LCPTPX$, we ran the algorithm $kGA(t, k, s, PS^*, NOGEN^*, ST^*, XT, MT, p_X, p_M)$ for all four combinations of each of the two crossover types with each of the two mutation types taking probability of crossover $p_X = 0.66$ and probability of mutation $p_M = 0.33$. The average accuracies for the combinations of crossover and mutation, $LCPSPX$ with SRM, $LCPSPX$ with $LCPRM$, $LCPTPX$ with SRM and $LCPTPX$ with $LCPRM$ are $1.19464, 1.25615, 1.19646$ and 1.27003 respectively. The test indicates that the best mutation scheme is $MT^* = SRM$.

Table I shows the approximate values of the 5-error linear complexity found by the GA algorithm applied to each of the test sequences $s^{(i)}$ ($i = 0, 1, \ldots, 9$) of length 32, when population size is 110, number of generations is 100, selection scheme is the elitist selection with a level of 25%, crossover is two point crossover using the linear complexity information with probability of crossover 66% and mutation is standard random mutation with probability of mutation 33%. The table displays the exact values of the 5-error linear complexity and the generation number when the GA has found the solution.

Studying the details of each generation we noticed that sometimes very fit solutions produced by crossover are turned into unfit solutions through mutation (since mutation keeps the mutated individual for the next population for the sake of variety). This suggests that the percent that we considered for our tests, 33%, is quite large. Whilst the mutation process is needed in order to prevent convergence to a potential local minimum, we consider future work to experiment different mutation probability values as well as different crossover probability values.

TABLE I

THE RESULTS OF $kGA(32, 5, s_i, 110, 100, ELSEL(25\%), LCPTPX, SRM, 0.66, 0.33)$ COMPARED TO THE EXACT VALUES

s	(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
Exact value	8	9	4	9	7	8	5	7	7	8
GA Value	8	9	8	9	9	10	5	7	9	9
Generation	42	99	73	86	13	10	61	1	69	4

V. CONCLUSIONS

We proposed a genetic algorithm for computing the k-error linear complexity of a sequence over a finite field. We implemented various techniques for each of the evolutionary operators and we investigated the best choice of parameters for the problem. From our preliminary experiments we conclude that the genetic algorithm approach is suitable to the problem and that a good scheme would use a medium sized population, an elitist type of selection with a level of 25%, two point random crossover which uses the linear complexity profile information with a probability of 0.66 and a standard random mutation with a probability of 0.33. With these choices, the algorithm outputs an approximative value of the k-error linear complexity which is on average only 19.5% higher than the exact value.

As future work we intend to experiment more strategies and more choices of parameters and also run experiments on longer sequences.

ACKNOWLEDGMENT

The authors would like to thank Dr. Chris Hinde for the useful discussions on the subject of this paper.

REFERENCES

[1] E. R. Berlekamp. *Algebraic Coding Theory*. McGraw-Hill, NY, 1968.
[2] Jason Cooper. *Improving Performance of Genetic Algorithms by Using Novel Fitness Functions*. PhD thesis, Loughborough University, 2004.
[3] C. Ding, G. Xiao, and W. Shan. *The Stability Theory of Stream Ciphers*. Springer-Verlag, Heidelberg, 1992.
[4] R. A. Games and A. H. Chan. A Fast Algorithm for Determining the Complexity of a Binary Sequence with Period 2^n. *IEEE Trans. Information Theory*, 29(1):144–146, 1983.
[5] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, USA, 1989.
[6] T. Kaida. On the Generalized Lauder-Paterson Algorithm and Profiles of the k-error linear complexity for Exponent Periodic Sequences. In *Proceedings of SETA 2004*, volume LNCS 3486, pages 166–178, Berlin, 2005. Spinger-Verlag.
[7] T. Kaida, S. Uehara, and K. Imamura. *An Algorithm for the k-error linear complexity of Sequences over $GF(p^m)$ with Period p^n, p a Prime*, volume 151 of *Information and Computation*, pages 134–147. Academic Press, 1999.
[8] A. G. B. Lauder and K. G. Paterson. Computing the Error Linear Complexity Spectrum of a Binary Sequence of Period 2^n. *IEEE Trans. Information Theory*, 49(1):273–2803, 2003.
[9] R. Lidl and H. Niederreiter. *Introduction to finite fields and their applications*. Cambridge University Press, 1994.
[10] J. L. Massey. Shift-Register Synthesis and BCH Decoding. *IEEE Trans. Information Theory*, 15(1):122–127, 1969.
[11] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, Berlin, 1999.
[12] Colin R. Reeves. Using genetic algorithms with small populations. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 92–99, San Francisco, CA, USA, 1993. Morgan Kaufmann Inc.
[13] M. Stamp and C. F. Martin. An Algorithm for the k-Error Linear Complexity of Binary Sequences with Period 2^n. *IEEE Trans. Information Theory*, 39(4):1398–1401, 1993.
[14] Gilbert Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 2–9, San Francisco, CA, USA, 1989. Morgan Kaufmann Inc.