Algorithms for the k-error linear complexity of cryptographic sequences over finite fields

by

Alexandra Alecu

A Doctoral Thesis

Submitted in partial fulfilment
of the requirements for the award of

Doctor of Philosophy
of
Loughborough University

19th December 2008

Linearity may be the curse of the cryptographer, but it is also his best guide to perfect secrecy.

James L. Massey, Cryptography and System Theory, 1986

Contents

List of Tables

List of Figures

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. The security of the sequences is translated into several measurable characteristics. For example they should have a large linear complexity and also a large k-error linear complexity.

This thesis focuses on the k-error linear complexity of sequences. Currently, efficient algorithms for computing the k-error linear complexity of a sequence only exist for special classes of sequences, e.g. 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.

Firstly we present a general heuristic algorithm which approximates the k-error linear complexity of sequences by taking advantage of the incremental nature of the Berlekamp-Massey Algorithm. Secondly, we investigate the application of evolutionary techniques for the approximation of the k-error linear complexity. While the complexity of these heuristic algorithms is still exponential, they are consistently more efficient than the exhaustive search and they are working on general sequences over arbitrary finite fields. The accuracy of the results of the algorithms is experimentally analysed.

Finally, we investigate using the Discrete Fourier Transform and Blahut's Theorem for calculating the k-error linear complexity of sequences. We present a new concept, a natural extension of the k-error linear complexity, denoted the extension field k-error linear complexity and devise algorithms to compute it.

While the problem of computing the k-error linear complexity remains open, a collection of algorithms to use in different situations is provided and the approximate results obtained can be useful in the design stage of the cryptographic sequences in order to quickly eliminate the insecure ones.

Notation

These are some of the most important notations used throughout this thesis.

$s = s_0 s_1 \dots s_{t-1} \dots$	an infinite sequence of size t
$s = (s_0 s_1 \dots s_{N-1})$	an infinite periodic sequence
$s = s_0 s_1 \dots s_{t-1}$	finite sequence
t	the size of the finite sequence s
k	the number of errors
N	the period of the infinite periodic sequence s
$w_H(s)$	Hamming weight of a sequence s
$C(X)$	the characteristic polynomial
$GF(p^m)$	a finite field of characteristic p and order p^m, where p is prime and $m \geq 1$
K	the base finite field
F	the extension field of K which contains an N-th root of unity, where N is the period of the sequence s
$L(s)$	the linear complexity of s
$L_k(s)$	the k-error linear complexity of s
$L^{(n)}(s)$	the linear complexity of the initial segment of size n for the sequence s
$L_k^{(n)}(s)$	the k-error linear complexity of the initial segment of size n for the sequence s
$EL_{k,N}(s)$	the extension field k-error linear complexity of the infinite periodic sequence s of period N

Abbreviations

BMA Berlekamp-Massey Algorithm

DFT Discrete Fourier Transform

EESA Efficient Exhaustive Search Algorithm

FFT Fast Fourier Transform

FSM Finite State Machine

GAP Groups, Algebra and Programming

kGA k-Error Genetic Algorithm

kSAA k-Error Simulated Annealing Algorithm

LFSR Linear Feedback Shift Register

MBMA Modified Berlekamp-Massey Algorithm

NESA Naïve Exhaustive Search Algorithm

PRNG Pseudo-Random Number Generator

Chapter 1

Introduction

1.1 Motivation

In this thesis we study the k-error linear complexity of sequences over finite fields. The k-error linear complexity 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 that generates a sequence which differs from the original in at most k positions. Formal definitions of these concepts are included in section 2.2.

Our aim is to investigate theoretical results and algorithms for computing and/or approximating the k-error linear complexity of sequences (see the definition of the problem in section 3).

We motivate this study by considering the following application from cryptographic sequence design and cryptanalysis.

When designing a stream cipher, the keystream sequence has to have a large linear complexity. The reason is that using the Berlekamp-Massey Algorithm (Berlekamp [4], Massey [42]), 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.

From a design point of view it is also useful to predetermine the level of security of the keystream sequence which is used in a cipher in order to prevent successful cryptographic attacks.

It is therefore important to have tools to evaluate the k-error linear complexity for cryptographic sequences.

1.2 State of the art

Efficient exact algorithms to compute the k-error linear complexity exist for certain classes of sequences, e.g. periodic sequences over a finite field $GF(p^m)$ and with period of a certain form, namely equal to a power of the characteristic of the field p, p being prime and $m \geq 1$ (see Stamp and Martin [79], Lauder and Paterson [38] for $p = 2$ and Kaida, Uehara and Imamura [31] for an arbitrary p). These results are based on the algorithms of Games and Chan [17] and Ding, Xiao, Shan [12] for computing the linear complexity of such sequences.

These algorithms have as input a full period of the sequence, i.e. the whole sequence should be known apriori, they are therefore useful mostly in the design stage for cryptographic sequences and not so much in cryptanalysis applications.

1.3 Contribution

We firstly show some heuristic methods for approximating the k-error linear complexity for finite sequences over finite fields. The advantage of these algorithms is that they work on arbitrary sequences and even if they only approximate the exact result, the approximation is accurate and the computational time complexity is at a manageable level. Therefore these algorithms can be a useful tool to quickly eliminate unsecure cryptographic sequences. We implemented a few heuristic algorithms. The first uses a recursive version of the Berlekamp-Massey Algorithm (Chapter 4). Further we present and analyse two which are using genetic algorithms and simulated annealing techniques, respectively (sections 5.1 and 5.2). We investigate and estimate the efficiency and accuracy of these methods and we compare these techniques showing advantages and disadvantages. The accuracy of the approximation is compared to the exact result returned by an efficient version of the exhaustive search algorithm, which we present in Chapter 3.

Secondly, we show how to use Blahut's Theorem (Rueppel [70]) which relates the Discrete Fourier Transform of periodic sequences to their linear complexity, for approximating the k-error linear complexity. This work has led us to a new concept, the extension field k-error linear complexity, which is useful from a cryptanalysis point of view. We motivate the definition and the applications of this concept and we create an algorithm to approximate its value (Chapter 6).

Parts of the research findings in this thesis are joint work with Ana Sălăgean and have been (or will be) published by Springer-Verlag in Lecture Notes in Computer Science (LNCS) series [2] and by IEEE Computer Society Press, [1] and [3]. The content of [2] forms the basis of chapter 4 and the content of [1] is extended in chapter 5. In addition, the notion and algorithm presented in [3] are extended

in chapter 6.

1.4 Thesis structure

This thesis details our research outcomes on *Algorithms for the k-error linear complexity of cryptographic sequences over finite fields.*

We start with a brief introduction, a motivation of our research and highlight the main contributions of the thesis (Chapter 1).

Chapter 2 sets out the general and mathematical background for the problem. Also, for each of the concepts introduced it includes a short review of the current known results which are relevant to this thesis.

In chapter 3 we clearly define the problems that we are interested in, when designing the algorithms presented in this thesis. The chapter includes a detailed description and analysis of an exact and general method of calculating the k-error linear complexity of sequences over finite fields, the exhaustive search. We introduce and analyse a more efficient version of the exhaustive search (the Efficient Exhaustive Search Algorithm), the exact results produced by this algorithm being used throughout the thesis as a reference for the efficiency and accuracy of the results of the proposed heuristic algorithms.

In chapter 4, we propose adapting the Berlekamp-Massey Algorithm (see Berlekamp [4], Massey [42]) which computes the linear complexity, in order to approximate the k-error linear complexity profile for a finite sequence over an arbitrary finite field. The main idea in a heuristic algorithm is to explore only some of all the possible error sequences. The choice of the positions of the errors in this case is guided by the steps of the Berlekamp-Massey Algorithm in which the complexity increases.

Chapter 5 presents two evolutionary techniques applied to the problem of computing the k-error linear complexity of sequences, genetic algorithms (Chapter 5.1) and simulated annealing (Chapter 5.2). We are focusing on finding best choices for the parameters involved in each technique (e.g. population size, number of generations, technique of selection, crossover or mutation for genetic algorithms or cooling schedule and evaluation function for simulated annealing) such that the resulting approximation is accurate and the computational time stays polynomial.

Chapter 6 of the thesis includes techniques for infinite periodic sequences and shows how to use the Discrete Fourier Transform and Blahut's Theorem to calculate the k-error linear complexity. A new concept is introduced, the extension field k-error linear complexity, $EL_{k,N}(s)$, and algorithms to approximate it are presented and experimentally analysed.

We conclude and suggest some possible developments of our work in Chapter 7.

Chapter 2

Background

2.1 General Background

The wide area of cryptology includes two main branches: cryptography on one side, e.g. methods for securing the communication over an insecure channel, between a sender (usually named Alice) and a receiver (Bob) (see figure 2.1), and cryptanalysis on the other side, e.g. methods for secretly eavesdropping or interfering in a transmission between a sender and a receiver.

The message before encryption is called plaintext and the encrypted message is called ciphertext.

A system designed for the secure communication between a sender and a receiver can be formally described by the notion of cryptosystem (cryptographic system) or cipher. The encryption/decryption is the controlled modification of the plaintext/ciphertext using one or more characters, called key. Cryptosystems are usually classified in secret-key (symmetric) cryptosystems (see figure 2.2) or public-key (asymmetric) cryptosystems.

The public-key cipher allows the sender to use publicly known information in order to encrypt and send a message to the receiver, such that only the latter can decrypt it. On the other hand, the secret-key ciphers require the secure transmission of a secret key in advance, which is only known by the sender and receiver (see figure 2.2). A sequence of values used as key is called a key stream. This thesis only deals with symmetric cryptosystems, therefore in the following

Figure 2.1: Cryptography

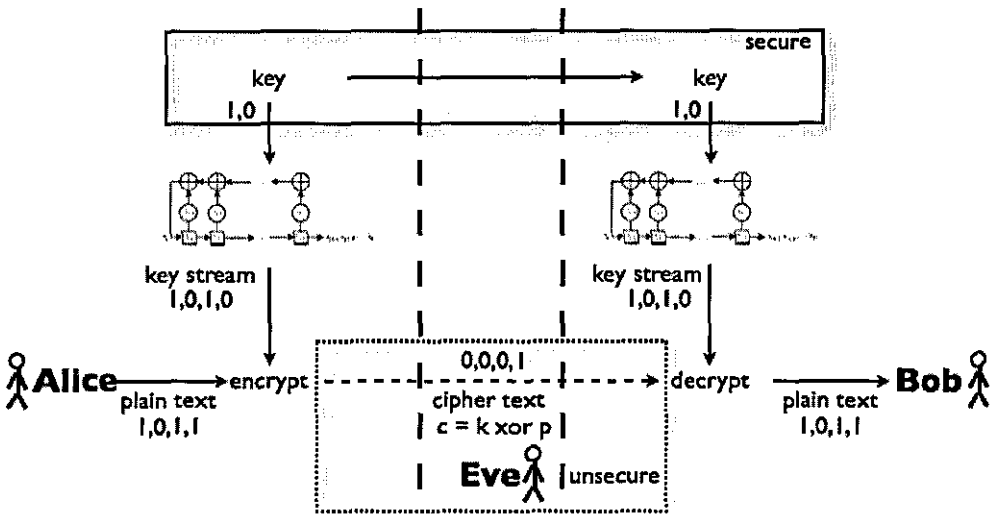

Figure 2.2: A symmetric cryptosystem

we will refer to secret-key cryptosystems simply as cryptosystems.

The cryptanalysis of a cryptosystem usually relies on the nature of the encryption/decryption algorithm and some knowledge of the nature of the plaintext. Depending on the level of involvement of the intruder in the encryption system, the attacks can be passive or active attacks.

The starting point for any type of cryptanalysis is the encryption algorithm and a portion of intercepted ciphertext. Depending on the amount of additional information known by the cryptanalyst, there are five types of attacks: ciphertext only, known plaintext, chosen plaintext, chosen ciphertext and chosen text. They all assume that at least the encryption algorithm and the ciphertext to decode are known. Generally, a cipher is considered secure if it is designed to resist a known plaintext attack. However, there are other types of security, for example (Menezes et al. [52]):

- A cipher is *unconditionally secure* if the ciphertext generated using that cipher is completely independent from the corresponding plaintext, i.e. it is equally probable for any plaintext to be encrypted to obtain that ciphertext.
- A cipher is *computationally secure* if the computational cost of breaking the cipher exceeds the importance of the encrypted information and if the time required to break the cipher exceeds the useful lifetime of the information.

Definition 2.1. *(Stinson [80]) A symmetric cryptosystem is a 5-tuple* $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ *where the following conditions are met:*

- *$\mathcal{P}$ is a finite set of plaintext messages (text before encryption),* $\mathcal{P} = \{p | p = p_1 p_2 \dots p_m, m \geq 1, p_i \in A, i = 1, 2, \dots, m\}$, *where A is a finite alphabet for the plaintext messages.*

- *$\mathcal{C}$ is a finite set of ciphertext messages (encrypted text),* $\mathcal{C} = \{c | c = c_1 c_2 \dots c_n, n \geq 1, c_i \in B, i = 1, 2, \dots, n\}$, *where B is a finite alphabet for the ciphertext messages.*

- *$\mathcal{K}$ is a finite set of keys over a finite alphabet K.*

- $(\forall) k \in \mathcal{K}$, *$(\exists)$ an encryption function $e_k \in \mathcal{E}$, $e_k : \mathcal{P} \rightarrow \mathcal{C}$ and a corresponding decryption function $d_k \in \mathcal{D}$, $d_k : \mathcal{C} \rightarrow \mathcal{P}$ such that* $d_k(e_k(p)) = p$, $(\forall) p \in \mathcal{P}$.

Symmetric cryptosystems can be classified further in block ciphers and stream ciphers. Essentially, the difference between the two is that first processes blocks of characters from the plaintext at a time whereas the second processes one character at a time. Here we are only interested in stream ciphers which are devices with internal memory, encrypting one character at a time by combining it with a character from the secret key stream. The j^{th} character of the plaintext, p_j, is enciphered into the j^{th} character of the ciphertext, c_j. Due to the fact that in a known-plaintext attack scenario, the stream cipher is fully characterised by the key stream employed, a stream cipher is considered to be secure if knowledge of a small number of subsequent bits of the key stream is not sufficient to recover the entire key stream (e.g. Rueppel [70], Robshaw [67]).

An effort to standardise the stream ciphers has been undertaken by a recent *eCRYPT* project called *eSTREAM* [13]. The project has resulted into a portfolio of stream ciphers which are advisable to use: HC-128, Rabbit, Salsa20/12, SOSEMANUK, Grain v1, Mickey v2 and Trivium.

Stream ciphers are widely used, especially when it is necessary to encrypt large amounts of data very quickly. The main advantages of stream ciphers are that they are fast, easy to implement in hardware and appropriate for limited buffering conditions. They allow limited error propagation, detection of active attacks and a good diffusion of plaintext statistics. Also their definition is straightforward and well fundamented from a mathematical point of view.

One of the most famous stream ciphers is the one-time pad. The one-time pad cipher was derived from the Vernam cipher devised in 1926 (Vernam [82]) and, as proved mathematically by Shannon [75] using information theory methods, providing the key is truly random, never reused and kept secret, it constitutes the only unconditionally secure cipher. However, the one time pad has a low practical value since it is very expensive to implement, firstly because it needs

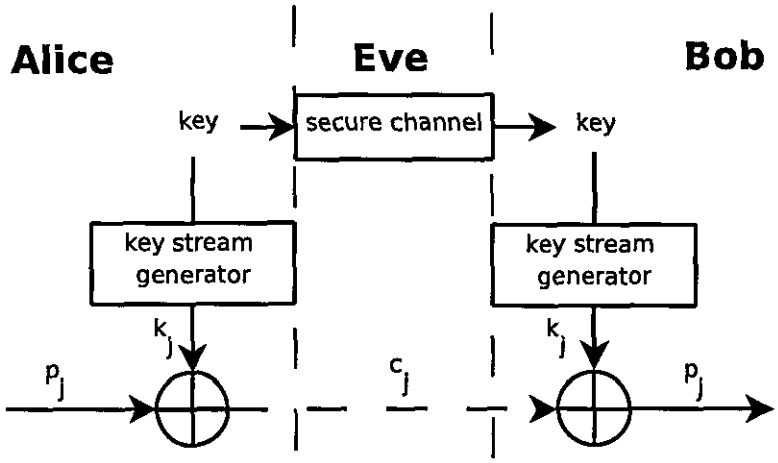

Figure 2.3: Stream cipher system based on addition.

the generation of a truly random key stream at least as long as the plaintext and secondly, because it requires that this long key stream to be transmitted securely to the receiver.

It is expensive and unsecure to exchange large key streams between the sender and the receiver. This is why a key stream generator is usually employed which expands a short, truly random key k into a long pseudo-random sequence. Pseudo-random sequences have statistical randomness properties while being generated by an entirely deterministic causal process. Apart from allowing for better analysis, the determinism is particularly useful in cryptographic applications since it allows the sender and the receiver to (re)generate the same key to use for encryption/decryption.

Usually, in stream ciphers the plain text bits are encrypted one at a time by adding (XOR in binary terminology) them with a bit from the secret key stream. The simplest and most widely used stream cipher is the binary cipher based on addition (see figure 2.3). The encryption consists of adding (XOR) the key value to the plaintext character $(c_j = k_j \oplus p_j)$ and the decryption consists of adding (XOR) the key to the ciphertext character $(p_j = k_j \oplus c_j)$.

One way to describe a key stream generator is to use a Finite State Machine (FSM) with output (Ding, Xiao, Shan [12]). However a widely used mechanism of generating a key stream is the Linear Feedback Shift Register (LFSR), which is a special case of an FSM. It provides a simple way of generating an infinite (eventually periodic) sequence of terms over a field, the sequence having a non-trivial structure.

A LFSR can be implemented in software or hardware and it can generate sequences with 'good' statistical properties, therefore it is commonly used as a

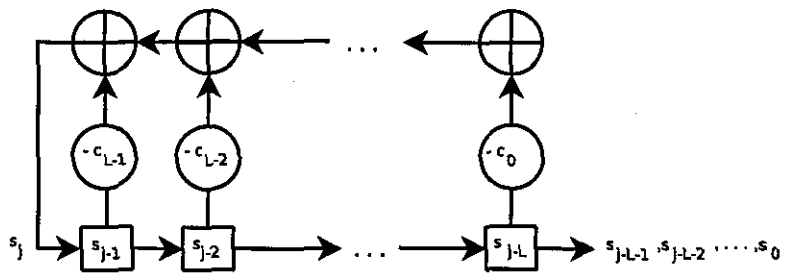

Figure 2.4: A general L-stage Linear Feedback Shift Register.

Pseudo-Random Number Generator (PRNG).

A general LFSR of length L consists of a cascade of L unit delays[1] or stages, linked (using constant adders[2], constant multipliers [3] and wires) so that they allow the computation of a linear combination of cell contents whose value then serves as the input back to the first stage (see figure 2.4). The output of the LFSR is taken from the rightmost unit. The initial content $s_0, s_1, \ldots, s_{L-1}$ of the L unit cells is called the seed, initial load or key and it coincides with the first L output digits (Lidl and Niederreiter [39]). The remaining output digits are uniquely determined by the following linear recurrence relation:

$$s_j = -\sum_{i=0}^{L-1} c_i s_{i+j-L}, \text{ for all } j = L, L+1, \ldots \tag{2.1}$$

The output terms and the feedback coefficients $c_0, c_1, \ldots, c_{L-1}$ lie in the same field as the initial terms and the generated sequence. The sequence generated by a LFSR or by a linear recurrence (2.1) is called linear recurrent. If the initial values are over a finite field K of order q, then the sequence is periodic with period of at most $q^L - 1$, where L is the number of stages in the LFSR (see property 2.5 in section 2.2 for more details). A sequence generated by a Linear Feedback Shift Register of size L and with terms in a finite field of order q is called a maximum length(ML) sequence if its smallest period takes the maximum value $q^L - 1$.

The function defined by the recurrence relation is called the feedback function

[1] An *unit delay* is an electronic device which has one input, one output and is regulated by an external clock so that its input as a particular time appears as its output one time unit later.

[2] A *constant adder* is a special kind of electronic switching circuit which has two inputs and one output, the output being the sum (over the appropriate field) of the two inputs.

[3] A *constant multiplier* is an electronic device with an internal value, an input and an output, the output being the product of the input value and the internal value (over the appropriate field).

of the LFSR.

$$f(s_{j-L}, s_{j-L-1}, \ldots, s_{j-1}) = -\sum_{i=0}^{L-1} c_i s_{i+j-L}, \text{ for all } j = L, L+1, \ldots \quad (2.2)$$

The stages which participate in the feedback are called *taps* and the list of taps is known as the *tap sequence.*

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 pseudo-random sequence. However, any recurrent sequence over a finite field is linearly recurrent and can therefore be generated by one single (usually much larger) LFSR.

The effort of building secure stream ciphers equates to generating a secure key stream. The following is a list of the main properties mentioned in the literature which support and characterise the psuedo-randomness of a sequence (see Golomb [22], Rueppel [77], Ding, Xiao and Shan [12], Menezes et al. [52], Stallings [78], Stinson [80]).

1. The sequence needs to be *balanced*. For a sequence over an arbitrary finite field $GF(q)$ this translates to the fact that the probability of a term of the sequence to be equal to any value $a \in GF(q)$ is constant (not depending on a). In particular, for binary sequences balanced means that the number of ones equals the number of zeros in the sequence. (Golomb [22], Stallings [78])

2. The key stream must have a 'large' *period*, since one full period defines the whole sequence. A linear recurrence for a sequence with period n is simply $s_{i+n} = s_i, (\forall) i = 0, 1, \ldots$, so if n is small it is easy to recover the whole sequence by only intercepting n terms. (Ding et al. [12], Rueppel [77], Stallings [78], Menezes et al. [52])

3. The key stream terms need to appear as being drawn from an *uniform distribution*; the key stream should have uniform statistics, i.e., an equal distribution of single bits, of pairs, triplets of bits, etc. (Golomb [22], Ding et al. [12], Rueppel [77], Menezes et al. [52])

4. For binary sequences, the sequence needs to have a two levelled auto-correlation function. The auto-correlation function of a binary sequence s is defined as

$$C(\theta) = \lim_{N \to \infty} \frac{1}{N} \sum_{n=1}^{N} s_n s_{n+\theta},$$

where the binary sequence is considered as a string of 1s and -1s, rather then

1s and 0s and θ is an integer called the phase shift. For random sequences $C(\theta)$ is very close to 0 for $\theta \neq 0$ and a constant, high value for $\theta = 0$. (Golomb [22, Chapter 3])

5. The *linear complexity* of the sequence (i.e. the size of the shortest LFSR which generates that sequence) needs to be large. The reason is that there is an efficient algorithm (Berlekamp-Massey Algorithm) for finding the shortest LFSR corresponding to a sequence of linear complexity L, having as input only $2L$ consecutive terms of the sequence. (Ding et al. [12], Rueppel [77], Menezes et al. [52])

6. The *k-error linear complexity* of the sequence (i.e. the size of the shortest LFSR which generates the sequence in which at most k bits are changed in each period) needs to be large enough for all relatively small k. (Ding et al. [12], Stamp and Martin [79])

7. The sequence needs to meet the principle of *confusion*, every key stream bit must be a complex transformation of all or most of the key bits. (Rueppel [77])

8. The sequence needs to meet the principle of *diffusion*, i.e. redundancies in segments of the sequence must be dissipated on the whole length of the sequence. For example, if the sequence is generated using a combination of LFSRs, it is better to interleave characters produced by different LFSRs than to concatenate the keystreams generated by each LFSR. (Rueppel [77])

9. For sequences generated by a non-linear filtered shift register, the boolean function used to filter the output of the shift register needs to be 'highly' nonlinear, taking into account properties like the m^{th}-order correlation immunity (Siegenthaler [76]), the distance to linear functions, the avalanche criterion etc. (Rueppel [77], Ding et al. [12])

2.2 Mathematical Background

In the following we include the definitions and some properties of the main concepts used throughout the thesis. We will consider infinite or finite sequences usually denoted as s. The sequence terms lie in a finite field $GF(q), q = p^m$, where p is a prime number and $m \geq 1$. Some of the basic algebraic concepts are briefly summarised in Appendix A.

More details about linear recurrent sequences and terminology can be found for example in Lidl and Niederreiter [39].

2.2.1 Linear recurrences

Definition 2.2. *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*

$$s_j + c_{L-1}s_{j-1} + \ldots + c_1 s_{j-L+1} + c_0 s_{j-L} = 0 \tag{2.3}$$

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 (2.3) is called a homogeneous linear recurrence relation of order *L. A recurrence relation of minimal order is called a* minimal recurrence relation.

If a finite or infinite sequence s satisfies one linear recurrence relation then it satisfies an infinite number of linear recurrence relations. However, for an infinite sequence the minimal recurrence relation is unique, whereas for finite sequences, a minimal recurrence relation has a fixed order, but it is not necessarily unique.

Knowledge of a recurrence relation of order L for a sequence s and any L successive terms of that sequence is enough for generating all the terms of the sequence. Therefore, a finite or infinite linear recurrent sequence is fully specified by its characteristic recurrence relation of size L and by the L initial terms.

The mathematical foundations of linear recurrent sequences have been firstly set by Golomb [22]. Besides the characteristic polynomial (section 2.2.2) some other methods of formalising the linear recurrent sequences are proposed: the generating function and the matrix method (Lidl and Niederreiter [39]).

2.2.2 Characteristic polynomial

To any linear recurrence relation of a sequence, we can associate a polynomial whose coefficients are the feedback coefficients c_i for all $i = 0, 1, \ldots, L-1$. This polynomial is called a characteristic polynomial for that sequence and it can be defined in one of several ways. Two options are commonly used in literature:

$$C(X) = X^L + c_{L-1}X^{L-1} + c_{L-2}X^{L-2} + \ldots + c_1 X + c_0 \tag{2.4}$$

$$C(X) = X^L + c_0 X^{L-1} + c_1 X^{L-2} + \ldots + c_{L-2}X + c_{L-1} \tag{2.5}$$

We denote the characteristic polynomial associated to the minimal linear recurrence relation, the minimal characteristic polynomial.

We chose in this thesis the first definition (2.4) for mathematical reasons. Using this definition, it is well known that the minimal characteristic polynomial of a

sequence s is unique and that any other characteristic polynomial of that sequence is a multiple of the minimal characteristic polynomial.

Definition 2.3. *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 and a polynomial*

$$C(X) = X^L + c_{L-1}X^{L-1} + c_{L-2}X^{L-2} + \ldots + c_1X + c_0,$$

with $L \geq 0$ we say that $C(X)$ is a characteristic polynomial of s if the associated linear recurrence is satisfied by the sequence s

$$s_j + c_{L-1}s_{j-1} + \ldots + c_1s_{j-L+1} + c_0s_{j-L} = 0 \tag{2.6}$$

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.

If the linear recurrence is the minimal recurrence for the sequence, then the characteristic polynomial is called the minimal characteristic polynomial. In these conditions, we say that the polynomial $C(X)$ generates the sequence s.

A characteristic polynomial is also called in literature a connection polynomial (Ding, Xiao and Shan [12]) or a generator polynomial (Massey [42]).

For an infinite sequence the minimal characteristic polynomial is unique, whereas for finite sequences, the minimal characteristic polynomial has a fixed degree, but it is not necessarily unique.

Golomb [22] proves that the minimal characteristic polynomial of a maximum length (ML) sequence is primitive and therefore, the search for pseudo-random sequences can be reduced to finding primitive characteristic polynomials.

2.2.3 Periodic sequences

Definition 2.4. *A sequence $s = s_0, s_1, \ldots$ over a finite field is called* ultimately periodic *if there exists the integers $N > 0$ and $n_0 \geq 0$ such that $s_{n+N} = s_n$, for all $n \geq n_0$. The number N is called a* period *of the sequence and the smallest N with the previous property is called the* minimal period *of the sequence.*

Property 2.5. *For infinite sequences over finite fields the following three properties are equivalent:*

(i) ultimately periodic;

(ii) recurrent;

(iii) linear recurrent.

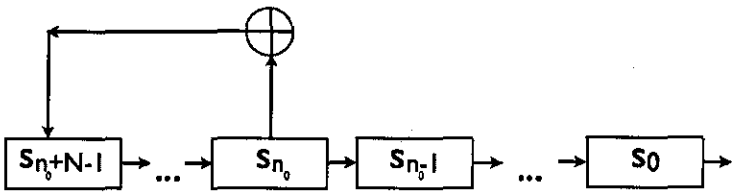

Figure 2.5: A LFSR for a ultimately periodic sequence.

Proof. $(i) \Rightarrow (ii), (iii)$ Suppose s is an ultimately periodic sequence. Therefore there is a linear recurrence relation of order $N + n_0$ for the sequence (see definition 2.4):

$$s_j = s_{j-N}, \text{ for all } j = n_0 + N, n_0 + N + 1, \ldots, \text{ where } N > 0, n_0 \geq 0.$$

The corresponding characteristic polynomial is $C(X) = X^{N+n_0} - X^{n_0}$ and the corresponding LFSR is as shown in figure 2.5. Therefore, the sequence is also recurrent and linear recurrent.

A linear recurrent sequence is obviously recurrent, therefore $(iii) \Rightarrow (ii)$ is also easily proved.

$(ii) \Rightarrow (i)$ Suppose the sequence s is recurrent. Therefore there is a value L so that $s_j = f(s_{j-L}, s_{j-L+1}, \ldots, s_{j-1})$, for all $j = L, L+1, \ldots$, where $f : (GF(q))^L \rightarrow GF(q)$ is an arbitrary function. There are q^L possible combinations of L terms from the considered finite field (where q is the order of the field).

The sequence is infinite and the field is finite, therefore there must be a segment of L values which repeats in the sequence at least once, after at most q^L steps (all possible combinations of L values from the finite field of order q). When a segment of size L repeats, it means that using the function f, the same sequence of terms will be generated. It follows that the sequence is ultimately periodic.

A similar justification, with the single change that the feedback function needs to be linear is valid for the implication $(iii) \Rightarrow (i)$. □

Proposition 2.6. *The minimal characteristic polynomial of any periodic infinite linear recurrent sequence over a finite field has a constant term. Moreover, the characteristic polynomial of a sequence with period N divides $X^N - 1$.*

Proof. Suppose s is an infinite periodic linear recurrent sequence and that the minimal period of the sequence is N. It follows that a linear recurrence relation for the sequence would be $s_{i+N} = s_i$, for all $i \geq 0$ which implies a characteristic polynomial of s would be $C(X) = X^N - 1$.

All characteristic polynomials of the sequence s are multiples of the minimal

characteristic polynomial. Therefore, if we denote $D(X)$ the minimal characteristic polynomial of s, $C(X) = X^N - 1 = D(X)h(X)$, for some $h(X) \in GF(q)[X]$, it follows that the minimal characteristic polynomial of s has a constant term and that it divides $X^N - 1$. □

Note that if the minimal characteristic polynomial does not have a constant term, it means that by taking a power of X as factor (suppose the maximal factor is X^{n_0}), a reduced characteristical polynomial can be found for the periodic part of the sequence. This defines the recurrence which generates the sequence starting with s_{n_0} and ignoring the first n_0 terms, $s_0, s_1, \ldots, s_{n_0-1}$ not participating in the recurrence (see the corresponding LFSR in figure 2.5 for a graphical representation).

Therefore, if the sequence is ultimately periodic, by removing the initial segment which does not participate into the recurrence, a characteristic polynomial with a constant term can be obtained for the periodic part of the sequence. This is an important remark showing that in applications the initial non periodic part of sequences can be ignored.

This is why in most cryptographic applications the key streams can be considered periodic; even more so for cryptanalysis since the attacks do not usually involve intercepting the first few terms of the key stream. Sălăgean [74] presents an algorithm which finds the minimal characteristic polynomial with a non zero constant term for a sequence.

The interpretation of proposition 2.6 is that finding the linear recurrence which generates the periodical part of the sequence would be enough to break the cipher, therefore, the sequences should be designed to have both a characteristic polynomial of large degree and with a constant term; only one of these conditions is not sufficient.

2.2.4 Linear Feedback Shift Registers

To any linear recurrence relation we can assign a Linear Feedback Shift Register (see figure 2.4 in section 2.1).

We say that a LFSR generates a finite sequence $s_0, s_1, \ldots, s_{t-1}$ if the sequence coincides with the first t output digits of the LFSR for some initial loading of the LFSR. A sequence generated by a LFSR can be defined by a linear recurrence relation or, equivalently, by a characteristic polynomial. If the size of a LFSR is L with $L \geq t$, then the LFSR can always generate the sequence by including the sequence terms in the initial loading. If $L < t$, then the LFSR generates a sequence s if and only if the linear recurrence relation corresponding to the LFSR is a recurrence relation for the given sequence (see relation (2.3)).

The feedback coefficients of a linear recurrence relation can be zero. If the ones corresponding to the unit cells next to the output of the corresponding LFSR are zero then the values included in these unit cells do not get included in the recurrence and the first terms of the generated sequence do not participate in the generation of the following terms of the sequence. Further, note that a LFSR with L stages and all feedback coefficients equal to zero generates the sequence: $s_0, s_1, \ldots, s_{L-1}, 0, 0, \ldots$.

By convention, the all-zero sequence is generated by a LFSR with length $L = 0$. The linear recurrence relation corresponding to the LFSR of length $L = 0$ $0 \cdot s_j = 0$, for all $j = 0, 1, \ldots$ and the characteristic polynomial is $C(X) = 1$.

We will assume in the following that at least one of the feedback coefficients is non zero.

2.2.5 Linear complexity

Definition 2.7. *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, the linear complexity of s, denoted $L(s)$ can be equivalently defined as:*

(i) the order of the minimal linear recurrence of s

(ii) the degree of the minimal characteristic polynomial of s

(iii) the size of the smallest Linear Feedback Shift Register which generates s.

The linear complexity of a finite or infinite sequence is unique. Note that $L(s) = 0$ if and only if the terms of the sequence are all zero.

If a sequence has linear complexity L, the minimal linear recurrence relation, the minimal characteristic polynomial or the Linear Feedback Shift Register that generates that sequence can be determined knowing only $2L$ consecutive terms.

Determining the linear complexity and the minimal characteristic polynomial of a finite sequence can be done by solving the system of linear equations obtained by writing the recurrence relation (2.3) for all $j = L, L+1, \ldots, 2L-1$. Commonly though, the efficient and intuitive method for computing the linear complexity of a sequence is by using the Berlekamp-Massey Algorithm (see section 2.2.6).

Property 2.8. *For an infinite sequence $s = s_0, s_1, \ldots$ (or a finite sequence $s = s_0, s_1, \ldots, s_{t-1}$) and for t_1, t_2 with $0 \leq t_1 \leq t_2 < n$, $L(s_0, s_1, \ldots, s_{t_1-1}) \leq L(s_0, s_1, \ldots, s_{t_2-1})$, i.e. $L(s)$ is increasing with the number of terms for a fixed s.*

Property 2.9. *For an infinite sequence $s = s_0, s_1, \ldots$ (or a finite sequence $s = s_0, s_1, \ldots, s_{t-1}$) $L(s_0, s_1, \ldots, s_{t-1}) \leq t$, for all $t \geq 0$ (or for all $t = 0, 1, \ldots, t-1$, respectively).*

Definition 2.10. *Suppose s is an infinite sequence $s = s_0, s_1, \ldots$ (or a finite sequence $s = s_0, s_1, \ldots, s_{t-1}$). The set $\{L(s_0, s_1, \ldots, s_{t-1}) | \text{ for all } t \geq 0\}$ (or $\{L(s_0, s_1, \ldots, s_{n-1}) \mid \text{for all } 1 \leq n \leq t\}$, respectively) is called the* linear complexity profile *of the sequence s.*

The first paper which outlines the notion of linear complexity of a sequence is Massey [42] which defines it as the size of the shortest Linear Feedback Shift Register which generates the sequence.

The notions of linear complexity and linear complexity profile were closely analysed by Rueppel [70, 69]. In an effort to see the relation between the value of the linear complexity and the randomness of a sequence, Rueppel shows that for random binary sequences which are independent and uniformly distributed the expected value of the linear complexity is approximately $\frac{t}{2}$ for a sequence of size t, where the linear complexity is seen as a random variable. The variance of the linear complexity is very close to a constant which depends on the length of the sequence.

Rueppel claims that a good random sequence generator should have linear complexity close to the period length, and also the linear complexity profile should follow closely but irregularly the $\frac{t}{2}$ line exhibiting therefore average step lengths and heights of 4 and 2, respectively. Following this observation, a sequence with a perfect linear complexity profile is defined by Rueppel as follows.

Definition 2.11. *A sequence s of length n is said to have a perfect linear complexity profile if $L(s_0, s_1, \ldots, s_m) = \lfloor \frac{m+1}{2} \rfloor$, for all $m = 1, 2, \ldots, n$, where $\lfloor x \rfloor$ denotes the largest integer not greater than x.*

While giving this general rule, Rueppel conjectures that there are sequences of length t which are highly non random and do however have a linear complexity profile which follows closely $\frac{t}{2}$. For example the sequence given by (2.7) meets this description. The conjecture was proven and it was shown that the sequence of size n defined by (2.7) has linear complexity $\lfloor (n+1)/2 \rfloor$ (Dai [11]) as well as a perfect linear complexity profile as defined by Rueppel. Even with a high linear complexity and with a good linear complexity profile, the sequence is very sparse and not random.

$$s_i = \begin{cases} 1, & \text{when } i \text{ has the form } i = 2^t - 1, \text{ and } t = 0, 1, 2, \ldots \\ 0, & \textit{otherwise.} \end{cases} \tag{2.7}$$

A full characterisation of the sequences with a perfect linear complexity profile can be obtained using Berlekamp-Massey Algorithm (Wang and Massey [83]). It

can be shown that all the sequences with a perfect linear complexity profile can be obtained by solving a system of linear equations, see (2.8).

$$\begin{cases} s_0 = 1 \\ s_2 = s_1 + s_0 \\ \cdots \\ s_{2i} = s_{2i-1} + s_{i-1} \\ \cdots \\ s_{2(n-1)} = s_{2(n-1)-1} + s_{n-2} \end{cases} \tag{2.8}$$

Niederreiter [55] establishes the connection between the linear complexity of a sequence and the continued fraction expansion of the generating function corresponding to that sequence.

Taking into account the full characterisation of the sequences with a perfect linear complexity profile (see system (2.8), Wang and Massey [83]), Niederreiter argues that this condition is too restrictive to be useful in building pseudo-random sequences and that the linear complexity should have larger deviations from the expected value, i.e. from $\frac{t}{2}$. This is why he introduces the notion of an 'almost perfect linear complexity profile' using a remark made by Rueppel [70] concerning the variance of the linear complexity, namely the fact that the linear complexity needs to follow closely but irregularly $\frac{t}{2}$. Also, Niederreiter [55] suggests a way of building sequences with a prescribed linear complexity profile.

Extending the results from [55], Niederreiter [56] develops a probabilistic theory of the linear complexity and the linear complexity profile for sequences over arbitrary finite fields, by using techniques from the probability theory and the theory of dynamical systems. He suggests a new type of randomness tests for sequences, the continued fraction tests. The main result of the paper is that the following relation holds for *almost* all random sequences s over a finite field.

$$\lim_{t\to\infty} \frac{L(s)}{t} = \frac{1}{2} \tag{2.9}$$

Interpreting (2.9), Niederreiter [57] defines a sequence to have a 'good linear complexity profile' if there exists a constant value $C(s)$, which depends only on the sequence, such that

$$|L(s) - \frac{t}{2}| \leq C(s) \max\{\log t, 1\}, \text{ for all } t = 0, 1, 2, \ldots. \tag{2.10}$$

Following a comment made by Piper [61], that a pseudo-random sequence should have an acceptable linear complexity profile for every starting point, Niederreiter defines the 'uniformly good linear complexity profile' as the property of a

sequence to have a good linear complexity profile for all its shifted versions. He also proves that almost none of the random sequences meet this requirement and leaves the open problem of finding sequences which have this property.

Massey [41] shows that the linear complexity is a system-theoretic concept, closely related to the Discrete Fourier Transform. Although it was firstly used by Blahut [7, 6] for error control codes, the explicit form of the link between the linear complexity and the Discrete Fourier Transform is clearly presented by Massey in [41].

A way of obtaining sequences with high linear complexity is by combining a number of LFSRs (combination) or by applying a non-linear filter function to a single LFSR (filtering). The linear complexity of a sequence produced by a combination of LFSRs can never exceed the product of the sizes of the LFSRs included in the combination.

Groth [23] and Key [32] present some theoretical methods of combining LFSRs with known characteristics in order to obtain generators of binary sequences with controllable linear complexity.

Herlestam [25] presents an overview of the results on the linear complexity of sequences obtained using filtering functions applied to sequences generated by an Linear Feedback Shift Register. By studying the linear complexity of the sum, the product with a constant, the Hadamard product and the Hadamard power[4] applied to linear shift register sequences, exact formulas and lower bounds on the linear complexity of sequences generated using a general function applied to a finite set of linear recurrent sequences are obtained.

Using the arithmetic properties of the period of the sequences, Rueppel and Staffelbach [71] obtain a set of conditions which guarantee that a product of linear recurrent sequences attain maximum linear complexity. Precisely, they show that for a finite number of maximum length sequences over a finite field $GF(q)$, if the degrees of the corresponding minimal polynomials are distinct and greater than two, then their product has maximum linear complexity. The result can be generalised for arbitrary linear combinations of sequences (Golic [21]).

There are efficient algorithms presented by Games and Chan [17], Ding, Xiao, Shan [12] for computing the linear complexity of sequences of period a power of the characteristic of the field.

The modulo p linear complexity is the linear complexity of a binary sequence when it is considered over a prime finite field of higher characteristic p, where p is

[4]Note that for two finite sequences $x = x_0, x_1, \ldots, x_{t-1}$ and $y = y_0, y_1, \ldots, y_{t-1}$ over a field K, the Hadamard product xy is $xy = x_0y_0, x_1y_1, \ldots, x_{t-1}y_{t-1}$. For a sequence $x = x_0, x_1, \ldots, x_{t-1}$ over a field K and a constant integer $a \geq 0$ the Hadamard power x^a is $x^a = x_0^a, x_1^a, \ldots, x_{t-1}^a$.

prime. Klapper [34] shows that some binary sequences of high linear complexity could have low modulo p linear complexity and for a certain class of sequences, geometric sequences, he shows how to choose the value of p. An algorithm for computing the modulo p linear complexity of sequences is presented by Boztas [8].

Robshaw [68] presents an algorithm for binary sequences of period a power of two which is very efficient on sequences with high linear complexity, involving on average the computation of only two parity checks in such a case.

Generalising both the Discrete Fourier Transform and the Games-Chan Algorithm [17], Blackburn [5] presents an algorithm which calculates the minimal characteristic polynomial of an arbitrary periodic sequence. The computational complexity of the algorithm is asymptotically equal to the complexity of the Berlekamp-Massey Algorithm [42], however the Blackburn Algorithm can perform faster on sequences with a large complexity.

Fleischmann [16] modifies the Berlekamp-Massey Algorithm [42] to work in both directions by creating an algorithm suitable for real time applications where the sequence provided as input is not fully known apriori, for example $\ldots, s_{-1}, s_0, s_1, \ldots$.

2.2.6 Berlekamp-Massey Algorithm

The Berlekamp-Massey Algorithm (Berlekamp [4], Massey [42]) 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, $\mathcal{O}(L^2)$, where L is the linear complexity of the input sequence (for more details on computational complexity see Gustavson [24]).

The algorithm takes iteratively 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 $C^{(n)}(X)$ can generate the n sequence terms $s_0, s_1, \ldots, s_{n-1}$ processed so far. After all terms are processed, a minimal characteristic polynomial of the input sequence is obtained. The linear complexity is the degree of the resulting characteristic polynomial.

At each step n, 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 $L^{(i)} = \deg(C^{(i)}(X))$ and $C^{(n)}(X) = X^{L^{(n)}} + c^{(n)}_{L^{(n)}-1} X^{L^{(n)}-1} + \ldots + c^{(n)}_0$. The discrepancy $d^{(n)}$ is calculated using the

following formula:

$$d^{(n)} = s_n + \sum_{i=0}^{L^{(n)}-1} c_i^{(n)} s_{i+n-L^{(n)}} \tag{2.11}$$

This discrepancy represents the difference between the term s_n which is currently processed and the term which would be expected using the current polynomial $(-\sum_{i=0}^{L^{(n)}-1} c_i^{(n)} s_{i+n-L^{(n)}})$. Depending on the value of the discrepancy, three possible cases are identified:

1. If $d^{(n)} \neq 0$ then s_n cannot be generated using $C^{(n)}(X)$. Further:

   a) If $2L^{(n)} > n$ then the new characteristic polynomial is computed as $C^{(n+1)}(X) \leftarrow C^{(n)}(X) - \frac{d^{(n)}}{d^{(m)}} X^{(m-L^{(m)})-(n-L^{(n)})} C^{(m)}(X)$ and it has the same degree as the previous one, $L^{(n+1)} = L^{(n)}$;

   b) If $2L^{(n)} \leq n$ then the new characteristic polynomial is computed as $C^{(n+1)}(X) \leftarrow X^{(n-L^{(n)})-(m-L^{(m)})} C^{(n)}(X) - \frac{d^{(n)}}{d^{(m)}} C^{(m)}(X)$ and it has 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).

We present the classic iterative version of the algorithm in Algorithm 1. The algorithm has as input a finite sequence $s = s_0, s_1, \ldots, s_{t-1}$ over a field and it returns $C^*(X)$, the characteristic polynomial and L^*, the linear complexity of the sequence.

A recursive version of the Berlekamp-Massey Algorithm is presented in Algorithm 2 (using the recursive procedure in Algorithm 3). The most important element of the recursive implementation is the recursive procedure *bmR(m, $D(X), d_m, C(X)$, n)* with the following parameters: m, the position in the sequence where the last degree change has occurred, $D(X)$, the characteristic polynomial at the last change, d_m, the discrepancy value at the last change, $C(X)$, the characteristic polynomial corresponding to the current position and n, the currently processed position in the sequence.

Algorithm 1 Berlekamp-Massey Algorithm

Input: A finite sequence $s = s_0, s_1, \ldots, s_{t-1}$
Output: Linear complexity, L^* and characteristic polynomial, $C^*(X)$ of s.
$n \leftarrow 0$
while $s_n = 0$ and $n < t$ **do**
    $n \leftarrow n + 1$
end while
if $n = t$ **then**
    $C^*(X) \leftarrow 1$
    $L^* \leftarrow 0$
    return $C^*(X), L^*$
else
    $m \leftarrow n$
    $D(X) \leftarrow 1$
    $d_m \leftarrow s_n$
    $n \leftarrow n + 1$
    $C(X) \leftarrow X^n$
    $L_n \leftarrow n$
    if $n \leq t - 1$ **then**
        repeat
            $d_n \leftarrow s_n + \sum_{i=0}^{L_n - 1} c_i s_{i+n-L_n}$
            if $d_n <> 0$ **then**
                if $2L_n > n$ **then** ▷ (1a) degree and complexity does not change
                    $C(X) \leftarrow C(X) - \frac{d_n}{d_m} X^{(m-L_m)-(n-L_n)} D(X)$
                else ▷ (1b) degree and complexity do change
                    $T(X) \leftarrow C(X)$
                    $C(X) \leftarrow X^{(n-L_n)-(m-L_m)} C(X) - \frac{d_n}{d_m} D(X)$
                    $D(X) \leftarrow T(X)$
                    $d_m \leftarrow d_n$
                    $m \leftarrow n$
                    $L_n \leftarrow n + 1 - L_n$
                end if
            else ▷ (2) current characteristic polynomial does not change
                $n \leftarrow n + 1$
            end if
        until $n = t$
    end if
    $C^*(X) \leftarrow C(X)$
    $L^* \leftarrow L_n$
    return $C^*(X), L^*$
end if

Algorithm 2 Recursive Berlekamp-Massey Algorithm

Input: A finite sequence $s = s_0, s_1, \ldots, s_{t-1}$
Output: Linear complexity, L^* and characteristic polynomial, $C^*(X)$ of s.
$n \leftarrow 0$
while $s_n = 0$ and $n < t$ **do**
  $n \leftarrow n + 1$
end while
if $n = t$ **then**
  $C^*(X) \leftarrow 1$
  $L^* \leftarrow 0$
  return $C^*(X), L^*$
else
  $m \leftarrow n$
  $D(X) \leftarrow 1$
  $d_m \leftarrow s_n$
  $n \leftarrow n + 1$
  $C(X) \leftarrow X^n$
  $C^*(X) \leftarrow C(X)$
  $L^* \leftarrow n$
  if $n \leq t - 1$ **then**
    $bmR(m, D(X), d_m, C(X), n)$
  end if
  return $C^*(X), L^*$
end if

Algorithm 3 The $bmR(m, D(X), d_m, C(X), n)$ procedure

procedure BMR$(m, D(X), d_m, C(X), n)$
  if $n = t$ **then**
    $L^* \leftarrow \deg(C(X))$
    $C^*(X) \leftarrow C(X)$
  else
    $L_n \leftarrow \deg C(X)$
    $d_n \leftarrow s_n + \sum_{i=0}^{L_n - 1} c_i s_{i+n-L_n}$
    if $d_n \neq 0$ **then**
      if $2L_n > n$ **then** ▷ (1a)
        $C(X) \leftarrow C(X) - \frac{d_n}{d_m} X^{(m-L_m)-(n-L_n)} D(X)$
        $bmR(m, D(X), d_m, C(X), n + 1)$
      else ▷ (1b)
        $T(X) \leftarrow C(X)$
        $C(X) \leftarrow X^{(n-L_n)-(m-L_m)} C(X) - \frac{d_n}{d_m} D(X)$
        $bmR(n, T(X), d_n, C(X), n + 1)$
      end if
    else ▷ (2)
      $bmR(m, D(X), d_m, C(X), n + 1)$
    end if
  end if
end procedure

2.2.7 k-error linear complexity

The concept of linear complexity of a sequence can be generalised to k-error linear complexity, which is the minimal linear complexity of that sequence in which the values on at most k positions are changed. The concept was firstly outlined by Ding, Xiao, Shan [12] under the name of k-sphere complexity in the context of the stability of stream ciphers, and defined under the name of k-error linear complexity by Stamp and Martin [79]. Note that the 0-error linear complexity coincides with the linear complexity.

Definition 2.12. *For a given finite sequence s of size t (or an infinite sequence s of period N) we denote $w_H(s) = \#\{i|i = 0, 1, \ldots, t-1, s_i \neq 0\}$ (or $w_H(s) = \#\{i|i = 0, 1, \ldots, N-1, s_i \neq 0\}$, respectively) the Hamming weight of s i.e. the number of non-zero terms of s (or the number of non-zero terms in a period of s). For periodic sequences the notations $w_H(s)$ and $w_H((s_0, s_1, \ldots, s_{N-1}))$ are equivalent. For a given finite set A, we denote by $\#A$ the number of elements in the set A.*

In definition 2.13, $s + e$ is the sum between two sequences. We consider the term by term addition, namely: if $a = (a_0, \ldots, a_{t-1})$ and $b = (b_0, \ldots, b_{t-1})$ then the sum sequence $a + b = (a_0 + b_0, \ldots, a_{t-1} + b_{t-1})$ where the addition is in the field which includes the terms of a and b.

Definition 2.13. *For a 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*

$$L_k(s) = \min\{L(s+e)|e \in K^t, w_H(e) \leq k\} \tag{2.12}$$

For an infinite sequence $s = s_0, s_1, \ldots$ of period N, with elements in a field K and for 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*

$$L_k(s) = \min\{L(s+e)|\ e \text{ sequence of period } N \text{ over } K, w_H((e_0, e_1, \ldots, e_{N-1})) \leq k\} \tag{2.13}$$

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 $0 \leq k \leq w_H(s)$.

The following property shows that the k-error linear complexity decreases with k for a fixed sequence s.

Property 2.14. *Given a (finite or infinite periodic) sequence s with elements in a finite field K, we have $L_i(s) \geq L_j(s)$, for all $i < j$.*

Proof. The proof is immediate using the definition of k-error linear complexity and the property that for any two sets S_1 and S_2 of sequences such that $S_1 \subseteq S_2$, it is true that $\min\{L(s)|s \in S_1\} > \min\{L(s)|s \in S_2\}$. □

Property 2.15. *Given a (finite or infinite periodic) sequence s with elements in a finite field K, we have $L_k(s) = 0$, for all $k \geq w_H(s)$.*

Proof. We choose e such that

$$e_i = \begin{cases} 0, & \text{for all } i \in 0, 1, \ldots, t-1 \text{ with } s_i = 0, \\ -s_i, & \textit{otherwise.} \end{cases} \tag{2.14}$$

Therefore $w_H(e) = w_H(s)$ and $s + e = (0, 0, \ldots, 0)$, so $L_{w_H(s)} = 0$. It follows using property 2.14 that $L_k(s) = 0$, for all $k \geq w_H(s)$. □

If the k-error linear complexity of a sequence is very low for small values of k (e.g. k less than 5% of the length of the sequence), then that sequence is likely to be easily recovered when only knowing a short segment of the sequence. The k-error linear complexity is therefore an important parameter when analysing the security of cryptographic sequences.

Sequences with a high linear complexity and good linear complexity profile can have a very low k-error linear complexity for small values of k, making them unsecure in cryptographical applications. Consider a binary sequence of N zeros followed by a 1, $s = (\underbrace{0, \ldots, 0}_{\text{N times}}, 1)$. This sequence has a linear complexity of $N+1$, but a 1-error linear complexity of 0. Nontrivial examples of sequences with high linear complexity and low k-error linear complexity can be obtained using the results presented by Safavi-Naini and Seberry [72].

Fell's paper [14] reflects on the k-error linear complexity by considering bijections on the set of sequences of a certain length and by studying the average difference in the linear complexity of a sequence and its image using one of these bijections. These results are applied to the computation of an average k-error linear complexity of a sequence. The sequences considered are binary and the methods are probabilistic. Fell's paper provides an upper bound of the average change in linear complexity for sequences where at most k errors are forced. Fell concludes that for large values of t there are many sequences of size t which are far, in terms of Hamming distance, from low linear complexity sequences. It is therefore not straightforward to find a direct algorithm which produces for most sequences the nearby sequences of low linear complexity. Fell calls two sequences

k-close, if one can be obtained from the other by changing k terms. It is suggested to search for sequences k-close to sequences of low complexity in order to exclude them from being used in stream ciphers as key streams.

By extending the Games-Chan Algorithm (Games and Chan [17]), which computes the linear complexity of a periodic binary sequence with period a power of two, Stamp and Martin (Stamp and Martin [79]) 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 two. The Stamp-Martin Algorithm was further extended to compute the whole k-error linear complexity profile by Lauder and Paterson [38]. 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 [12], Kaida, Uehara, Imamura [30, 31] and Kaida [29]. An efficient algorithm for computing the k-error linear complexity of periodic sequences over a finite field $GF(q)$ when the period is of the form $2p^n$, with p prime and $n > 0$ is presented in [84] (the special case treated is where p and q are odd primes, and q is a primitive root modulo p^2). All these algorithms, unlike the Berlekamp-Massey Algorithm, need a whole period as input, i.e. the whole sequence is known, which would not be the case in cryptanalysis applications. However from a design point of view it is useful to predetermine the level of security of the sequences which are used in a cipher in order to prevent successful cryptographic attacks.

Efficient techniques to compute the 1-error linear complexity of periodic binary sequences are available (Kolokotronis et al. [35]).

Some research has been invested in trying to find periodic sequences with high linear complexity and high k-error linear complexity for small values of k as well as bounds for the two measurements. A unified derivation of the bounds of the k-error linear complexity is available for N-periodic sequences s, with tight bounds when $L(s) < \frac{N}{k}$ (Jiang et al. [28]). It was conjectured by Ding et al. [12] that there may be a tradeoff between the linear complexity and the k-error linear complexity. However, Niederreiter proves the existence of periodic sequences which simultaneously achieve maximum value for the linear complexity and k-error linear complexity (Niederreiter [59]). Moreover, a lower bound on the number of N-periodic sequences with maximum linear complexity (i.e. N) and k-error linear complexity at least a fixed value $N-c$, with $c > 0$, is given to show that a consistent number of such sequences exist (Meidl and Niederreiter [51]). A constructive proof is yet to be found, however using such a constructive technique for a real cipher would not improve the security, on the contrary it would restrict the search of a cryptanalyst to a well specified set of sequences.

A survey of recent work on the linear complexity, linear complexity profile and

the k-error linear complexity of periodic sequences is included in Niederreiter [58]. The study focuses mostly on the recent results regarding the statistical theory of complexity measurements, namely the expected value of the linear complexity and k-error linear complexity, as well as lower and upper bounds for the number of sequences with a designated linear complexity or k-error linear complexity.

Meidl [47] obtains for sequences of period of a certain form, the relation between the linear complexity and the minimum value k, for which the k-error linear complexity is strictly less than the linear complexity. This is useful as it gives a way of finding how many bits need to be changed to decrease the linear complexity of a sequence. Further statistical results are published by Meidl [48] and exact formulas for the expected value of the linear complexity and for the 1-error linear complexity are given for sequences of period a power of two. For $k \geq 2$ lower and upper bounds of the expected value of the k-error linear complexity are given.

Kurosawa et al. [36] present results regarding the relation between the linear complexity and the minimum value k for which the k-error linear complexity is strictly less than the linear complexity.

Sălăgean [73] presents theoretical results showing how to apply Games-Chan and Stamp-Martin Algorithms to an infinite sequence with the period a power of two, when a whole period is not known apriori.

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. See Chapter 3 for a description of the exhaustive techniques available.

Chapter 3

k-error linear complexity problem

In this chapter we define and analyse the problems that we are interested in with respect to the k-error linear complexity. We also describe and analyse the exhaustive search techniques that can be used for solving these problems in order to obtain an exact result. We show the way that the exhaustive search method can be optimised without losing the accuracy of the result.

First, let us clearly define the problems we will be investigating.

k-error linear complexity problem

Input: A finite sequence $s = s_0, s_1, \ldots, s_{t-1}$ of size $t > 0$ with terms over a finite field $GF(q)$, q a prime power; an integer value k_0, with $0 < k_0 \leq w_H(s) - 1$.

Output: The k_0-error linear complexity of s, $L^*_{k_0}$; the error pattern corresponding to this linear complexity, $e^*_{k_0}$; a minimal characteristic polynomial $C^*_{k_0}(X)$ corresponding to the sequence $s + e^*_{k_0}$.

While the k-error linear complexity problem is interesting in its own right we will implement whenever possible a solution for the following problem which determines the whole k-error linear complexity profile for a certain sequence s and limit of errors k_0. Note that any algorithm which solves the k-error linear complexity profile problem is also an algorithm for the previous problem, the k-error linear complexity problem.

k-error linear complexity profile problem

Input: A finite sequence $s = s_0, s_1, \ldots, s_{t-1}$ of size $t > 0$ with terms over a finite field $GF(q)$, where q is a prime power; an integer value k_0, with $0 < k_0 \leq w_H(s) - 1$.

Output: The k_0-error linear complexity profile of s containing for each i, $i = 0, 1, \ldots, k_0$: L^*_i, the i-error linear complexity; e^*_i, the error pattern producing the linear complexity L^*_i on s; $C^*_i(X)$ a minimal characteristic polynomial corresponding to the sequence $s + e^*_i$.

Note that we exclude from the previous problems the marginal cases when $k_0 = 0$ or when $k_0 \geq w_H(s)$, since for these values the k-error linear complexity can be immediately evaluated in polynomial time, $L_0(s) = L(s)$ or in constant time, $L_k(s) = 0$, for all $k \geq w_H(S)$ (see property 2.15 in section 2.2.7), respectively. We impose these restrictions in order to concentrate on the interesting part of the problem and separate any special cases with an immediate solution.

Also, note that in these problems, we are not interested only in the value of the k-error linear complexity for a specific k but also the error pattern e which produces that complexity and the characteristic polynomial of the sequence $s + e$. In general, for an integer i with $0 < i \leq w_H(s) - 1$, we denote in the output, L_i^* as the i-error linear complexity, e_i^*, an error pattern which achieves this complexity, i.e. $L(s+e_i^*) = L_i^*$ and $C_i^*(X)$, a minimal characteristic polynomial of the sequence $s + e_i^*$.

For the all zero sequence $\mathbf{0} = (0, 0, \ldots, 0)$, the k-error linear complexity and the k-error linear complexity profile can be immediately obtained, since $L_k(\mathbf{0}) = 0$ for any integer k. Moreover for this very reason, since in the problems above we constrain the values of k_0 to be such that $0 < k_0 \leq w_H(s) - 1$ then the all zero sequence is not a valid input ($w_H(\mathbf{0}) = 0$). This is deliberate and we consider this a special case which can be excluded from the input as its solution is trivial, $L_k(\mathbf{0}) = 0$ and the corresponding minimal characteristic polynomial and error pattern are $C_k(X) = 1$ and $e_k = (0, 0, \ldots, 0)$, respectively, for any integer $k > 0$.

In this thesis, we will consider that an approximation algorithm for the k-error linear complexity profile problem is correct if the profile returned by that algorithm is composed of correct linear complexities and minimal characteristic polynomials corresponding to the input sequence and each of the error patterns.

Definition 3.1. *An approximation algorithm for the k-error linear complexity profile problem is correct if for all $i = 0, 1, \ldots, k_0$, the value returned by the algorithm L_i^* is equal to $L(s+e_i^*)$ and $C_i^*(X)$ is the minimal characteristic polynomial of the sequence $s + e_i^*$.*

It is difficult to set a correctness check for the accuracy of the result as this would naturally focus more on the 'goodness' rather than the correctness of the solution. However it is safe to assume that for each $k = 0, 1, \ldots, k_0$ the k-error linear complexity returned by the approximation algorithm, L_k^* is lower bounded by the exact value of the k-error linear complexity, $L_k(s)$ and it is upper bounded by the linear complexity of the input sequence, $L(s)$. Formally that means that a solution of an approximation algorithm for the k-error linear complexity profile problem satisfies the following condition: for each $i = 0, 1, \ldots, k_0$, $L_i(s) \leq L_i^* \leq L(s)$,

where $L_i(s)$ and $L(s)$ is the exact i-error linear complexity and the exact linear complexity of the input sequence, respectively.

A special, reversed problem can be formulated with respect to the k-error linear complexity, in the context of the application of the design of cryptographic sequences. This problem assumes a practical application where a sequence is tested, to check if its k-error linear complexity does not fall under a specified threshold L_0 when a certain number of errors k are allowed. Therefore, if this happens for small values of k and L_0, the sequence is unsafe to be used as it is easy to break.

The problem is: what is the minimum number of errors k that need to be applied on the terms of a sequence s such that the linear complexity decreases below a specified value L_0 (Sălăgean [73]). Namely, what is the minimum k such that there is an error pattern e of weight k with $L(s+e) \leq L_0$.

We denote this problem the L-constrained k-error linear complexity problem.

L-constrained k-error linear complexity problem

Input: A finite sequence $s = s_0, s_1, \ldots, s_{t-1}$ of size $t > 0$ with terms over a finite field $GF(q)$, where q is a prime power; an integer value k_0, with $0 < k_0 \leq w_H(s) - 1$; an integer value L_0, with $0 \leq L_0 \leq t$.

Output: The minimal number of errors k^*, such that $k^* \leq k_0$, necessary to lower the linear complexity of s below L_0, i.e. $L_k^*(s) \leq L_0$; the k^*-error linear complexity, L^*; e^* the error pattern producing the linear complexity L_k^*; $C^*(X)$ a minimal characteristic polynomial corresponding to the sequence $s + e^*$. If there is no such k^* then an error value, e.g. -1, is returned.

Note that the L-constrained k-error linear complexity problem might not have a solution when k_0 errors are not enough to lower the complexity of the input sequence below the given L_0. In such a case, an error code is expected, e.g. -1.

3.1 Naïve Exhaustive Search Algorithm

Determining the k-error linear complexity of a finite sequence of length t over a finite field of order q using an exhaustive search approach would mean investigating all the $\sum_{i=0}^{k}(q-1)^i\binom{t}{i}$ possible error patterns of up to k errors, computing the linear complexity of each of the sequences obtained by adding these error patterns to the original sequence and, finally, the error pattern which corresponds to the minimum linear complexity would be the solution.

The Naïve Exhaustive Search Algorithm for the k-error linear complexity profile problem is presented in listing 4.

Algorithm 4 Naïve Exhaustive Search Algorithm for the k-error linear complexity profile problem

Input: A finite sequence $s = s_0, s_1, \ldots, s_{t-1}$ over $GF(q)$; an integer k_0, with $0 < k_0 \leq w_H(s) - 1$.
Output: L_i^*, $C_i^*(X)$ and e_i^*, for all $i = 0, 1, \ldots, k_0$
for $i = 0, 1, \ldots, k_0$ **do**
    $L_i^* \leftarrow L(s)$
    $C_i^*(X) \leftarrow C(X)$, a minimal characteristic polynomial
    $e_i^* \leftarrow (\underbrace{0, 0, \ldots, 0}_{t \text{ times}})$
end for
for all $\{e | e \in GF(q)^t$ with $w_H(e) \leq k_0\}$ **do**
    Calculate $L(s+e)$ and $C(X)$ corresponding to $s+e$
    $k \leftarrow w_H(e)$
    if $L_k^* > L(s+e)$ **then**
        $L_k^* \leftarrow L(s+e)$
        $C_k^*(X) \leftarrow C(X)$
        $e_k^* \leftarrow e$
    end if
end for
return L_i^*, $C_i^*(X)$ and e_i^*, for all $i = 0, 1, \ldots, k_0$

Since the k-error linear complexity is calculated as a minimum value (from the definition), the profile is initialised with the maximum possible value that $L_i^*(s) = L(s)$, $C_i^*(X) = C(X)$ and $e_i^* = (\underbrace{0, 0, \ldots, 0}_{t \text{ times}})$, for all $i = 0, 1, \ldots, k_0$, where $C(X)$ is a minimal characteristic polynomial of s.

All possible error patterns $e \in GF(q)^t$ of weight less than or equal to k_0 are processed. The linear complexity and the characteristic polynomial of these $s+e$ are calculated using the Berlekamp-Massey Algorithm and the error patterns corresponding to the minimum value $L(s+e)$ for each number of errors are saved in the k_0-error linear complexity profile.

From the above considerations it is immediate to obtain the following theorem.

Theorem 3.2. *The Naïve Exhaustive Search Algorithm for k-error linear complexity profile problem (listing 4) is correct.*

Proof. The Naïve Exhaustive Search Algorithm is a direct implementation of the definition of the k-error linear complexity (see definition 2.13). All the error patterns of weight less than k_0 are selected, added to the input sequence and the linear complexity of the resulting sequence is calculated. The minimum for each weight is saved. Therefore it follows immediately that the Naïve Exhaustive Search Algorithm for the k-error linear complexity profile problem is correct. □

The Naïve Exhaustive Search Algorithm for the L-constrained k-error linear

complexity problem can be easily obtained with a similar exhaustive approach. We present it in listing 5 and its correctness is immediate.

Theorem 3.3. *The Naïve Exhaustive Search Algorithm for the L-constrained k-error linear complexity problem (listing 5) is correct.*

Proof. The Naïve Exhaustive Search Algorithm for the L-constrained k-error linear complexity problem is a direct implementation of

$$k^* = \min\{k | 0 \leq k \leq k_0 \text{ so there is an } e \in GF(q)^t, w_H(e) = k \text{ and } L(s+e) \leq L_0\}.$$

All the error patterns of weight less than k_0 are selected, added to the input sequence and the linear complexity of the resulting sequence is calculated. We save the minimum value of k for which an error pattern e of weight k and with $L(s+e) \leq L_0$ is found. Therefore it is immediate that the Naïve Exhaustive Search Algorithm for the L-constrained k-error linear complexity is correct. If no such k is found then the value of k^* will remain $w_H(s)$ and an error code is returned. □

Algorithm 5 Naïve Exhaustive Search Algorithm for the L-constrained k-error linear complexity problem

1: **Input:** A finite sequence $s = s_0, s_1, \ldots, s_{t-1}$ over $GF(q)$; an integer k_0, with $0 < k_0 \leq w_H(s) - 1$; an integer L_0, with $0 \leq L_0 \leq t$.
2: **Output:** k^*, L^*, e^* and $C^*(X)$.
3: $k^* \leftarrow w_H(s)$
4: $L^* \leftarrow 0$
5: $C^*(X) \leftarrow 1$
6: $e^* \leftarrow (-s_0, -s_1, \ldots, -s_{t-1})$
7: **for all** $\{e | e \in GF(q)^t$ with $w_H(e) \leq k_0\}$ **do**
8: Calculate $L(s+e)$ and $C(X)$ corresponding to $s+e$
9: $k \leftarrow w_H(e)$
10: **if** $k < k^*$ and $L(s+e) \leq L_0$ **then**
11: $k^* \leftarrow k$
12: $L^* \leftarrow L(s+e)$
13: $C^*(X) \leftarrow C(X)$
14: $e^* \leftarrow e$
15: **end if**
16: **end for**
17: **if** $k^* < w_H(s)$ **then**
18: **return** k^*,L^*, $C^*(X)$ and e^*
19: **else**
20: **return** -1
21: **end if**

3.2 Efficient Exhaustive Search Algorithm

Some computational savings can be made in an exhaustive search approach by taking advantage of the incremental nature of the Berlekamp-Massey Algorithm which is used for computing the linear complexity and the characteristic polynomial of each sequence $s+e$. Namely, for error patterns which coincide on the first say i positions, we can reuse the computations made on the first i terms of the sequence $s+e$.

This more efficient version of an exhaustive search can be implemented for example by extending the recursive version of the Berlekamp-Massey Algorithm (algorithm 2 in section 2.2.6). We describe this alternative method for the k-error linear complexity problem below (see listings 6 and 7). This method follows closely the Berlekamp-Massey Algorithm.

Similarly to the initialisation step in the naïve version the k_0-error linear complexity profile is initialised with the maximum possible value that $L_i^*(s) = L(s)$, $C_i^*(X) = C(X)$ and $e_i^* = (\underbrace{0,0,\ldots,0}_{t \text{ times}})$, for all $i = 0,1,\ldots,k_0$, where $L(s)$ is the linear complexity of s and $C(X)$ is a minimal characteristic polynomial of s (note that $\deg(C(X)) = L(s)$.

The position of the first non zero element of s is found. If it would coincide with the size of the sequence then it would mean that the input sequence is all zero, a special case which we have treated and excluded in the definition of the problem.

Once the position of the first non zero term is found the initialisation step needs to be performed. Since the exhaustive algorithm for the k-error linear complexity calculates the linear complexity for sequences of the form $s+e$ where s is the input sequence and $e \in GF(q)^t$ is an error pattern of weight at most k_0, the position of the first non zero element in $s+e$ varies depending on the error pattern e.

The recursive procedure $exhefR(m, D(X), d_m, C(X), n, e, init)$ covers all the error patterns e of weight at most k_0, calculating the linear complexities of $s+e$ with the Berlekamp-Massey Algorithm and saving the minimum values for each weight, building this way the k_0-error linear complexity profile of s.

The initial value of the error pattern e is all zero. Subsequently, the terms of the error patterns take all the possible values from $GF(q)^*$. In order to select iteratively all or a subset of values from $GF(q)$ (e.g. line 24 in procedure $exhefR$ from listing 7), we need to consider an arbitrary but fixed total ordering, which we denote $\prec$, of the finite field $GF(q) = \{f_0 = 0, f_1, \ldots, f_{q-1}\}$, i.e. $f_0 = 0 \prec f_1 \prec \ldots \prec f_{q-1}$.

At each step of the recursion, as the current error pattern e is built and the corresponding sequence $s+e$ is processed, the parameters $m, D(X), d_m, C(X)$

Algorithm 6 Efficient Exhaustive Search Algorithm for the k-error linear complexity profile problem

1: **Input:** A finite sequence $s = s_0, s_1, \ldots, s_{t-1}$ over GF(q); an integer k_0, with $0 < k_0 \leq w_H(s) - 1$
2: **Output:** L_i^*, $C_i^*(X)$ and e_i^*, for all $i = 0, 1, \ldots, k_0$
3: **for** $i = 0, 1, \ldots, k_0$ **do**
4: $L_i^* \leftarrow L(s)$
5: $C_i^*(X) \leftarrow C(X)$, the characteristic polynomial of s
6: $e_i^* \leftarrow (\underbrace{0, 0, \ldots, 0}_{t \text{ times}})$
7: **end for**
8: $e \leftarrow (\underbrace{0, 0, \ldots, 0}_{t \text{ times}})$
9: $m \leftarrow 0$
10: $D(X) \leftarrow 1$
11: $d_m \leftarrow 1$
12: $n \leftarrow 0$
13: $C(X) \leftarrow 1$
14: $init \leftarrow false$
15: **call** $exhefR(m, D(X), d_m, C(X), n, e, init)$
16: **return** L_i^*, $C_i^*(X)$ and e_i^*, for all $i = 0, 1, \ldots, k_0$

are updated. These parameters have the same meaning as in the Berlekamp-Massey Algorithm (section 2.2.6), namely m is the last change index, $D(X)$ is the characteristic polynomial at the last change, d_m is the discrepancy at the last change and $C(X)$ is the current characteristic polynomial.

The parameter *init* has a boolean value which reflects if the initialisation step has been performed ($init = true$) or not ($init = false$).

At each step n, if the initialisation has not been done yet ($init = false$) we can either consider an error of magnitude $-s_n$ which will delay the initialisation to a later step, since $s_n + e_n = 0$ (lines 12-13 in algorithm 7) or apply any other value from $GF(q) \setminus \{-s_n\}$ ($s_n + e_n \neq 0$), perform the initialisation and proceed to the next element (lines 14-21 in algorithm 7).

If the initialisation has been done ($init = true$), similarly with the Berlekamp-Massey Algorithm, the procedure firstly calculates the current discrepancy d_n, where c_n are the coeffcients of the current characteristic polynomial $C(X) = X^{L_n} + c_{L_n-1}X^{L_n-1} + \ldots + c_1X + c_0$ and L_n is the linear complexity of the sequence $s + e$ up to element n, $L_n = L(s_0+e_0, s_1+e_1, \ldots, s_{n-1}+e_{n-1})$. Secondly the new values of the intermediary parameters $m, D(X), n, C(X)$ are calculated.

For each element s_n, all possible error values are considered on that position, thus calculating the linear complexity of the sequence $s+e$ where e is each possible error pattern (lines 23-39 in algorithm 7).

The statement $E \leftarrow (k = k_0)\ ?\ \{f_0\} : GF(q)$ in line 23 (listing 7) has the usual

Algorithm 7 The $exhefR(m, D(X), d_m, C(X), n, e, init)$ procedure

procedure EXHEFR$(m, D(X), d_m, C(X), n, e, init)$
  $k \leftarrow w_H(e)$
  $L_n \leftarrow \deg(C(x))$
  $L_m \leftarrow \deg(D(x))$
  if $(n = t)$ or $(k > k_0)$ or $(L_k^* \leq L_n)$ **then** ▷ Stop cond
    if $((n = t)$ and $(k \leq k_0)$ and $(L_k^* > L_n))$ **then**
      $(L_k^*, C_k^*(X), e_k^*) \leftarrow (L_n, C(X), e)$
      call $adjustProfile(L_n, C(X), e, k)$
    end if
  else
    if $init = false$ **then** ▷ Not initialised yet
      $e_n \leftarrow -s_n$
      call $exhefR(m, D(X), d_m, C(X), n+1, e, false)$
      for all $g \in GF(q) \backslash \{-s_n\}$ in increasing order relative to $\prec$ **do**
        $e_n \leftarrow g$
        $m \leftarrow n$
        $D(X) \leftarrow 1$
        $d_m \leftarrow (s+e)_m$
        $C(X) \leftarrow X^{n+1}$
        call $exhefR(m, D(X), d_m, C(X), n+1, e, true)$
      end for
    else ▷ Already initialised
      $E \leftarrow (k = k_0)\ ?\ \{f_0\} : GF(q)$
      for all $g \in E$ in increasing order relative to $\prec$ **do**
        $e_n \leftarrow g$
        $d_n \leftarrow (s+e)_n + \sum_{i=0}^{L_n - 1} c_n (s+e)_{i+n-L_n}$
        if $d_n \neq 0$ **then**
          if $2L > n$ **then**
            $C(X) \leftarrow C(X) - \frac{d_n}{d_m} X^{(m-L_m)-(n-L_n)} D(X)$
            call $exhefR(m, D(X), d_m, C(X), n+1, e, true)$
          else
            $T(X) \leftarrow C(X)$
            $C(X) \leftarrow X^{(n-L_n)-(m-L_m)} C(X) - \frac{d_n}{d_m} D(X)$
            call $exhefR(n, T(X), d_n, C(X), n+1, e, true)$
          end if
        else
          call $exhefR(m, D(X), d_m, C(X), n+1, e, true)$
        end if
      end for
    end if
  end if
end procedure

C/C++ meaning of the ternary conditional operator ? :, namely, if $k = k_0$ then $E \leftarrow \{f_0\}$, otherwise $E \leftarrow GF(q)$. This way only error patterns of weight at most k_0 are considered.

The stopping condition of the recursive call is a disjunction of subconditions.

Firstly, if either the end of the sequence is reached, or if the current error pattern has a weight higher than k_0, the processing of that recursive path needs to stop in order to save the result, in the former case, or to discard the current error pattern since the limit of errors was reached, in the latter. Concerning the latter condition, note that due to the ascending property of the linear complexity (see property 2.8) and since any error pattern on the current recursion path would have the initial segment equal to the current error pattern e, this means that if the recursion would not stop then it would either progress to an error pattern with a higher weight than allowed, or the complexity of $s + e$ would become higher than the current best for k_0-error linear complexity.

Additionally, when the linear complexity of the errored sequence $s + e$ up to term n (in the algorithm denoted L_n) is higher or equal to the current best for $k = w_H(e)$ errors, i.e. $L_k^* \leq L_n$, then in this case the calculations on that path can stop since it is impossible to improve the current solution using an error with the initial segment e. This is based on the fact that the linear complexity of a fixed sequence increases with the index of the processed term (property 2.8 in section 2.2.5) and that the k-error linear complexity decreases with the value of k (property 2.14). If a certain error pattern e of weight k up to term n gives a linear complexity $L_n = L(s_0 + e_0, \ldots, s_{n-1} + e_{n-1})$ such that $L_n \geq L_k^*$, then any error pattern e' coinciding with e on the first n terms would not contribute to the solution profile since $L(s + e') \geq_{\text{prop 2.8}} L(s + e) \geq L_k^* \geq_{\text{prop 2.14}} L_{w_H(e')}^*$ since $w_H(e') \geq k$.

In order to make this latter condition more efficient ($L_k^* \leq L_n$), the currently stored k_0-error linear complexity profile can be maintained whenever a new solution is found, using the property of the k-error linear complexity of decreasing with the number of errors k (property 2.14 in section 2.2.7). Whenever a new solution is found, some of the k-error linear complexities stored in the profile can be checked to see if there are any possible adjustments (see implementation in listing 8). This ensures that the current profile solution is a valid one (decreasing with k) and also as close to the exact value as possible, so the comparison of the currently found k-error linear complexity (L_n) with the current solution for errors up to k (L_k^*) is as fruitful as possible.

The adjustment of the solution k-error linear complexity profile contains one loop which implements the following logic. When an error pattern e of weight k which improves the current solution is found, then execute:

- For all $i = k+1, \ldots, w_H(s)-1$, if the currently stored i-error linear complexity is more than the newly found k-error linear complexity (i.e. $L_i^*(s) > L(s+e)$) then make the i-error solution equal to the k-error solution (copy the i-error linear complexity, the characteristic polynomial and error sequence), as $L_i(s) \leq L_k(s)$.

Note that for all $i = 0, 1, \ldots, k-1$, the currently stored i-error linear complexity is larger then the newly found k-error linear complexity (i.e. $L_i^*(s) \geq L(s+e)$) in compliance with property 2.14 in section 2.2.7.

Algorithm 8 The maintenance of the k-error linear complexity profile

```
procedure ADJUSTPROFILE(L*, C*(X), e*, k)
    for i = k + 1, ..., k_0 do
        if L_i^* > L_k^* then
            L_i^* ← L_k^*
            C_i^*(X) ← C_k^*(X)
            e_i^* ← e_k^*
        end if
    end for
end procedure
```

Figure 3.1 is a graphical representation of an execution tree of recursive calls within the Efficient Exhaustive Search Algorithm (EESA), in the *exhefR* procedure. The root represents the initial state of the algorithm and each node represents a recursive transition from one index n to the next one, $n+1$, for all $0 \leq n < t-1$. Each arc between two nodes say on levels n and $n+1$ is labeled with a value from $GF(q) = \{f_0 = 0, f_1 = 1, \ldots, f_{q-1}\}$, for all $0 \leq n < t-1$, the value being the one attributed to e_n in the algorithm when processing the next term $(s+e)_n$. The leaves correspond to the end of an error pattern, i.e. reaching one of the stopping conditions in the recursive procedure, for example $n = t$.

Any path from the root to a leaf will contain at most k_0 arcs labeled with non zero values, e.g. values from the set $GF(q)^*$. The paths of length t from the root to a leaf correspond to error patterns which are processed in full by the algorithm. The error $e = (\underbrace{0, 0, \ldots, 0}_{t \text{ times}})$ will always be considered regardless of the value of k_0 $(0 < k_0 \leq w_H(s)-1)$, and this error pattern corresponds to the leftmost path. The depth[1] of the tree is therefore t. Also, note that the rightmost path corresponds to the error sequences of weight k_0, $e = (\underbrace{f_{q-1}, f_{q-1}, \ldots, f_{q-1}}_{k_0 \text{ times}}, \underbrace{0, \ldots, 0}_{t-k_0 \text{ times}})$.

[1]The depth of a tree is the number of edges in the path from the root node to its furthest leaf.

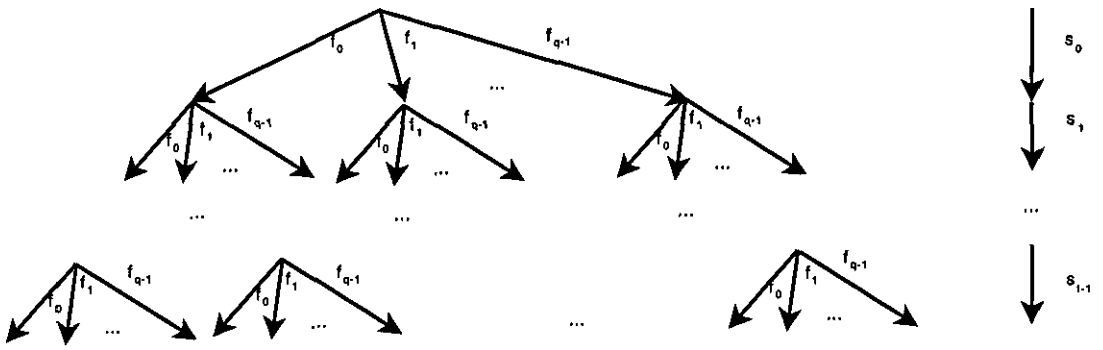

Figure 3.1: The execution tree of the efficient exhaustive search algorithm.

The Efficient Exhaustive Search Algorithm for the k-error linear complexity profile problem calculates the linear complexity of the input sequence s to which we add every relevant error pattern $e \in GF(q)^t$ with $w_H(s) \leq k_0$. The error pattern which produces the minimum complexity for each number of errors $i = 0, 1, \ldots, k_0$ is saved. Therefore the algorithm correctly returns the k_0-error linear complexity profile of the sequence along with the corresponding minimal characteristic polynomials and error patterns.

Theorem 3.4. *The Efficient Exhaustive Search Algorithm for the k-error linear complexity profile problem (listings 6 and 7) is correct.*

Proof. Note that the stop condition in procedure $exhefR$ ensures that all the error patterns $e \in GF(q)^t$ processed by the algorithm are such that $w_H(e) \leq k_0$.

We can prove by induction on n, where $n = 0, 1, \ldots, t-1$, that for each error sequence $e = e_0, e_1, \ldots, e_n$ with $w_H(e) \leq k_0$ processed by the algorithm, the values of the intermediary parameters m, $D(X)$, d_m, $C(X)$ and $init$ calculated by the recursive procedure are correct for the sequence
$(s+e)^{(n)} = (s_0 + e_0, s_1 + e_1, \ldots, s_n + e_n)$. Specifically, m is the last change index, i.e. the maximum value such that $m < n$ and $L((s+e)^{(m-1)}) < L((s+e)^{(m)} = L((s+e)^{(n-1)})$; $D(X)$ and d_m are a characteristic polynomial and the discrepancy corresponding to the last change term m; $C(X)$ is a minimal characteristic polynomial for $(s+e)^{(n)}$; $init$ reflects if the sequence $(s+e)^{(n)}$ is all zero $(init = false)$ or not $(init = true)$.

For $n = 0$ and for an $e_0 \in GF(q)$, the values of m, $D(X)$, d_m, $C(X)$ and $init$ at the end of the procedure can be easily verified to be correct for $(s+e)^{(0)}$.

We discuss the correctness of these values for an $n > 0$ and any error pattern $e_0, e_1, \ldots, e_n$ of weight at most k_0, provided that they are correct at the end of the recursion for $n-1$ and error pattern $e_0, e_1, \ldots, e_{n-1}$ of weight at most k_0.

Assuming the values of m, $D(X)$, d_m, $C(X)$ and *init* are correct for $(s+e)^{(n-1)}$, we prove that their values at the end of the next recursion are correct for $(s+e)^{(n)}$.

If $w_H(e_0, \ldots, e_{n-1}) > k_0$ then the recursion stops as the current error pattern is not eligible for calculating the k_0-error linear complexity profile.

If $\deg(C(X)) \geq L_k^*$ then the algorithm stops since any error pattern having as initial segment $(e_0, \ldots, e_{n-1})$ does not contribute to the result and can therefore be discarded.

If none of the stop criteria are met and the initialisation is not yet performed $(init = false)$ it means that the current $(s+e)^{(n-1)} = (\underbrace{0, 0, \ldots, 0}_{n-1 \text{ times}})$. There are two different situations in this case:

1. $e_n \leftarrow -s_n$, therefore $s_n + e_n = 0$ and the initialisation is delayed to one of the next steps, no change is needed on any of $m, D(X), d_m, C(X)$ and *init* remains $false$.

2. $e_n \leftarrow g$ for all $g \in GF(q) \backslash \{-s_n\}$ so that $s_n + e_n \neq 0$ and therefore the initialisation is performed. The values of the parameters can be immediately verified to be correct for $(s+e)^{(n)}$

   - $m \leftarrow n$
   - $D(X) \leftarrow 1$
   - $d_m \leftarrow (s+e)_m$
   - $C(X) \leftarrow X^{n+1}$
   - $init \leftarrow true$

If the initialisation has been performed $(init = true)$ then each possible value for e_n from $GF(q)$ is treated similarly, whether it means no error $(e_n = 0)$ or error $(e_n \in GF(q)^*)$. The discrepancy corresponding to term $s_n + e_n$ is calculated and the values of $m, D(X), d_m, C(X)$ are updated accordingly.

The correctness of the new values of $m, D(X), d_m, C(X)$ is implied by the correctness of Berlekamp-Massey Algorithm and of the initial values corresponding to $(s+e)^{(n-1)}$.

The previous considerations imply that the values $m, D(X), d_m, C(X)$ are correct for $(s+e)^{(t-1)}$ and since the minimum value is saved for each k less than or equal to k_0, each time a solution is obtained, the correctness of the resulting profile for the input sequence s and k_0 is immediate. □

We are now interested in an Efficient Exhaustive Search Algorithm for the L-constrained k-error linear complexity problem. Note that the L-constrained k-error linear complexity problem can be solved by checking for each integer k with

$k \leq k_0$, if there is any error pattern of weight at most k which produces a linear complexity less than or equal to L_0 on the input sequence and then taking the minimum such k. See listing 9 for the implementation of such an algorithm. With no changes, the $exhefR$ procedure could be used in algorithm 9 however some improvements can be made and we will show these in a new procedure, denoted $exhefRL$ (see listing 10).

Algorithm 9 Efficient Exhaustive Search Algorithm for the L-constrained k-error linear complexity problem

1: **Input:** A finite sequence $s = s_0, s_1, \ldots, s_{t-1}$ over GF(q); an integer k_0, with $0 < k_0 \leq w_H(s) - 1$; an integer L_0, with $0 \leq L_0 \leq t$.
2: **Output:** k^*, L^*, e^* and $C^*(X)$.
3: **for** $i = 0, 1, \ldots, k_0$ **do**
4: $\quad L_i^* \leftarrow L(s)$
5: $\quad C_i^*(X) \leftarrow C(X)$, the characteristic polynomial of s
6: $\quad e_i^* \leftarrow (\underbrace{0, 0, \ldots, 0}_{t \text{ times}})$
7: **end for**
8: $e \leftarrow (\underbrace{0, 0, \ldots, 0}_{t \text{ times}})$
9: $m \leftarrow 0$
10: $D(X) \leftarrow 1$
11: $d_m \leftarrow 1$
12: $n \leftarrow 0$
13: $C(X) \leftarrow \mathbf{1}$
14: $init \leftarrow false$
15: **call** $exhefRL(m, D(X), d_m, C(X), n, e, init)$
16: $k^* \leftarrow \min\{k | L_k^*(s) \leq L_0, 0 \leq k \leq k_0\}$
17: **if** $k^* > 0$ **then**
18: $\quad$ **return** k^*,L_k^*, $C_k^*(X)$ and e_k^*
19: **else**
20: $\quad$ **return** -1
21: **end if**

We can define an L_0-truncated k-error linear complexity profile as being the set $\{(i, L_i^{\sim}(s))|$ for all $0 \leq i \leq k_0\}$, where $L_i^{\sim}(s) = \begin{cases} L(s), & \text{when } L_i(s) > L_0 \\ L_i(s), & \text{otherwise.} \end{cases}$ and $L_i(s)$ is the i-error linear complexity of the sequence s. In an L_0-truncated k-error linear complexity we mean to ignore all the k-error linear complexities greater than L_0 and use a certain maximum value instead for these values of k, here we choose the maximum value to be $L(s)$. Having such a truncated profile, the minimal number of errors k^* for which the k-error linear complexity is less than or equal to L_0 can still be found by taking $k^* = \min\{i | L_i^{\sim}(s) \leq L_0\}$.

Example 3.5. *Suppose s is a binary sequence $s =$ 0110111101110101 of length 16. The linear complexity of s is 8 and the characteristic polynomial $C(X) =$*

$X^8 + X^6 + X^5 + X^4 + X + 1$.

The exact k-error linear complexity profile of s is:

$$\{(0,8),(1,7),(2,6),(3,4),(4,2),(5,1),(6,1),(7,1),(8,1),(9,1),(10,1),(11,0)\}.$$

The 4-truncated k-error linear complexity profile of s is the following, and even if the values of the 1-error or 2-error linear complexity are unknown we can conclude that 3 errors are necessary to lower the linear complexity of s to 4:

$$\{(0,8),(1,8),(2,8),(3,4),(4,2),(5,1),(6,1),(7,1),(8,1),(9,1),(10,1),(11,0)\}.$$

The Efficient Exhaustive Search Algorithm needs minimal changes (some recursive calls are unnecessary) in order to calculate an L_0-truncated k-error linear complexity profile as defined above and therefore solve the L-constrained k-error linear complexity profile.

Two stop conditions need to be added to the recursive procedure $exhefRL$ so that a truncated profile is returned (see listing 10).

The first condition ($L_n > L_0$) is necessary since once the linear complexity of $s + e$ reaches a value greater than L_0, there is no need to carry on with that error pattern e which will not contribute to the resulted L_0-truncated k_0-error linear complexity profile.

Secondly, when the found solution for a certain number of errors k is already smaller than the given threshold L_0 ($L_k^* < L_0$) the investigations on error patterns of weight greater than or equal to k should stop since it would not help with finding the minimum value k^* that we are interested in. For this reason, the algorithm should concentrate on error patterns of weight lower than the current k^* so if $k^* < k$ then the current error sequence should be abandoned.

See listings 9 and 10 where the changes are highlighted in bold.

Given the above remarks, the following correctness theorem is immediate.

Theorem 3.6. *The Efficient Exhaustive Search Algorithm for the L-constrained k-error linear complexity problem (listings 9 and 10) is correct.*

Proof. The correctness of the linear complexities calculated by the Efficient Exhaustive Search Algorithm has already been proven in theorem 3.4 for the k-error linear complexity profile problem. The new version of the procedure $exhefRL$ for the L-constrained k-error linear complexity problem (listing 10) has additional stop conditions which are included for efficiency purposes and which do not affect the correctness of the result. □

Algorithm 10 The $exhefRL(m, D(X), d_m, C(X), n, e, init)$ procedure

1: **procedure** EXHEFRL$(m, D(X), d_m, C(X), n, e, init)$
2: $k \leftarrow w_H(e)$
3: $L_n \leftarrow \deg(C(X))$
4: $L_m \leftarrow \deg(D(X))$
5: **if** $(n = t)$ or $(k > k_0)$ or $(L_k^* \leq L_n)$ or $(\mathbf{L_n > L_0})$ or $(\mathbf{L_k^* < L_0})$ or $(\mathbf{k^* < k})$ **then** ▷ Stop cond
6: **if** $((n = t)$ and $(k \leq k_0)$ and $(L_k^* > L_n))$ **then**
7: $(L_k^*, C_k^*(X), e_k^*) \leftarrow (L_n, C(X), e)$
8: $k^* \leftarrow \min\{i | L_i^*(s) \leq L_0\}$
9: **call** $adjustProfile(L_n, C(X), e, k)$
10: **end if**
11: **else**
12: **if** $init = false$ **then** ▷ Not initialised yet
13: $e_n \leftarrow -s_n$
14: **call** $exhefRL(m, D(X), d_m, C(X), n+1, e, false)$
15: **for all** $g \in GF(q) \backslash \{-s_n\}$ in increasing order relative to $\prec$ **do**
16: $e_n \leftarrow g$
17: $m \leftarrow n$
18: $D(X) \leftarrow 1$
19: $d_m \leftarrow (s+e)_m$
20: $C(X) \leftarrow X^{n+1}$
21: **call** $exhefRL(m, D(X), d_m, C(X), n+1, e, true)$
22: **end for**
23: **else** ▷ Already initialised
24: $E \leftarrow (k = k_0)?\{f_0\} : GF(q)$
25: **for all** $g \in E$ in increasing order relative to $\prec$ **do**
26: $e_n \leftarrow g$
27: $d_n \leftarrow (s+e)_n + \sum_{i=0}^{L_n - 1} c_n (s+e)_{i+n-L_n}$
28: **if** $d_n \neq 0$ **then**
29: **if** $2L > n$ **then**
30: $C(X) \leftarrow C(X) - \frac{d_n}{d_m} X^{(m-L_m)-(n-L_n)} D(X)$
31: **call** $exhefRL(m, D(X), d_m, C(X), n+1, e, true)$
32: **else**
33: $T(X) \leftarrow C(X)$
34: $C(X) \leftarrow X^{(n-L_n)-(m-L_m)} C(X) - \frac{d_n}{d_m} D(X)$
35: **call** $exhefRL(n, T(X), d_n, C(X), n+1, e, true)$
36: **end if**
37: **else**
38: **call** $exhefRL(m, D(X), d_m, C(X), n+1, e, true)$
39: **end if**
40: **end for**
41: **end if**
42: **end if**
43: **end procedure**

3.3 Algorithm analysis

In this section we formally analyse the computational complexity of the algorithms presented in sections 3.1 and 3.2.

For both problems considered, the Naïve Exhaustive Search Algorithm takes each of the different error patterns e of weight at most k_0 (there are $\sum_{i=0}^{k_0}(q-1)^i\binom{t}{i}$ possibilities) and computes the linear complexity values of the sequences $s+e$ using Berlekamp-Massey Algorithm which has computational complexity $\mathcal{O}(t^2)$, where t is the size of the input sequence. Therefore the order of the number of operations performed by Naïve Exhaustive Search Algorithm is $t^2\sum_{i=0}^{k_0}(q-1)^i\binom{t}{i}$.

We can broadly approximate:

$$\sum_{i=0}^{k_0}(q-1)^i\binom{t}{i} < (q-1)^{k_0}\sum_{i=0}^{k_0}\binom{t}{i}$$

There is no closed form for sums of the form $\sum_{i=0}^{k}\binom{n}{i}$, so we will use bounds:

Lemma 3.7 (Alecu, Sălăgean [2]). *The following bound stands*

$$\sum_{i=0}^{k}\binom{n}{i} \leq \begin{cases} 2\binom{n}{k}, & \text{if } k \leq \left\lfloor\frac{n+1}{3}\right\rfloor, \\ \left(k-\left\lfloor\frac{n+1}{3}\right\rfloor+2\right)\binom{n}{k}, & \text{if } \left\lfloor\frac{n+1}{3}\right\rfloor < k \leq \left\lfloor\frac{n-1}{2}\right\rfloor \end{cases}$$

Proof. The first case follows by induction on k, using the fact that $\binom{n}{k} = \binom{n}{k-1}\frac{n-k+1}{k}$ for all integers n and k, $k \neq 0$. Also $\frac{n-k+1}{k} \geq 2$ if $k \leq \lfloor(n+1)/3\rfloor$. The remaining inequalities follow from the first using elementary properties of the binomial coefficients. □

We can approximate binomial coefficients by using Stirling's approximation (see for example [15, Section 2.9]) which specifies that for $n \geq 3$ the following estimation stands

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{c_n}, \tag{3.1}$$

where c_n is such that $\frac{1}{12n+1} < c_n < \frac{1}{12n}$.

Since we need to estimate a binomial coefficient $\binom{n}{k} = \binom{n}{\lambda n}$ where $n \geq 3$ and

λn is an integer then

$$\begin{aligned}\binom{n}{\lambda n} &= \frac{n!}{(\lambda n)!(n(1-\lambda))!} \\ &= \frac{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{c_n}}{\sqrt{2\pi\lambda n}\left(\frac{\lambda n}{e}\right)^{\lambda n} e^{c_{\lambda n}}\sqrt{2\pi(1-\lambda)n}\left(\frac{(1-\lambda)n}{e}\right)^{(1-\lambda)n} e^{c_{(1-\lambda)n}}} \\ &= \frac{1}{\sqrt{2\pi}}\frac{1}{\sqrt{n}(1-\lambda)^{n(1-\lambda)+\frac{1}{2}}\lambda^{\lambda n+\frac{1}{2}}} e^{c_n - c_{\lambda n} - c_{(1-\lambda)n}}\end{aligned}$$

It follows that

$$\binom{n}{k} \approx \frac{c}{\sqrt{2\pi}}\frac{1}{\sqrt{n}(1-\lambda)^{n(1-\lambda)+\frac{1}{2}}\lambda^{\lambda n+\frac{1}{2}}} \tag{3.2}$$

where $n \geq 3$, $0 < k < n$, $\lambda = k/n$ and c is a constant such that, $e^{-\frac{2}{37}} \leq c \leq e^{\frac{1}{18}}$.

When assessing exponential time complexities of algorithms we will also use the following property (see for example Garey and Johnson [19]).

Property 3.8. *For any $a > 1$ and $i > 0$ the function $f(n) = n^i a^n \in \mathcal{O}((a+\varepsilon)^n)$ where $\varepsilon > 0$ is an arbitrarily small constant.*

We are now ready to estimate the complexity of the algorithms presented.

Theorem 3.9. *The worst case time complexity of the Naïve Exhaustive Search Algorithm for sequences over $GF(q)$ of length t and number of errors at most $k_0 = vt$ with $0 < v < 1/3$ is $\mathcal{O}(t\sqrt{t}\lambda^t)$ where $\lambda = \frac{(q-1)^v}{v^v(1-v)^{1-v}}$. This can also be expressed as $\mathcal{O}((\lambda+\varepsilon)^t)$ with $\varepsilon > 0$ an arbitrarily small constant. For a typical value of $v = 0.1$ (i.e. errors in at most 10% of the positions) on a binary sequence of length t (i.e. $q = 2$) the time complexity is $\mathcal{O}(t\sqrt{t}1.384145^t)$.*

Proof. For a sequence of length t over $GF(q)$ and for a number of errors k_0, the Naïve Exhaustive Search Algorithm computes the linear complexity (using the Berlekamp-Massey Algorithm) for $\sum_{i=0}^{k_0}(q-1)^i\binom{t}{i}$ sequences obtained by adding different error patterns to s. The Berlekamp-Massey Algorithm involves at most t^2 operations so the order of the number of operations is $t^2\sum_{i=0}^{k_0}(q-1)^i\binom{t}{i}$ (see algorithm 4 in section 3.1). Therefore, using lemma 3.7 the number of operations is at most $2t^2(q-1)^{k_0}\binom{t+1}{k_0+1}$.

Using relation (3.2) we obtain the following approximation:

$$\begin{aligned}
2(q-1)^{k_0}t^2\binom{t+1}{k_0+1} &= 2(q-1)^{k_0}\frac{t^2(t+1)}{k_0+1}\binom{t}{k_0} \\
&= 2(q-1)^{vt}\frac{t^2(t+1)}{vt+1}\binom{t}{vt} \\
&\approx \frac{c\sqrt{2}(q-1)^{vt}}{\sqrt{\pi}}\frac{t^2(t+1)}{vt+1}\frac{1}{\sqrt{t}(1-v)^{t(1-v)+\frac{1}{2}}v^{vt+\frac{1}{2}}} \\
&\approx \frac{c\sqrt{2}}{\sqrt{\pi}}\frac{t\sqrt{t}}{v\sqrt{v(1-v)}}\frac{(q-1)^{vt}}{(1-v)^{(1-v)t}v^{vt}} \\
&\approx \frac{c\sqrt{2}}{\sqrt{\pi}v\sqrt{v(1-v)}}t\sqrt{t}\left(\frac{(q-1)^v}{(1-v)^{(1-v)}v^v}\right)^t
\end{aligned}$$

which is $\mathcal{O}(t\sqrt{t}\lambda^t)$ where $\lambda = \frac{(q-1)^v}{v^v(1-v)^{1-v}}$.

□

For analysing the computational complexity of the Efficient Exhaustive Search Algorithm we will use the trees described earlier in figure 3.1. From the algorithm one can notice that these trees have the property that any path from the root to a leaf has at most k_0 branches labelled with a value $f \in GF(q)^*$.

Note that in terms of the Efficient Exhaustive Search Algorithm the label on each arc between level n and $n+1$ represents e_n in a certain error pattern considered by the algorithm.

We estimate their number of nodes using the following Lemma (the case $q = 2$ was treated in Alecu and Sălăgean [2]).

Lemma 3.10. *Consider a tree of depth t where any node has at most q children and the edges to these children are labeled with values from a finite field $GF(q) = \{f_0 = 0, f_1, \ldots, f_{q-1}\}$ like in figure 3.1. If on any path from a root to a leaf the number of non zero labels is at most k_0 then the tree has at most $\sum_{i=0}^{k_0}(q-1)^i\binom{t+1}{i+1}$ nodes.*

Proof. We associate to each node a description of the path from the root to that node using the labels of the edges, i.e. each node on level n is characterised by a sequence of length n over the alphabet $\{f_0 = 0, f_1, \ldots, f_{q-1}\}$. Also we denote the levels in the tree from 0 to t, 0 being the root level and t being the maximum level.

With these notations, let us compute the maximum number of nodes on each of the levels of the tree.

There are two situations depending on the value of the level n.

Firstly, if $1 \leq n \leq k_0$, then the number of nodes on the level n is equal to q^n. Because the level is less or equal to k_0, any path from root to the nodes on the level n would have at most k_0 edges with non zero labels. Therefore the tree is actually a complete tree up to level n.

Secondly, if $k_0 + 1 \leq n \leq t$ then the number of nodes on the level n is equal to the number of sequences of length n with at most k_0 non zero terms, i.e. from the set $GF(q)^* = \{f_1, f_2, \ldots, f_{q-1}\}$. Therefore it can be represented as:

$$\sum_{i=0}^{k_0} (q-1)^i \binom{n}{i}.$$

We note that even in the first situation when $1 \leq n \leq k_0$, since the convention is that $\binom{n}{k_0} = 0$, we can express the number of nodes as:

$$\sum_{i=0}^{k_0} (q-1)^i \binom{n}{i} = \sum_{i=0}^{n} (q-1)^i \binom{n}{i} = (1+q-1)^n = q^n$$

Therefore we can unify the two formulas for all levels n, where $n = 1, 2, \ldots, k$ to obtain the total number of nodes in the tree, except the root:

$$\sum_{n=1}^{t} \sum_{i=0}^{k_0} (q-1)^i \binom{n}{i} = \sum_{i=0}^{k_0} (q-1)^i \sum_{n=1}^{t} \binom{n}{i} = \sum_{i=0}^{k_0} (q-1)^i \sum_{n=i}^{t} \binom{n}{i} = \sum_{i=0}^{k_0} (q-1)^i \binom{t+1}{i+1}.$$

Since the number of edges in a tree is equal to the number of nodes minus the root, it means that the number of vertices in the tree is equal to:

$$\sum_{i=0}^{k_0} (q-1)^i \binom{t+1}{i+1}.$$

□

Theorem 3.11. *The worst case time complexity of the Efficient Exhaustive Search Algorithm for sequences over $GF(q)$ of length t and number of errors at most $k_0 = vt$ with $0 < v < 1/3$ is $\mathcal{O}(\sqrt{t}\lambda^t)$ where $\lambda = \frac{(q-1)^v}{v^v(1-v)^{1-v}}$. This can also be expressed as $\mathcal{O}((\lambda+\varepsilon)^t)$ with $\varepsilon > 0$ an arbitrarily small constant. For a typical value of $v = 0.1$ (i.e. errors in at most 10% of the positions) on a binary sequence of length t (i.e. $q = 2$) the time complexity is $\mathcal{O}(\sqrt{t}1.384145^t)$.*

Proof. A run of the Efficient Exhaustive Search Algorithm can be represented as a tree of depth t and at most k_0 branches labelled with non zero values on any path from the root to a leaf. Using Lemma 3.10, this tree will have at most $\sum_{i=0}^{k_0} (q-1)^i \binom{t+1}{i+1}$ nodes. So the number of nodes is bounded by $2(q-1)^{k_0} \binom{t+1}{k_0+1}$,

from Lemma 3.7. For any node, the algorithm computes a discrepancy and possibly adjusts the characteristic polynomial, so there are $\mathcal{O}(t)$ computational steps. Therefore the complexity is $\mathcal{O}(t(q-1)^{k_0}\binom{t+1}{k_0+1})$.

Using (3.2) we obtain the following approximation:

$$\begin{aligned}
2(q-1)^{k_0}t\binom{t+1}{k_0+1} &= 2(q-1)^{k_0}\frac{t(t+1)}{k_0+1}\binom{t}{k_0} \\
&= 2(q-1)^{vt}\frac{t(t+1)}{vt+1}\binom{t}{vt} \\
&\approx \frac{c\sqrt{2}(q-1)^{vt}}{\sqrt{\pi}}\frac{t(t+1)}{vt+1}\frac{1}{\sqrt{t}(1-v)^{t(1-v)+\frac{1}{2}}v^{vt+\frac{1}{2}}} \\
&\approx \frac{c\sqrt{2}}{\sqrt{\pi}}\frac{\sqrt{t}}{v\sqrt{v(1-v)}}\frac{(q-1)^{vt}}{(1-v)^{(1-v)t}v^{vt}} \approx \\
&\approx \frac{c\sqrt{2}}{\sqrt{\pi}v\sqrt{v(1-v)}}\sqrt{t}\left(\frac{(q-1)^v}{(1-v)^{(1-v)}v^v}\right)^t
\end{aligned}$$

which is $\mathcal{O}(\sqrt{t}\lambda^t)$ where $\lambda = \frac{(q-1)^v}{v^v(1-v)^{1-v}}$. □

The Efficient Exhaustive Algorithm is used throughout this thesis as a reference for the accuracy of the algorithms that we design.

3.4 Conclusion

We have clearly defined the problems that are trying to solve in this thesis: the k-error linear complexity problem, the k-error linear complexity profile problem and the L-constrained k-error linear complexity problem.

We present and analyze two exhaustive search techniques for computing the k-error linear complexity of sequences over finite fields. Both the Naïve Exhaustive Search Algorithm and Efficient Exhaustive Search Algorithm are general and exact algorithms. The computational complexity is exponential for both algorithms, however important time improvements with no accuracy costs are registered in Efficient Exhaustive Search Algorithm.

Efficient Exhaustive Search Algorithm will be used throughout this thesis to compute the exact values of the k-error linear complexity when evaluating the heurstic algorithms presented in chapters 4 and 5.

Chapter 4

Modified Berlekamp-Massey Algorithm

In this chapter, we present heuristic algorithms which approximate the solution of the k-error linear complexity profile problem and the L-constrained k-error linear complexity problem for general finite sequences over a finite field by adapting the Berlekamp-Massey Algorithm (Berlekamp [4], Massey [42]). The method explores only some of all the possible error sequences, the choice of the positions of the errors being guided by the steps of the Berlekamp-Massey Algorithm in which the complexity is increased.

4.1 Introduction

In chapter 3 we described the exhaustive techniques for solving the k-error linear complexity profile and the L-constrained k-error linear complexity problems. These techniques process all the possible error patterns in order to obtain an exact result.

A heuristic approach would only explore a subset of all the possible error patterns. We investigate a method of choosing these selected few error patterns in such way to maximise the accuracy of the result. The heuristic in this chapter uses the Berlekamp-Massey Algorithm to choose these patterns. Namely, during the algorithm (see section 2.2.6), only the case when the discrepancy is not zero $d^{(n)} \neq 0$ and when the current linear complexity is less than or equal to half the length of the sequence, $2L^{(n)} \leq n$ (case (1b)) yields an increase in the current complexity of the sequence. It therefore seems natural to concentrate on what would happen if the current term of the sequence, which creates this increase in complexity, would be changed in such a way as to make the discrepancy $d^{(n)}$ zero, and therefore make an increase in complexity unnecessary. If we introduce

these changes to the sequence early in the algorithm, we would soon run out of the k_0 allowed errors, and we would not be able to explore the effect of errors on later terms of the sequence. Whenever case (1*b*) occurs in the algorithm we do therefore consider both possibilities: changing the current term of the sequence, or not changing it, and we continue exploring both branches. This approach will therefore still have an exponential complexity, but will substantially decrease the number of error patterns investigated, with the savings becoming larger for fields of larger cardinality.

In the case of fields of larger cardinality, this approach has the advantage that even if the field has more than two elements, there are still only two choices that are investigated: introducing no error, or introducing an error of magnitude $-d^{(n)}$, where $d^{(n)}$ is the discrepancy. An exhaustive search approach (for example Efficient Exhaustive Search Algorithm described in section 3.2) would have to investigate all the possible error magnitudes along with the zero value for each error position, i.e. q possibilities for a field of q elements. The computational complexity of the heuristic method is further discussed in Section 4.3.

Our approach is not guaranteed to give the exact result of each k-error linear complexity, as the error pattern that decreases the complexity the most may well not have the errors in those positions suggested by the Berlekamp-Massey Algorithm. Since we investigate only some of all the possible error patterns and the k-error linear complexity is defined as a minimum over the set of all error patterns (see definition 2.13), our results are always larger than or equal to the exact ones. Unfortunately we were unable to prove a bound on the approximation quality. Therefore, we investigate experimentally in Section 4.4 how close the approximation is to the exact values composing the k-error linear complexity profile.

We firstly illustrate our algorithm with an example:

Example 4.1. *Suppose we take a binary sequence* $s = 0110111101110101$ *of length 16. Table 4.1 shows the intermediate results of the Berlekamp-Massey Algorithm for this sequence. The resulting linear complexity is 8 and the characteristic polynomial* $C(X) = X^8 + X^6 + X^5 + X^4 + X + 1$.

Figure 4.1 shows the tree of recursive calls which would be considered as suggested by the Berlekamp-Massey Algorithm, calls which correspond to the situations when the current linear complexity needs to be increased. The internal nodes and the root in figure 4.1 show the current position in the sequence and the current linear complexity at the moment when a raise in the linear complexity is needed. The left child of each internal node corresponds to not forcing an error and the right child corresponds to introducing the error $-d^{(n)}$ *such that the discrepancy becomes zero. The leaves in the tree show the final result on each path in the tree:*

Table 4.1: Intermediate results for the Berlekamp-Massey Algorithm applied to the sequence $s = 0110111101110101$

n	s_{n-1}	d_n	L_n	d_m	m	L_m	$C_m(X)$	$C_n(X)$
1	0	-	0	-	-	-	-	1
2	1	-	2	1	1	0	1	X^2
3	1	1	2	1	1	0	1	X^2+X
4	0	0	2	1	1	0	1	X^2+X+1
5	1	0	2	1	1	0	1	X^2+X+1
6	1	1	2	1	1	0	1	X^2+X+1
7	1	0	5	1	6	2	X^2+X+1	$X^5+X^4+X^3+1$
8	1	0	5	1	6	2	X^2+X+1	$X^5+X^4+X^3+1$
9	0	1	5	1	6	2	X^2+X+1	$X^5+X^4+X^3+1$
10	1	0	5	1	6	2	X^2+X+1	$X^5+X^4+X^3+X^2+X$
11	1	0	5	1	6	2	X^2+X+1	$X^5+X^4+X^3+X^2+X$
12	1	1	5	1	6	2	X^2+X+1	$X^5+X^4+X^3+X^2+X$
13	0	1	8	1	12	5	$X^5+X^4+X^3+X^2+X$	$X^8+X^7+X^6+X^5+X^4+X^2+X+1$
14	1	1	8	1	12	5	$X^5+X^4+X^3+X^2+X$	$X^8+X^3+X^2+X+1$
15	0	0	8	1	12	5	$X^5+X^4+X^3+X^2+X$	$X^8+X^6+X^5+X^4+X+1$
16	1	0	8	1	12	5	$X^5+X^4+X^3+X^2+X$	$X^8+X^6+X^5+X^4+X+1$

the number of errors which were introduced and the corresponding k-error linear complexity for that error pattern. In Table 4.1 we can see that the first change of complexity happens when the term $s_6 = 1$ is processed. At this moment two paths need to be taken, either an error is introduced and the linear complexity and characteristic polynomial remain unchanged (corresponding to the right subtree), or no error is introduced and the algorithm carries on just as in the classic Berlekamp-Massey Algorithm (left subtree). Having the path of subsequent decisions to get to a certain solution (represented by a leaf in the tree), the error sequence can be built using a bottom-up technique. For example, if we need to rebuild the error sequence corresponding to the found 3-error linear complexity ($L_3(s) = 5$), this one will contain 1's in the positions 14,13 and 12, corresponding to the error sequence 0000000000001110. *Note that there is a second solution contributing to the 3-error linear complexity ($L_3(s) = 13$), corresponding to an error sequence with 1's in the positions 14, 9 and 8 but this is not the optimum value.*

By taking the minimum value of the linear complexity for each number of errors, the results in the tree in Figure 4.1 give an incomplete k-error linear complexity profile $\{(0,8),(1,9),(2,7),(3,5),(5,2)\}$. *Applying the monotony property of the k-error linear complexity (property 2.14) and the fact that $L_{w_H(s)}(s) = 0$ for any sequence s (property 2.15) an approximation of the full k-error linear complexity profile can be found:*

$$\{(0,8),(1,8),(2,7),(3,5),(4,5),(5,2),(6,2),(7,2),(8,2),(9,2),(10,2),(11,0)\}.$$

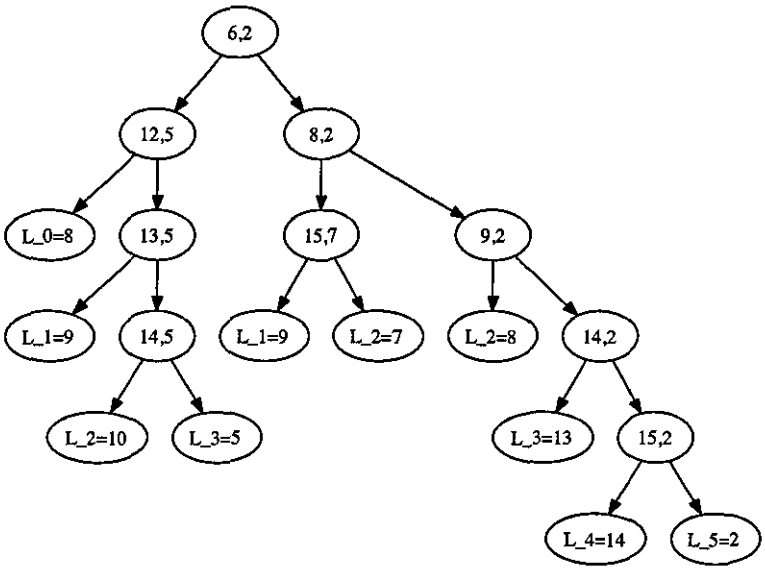

Figure 4.1: Example of the Modified Berlekamp-Massey Algorithm tree of error and no-error recursive calls for the sequence $s = 0110111101110101$

The exact k-error linear complexity profile for the sequence s is:

$$\{(0,8),(1,7),(2,6),(3,4),(4,2),(5,1),(6,1),(7,1),(8,1),(9,1),(10,1),(11,0)\}.$$

One can notice that the approximation is very close to the exact values even though the difference in the number of error patterns processed is significant (10 error patterns for the proposed approach and $\sum_{i=0}^{11}\binom{16}{i} = 33215$ for the Naïve Exhaustive Search Algorithm).

4.2 Modified Berlekamp-Massey Algorithm

Based on the considerations in section 4.1 we will describe an implementation, denoted Modified Berlekamp-Massey Algorithm, which outputs an approximation of the k-error linear complexity profile problem and which is based on the recursive version of the Berlekamp-Massey Algorithm (see listing 2 and 3 in section 2.2.6).

For a sequence s and an integer k_0 with $0 < k_0 \leq w_H(s)$, the approximate k_0-error linear complexity profile of s returned by the Modified Berlekamp-Massey Algorithm contains for each $i = 0, 1, \ldots, k_0$: L_i^*, the approximate i-error linear complexity; e_i^*, the error pattern producing the linear complexity L_i^* on s; $C_i^*(X)$ a minimal characteristic polynomial corresponding to the sequence $s + e_i^*$.

The algorithm initially locates the position of the first non zero term. This cannot coincide with the end of the sequence since the all zero sequence is not a valid input sequence.

Since the k-error linear complexities in the profile are minimum values, the k-error linear complexity profile is initialised with the maximum possible value, namely $L_i^* = L(s)$, $C_i^*(X) = C(X)$ and $e = (\underbrace{0,0,\ldots,0}_{t \text{ times}})$, for all $i = 0,1,\ldots,k_0$, where $L(s)$ is the linear complexity of s and $C(X)$ is a minimal characteristic polynomial of the input sequence s, i.e. $\deg(C(X)) = L(s)$.

The significant changes from the Berlekamp-Massey Algorithm are in the recursive procedure, denoted now *mbmR* (listing 12). See listing 3 in section 1 for procedure *bmR*. The parameters of the procedure *mbmR* are:

- e, the current error sequence of weight k ($w_H(e) = k$).
- n, the current position in the sequence.
- $C(X)$, the current characteristic polynomial ($L_n = \deg(C(X))$).
- $D(X)$, the characteristic polynomial at the last change in degree ($L_m = \deg(D(X))$).
- m, the position where the last change in degree occurred ($m < n$, $L_m < L_{m+1}$ and $L_{m+1} = L_n$).
- d_m, the discrepancy value at the time of the last change in degree.

The procedure works in a similar way with the recursive version of the Berlekamp-Massey Algorithm, processing the sequence $s + e$, where s is the input sequence and e is an error pattern. The error pattern is built by considering at every position n where $2L_n \leq n$ and $d_n \neq 0$, two alternatives for the current e_n value, $e_n = 0$, corresponding to introducing no error, and $e_n = -d_n$, the reverse of the discrepancy value, corresponding to introducing an error.

I_n (line 21 in algorithm 12) is the unit vector of size n having all terms zero except for the position n where the term is 1 ($I_n = (0,0,\ldots,1,\ldots,0)$). Therefore $e - d_n I_n = (e_0, e_1, \ldots, e_{n-1}, e_n - d_n, e_{n+1}, \ldots, e_{t-1})$.

The stop condition is a disjunction of conditions. If the end of the sequence has been reached ($n = t$) then it means an error pattern e of weight k at most k_0 has been processed and if $L(s+e)$ is smaller than the currently stored L_k^*, the new solution needs to be saved.

If the current error pattern has a weight larger than k_0 ($k > k_0$) then that path needs to be discarded since it does not contribute to the k_0-error linear complexity profile.

Additionally, some of the paths taken by the recursion calls might get to an error pattern e such that $L(s+e)$ is greater than or equal to the currently stored

Algorithm 11 Modified Berlekamp-Massey Algorithm for the k-error linear complexity profile problem

1: **Input:** A finite sequence $s = s_0, s_1, \ldots, s_{t-1}$; an integer k_0 with $0 < k_0 \leq w_H(s) - 1$;
2: **Output:** The approximate k-error linear complexity profile, L_i^*, $C_i^*(X)$ and e_i^*, for all $i = 0, 1, \ldots, k_0$.
3: $n_z \leftarrow 0$
4: **while** $s_{n_z} = 0$ and $n_z < t$ **do** ▷ go over the initial zeros
5: $\quad n_z \leftarrow n_z + 1$
6: **end while**
7: **for** $i = 0, 1, \ldots, k_0$ **do**
8: $\quad L_i^* \leftarrow L(s)$
9: $\quad C_i^*(X) \leftarrow C(X)$, a minimal characteristic polynomial
10: $\quad e_i^* \leftarrow (\underbrace{0, 0, \ldots, 0}_{t \text{ times}})$
11: **end for**
12: $k \leftarrow 0$
13: $e = (\underbrace{0, 0, \ldots, 0}_{t \text{ times}})$
14: $m \leftarrow n_z$
15: $D(X) \leftarrow 1$
16: $d_m \leftarrow s_{n_z}$
17: $n_z \leftarrow n_z + 1$
18: $C(X) \leftarrow X^{n_z}$
19: **if** $n_z \leq t - 1$ **then** ▷ otherwise $k_0 = w_H(s) - 1 = 0$
20: $\quad$ **call** $mbmR(m, D(X), d_m, C(X), n_z, e)$
21: **end if**
22: **return** L_i^*, $C^*(X)_i$ and e_i^*, for all $i = 0, 1, \ldots, k_0$

solution for that number of errors k (L_k^*). Continuing the investigation on that recursion path is of no use as it will not give a better solution. The condition ($L_k^* \leq L_n$) can therefore be added to avoid these unnecessary recursive calls.

Finally, we apply an adjustment after each update of the current approximation of the k-error linear complexity profile, in order to maintain its monotony property. See procedure *adjustProfile* and the associated discussion in section 3.2, listing 8.

Additionally, we can combine iteration and recursion in order to minimise the stack size. Since the recursive calls are useful only when there is a decision to force or not an error, a level of iteration can be introduced for the other cases (lines 12-32 in Algorithm 13). A boolean flag is used to indicate when to reiterate and when there is need for a recursive call (lines 11, 19 and 30 in Algorithm 13). This last remark is implemented in procedure *mbmROpt* from listing 13.

As opposed to the Efficient Exhaustive Search Algorithm, the algorithm presented in this section only considers a subset of all the possible error patterns $e \in GF(q)^t$. These error patterns $e = (e_0, e_1, \ldots, e_{t-1})$ are such that for each

Algorithm 12 The $mbmR$ procedure

procedure MBMR$(m, D(X), d_m, C_n(X), n, e)$
  $k \leftarrow w_H(e)$
  $L_m \leftarrow \deg(D(X))$
  $L_n \leftarrow \deg(C(X))$
  if $(n = t)$ or $(k > k_0)$ or $(L_k^* \leq L_n)$ **then**
    if $((n = t)$ and $(k \leq k_0)$ and $(L_k^* > L_n))$ **then**
      $(L_k^*, C_k^*(X), e_k^*) \leftarrow (L_n, C(X), e)$
      call $adjustProfile(L_n, C(X), e, k)$
    end if
  else
    $d_n \leftarrow (s+e)_n + \sum_{i=0}^{L_n-1} c_i (s+e)_{i+n-L_n}$
    if $d_n \neq 0$ **then**
      if $2L_n > n$ **then** ▷ (1a) the complexity does not change
        $C(X) \leftarrow C(X) - \frac{d_n}{d_m} X^{(m-L_m)-(n-L_n)} D(X)$
        call $mbmR(m, D(X), d_m, C(X), n+1, e)$
      else ▷ (1b) the complexity does change
        $T(X) \leftarrow C(X)$
        $C(X) \leftarrow X^{(n-L_n)-(m-L_m)} C(X) - \frac{d_n}{d_m} D(X)$
        call $mbmR(n, T(X), d_n, C(X), n+1, e)$
        if $k < k_0$ **then**
          call $mbmR(m, D(X), d_m, C(X), n+1, (e - d_n I_n))$
        end if
      end if
    else ▷ (2) the current characteristic polynomial does not change
      call $mbmR(m, D(X), d_m, C(X), n+1, e)$
    end if
  end if
end procedure

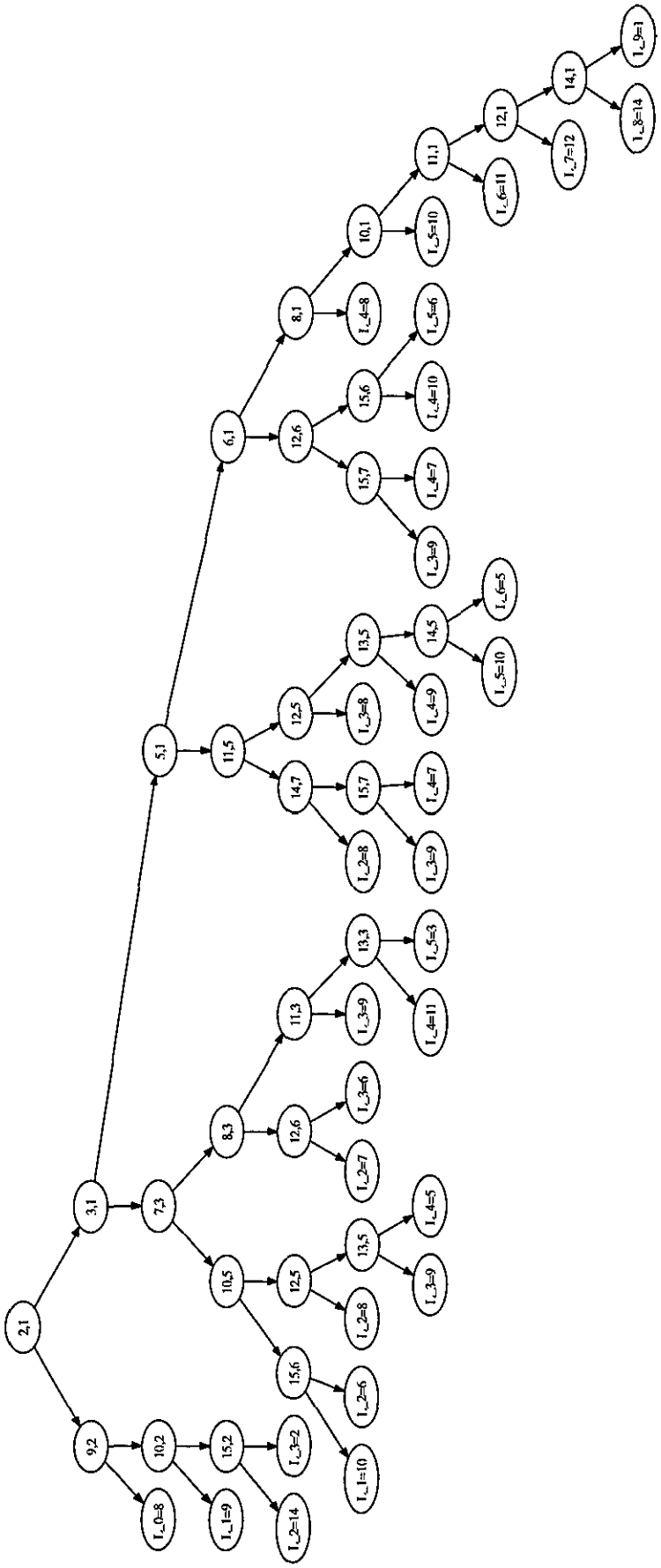

Figure 4.2: Solution tree generated with the Modified Berlekamp-Massey Algorithm for the sequence $s = 1011011010111010$

Algorithm 13 The *mbmROpt* procedure - Optimised version

1: **procedure** MBMROPT($m, D(X), d_m, C(X), n, e$)
2: $k \leftarrow w_H(e)$
3: $L_m \leftarrow \deg(D(X))$
4: $L_n \leftarrow \deg(C(X))$
5: **if** $(n = t)$ or $(k > k_0)$ or $(L_k^* \leq L_n)$ **then**
6: **if** $((n = t)$ and $(k \leq k_0)$ and $(L_k^* > L_n))$ **then**
7: $(L_k^*, C_k^*(X), e_k^*) \leftarrow (L_n, C(X), e)$
8: **call** $adjustProfile(L_n, C(X), e, k)$
9: **end if**
10: **else**
11: $loopFlag \leftarrow true$
12: **repeat**
13: $d_n \leftarrow (s + e)_n + \sum_{i=0}^{L_n - 1} c_i (s + e)_{i+n-L_n}$
14: **if** $d_n \neq 0$ **then**
15: **if** $2L_n > n$ **then** ▷ (1a)
16: $C(X) \leftarrow C(X) - \frac{d_n}{d_m} X^{(m-L_m)-(n-L_n)} D(X)$
17: $n \leftarrow n + 1$
18: **else** ▷ (1b)
19: $loopFlag \leftarrow false$
20: $T(X) \leftarrow C(X)$
21: $C(X) \leftarrow X^{(n-L_n)-(m-L_m)} C(X) - \frac{d_n}{d_m} D(X)$
22: **call** $mbmROpt(n, T(X), d_n, C(X), n + 1, e)$
23: **if** $k < k_0$ **then**
24: **call** $mbmROpt(m, D(X), d_m, C(X), n + 1, (e - d_n I_n))$
25: **end if**
26: **end if**
27: **else** ▷ (2)
28: $n \leftarrow n + 1$
29: **end if**
30: **until** $loopFlag = false$ or $n = t$
31: **end if**
32: **if** $((n = t)$ and $(k \leq k_0)$ and $(L_k^* > L_n))$ **then**
33: $(L_k^*, C_k^*(X), e_k^*) \leftarrow (L_n, C(X), e)$
34: **call** $adjustProfile(L_n, C(X), e, k)$
35: **end if**
36: **end procedure**

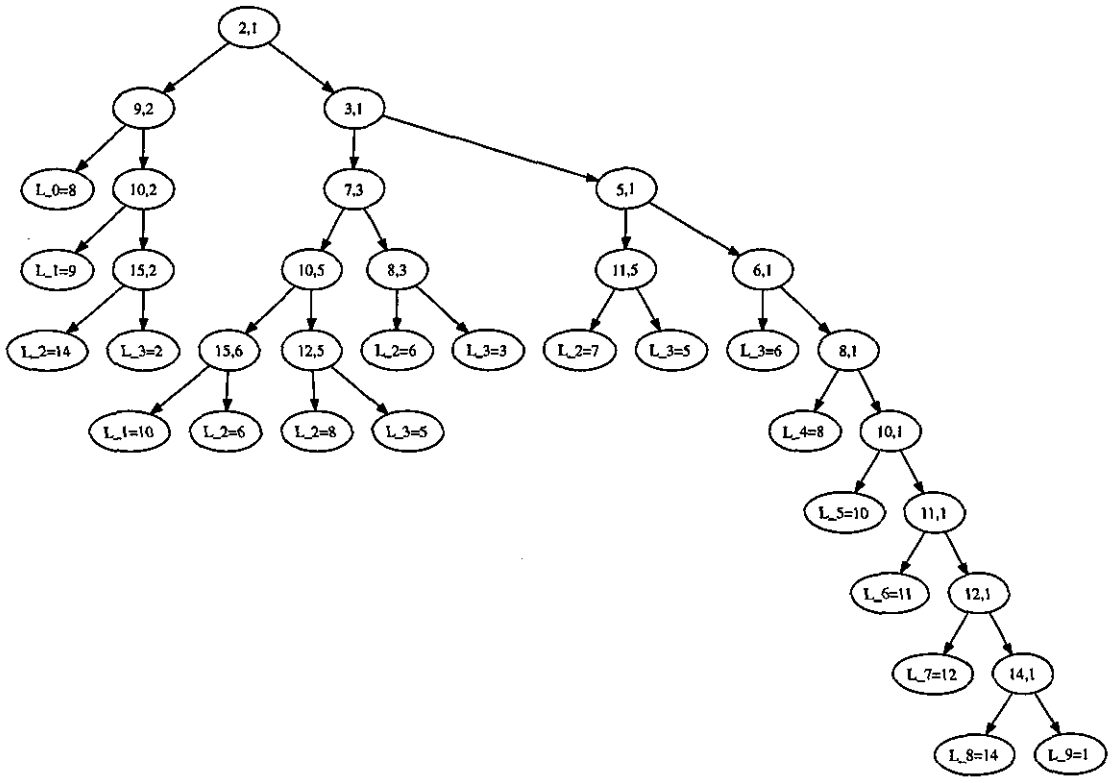

Figure 4.3: Solution tree generated with the optimised Modified Berlekamp-Massey Algorithm for the sequence $s = 1011011010111010$

$i = 0, 1, \ldots, t-1$, either $e_i = 0$ or $e_i = -d_i$ where d_i is the discrepancy when processing term i for the sequence $(s+e)^{(i)} = (s_0 + e_0, s_1 + e_1, \ldots, s_i + e_i)$.

As already shown, the set of error patterns processed by the Modified Berlekamp-Massey Algorithm can be represented hierarchically as a binary tree. In this tree each internal node (n, L_n) represents a moment in the algorithm when case (1b) from Berlekamp-Massey Algorithm is encountered ($2L_n \leq n$ and $d_n \neq 0$) (see section 2.2.6), where the discrepancy, d_n is calculated for the sequence $s+e$. Each of the two branches of a non-final node will correspond to one of the values which can be considered for e_i, namely 0 or $-d_i$. Some of the indices do not appear in the tree and that means that the error term on those positions is zero for the corresponding error pattern considered by the algorithm.

This tree representation is useful not only for visualisation purposes but also since it enables us to describe and estimate the number of error patterns considered by the algorithm. The number of error patterns considered is the number of leaves in the tree (see section 4.3 for the estimation).

For instance, the tree in Example 4.1 in section 4.1 shows that 10 error patterns have been considered and that these are: 0000000000000000, 0000000000001000, 0000000000001100, 0000000000001110, 0000001000000000, 0000001010000000, 0000001000000001, 0000001011000000, 0000001011000010 and 0000001011000011.

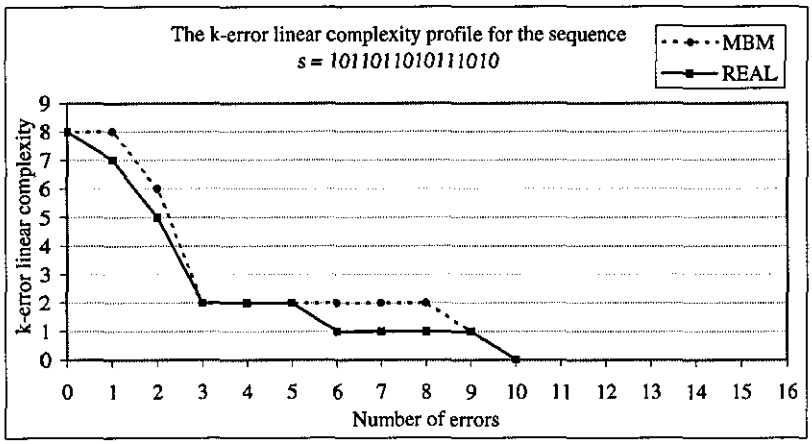

Figure 4.4: The k-error linear complexity profile for the sequence $s = 1011011010111010$

In order to justify the optimisations we have made to the algorithm, namely the stop condition ($L_k^* \leq L_n$), the procedure *adjustProfile* and the mix of recursion and iteration, we include the following example.

Example 4.2. *Consider the binary sequence* $s = 1011011010111010$ *of length 16. For the sequence* s *and no limitations on the number of errors* ($k_0 = w_H(s)$) *or the complexity* ($L_0 = t$), *the number of nodes in the tree of recursive calls decreases from 61 nodes to 37 nodes for the optimised version with no loss in accuracy (see figures 4.2 and 4.3 for a graphic representation of the trees).*

Figure 4.4 shows the difference between the approximation found by the Modified Berlekamp-Massey Algorithm and the exact values of the k-error linear complexity for the same sequence.

In theorem 4.3 we prove that the approximation algorithm Modified Berlekamp-Massey Algorithm is correct according to definition 3.1.

Theorem 4.3. *The Modified Berlekamp-Massey Algorithm (listings 11 and 13) is correct.*

Proof. We can prove by induction on n, where $n = n_z, n_z + 1, \ldots, t - 1$ (n_z is the position of the first non zero term in sequence s) that for each error sequence $e = e_0, e_1, \ldots, e_n$ with $w_H(e) \leq k_0$ processed by the algorithm, the values of m, $D(X)$, d_m, $C(X)$ calculated by the recursive procedure *mbmROpt* are correct for the sequence $(s+e)^{(n)} = (s_0+e_0, \ldots, s_n+e_n)$. Namely, m is the last change index, i.e. the maximum value such that $m < n$ and $L((s+e)^{(m-1)}) < L((s+e)^{(m)} = L((s+e)^{(n-1)}$; $D(X)$ and d_m are the characteristic polynomial and the discrepancy corresponding to the last change term m; $C(X)$ is a minimal characteristic polynomial for $(s+e)^{(n)}$.

For $n = n_z$ and for $e = (\underbrace{0, \ldots, 0}_{n_z \text{ times}})$ the initial values given to $m = n_z$, $D(X) = 1$, $d_m = s_{n_z}$, $C(X) = X^{n_z+1}$ are correct for $(s+e)^{(n_z)} = (0, \ldots, 0, s_{n_z})$.

Suppose that for a certain error pattern $e = e_0, e_1, \ldots, e_{n-1}$ the values m, $D(X)$, d_m, $C(X)$ are correct for $(s+e)^{(n-1)}$. We show that the values m, $D(X)$, d_m, $C(X)$ remain correct after a run of the procedure *mbmROpt*.

If $n = t$, $w_H(e) > k_0$ or $\deg(C(X)) \geq L_k^*$ then the recursion stops so the values remain unchanged.

If none of the conditions above are met then the discrepancy d_n is calculated. If $d_n \neq 0$ two cases are considered:

- If $2L_n > n$ then the last change polynomial $D(X)$, the last change index, m and discrepancy d_m remain the same. The current characteristic polynomial changes but its degree is the same, see line 14 in algorithm 12.
- If $2L_n \geq n$ then two further situations are considered:
  - Do not introduce an error, $e_n \leftarrow 0$, the last change polynomial becomes the current polynomial $D(X) \leftarrow C(X)$, the last change index, $m \leftarrow n$ and the corresponding discrepancy $d_m \leftarrow d_n$. The current characteristic polynomial and its degree change, see line 15 in algorithm 13. In this case, $L_n \leftarrow \deg(C(X)) = n - L_n + 1$.
  - Do introduce an error, $e_n \leftarrow -d_n$, on the term s_n then the discrepancy of the sequence $s+e$ at position n becomes 0 so there is no need for a change. All parameters $D(X), m, d_m, C(X)$ remain unchanged.

Finally if $d_n = 0$ then there is no need for a change so all the values of the parameters $D(X), m, d_m, C(X)$ remain unchanged.

It follows that the values of $C(X)$ are minimal characteristic polynomials corresponding to $(s+e)^{(t-1)}$ for all the error patterns e of different weights processed by the algorithm. Since the best values are saved for each k less than or equal to k_0 the correctness of the resulting profile is immediate. □

A version of the Modified Berlekamp-Massey Algorithm for the L-constrained k-error linear complexity problem can be immediately obtained (listings 14 and 15) by outputting $k^* = \min\{k | L_k^* \leq L_0\}$ with the corresponding error pattern, characteristic polynomial and linear complexity, instead of outputting the full k-error linear complexity.

Additionally, for optimisation reasons we can add a set of stop conditions to the recursion such that some of the recursive calls which are unnecessary to solving this problem are avoided. See section 3.2 in Efficient Exhaustive Search Algorithm

Algorithm 14 Modified Berlekamp-Massey Algorithm for the L-constrained k-error linear complexity problem

Input: A finite sequence $s = s_0, s_1, \ldots, s_{t-1}$; an integer k_0 with $0 < k_0 \leq w_H(s) - 1$; an integer L_0 with $0 \leq L_0 \leq t$
Output: The approximate value k^* with corresponding L^*, $C^*(X)$ and e^* such that k^* is minimum with $L_{k^*}(s) \leq L_0$.
$n_z \leftarrow 0$
while $s_{n_z} = 0$ and $n_z < t$ **do** ▷ go over the initial zeros
    $n_z \leftarrow n_z + 1$
end while
for $i = 0, 1, \ldots, k_0$ **do**
    $L_i^* \leftarrow L(s)$
    $C_i^*(X) \leftarrow C(X)$, a minimal characteristic polynomial
    $e_i^* \leftarrow (\underbrace{0, 0, \ldots, 0}_{t \text{ times}})$
end for
$k \leftarrow 0$
$e = (\underbrace{0, 0, \ldots, 0}_{t \text{ times}})$
$m \leftarrow n_z$
$D(X) \leftarrow 1$
$d_m \leftarrow s_{n_z}$
$n_z \leftarrow n_z + 1$
$C(X) \leftarrow X^{n_z}$
if $n_z \leq t - 1$ **then** ▷ otherwise $k_0 = w_H(s) - 1 = 0$
    call $mbmRL(m, D(X), d_m, C(X), n_z, e)$
end if
$k^* \leftarrow \min\{k | L_k^*(s) \leq L_0, 0 \leq k \leq k_0\}$
if $k^* > 0$ **then**
    return k^*,L_k^*, $C_k^*(X)$ and e_k^*
else
    return -1
end if

for L-constrained k-error linear complexity problem, for the explanation of these additional stop conditions.

4.3 Algorithm analysis

In this section we formally analyse the computational complexity of the Modified Berlekamp-Massey Algorithm.

For analysing the complexity of the Modified Berlekamp-Massey Algorithm we will use the recursion trees as described in section 4.2 and estimate their number of nodes using lemmas 3.7 and 3.10 from chapter 3.

For the Modified Berlekamp Massey Algorithm it is harder to estimate the

Algorithm 15 The *mbmRL* procedure

procedure MBMRL($m, D(X), d_m, C(X), n, e$)
  $k \leftarrow w_H(e)$
  $L_m \leftarrow \deg(D(X))$
  $L_n \leftarrow \deg(C(X))$
  if $(n = t)$ or $(k > k_0)$ or $(L_k^* \leq L_n)$ or $(L_n > L_0)$ or $(L_k^* < L_0)$ or $(k^* < k)$ **then**
    if $((n = t)$ and $(k \leq k_0)$ and $(L_k^* > L_n))$ **then**
      $(L_k^*, C_k^*(X), e_k^*) \leftarrow (L_n, C(X), e)$
      $k^* \leftarrow \min\{k | L_k^* \leq L_0\}$
      call $adjustProfile(L_n, C(X), e, k)$
    end if
  else
    $loopFlag \leftarrow true$
    repeat
      $d_n \leftarrow (s + e)_n + \sum_{i=0}^{L_n - 1} c_i (s + e)_{i+n-L_n}$
      if $d_n \neq 0$ **then**
        if $2L_n > n$ **then** ▷ (1a)
          $C(X) \leftarrow C(X) - \frac{d_n}{d_m} X^{(m-L_m)-(n-L_n)} D(X)$
          $n \leftarrow n + 1$
        else ▷ (1b)
          $loopFlag \leftarrow false$
          $T(X) \leftarrow C(X)$
          $C(X) \leftarrow X^{(n-L_n)-(m-L_m)} C(X) - \frac{d_n}{d_m} D(X)$
          call $mbmRL(n, T(X), d_n, C(X), n + 1, e)$
          if $k < k_0$ **then**
            call $mbmRL(m, D(X), d_m, C(X), n + 1, (e - d_n I_n))$
          end if
        end if
      else ▷ (2)
        $n \leftarrow n + 1$
      end if
    until $loopFlag = false$ or $n = t$
  end if
  if $((n = t)$ and $(k \leq k_0)$ and $(L_k^* > L_n))$ **then**
    $(L_k^*, C_k^*(X), e_k^*) \leftarrow (L_n, C(X), e)$
    $k^* \leftarrow \min\{k | L_k^* \leq L_0\}$
    call $adjustProfile(L_n, C(X), e, k)$
  end if
end procedure

depth of the recursion tree, as the number of terms processed in between two decision points will vary depending on each particular sequence (we call a decision point a moment in algorithm where $d_n \neq 0$ and $2L_n \geq n$ such that the recursion branches into two paths corresponding to introducing an error of magnitude $-d_n$ or not introducing an error on position n). We will assume that an average of u terms are processed between two decision points, i.e. between two points where the Berlekamp-Massey Algorithm would prescribe an increase in the current complexity of the sequence. In Rueppel [70, Chapter 4] it is shown that for random binary sequences the average number of bits that have to be processed between two changes in complexity is 4 and the change in complexity has an average of 2. While the sequences used in the cryptographic applications are not truly random, using a value of $u = 4$ for the average number of terms between two changes of complexity seems a reasonable choice. The following theorem has been proven by Sălăgean in our paper [2].

Theorem 4.4. *(Alecu, Sălăgean [2]) The average case time complexity of the Modified Berlekamp Massey Algorithm for sequences of length t, an average of u terms of the sequence processed between two changes in complexity, and a number of errors at most $k_0 = vt$ with $0 < v < \frac{1}{u}$ is*

$$\begin{cases} \mathcal{O}(\sqrt{t}\lambda_1^t) & \text{if } v < \frac{1}{3u} \quad \text{where } \lambda_1 = \frac{1}{uv^v(1-uv)^{\frac{1}{u}-v}}, \\ \mathcal{O}(t\sqrt{t}\lambda_1^t) & \text{if } \frac{1}{3u} \leq v < \frac{1}{2u} \quad \text{where } \lambda_1 = \frac{1}{uv^v(1-uv)^{\frac{1}{u}-v}}, \\ \mathcal{O}(t\lambda_2^t) & \text{if } \frac{1}{2u} \leq v \leq \frac{1}{u} \quad \text{where } \lambda_2 = \sqrt[u]{2}. \end{cases}$$

In all cases the complexity can also be written as $\mathcal{O}((\lambda_i + \varepsilon)^t)$ where $\varepsilon > 0$ is an arbitrarily small constant. For a typical value of $v = 0.1$ (i.e. errors in at most 10% of positions) and $u = 4$ the complexity is $\mathcal{O}(t\sqrt{t}1.189208^t)$.

Proof. Since u is the number of terms between two decision points and t is the total number of terms, the depth of the tree will be t/u. We bound the number of vertices in the tree by $(q-1)^{k_0}\sum_{i=0}^{k_0}\binom{\frac{t}{u}+1}{i+1}$, using lemma 3.10. When the number of right branches on any path, k_0, is at most half the depth of the tree, by applying the first or the second bound in lemma 3.7 (depending on whether k_0 is smaller or greater than a third of t/u), followed by the estimation (3.2), we obtain the first two computational complexities $\mathcal{O}$ of the theorem in a similar way as in the proof of theorem 3.11.

When the number of right branches allowed in the tree approaches the depth of the tree, i.e. k_0 approaches t/u, we will bound the number of nodes by $2^{\frac{t}{u}+1} - 1$ (the number of nodes in a complete binary tree of depth t/u). Combining this with $\mathcal{O}(t)$ operations in each node gives the third $\mathcal{O}$ of the theorem. □

The proposed algorithm has the advantage that even when the field has more than two elements, there are still only two choices that are investigated: introducing no error, or introducing an error of magnitude $-d_n$, where d_n is the discrepancy; an exhaustive search approach would have to investigate all the possible error magnitudes for each error position, i.e. $\sum_{i=0}^{k} \binom{t}{i}(q-1)^i$ possibilities for a field of q elements.

Both the computational complexities of Efficient Exhaustive Search Algorithm (theorem 3.11) and of Modified Berlekamp-Massey Algorithm (theorem 4.4) will increase by a factor of $(\log q)^2$ to account for the more costly operations in a field of q elements. However, the exponential part in the $\mathcal{O}$ estimate will remain unchanged in Theorem 4.4 (Modified Berlekamp-Massey Algorithm), whereas in Theorem 3.11 (Efficient Exhaustive Search Algorithm), λ^t is replaced by $(\lambda(q-1)^v)^t$.

For a typical value of $v = 0.1$ (i.e. errors in at most 10% of the positions) and an alphabet of $q = 16$ elements, the worst case time complexity is $\mathcal{O}(\sqrt{t}1.826^t)$ for Efficient Exhaustive Search Algorithm compared to $\mathcal{O}(t\sqrt{t}1.189208^t)$ for the proposed Modified Berlekamp-Massey Algorithm ($u = 4$).

Note that since similar considerations were used for the approximations in the computational complexity estimations for the two algorithms (Efficient Exhaustive Search Algorithm and Modified Berlekamp-Massey Algorithm) the comparision is fair.

4.4 Tests and results

Several tests are performed in order to estimate the efficiency and the accuracy of the proposed algorithm. We mainly do a comparison between the optimised Modified Berlekamp-Massey Algorithm (MBM) and the Efficient Exhaustive Search Algorithm (EES), but also a few other experiments are presented in order to better characterise the behaviour of the algorithm proposed in this chapter.

To measure how close to the exact result we get using the Modified Berlekamp-Massey Algorithm, we define the accuracy $ACC_{MBM,k}(s)$ as the ratio between $L_{MBM,k}(s)$, the approximate value of the k-error linear complexity obtained using the Modified Berlekamp-Massey Algorithm and $L_{EES,k}(s)$, the exact value obtained using the Efficient Exhaustive Search Algorithm:

$$ACC_{MBM,k}(s) = \frac{L_{MBM,k}(s)}{L_{EES,k}(s)}.$$

The running time improvement of the Modified Berlekamp-Massey Algorithm is computed as the ratio between the time taken by the Efficient Exhaustive Search

Algorithm and the time taken by the Modified Berlekamp-Massey Algorithm on the same processor:

$$improvement = \frac{time_{EES}}{time_{MBM}}.$$

In order to better understand the efficiency gain, the Modified Berlekamp-Massey Algorithm registers how many different error patterns it investigates and returns for each run their number (it includes in the count all the error patterns processed, not only the ones for which it reaches the last term e_{t-1}, which correspond to the stop condition $n = t$). Having the number of error patterns investigated by Modified Berlekamp-Massey Algorithm, let us denote it $patterns_{MBM}$, we can investigate if a random search through a randomly generated set of the same number of error patterns over the current finite field would give a better approximation and/or would take less running time (section 4.4.3).

4.4.1 Binary sequences

The first test involves running both algorithms (Modified Berlekamp-Massey Algorithm and Efficient Exhaustive Search Algorithm) on a number of 60 randomly chosen binary sequences of length 64 (each bit is generated with the C `rand()` linear congruential generator function).

Table 4.2: The average accuracy of the results of the MBM Algorithm.

Number of errors k	1	2	3	4	5	6	7	8	9
Average ACC_k	1.04	1.08	1.10	1.13	1.18	1.21	1.24	1.27	1.30
Best ACC_k	1	1	1	1	1	1	1	1	1
Worst ACC_k	1.22	1.26	1.31	1.35	1.35	1.47	1.5	1.57	1.62
Median	1.03	1.07	1.08	1.13	1.19	1.21	1.23	1.28	1.31
Frequency percentage of best accuracy	33.3%	20%	10%	5%	1.6%	3.3%	1.6%	1.6%	5%

Figure 4.5 presents the median, average, best and worst value of accuracy (ACC_k) over the 60 sequences tested for the case when the parameter $k_0 = w_H(s) - 1$. These results are given in Table 4.2 for $1 \leq k \leq 9$. For small values of k we notice that on average the k-error linear complexity obtained by the Modified Berlekamp-Massey Algorithm is pretty close to the actual value, being higher by only 4.49% for 1 error, increasing to 21.93% for 6 errors (i.e. errors in about 10% of the terms) and to 30.43% for 9 errors (i.e. about 15% of the terms). Additionally, the table 4.2 shows the frequency with which the proposed approximation algorithm has found the exact value for each number of errors (the total number of sequence is 60). The fact that the median (the 50% quartile[1]) is smaller or very

[1] The median is a certain value from a population which is separates the higher half of the

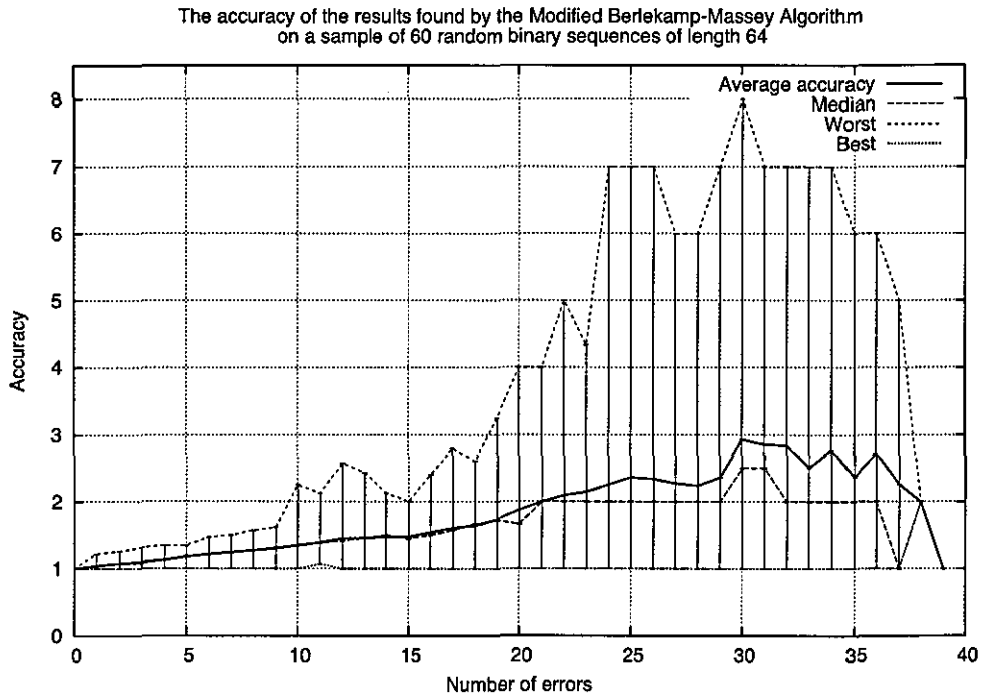

Figure 4.5: The accuracy of the Modified Berlekamp-Massey Algorithm for binary sequences of length 64 ($k_0 = w_H(s) - 1$)

close to the average for most of the numbers of errors (represented on the x axis in figure 4.5) shows that most times the algorithm approximates well the k-error linear complexity.

In many cases the Modified Berlekamp-Massey Algorithm determines the exact value of the k-error linear complexity. On the sequences on which the Modified Berlekamp-Massey Algorithm performs worst, it gets a k-error linear complexity up to 1.62 times the exact value, for $k \leq 9$.

As k increases, the quality of the results obtained by the Modified Berlekamp-Massey Algorithm deteriorates. Note however that the small values of k are the ones of practical interest.

Figure 4.6 shows the accuracy obtained when the input parameter given to the Modified Berlekamp-Massey Algorithm, k_0, is 15% of the length of the sequence, t.

The average running time improvement when calculating the full k_0-error linear complexity profile ($k_0 = w_H(s) - 1$) is 18,806 times, i.e. the Modified Berlekamp-Massey Algorithm was nearly 19,000 times faster than the Efficient Exhaustive

sample from the lower half.

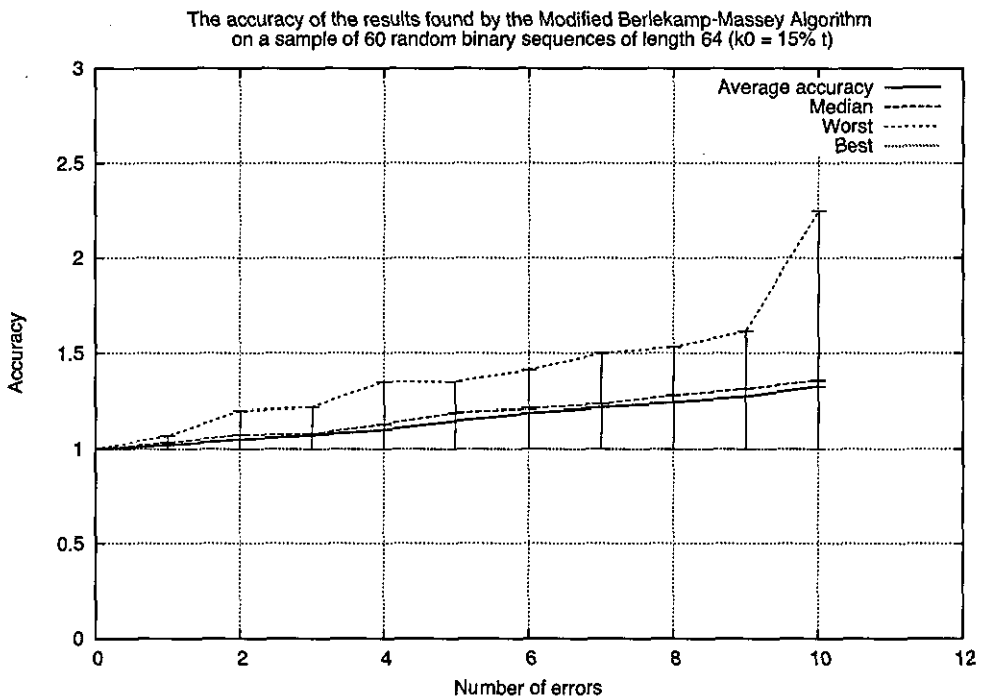

Figure 4.6: The accuracy of the Modified Berlekamp-Massey Algorithm for binary sequences of length 64 ($k_0 = 15\%t$)

Table 4.3: The running time in seconds for MBM and EES Algorithms for k-error linear complexity profile problem (different values for parameter k_0).

k_0	$time_{MBM}$			$time_{EES}$		
	min	avg	max	min	avg	max
$w_H(s)-1$	0	4.11	15	22,715	50,532.79	91,835
0.15t	0	3.80	9	17,685	40,683.62	103,869

Table 4.4: The running time improvement of the MBM Algorithm when compared with EES Algorithm for k-error linear complexity profile problem (different values for parameter k_0).

k_0	$\frac{time_{EES}}{time_{MBM}}$		
	min	avg	max
$w_H(s)-1$	4,881.13	18,805.85	91,835
0.15t	3,889.67	13,401.59	46,270

Search Algorithm. A similar time improvement is obtained when imposing a smaller limit on the number of errors (k_0). For example, when k_0 is 15% of the length of the sequence t, the time improvement is 13,402 times. See table 4.4 for details regarding the average running time improvement in each of the two cases considered. The time is expressed in seconds and the duration is rounded to the closest number of milliseconds.

A similar if not more accurate measure of the improvement is to look at the number of error patterns visited by each algorithm for the same sequence (see tables 4.5 and 4.6).

For the binary sequences of length 64 and when the input parameter $k_0 = w_H(s) - 1$, on average, the Modified Berlekamp-Massey Algorithm took approximately 4.11 seconds and processed 18,593 different error patterns while the Efficient Exhaustive Search Algorithm took approx. 50,532 seconds and processed 360,604,284 error patterns.

See tables 4.3, 4.4, 4.5, 4.6 showing full details regarding the running time and the number of error patterns visited by each algorithm.

Table 4.5: The number of error patterns visited by the MBM and EES Algorithms for k-error linear complexity profile problem (different values for parameter k_0).

k_0	$patterns_{MBM}$			$patterns_{EES}$		
	min	avg	max	min	avg	max
$w_H(s)-1$	902	18,593.36	58,572	220,854,390	360,604,284.10	592,530,235
0.15t	830	14,137.85	36,592	129,365,933	238,001,242.45	411,466,082

Table 4.6: The number of error patterns visited by the MBM Algorithm compared to EES Algorithm for k-error linear complexity profile problem (different values for parameter k_0).

k_0	$\frac{patterns_{EES}}{patterns_{MBM}}$		
	min	avg	max
$w_H(s)-1$	7,289.52	32,979.08	269,634.70
0.15t	5,879.20	26,553.13	239,322.82

4.4.2 L-constrained k-error linear complexity problem

For the case when the Modified Berlekamp-Massey Algorithm solves the L-constrained k-error linear complexity problem the focus of the problem is actually the value k^* representing the minimum number of errors to force on the sequence so that its linear complexity would drop below L_0. Since the Modified Berlekamp-Massey Algorithm returns an approximation, the value k_{MBM} will always be equal to or more than the exact value returned by the Efficient Exhaustive Search Algorithm, let us denote this exact value k_{EES}. Therefore in this case, for accuracy, it would be useful to compare the difference between these two values scaled by the length of the sequence: $(k_{MBM} - k_{EES})/t$.

We ran the Modified Berlekamp-Massey Algorithm and Efficient Exhaustive Search Algorithm for the L-constrained k-error linear complexity problem on the same set of 60 binary sequences of length 64 as in section 4.4.1, choosing L_0 to be 33% of the length of the sequence and for k_0 two different values, $k_0 = w_H(s) - 1$ or $k_0 = 15\%t$.

Figures 4.7 and 4.8 include the distribution of the accuracy $(k_{MBM} - k_{EES})/t$ when $k_0 = w_H(s) - 1$ or $k_0 = 15\%t$, respectively. Note that the approximation k_{MBM} is close to the exact value, in most cases, for both values of k_0, the difference between k_{MBM} and k_{EES} being between 1 and 2 (corresponding to the 0.02 and 0.03 in the two figures). In the worst cases, the approximate value is larger by 5 than the exact value.

In terms of efficiency, the runtime improvement from the Efficient Exhaustive Search Algorithm is consistent. For the 60 binary sequences of length 64, when the input parameter $k_0 = w_H(s) - 1$ and $L_0 = 33\%t$, on average, the Modified Berlekamp-Massey Algorithm took approx. 3.58 seconds and processed 17,547 different error patterns while the Efficient Exhaustive Search Algorithm took approx. 43,826 seconds and processed 347,452,754 error patterns.

See tables 4.7, 4.8, 4.9 and 4.10 which show full details regarding the running time and the number of error patterns visited by each algorithm.

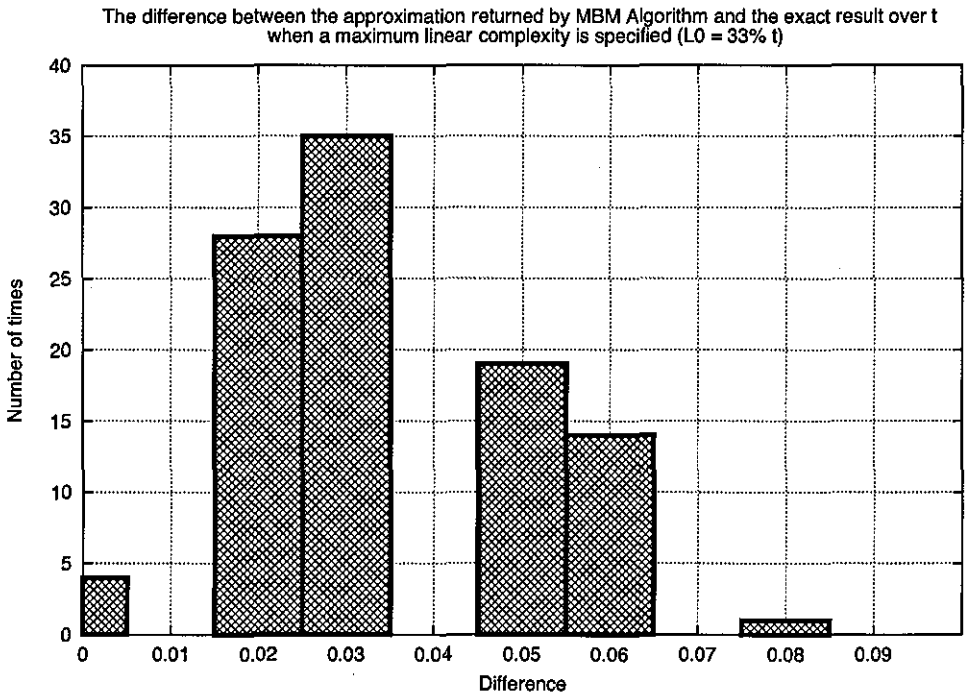

Figure 4.7: The difference between the approximation of the Modified Berlekamp-Massey Algorithm and the exact result for binary sequences of length 64 when $L_0 = 33\%t$

Table 4.7: The running time in seconds for MBM and EES Algorithms for the L-constrained k-error linear complexity problem.

k_0	L_0	$time_{MBM}$			$time_{EES}$		
		min	avg	max	min	avg	max
$w_H(s)-1$	$\frac{t}{3}$	0	3.58	15	23,400	43,825.97	79,291
0.15t	$\frac{t}{3}$	0	1.98	5.19	14,288	26,501.24	45,441

Table 4.8: The running time improvement of the MBM Algorithm when compared with EES Algorithm for the L-constrained k-error linear complexity problem (different values for parameters k_0 and L_0).

k_0	L_0	$\frac{time_{EES}}{time_{MBM}}$		
		min	avg	max
$w_H(s)-1$	$\frac{t}{3}$	2,588.60	28,838.55	659,611.11
0.15t	$\frac{t}{3}$	4,762.67	19,416.39	258,236.25

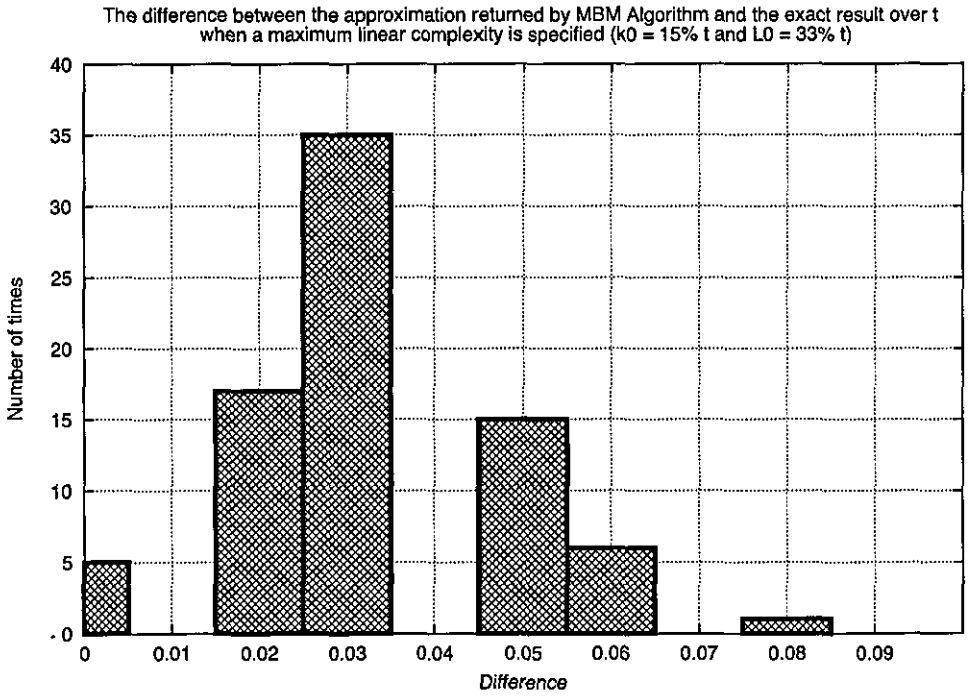

Figure 4.8: The difference between the approximation of the Modified Berlekamp-Massey Algorithm and the exact result for binary sequences of length 64 when $k_0 = 15\%t$ and $L_0 = 33\%t$

Table 4.9: The number of error patterns visited by the MBM and EES Algorithms (different values for parameters k_0 and L_0).

k_0	L_0	$patterns_{MBM}$			$patterns_{EES}$		
		min	avg	max	min	avg	max
$w_H(s)-1$	$\frac{t}{3}$	490	17,546.85	58,572	220,006,957	347,452,753.84	590,428,446
0.15t	$\frac{t}{3}$	386	11,210.80	30,839	128,234,300	238,965,677.38	410,103,809

Table 4.10: The number of error patterns visited by the MBM Algorithm compared to EES Algorithm for L-constrained k-error linear complexity problem (different values for parameters k_0 and L_0).

k_0	L_0	$\frac{patterns_{EES}}{patterns_{MBM}}$		
		min	avg	max
$w_H(s)-1$	$\frac{t}{3}$	5,634.26	42,427.07	748,075.28
0.15t	$\frac{t}{3}$	7,616.67	39,902.56	474,145.50

4.4.3 Significance of the heuristic selection

A purely random heuristic approach for the k-error linear complexity profile problem would be to generate randomly a certain number $N_{patterns}$ of error patterns e of weight k with $0 \leq k \leq k_0$ and save the minimum $L(s+e)$ for each k. We will call this method *KRandom* Algorithm, see listing 16 for the implementation.

Algorithm 16 KRandom Algorithm

1: **Input:** A finite sequence $s = s_0, s_1, \ldots, s_{t-1}$ over $GF(q)$; an integer k_0 with $0 < k_0 \leq w_H(s) - 1$; an integer $N_{patterns} > 0$
2: **Output:** L_i^*, $C_i^*(X)$ and e_i^*, for all $i = 0, 1, \ldots, k_0$
3: **for** $i = 0, 1, \ldots, k_0$ **do**
4: $L_i^* \leftarrow L(s)$
5: $C_i^*(X) \leftarrow C(X)$, a minimal characteristic polynomial
6: $e_i^* \leftarrow (\underbrace{0, 0, \ldots, 0}_{t \text{ times}})$
7: **end for**
8: $counter \leftarrow 0$
9: **while** $(counter < N_{patterns})$ **do**
10: Generate randomly an error pattern $e \in GF(q)^t$ with $w_H(e) \leq k_0$
11: Calculate $L(s+e)$ and $C(X)$ corresponding to $s+e$
12: **if** $L_k^* > L(s+e)$ **then**
13: $L_k^* \leftarrow L(s+e)$
14: $C_k^*(X) \leftarrow C(X)$
15: $e_k^* \leftarrow e$
16: **end if**
17: $counter \leftarrow counter + 1$
18: **end while**
19: **return** L_i^*, $C_i^*(X)$ and e_i^*, for all $i = 0, 1, \ldots, k_0$

We ran the *KRandom* Algorithm on the same set of 60 binary sequences of length 64 as in section 4.4.1 with the two values of k_0, $w_H(s) - 1$ and $15\%t$, and for each giving as input for $N_{patterns}$ the number of error patterns processed by the corresponding Modified Berlekamp-Massey Algorithm. We want to check that the method that we used in constructing the error patterns in Modified Berlekamp-Massey Algorithm is significant in that it does give consistently better results than if the same number of error patterns would be chosen randomly. For these tests, the random error patterns are generated using the C `rand()` linear congruential generator function.

Figure 4.9 shows the average accuracy of the *KRandom* algorithm and of the Modified Berlekamp-Massey Algorithm on the considered set of sequences. The accuracy of the approximation of the k-error linear complexity is good for a very low number of errors $k = 1$ or $k = 2$, however as the number of errors grows the *KRandom* Algorithm becomes unreliable and inaccurate, see figure 4.10 for a

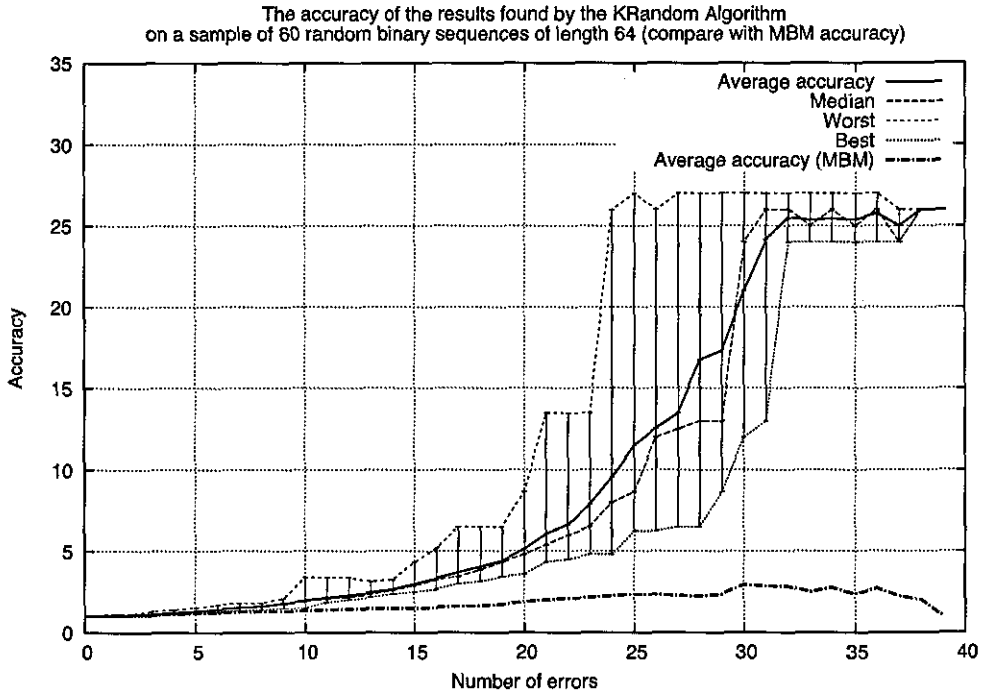

Figure 4.9: The accuracy of the KRandom for sequences of length 64.

zoom on the graph when $0 \leq k \leq 10$.

This gives us the indication that the approach used by Modified Berlekamp-Massey Algorithm in building and choosing the error sequences has a significant impact on the accuracy of the result.

4.4.4 Sequences of different lengths

In order to check how the accuracy of the results of the algorithm scales with the length of the input sequence, a second experiment involved running Modified Berlekamp-Massey Algorithm for binary sequences of different lengths. We used 20 random sequences of each even length between 8 and 64 and ran both Modified Berlekamp-Massey Algorithm and Efficient Exhaustive Search Algorithm for k-error linear complexity profile problem with $k_0 = w_H(s) - 1$.

The time improvement of the Modified Berlekamp-Massey Algorithm shows an exponential increase with the length of the sequence. Figure 4.11 contains the results concerning the running time improvement, $time_{EES}/time_{MBM}$, and figure 4.12 shows the improvement in number of error patterns processed,

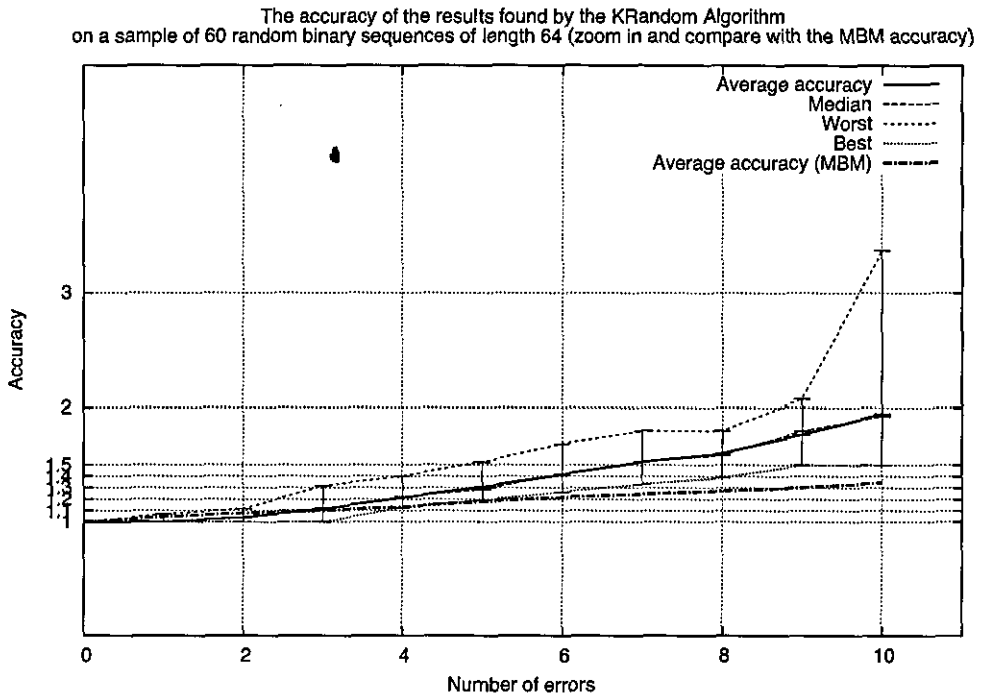

Figure 4.10: The accuracy of the KRandom for sequences of length 64 (zoom for $0 \leq k \leq 10$).

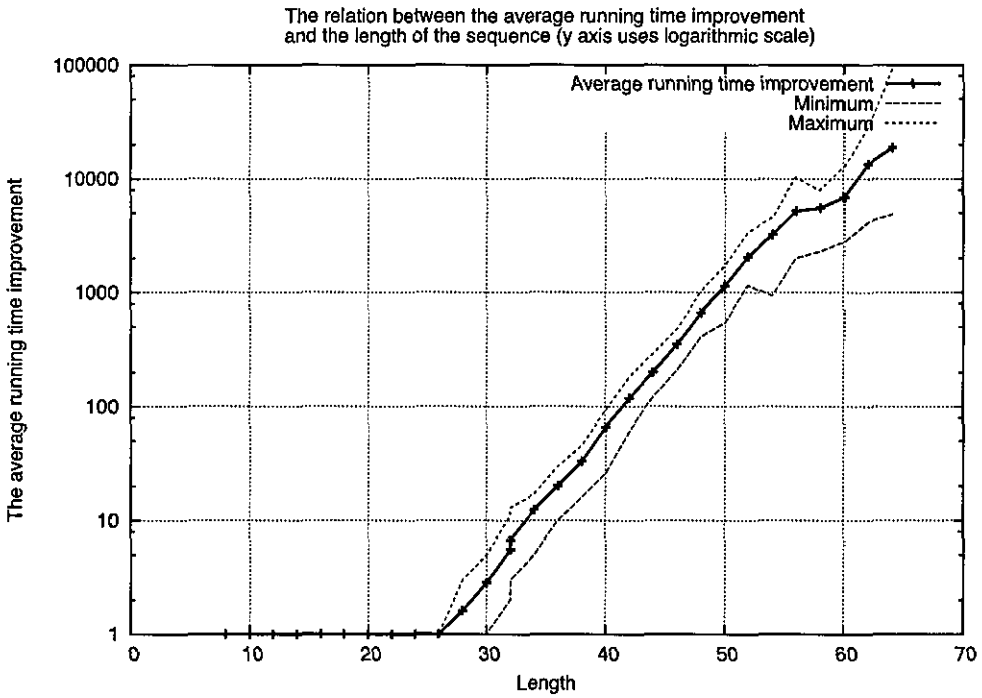

Figure 4.11: The relation between the average running time improvement on logarithmic scale and the length of the sequences.

$patterns_{EES}/patterns_{MBM}$ (figures 4.11 and 4.12 use a logarithmic scale on the y axis). The improvement, both in terms of running time and in terms of number of error patterns, is approximately exponential in relation to the length of the sequence. Figures 4.13 and 4.14 show the relation between the length of the input sequence and the running time or the number of patterns of the Modified Berlekamp-Massey Algorithm, respectively.

The quality of the approximation was measured for each sequence at different levels of number of errors: 5%, 10% and 15% of the length of the sequence. Namely for $k = 5\%$, 10% and 15% we compared the values returned by the two algorithms for the k-error linear complexity, evaluating the accuracy $ACC_k = L_{MBM,k}(s)/L_{EES,k}(s)$. The results are summarised in figure 4.15. We note that the approximate value of the k-error linear complexity found by the Modified Berlekamp-Massey Algorithm is consistently good on all lengths tested and it deteriorates only slightly as k increases as a percentage of the length of the sequence. This can be connected to the fact that the search space size increases with the value of k for $0 \leq k \leq \frac{t}{2}$ ($\binom{n}{k}$ is increasing with k when $0 \leq k \leq \frac{t}{2}$).

The accuracy follows a similar trend as the results obtained for the previous

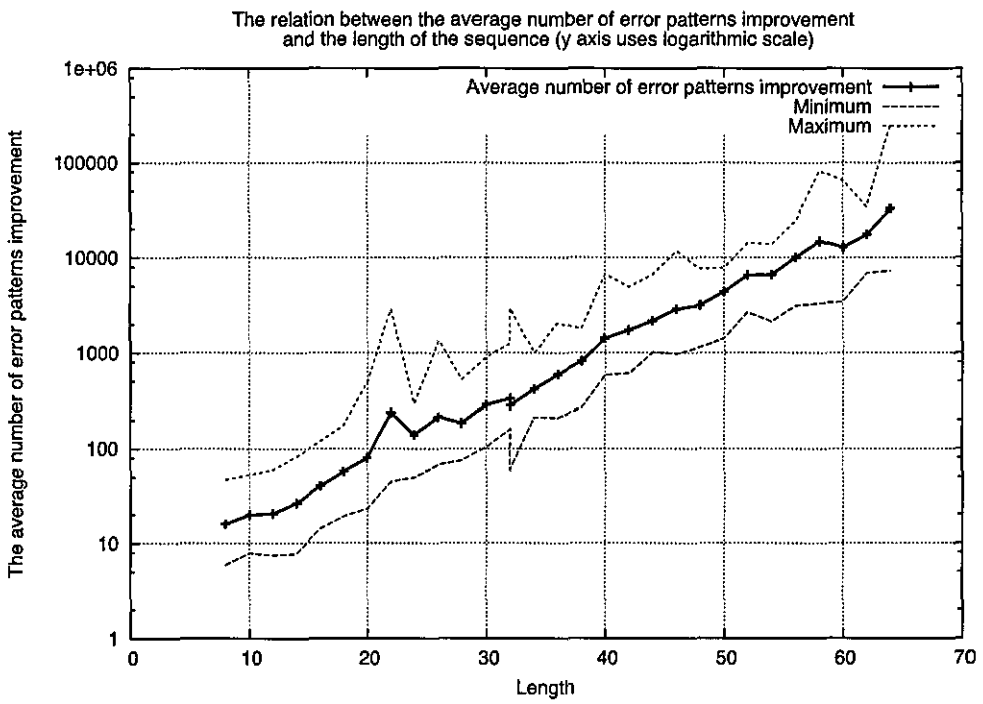

Figure 4.12: The relation between the number of error patterns on logarithmic scale and the length of the sequences.

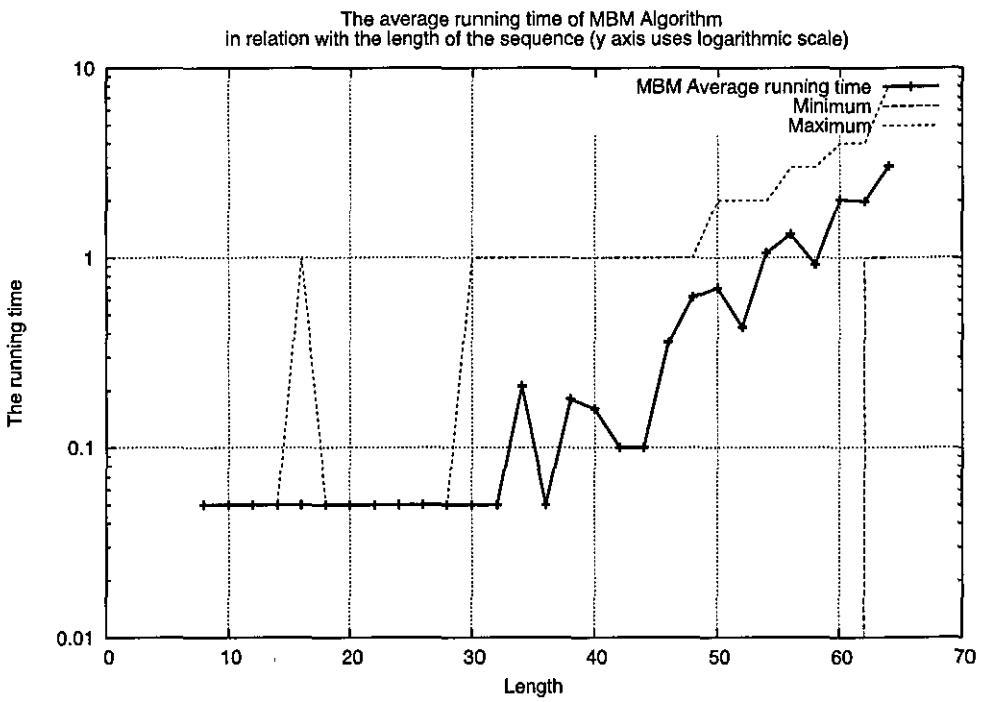

Figure 4.13: The average runtime of MBM Algorithm on a logarithmic scale in relation to the length of the sequence.

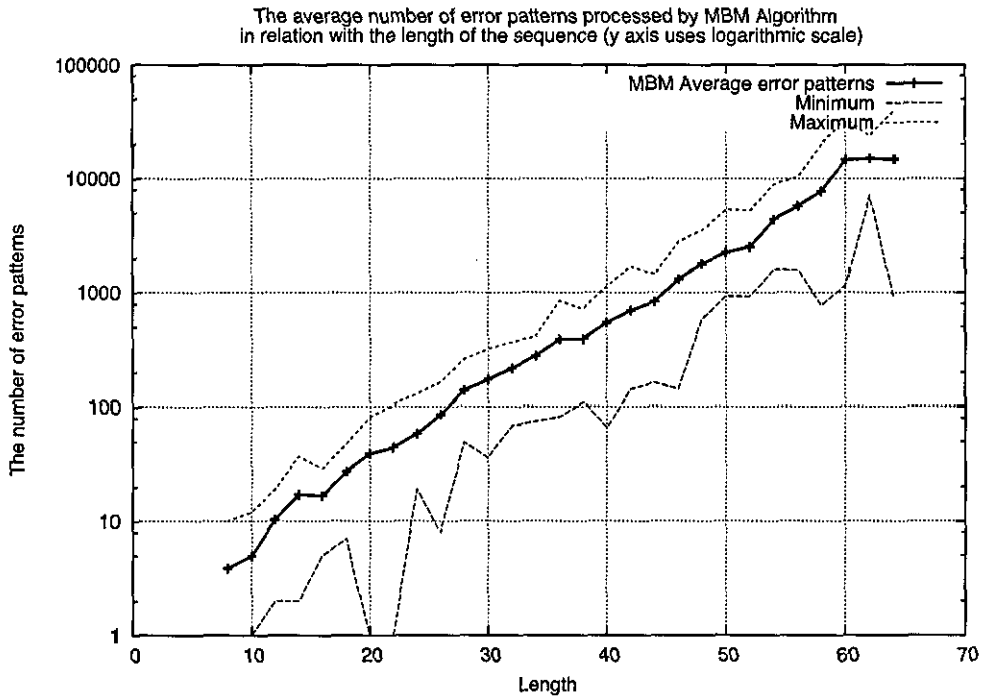

Figure 4.14: The average number of error patterns processed by MBM Algorithm on a logarithmic scale in relation to the length of the sequence.

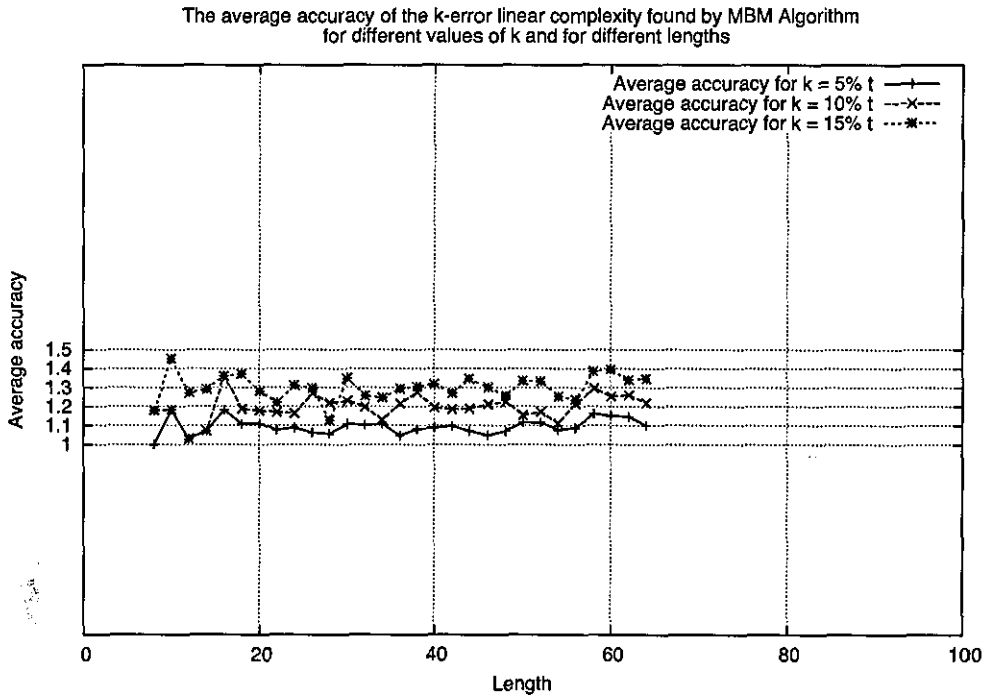

Figure 4.15: The average accuracy of the k-error linear complexity found by the MBM Algorithm for different values of k and for different lengths.

experiment including sequences of length 64 (see table 4.2). For 5% errors (i.e. k is 5% of the length), the k-error linear complexity found by the Modified Berlekamp-Massey Algorithm is on average not more than 10% higher than the actual value, for 10% errors it is at most 20% higher and for 15% it is at most 30% higher. This behaviour supports the assumption that the accuracy of the Modified Berlekamp-Massey Algorithm is approximately constant for binary sequences of any length when the k-error linear complexity is calculated for a k which is a certain fixed percentage of the length.

When running the KRandom Algorithm using the same number of error patterns as Modified Berlekamp-Massey Algorithm, we notice the significance of the method used in MBM (see figure 4.16). For a relatively low number of errors (5%) the approximation returned by KRandom Algorithm is only slightly worse than the approximation of the Modified Berlekamp-Massey Algorithm but for 10% or 15% the difference is noticeable.

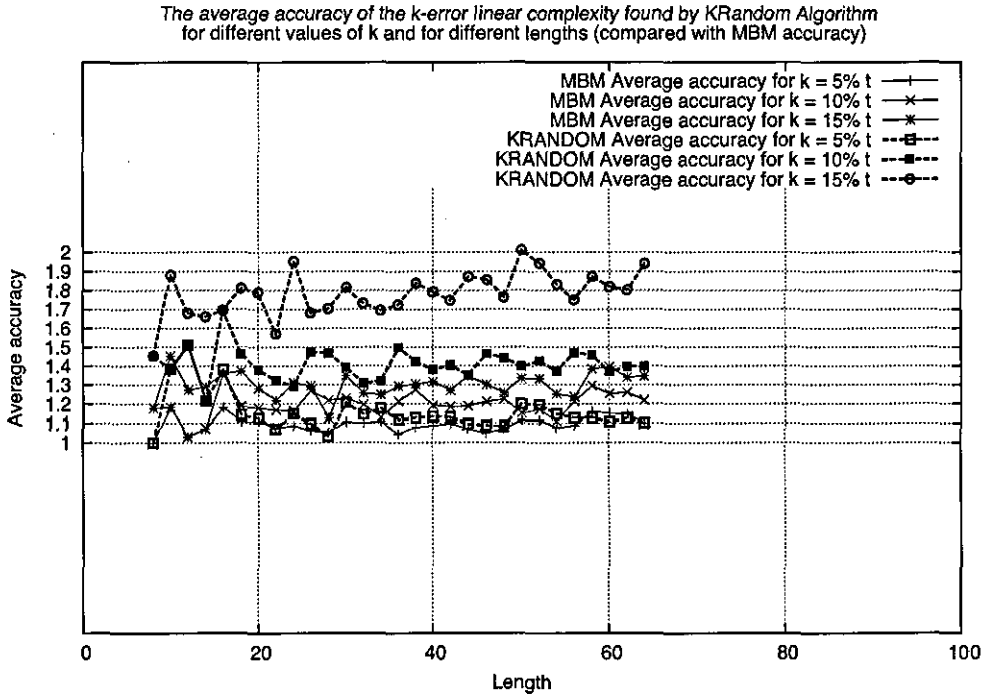

Figure 4.16: The average accuracy of the k-error linear complexity found by KRandom Algorithm for different values of k and for different lengths (compared with MBM accuracy)

4.4.5 Sequences of higher length

For evaluating the accuracy of the MBM algorithm for sequences of higher length, the exact k-error linear complexity profile can no longer be computed using exhaustive search due to time limitations. Instead, we carried out a controlled experiment.

The controlled experiment is based on the following set up. Suppose a sequence s of length t is generated using a randomly chosen recurrence of size L such that $L < \frac{t}{2}$. Note that the linear complexity of the sequence can be smaller than the size of the recurrence used to generate it, namely $L(s) < L$. We artificially apply an error sequence e of weight k such that the linear complexity of $s' = s + e$ is higher than $L(s)$. Obviously the k-error linear complexity of s' is equal or less than the initial linear complexity, $L_k(s') \leq L(s)$, so even though we do not know the exact k-error linear complexity of s', we do have a good upper bound. So if we apply the Modified Berlekamp-Massey Algorithm to s' and compute the fraction $\frac{L_{MBM,k}(s')}{L(s)}$ we aim for this value to be close to 1. Note that this time the accuracy ratio can be less than 1 because $L(s)$ is only an upper bound rather than the exact value of the k-error linear complexity of s', $L_k(s')$.

To illustrate, the tests include 100 binary sequences s of length 100, generated by a randomly chosen recurrence of size 33 (33% of the length). The linear complexity $L(s)$ of each sequence s is computed (this can be lower than 33). We artificially applied an error sequence e of weight k, such that the linear complexity of $s' = s + e$ is higher than $L(s)$. Obviously, $L_k(s') \leq L(s)$. We then applied the Modified Berlekamp-Massey Algorithm to s' and computed the ratio $L_{MBM,k}(s')/L(s)$.

Figure 4.17 presents the distribution of the values of this ratio in each interval of length 0.1. The three graphs considered correspond to limiting the view to the cases when the value of k takes random values up to 5%, 10% or 15% of the length of the sequence, respectively. Out of the 100 random values generated for k such that they are less than 15% of the length (figure 4.17(a)), 61 are less than 10% (figure 4.17(b)) and 24 are are less than 5% of the length (figure 4.17(c)). We notice that a high proportion of the ratios are below 1.3, i.e. the value found by the MBM algorithm is close, or even lower than the original complexity, $L(s)$. The results are better when k represents a higher proportion of the length of the sequence.

The results of a similar experiment applied to 26 sequences of length 128 are displayed in figure 4.18. Out of the 26 random values less than 15% of the length generated for k (figure 4.18(a)), 17 are less than 10% (figure 4.18(b)) and 4 are less than 5% of the length (figure 4.18(c)).

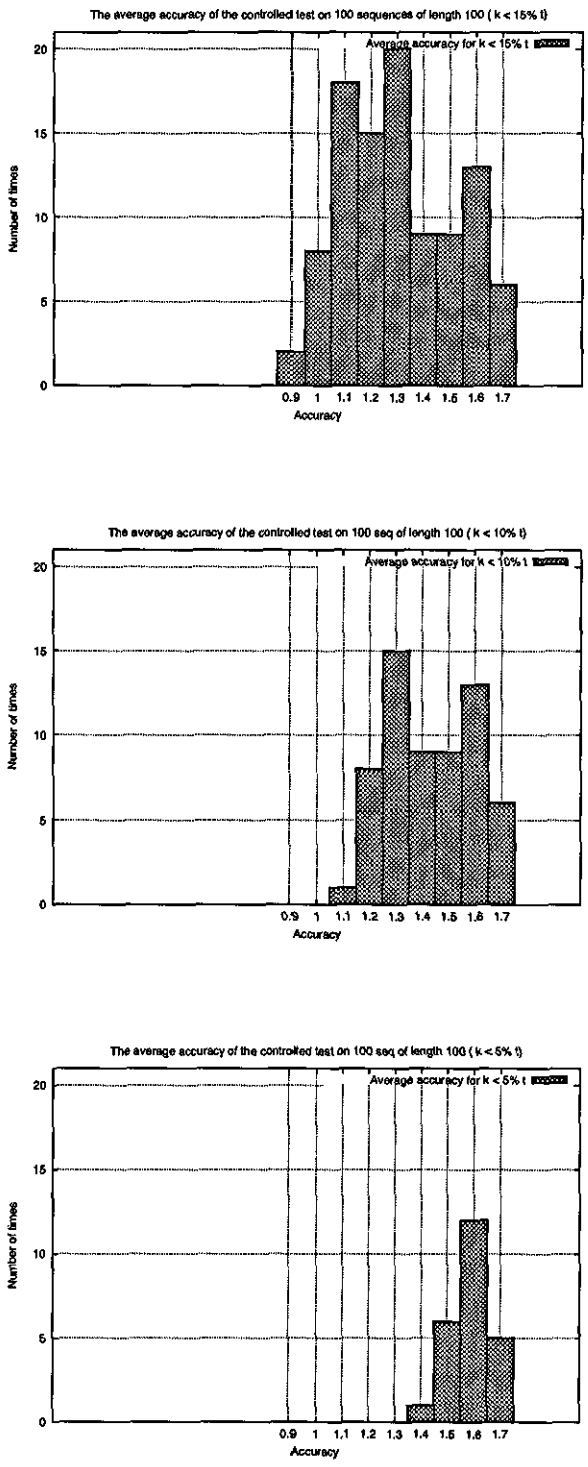

Figure 4.17: The accuracy of the results found by MBM Algorithm on 100 sequences of length 100, when the sequences were artificially modified with errors sequences of weight: (a) $k \leq 15\%$ of the length; (b) $k = 10\%$ of the length; (c) $k = 5\%$ of the length;

Table 4.11: The runtime and number of error patterns for MBM Algorithm applied to sequences over different finite fields.

Field	*MBM*		*EXHEF*	
	avg runtime	avg patterns	avg runtime	avg patterns
$GF(2)$	0.01	210.82	6.7	33,739.38
$GF(3)$	0.08	693.10	120.28	2,351,946.12
$GF(5)$	0.14	1,442.2	14,556.74	226,925,269.80

4.4.6 Sequences over finite fields of higher order

In order to see how the accuracy and efficiency of the Modified Berlekamp-Massey Algorithm scales when the finite field of the elements varies, we experimented with sequences from $GF(3)$ and $GF(5)$. The results concerning the accuracy and running time of the algorithm when run over 50 sequences of length $t = 32$ with elements in $GF(2)$, $GF(3)$ and $GF(5)$, with k_0 15% of the length of the sequence t are graphically presented in this section.

The accuracy of the Modified Berlekamp-Massey Algorithm approximation is constantly good for all three fields. See figures 4.19, 4.20 and 4.21 for the each of the fields and figure 4.22 for a unified view on the average accuracies for the three cases.

In order to validate the Modified Berlekamp-Massey Algorithm's heuristic method, the KRandom Algorithm has been applied to the same sequences using the same number of error patterns as used by Modified Berlekamp-Massey Algorithm. Figures 4.23, 4.24 and 4.25 show how the accuracy of Modified Berlekamp-Massey Algorithm is better than the accuracy of the purely random KRandom algorithm for each of the fields considered and the 50 sequences processed.

4.5 Conclusion

We propose a heuristic algorithm for approximating the k-error linear complexity, based on modifying the Berlekamp-Massey Algorithm. The modification consists in that, when choosing the error patterns, it considers introducing an error whenever there is an increase in the linear complexity of the sequence processed so far.

We implemented, tested and analysed this algorithm and the results are encouraging. The k-error linear complexity is closely approximated: on average it is only 16% higher than the exact value, for up to 6 errors on our test set of 60 ran-

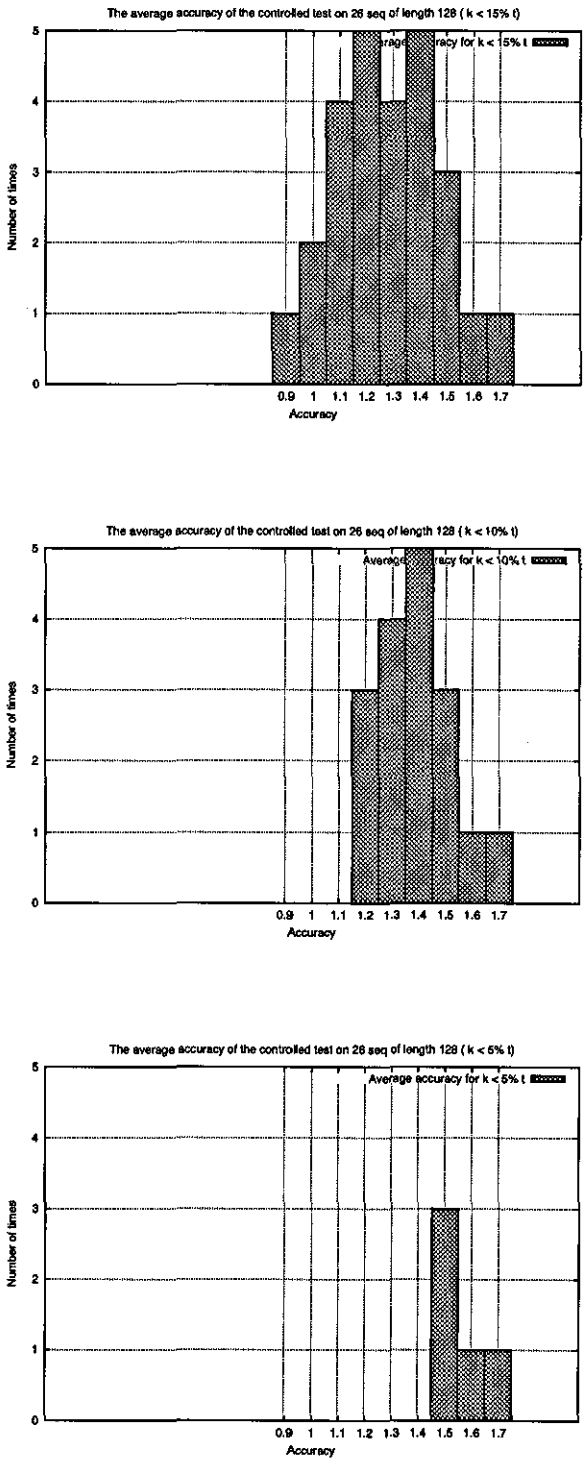

Figure 4.18: The accuracy of the results found by MBM Algorithm on 21 sequences of length 128, when the sequences were artificially modified with errors sequences of weight : (a) $k \leq 15\%$ of the length; (b) $k = 10\%$ of the length; (c) $k = 5\%$ of the length;

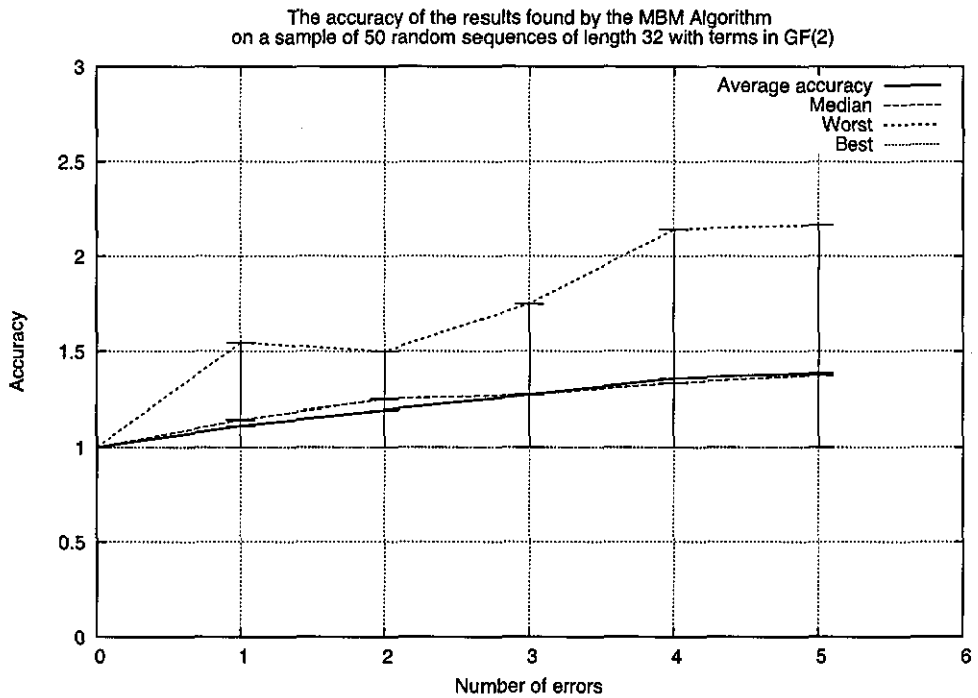

Figure 4.19: The accuracy of the results found by the MBM Algorithm on a sample of 50 random sequences of length 32 with terms in GF(2).

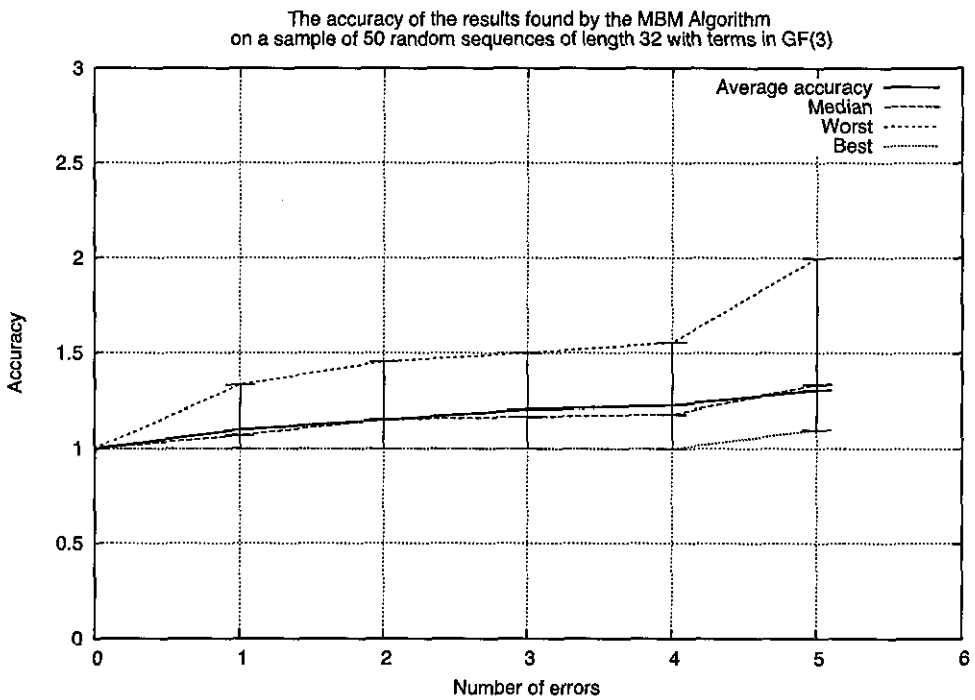

Figure 4.20: The accuracy of the results found by the MBM Algorithm on a sample of 50 random sequences of length 32 with terms in GF(3).

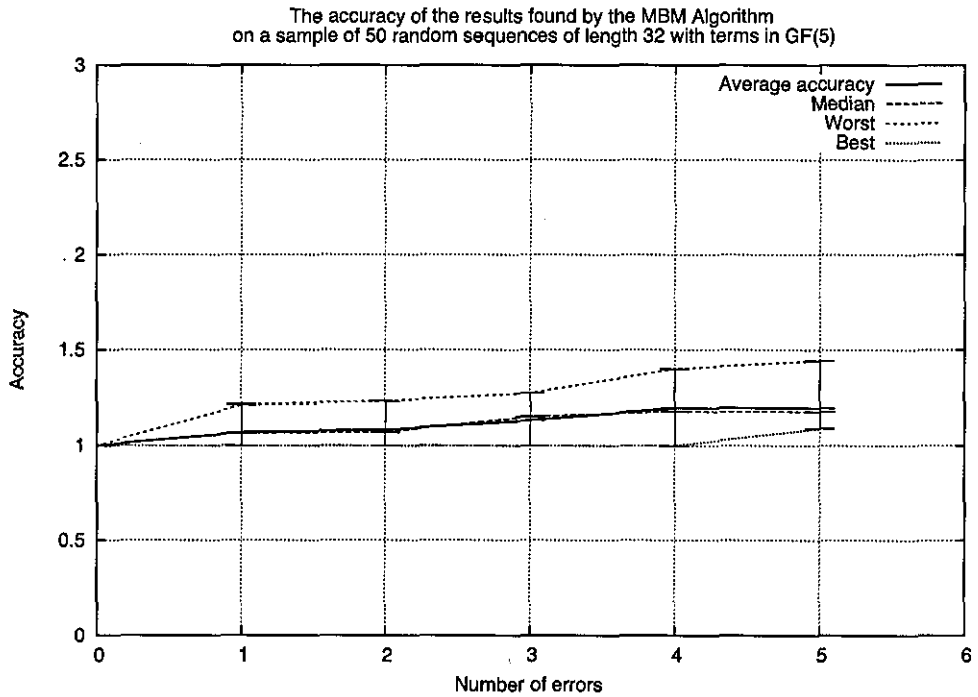

Figure 4.21: The accuracy of the results found by the MBM Algorithm on a sample of 50 random sequences of length 32 with terms in GF(5).

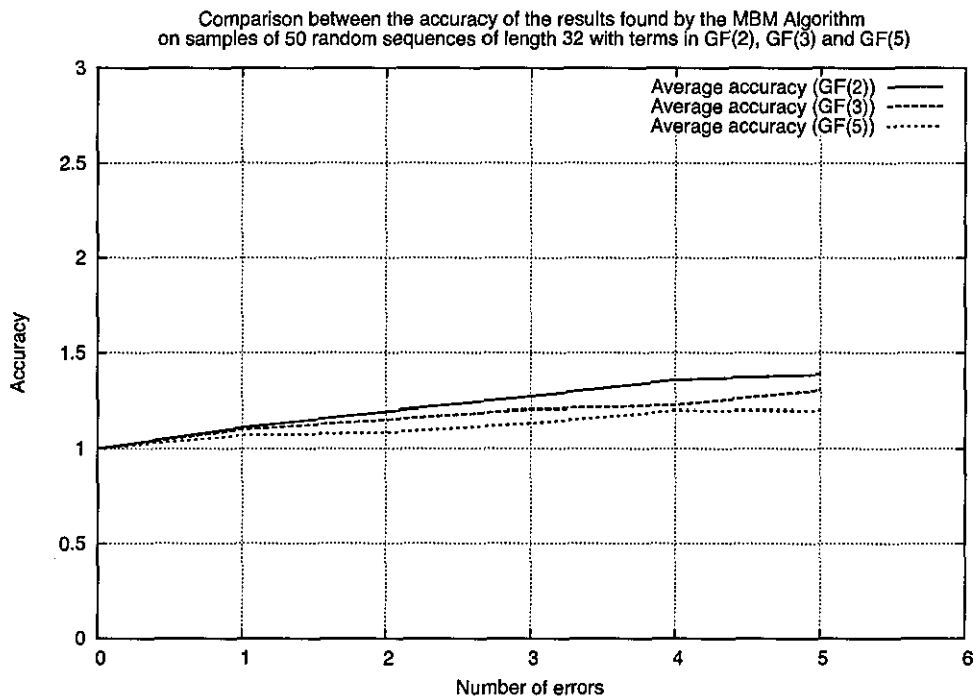

Figure 4.22: The accuracy of the results found by the MBM Algorithm on a sample of 50 random sequences of length 32 with terms in GF(2), GF(3), GF(5).

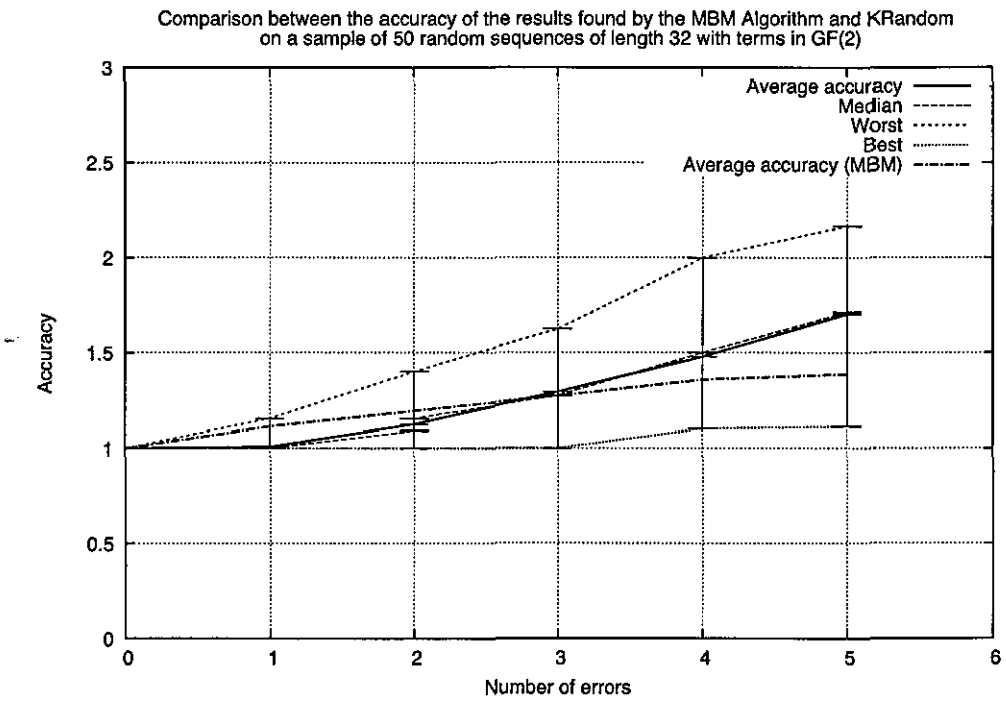

Figure 4.23: Comparison between the accuracy of the results found by the MBM Algorithm and KRandom Algorithm on a sample of 50 random sequences of length 32 with terms in GF(2).

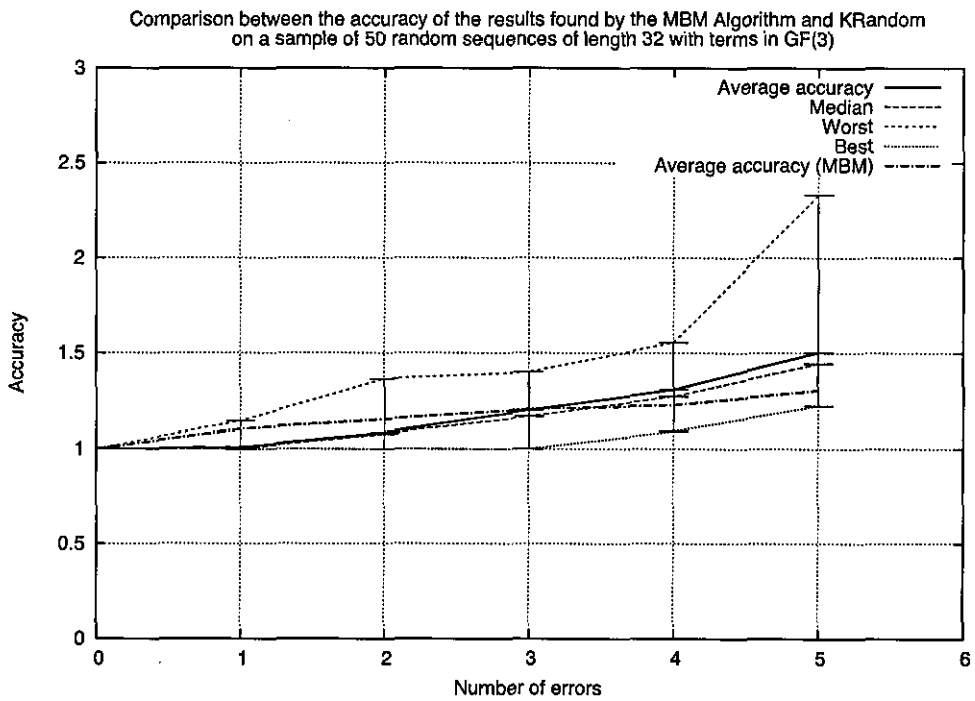

Figure 4.24: Comparison between the accuracy of the results found by the MBM Algorithm and KRandom Algorithm on a sample of 50 random sequences of length 32 with terms in GF(3).

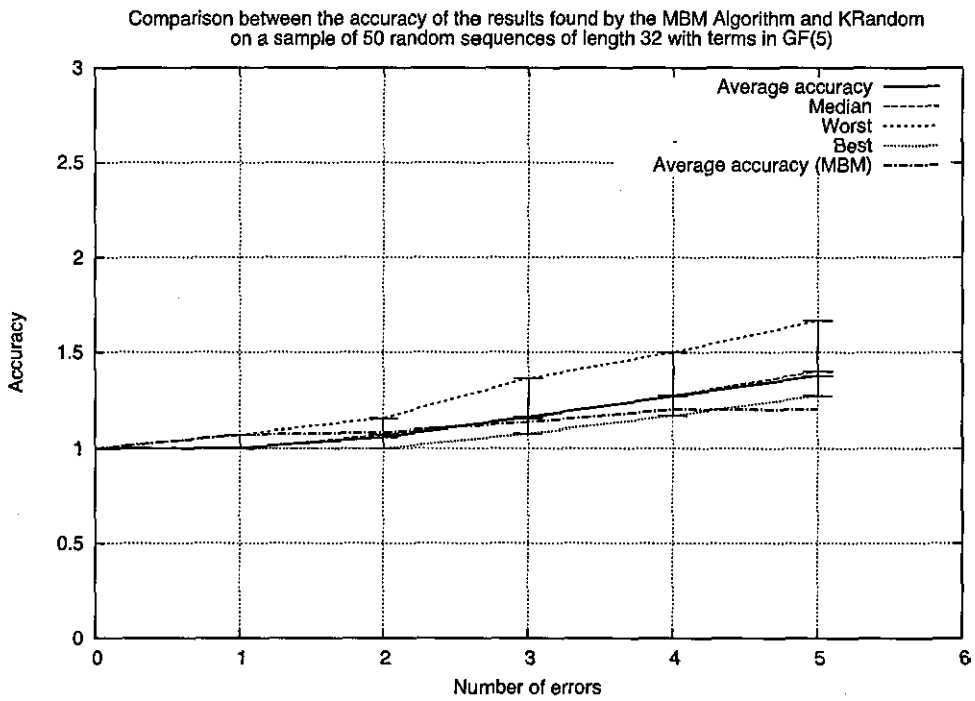

Figure 4.25: Comparison between the accuracy of the results found by the MBM Algorithm and KRandom Algorithm on a sample of 50 random sequences of length 32 with terms in GF(5).

dom sequences of length 64. The accuracy and the running time of the algorithm scales well with the size of the search space e.g. when considering long sequences or those over finite fields of higher order.

While the time complexity of the proposed algorithm is still exponential, it is considerably faster than an exhaustive search (on average about 19000 times faster for the set of 60 sequences of length 64). Even higher efficiency gains appear in the non-binary case and that has been experimentally validated.

In some special cases, the Modified Berlekamp-Massey Algorithm does not give a good approximation of the k-error linear complexity profile. For example, for sequences with many leading zeroes, it only processes a very small amount of error patterns. Future work would investigate the possibility of further improving the efficiency and accuracy of the algorithm by processing some more error patterns which are likely to reduce the linear complexity of the considered sequence.

Chapter 5

Evolutionary Computation Techniques

This chapter presents two evolutionary methods applied to solving the k-error linear complexity profile problem, as well as their implementations and their performance in the context of the heuristic approach.

5.1 Genetic Algorithm

This section presents a genetic algorithm to approximate the k-error linear complexity of a sequence over a finite field.

The algorithm follows the implementation of a classic Simple Genetic Algorithm (Goldberg [20]) and the focus is on finding the best choice of values for the different parameters involved, e.g. population size, number of generations, technique of selection, crossover or mutation, mutation probability, crossover probability. Some of the parameters need to scale with the size of the search space, e.g. the size of the input sequence and the number of errors. In addition, the choice of the evaluation function plays an important role in the design of the algorithm.

5.1.1 Background

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 (Goldberg [20]).

Genetic algorithms have proven to be useful in solving (or give reasonable solutions to) a big variety of problems. They have been successfully applied on

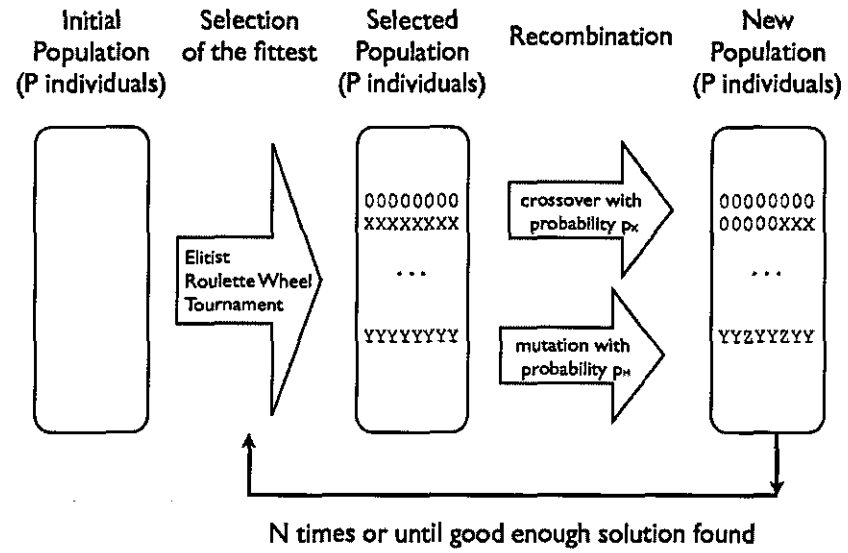

Figure 5.1: Schematic view of a Simple Genetic Algorithm

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 at 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 fittest individuals on that basis (the same individual can be duplicated in a population, the order of duplication being usually directly 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 found so far represents a reasonable solution, reasonable having different definitions for different problems (Michalewicz [54]). See figure 5.1 for a schematic view of a genetic algorithm and listing 17 for the pseudocode of a Simple Genetic Algorithm (SGA).

It is challenging and much experimental 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).

Algorithm 17 Simple Genetic Algorithm

```
Choose initial population of possible solutions (e.g. random sampling)
Evaluate all individuals' fitness and determine the population's fitness
while not reached target fitness or maximum number of generations do
    Select best ranking individuals from the current population
    Apply crossover with a certain probability
    Apply mutation with a certain probability
    Evaluate individuals' fitness and determine the new population's fitness
end while
return best solution so far
```

5.1.2 kGA Algorithm

We first argue why a genetic algorithm technique is suitable for the k-error linear complexity profile problem. Computing the k-error linear complexity of a sequence s with terms in a finite field K is an optimisation problem with a well defined search space, namely for a sequence of length t and a value k_0, the search space includes all the sequences of length t and Hamming weight at most k, $\{ e | e \in K^t, w_H(e) \leq k_0 \}$. The elements of the search space, the error patterns, can very naturally be seen as string encoded chromosomes.

We remind that the problem which we will approximate, the k-error linear complexity profile problem, has as input a finite sequence $s = s_0, s_1, \ldots, s_{t-1}$ of size $t > 0$ with terms over a finite field $GF(q)$, where q is a prime power and an integer value k_0, with $0 < k_0 \leq w_H(s) - 1$. The expected output is an approximate k_0-error linear complexity profile of s containing for each $i = 0, 1, \ldots, k_0$, L_i^*, the approximate i-error linear complexity; e_i^*, the error pattern producing the linear complexity L_i^* on s; $C_i^*(X)$ a minimal characteristic polynomial corresponding to the sequence $s + e_i^*$.

The algorithm starts with an initial population of PS possible solutions from the search space (denote the initial set POP_0), PS being an integer much smaller than the size of the search space. These initial individuals of the population are typically randomly generated. Each individual in the population is evaluated using a certain fitness function denoted f.

After the initialisation step, the following set of steps is repeated a fixed number of times called generations, let us denote $NOGEN$ the number of generations, or until the best found solution is 'good enough'. The fitter individuals from the current population are selected, crossover is applied to some of the selected individuals with a probability of crossover, p_X and mutation is applied to some of the selected individuals with the probability of mutation p_M. Finally some statistics regarding the current generation are gathered and the new population is evaluated using the fitness function; the iteration steps are repeated.

In the following, we will expand the different elements of the algorithm and their implementation. See listings 18 for a schematic view of the proposed algorithm.

Algorithm 18 Genetic Algorithm for computing the k-error linear complexity - A Schematic View

Input: A finite sequence $s = s_0, s_1, \ldots, s_{t-1}$ of size $t > 0$ with terms over a finite field $GF(q)$, where q is a prime power; an integer value k_0, with $0 < k_0 \leq w_H(s) - 1$.
Output: The approximate k_0-error linear complexity profile, L_i^*, $C_i^*(X)$ and e_i^*, for all $i = 0, 1, \ldots, k_0$.
Initialise the global solution
Initialise population POP_0 of size PS
Evaluate individuals in POP_0
$gen \leftarrow 0$
while $gen < NOGEN$ **do**
  Select new POP_{gen+1} from POP_{gen}
  Crossover individuals in POP_{gen+1} with probability p_X
  Mutate individuals in POP_{gen+1} with probability p_M
  Evaluate individuals in POP_{gen+1}
  Report statistics for current generation and update global solution
  $gen \leftarrow gen + 1$
end while
return global solution

The algorithm holds a global solution which is updated whenever necessary (i.e. when an individual improves the current global solution). Since the k-error linear complexity is calculated as a minimum value (from the definition), the profile is initialised with the maximum possible value, $L_i^*(s) = L(s)$, $C_i^*(X) = C(X)$ and $e_i^* = (\underbrace{0, 0, \ldots, 0}_{t \text{ times}})$, for all $i = 0, 1, \ldots, k_0$, where $C(X)$ is a minimal characteristic polynomial of s.

The algorithm processes a subset of the search space, therefore the returned value for the approximate k-error linear complexity will always be larger than or equal to the exact value. We will present the various components of the algorithm and then in section 5.1.3 we experimentally evaluate how close the approximation is to the exact value.

5.1.2.1 Individuals

Since the algorithm deals with finite sequences over finite fields, it is natural to use a string encoding for the individuals. We define a valid chromosome to be any 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 good chromosomes are the error patterns e which inflict smaller linear complexity on the input sequence s, e.g. $L(s+e)$ is smaller than $L(s)$. 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 E_k the set of sequences over the field $GF(q)$, with length t and Hamming weight at most k

$$E_k = \{e | e \in GF(q)^t, w_H(e) \leq k\},$$

therefore the search space size for the algorithm, SS, is given by the formula

$$SS = \sharp(E_{k_0}) = \sum_{i=0}^{k_0} \binom{t}{i} (q-1)^i \tag{5.1}$$

The initial population is randomly generated. The random number generator used is the C `rand()` linear congruential generator function. Algorithm 19 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 with PS. It is important to choose the right value for the population size but usually this is a value much smaller than the size of the search space. Some papers show that a moderate population size leads to fitter populations faster (e.g. Reeves [66]) however it is usually a case of experimental investigation.

It is desirable to scale the population size and number of generations with the size of the search space. Since the search space size depends on the input parameters t and k_0 and on the size of the field q, a formula can be devised to take into account these parameters as follows, where the coefficient c is such that $c > 0$.

$$PS = ck_0\lceil \ln(q^t) \rceil = ck_0\lceil t \ln q \rceil \tag{5.2}$$

The coefficient c is introduced to allow more tuning and we will experimentally check what value should c take in section 5.2.3. See figure 5.2 for a representation of the growth of the population size in relation to the growth of the space size, when working on the binary field ($q = 2$), taking $k_0 = \lceil 15\%t \rceil$ and the coefficient $c = 1$, for sequences of length 8, 16, 32, 64 or 128. The population size grows steadily 12, 33, 110, 440 and 1780, respectively, for the lengths considered, whereas the full search space size quickly becomes very large.

5.1.2.2 The fitness function

The quality of each individual is evaluated using a fitness function. The fitness function should reflect how good each solution is for the problem and provide a

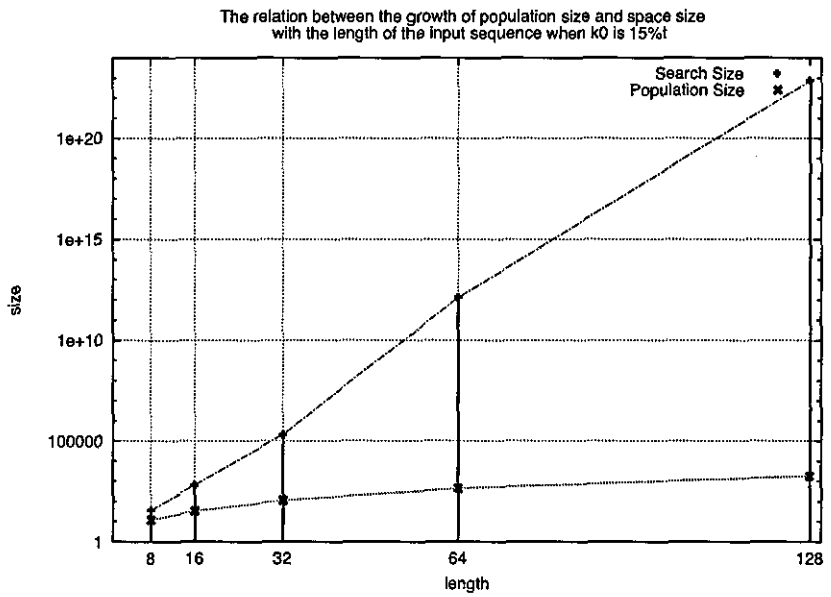

Figure 5.2: The relation between the growth of population size and search space size with the length of the input sequence when k_0 is $15\%t$.

good way of comparing two solutions. The goal of the k-error linear complexity problem is to find elements e in the set E_k which minimise the linear complexity of the sequence $s + e$. From this point of view, all possible error patterns, e, of Hamming weight up to k are comparable using the linear complexity of $s + e$, so this would be a natural choice for the fitness function.

In order to reflect the quality of each solution, the fitness of any error pattern e should therefore be proportional with the linear complexity of the sequence $s + e$. Since traditionally genetic algorithms are maximising and not minimising the fitness function, our choice for the fitness function of a valid error pattern e is the reverse of the linear complexity of the sequence onto which that error pattern has been applied, $s + e$, $f(e) = -L(s + e)$.

Formally, the definition of the fitness function for the kGA algorithm, f is

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

The fitness function for each element of the population can be computed using the Berlekamp-Massey Algorithm (section 2.2.6). The computational complexity of the evaluation step for one generation is therefore polynomial $\mathcal{O}(PS \cdot t^2) = \mathcal{O}(ck_0\lceil t \ln q \rceil t^2) \approx \mathcal{O}(t^4)$ (where we consider the following bound for k_0, $k_0 < t$).

Our experiments show that using this fitness function, the search space is

Algorithm 19 Generate a random sequence e of size t, $w_H(e) \leq k_0$

```
for i = 0, 1, ..., t − 1 do
    e_i ← 0
end for
k' ← a random number less than or equal to k_0
for i = 0, 1, ..., k' − 1 do
    pos ← a random position between 0 and t − 1
    val ← a random value in GF(q)
    e_pos ← val
end for
```

fragmented and there are many local minima and maxima. This fact can be a challenge 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 isolate the elements or set of elements from the domain E_{k_0} of function f which correspond to the minimum or the maximum values.

Example 5.1. *Figure 5.3 shows the shape of the distribution of linear complexities for a given binary sequence* $s = 1011110011010110$ *of size 16 when applying to it all the possible error patterns in the full search space* $GF(2)^{16}$. *The* x *and* y *axes correspond to each possible Hamming 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. We are interested in the error patterns corresponding to low* x *and low* y *coordinates.*

5.1.2.3 Selection

There are various possible schemes for the selection of best individuals for recombination. The general idea is that each chromosome will be copied zero, one or more times according to its fitness (more times if it is fitter) making sure that the population remains varied. The aim is to keep a good balance between the population diversity and the selective pressure.

Three alternative schemes of selection have been chosen for our experiments.

Elitist selection (ELSEL)

Elitist selection involves keeping only a certain top percentage of the population at each step, in decreasing order of the fitness values, and replacing the rest with completely new individuals. The higher the level of elitism (the percentage selected) the lower the efficiency of the algorithm as this will have to deal repeatedly with individuals which were previously processed. Further, since intuitively

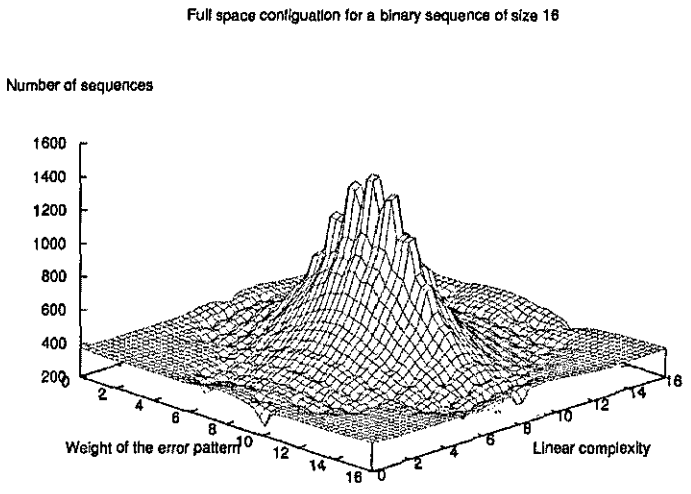

Figure 5.3: Distribution of linear complexities of $s = 0110111101110101$ when combined with all poosible error sequences over $GF(2)^{16}$

Table 5.1: Example of Elitist Selection of level 25% on sequence $s = 1011110011010110$ and $k_0 = 5$

No	Chromosomes before selection	Fitness	Chromosomes after selection	Fitness
0	0000100000101000	-8	**0000000000000100**	-7
1	**0000000000000000**	-8	**0000000000000000**	-8
2	0000100011001000	-8	0100001000100100	-7
3	0010010001010000	-11	0000001000000101	-9
4	0000000000100001	-9	0001100000101000	-8
5	0000001000000000	-9	0000000010010000	-9
6	0000000000000000	-8	0000010000010001	-9
7	0000100010000100	-8	1000000000000000	-7
8	**0000000000000100**	-7	0000010000000000	-6
9	0000000100010000	-8	0000010000100010	-8

the algorithm does not benefit from the overduplication of fit individuals, but more from the population diversity, the percentage kept is 25%; the rest of the individuals in the population are randomly generated using the same generation method as for the initial population (see listing 19). The computational complexity of this approach is polynomial, i.e. $\mathcal{O}(PS \cdot \ln(PS) + PS \cdot t^2) \approx \mathcal{O}(t^4)$ since it is necessary to order the individuals in the population by their fitness value and to evaluate the three quarters of population which are newly generated individuals. See table 5.1 for an example of Elitist Selection of level 25% for a sequence of size 16, $k_0 = 5$ when the population size is 10.

Roulette wheel with slots sized according to fitness (RWSEL)

The Roulette wheel selection technique allows a fair redistribution of the individuals based on their fitness. One disadvantage is that the population does not gain any new individuals during selection losing diversity, however this can be counteracted by a higher rate of crossover and mutation to compensate. This method duplicates the fitter individuals.

In the following, we summarise the classical roulette wheel method.

Denote the individuals in the population, $e^{(i)}$, with their fitness value $f(e^{(i)})$ for all $i = 0, 1, \ldots, PS - 1$. The total fitness of the population, denoted TF, represents the sum of the fitness values of all individuals:

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

The relative and the cumulative probability of each individual, denoted *rprob* and *cprob* respectively, are defined for each $i, 0 \leq i < PS$, as:

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

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

Note that the cumulative probabilities are increasing values between 0 and 1. A rotation of the roulette wheel consists of generating a random value r between 0 and 1. The element $e^{(i)}$ is chosen, such that i is minimal with the property that $r \leq cprob(e^{(i)})$. This step is repeated PS times to select all the individuals in the new population.

We will use a modified roulette wheel technique, one which emphasizes more the differences between the individuals in the population, based on their quality.

In order to give more strength to the error patterns e with low $L(s+e)$ for low Hamming weight $w_H(e)$ we apply an appropriate scaling to the slots of the wheel. In the context of the k-error linear complexity problem we need it to be more likely to choose error patterns with lower Hamming weight ($w_H(e)$ low) and which inflict a low linear complexity on the input sequence ($L(s+e)$ low). Therefore we want the size of the roulette wheel slots to be inversely proportional to the Hamming weight and to the absolute value of the fitness.

The total fitness scaled by Hamming weight, denoted TF', is defined as:

$$TF' = \sum_{i=0}^{PS-1} \frac{1}{(w_H(e^{(i)}) + 1)(|f(e^{(i)})| - 1)}$$

Adding 1 to the weight ($w_H(e) \geq 0$, for all $e \in K^t$) and subtracting 1 from the fitness function ($f(e) \leq 0$ for all $e \in K^t$) is necessary in order to avoid division by zero in the special cases when the weight or when the fitness function is 0. The relative and cumulative probabilities of each individual in this case, $rprob'$ and $cprob'$ respectively, are defined for each $i, 0 \leq i < PS$ as:

$$rprob'(e^{(i)}) = \frac{1}{(w_H(e^{(i)}) + 1)(f(e^{(i)}) - 1)TF'}$$

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

The selection process consists of spinning the roulette wheel PS times and each time selecting an existing individual, using the probabilities calculated above. This way, the fitter the individual, more likely it 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 $e^{(j)}$.

Algorithm 20 Roulette wheel with slots sized according to fitness (RWSEL)

```
for i = 0, 1, ..., PS − 1 do
    r ← a random value in [0, 1]
    if r < cprob(e^(0) then
        Select e^(0)
    else
        Find j such that cprob(e^(j−1)) < r ≤ cprob(e^(j)) and select e^(j)
    end if
end for
```

Tables 5.2 and 5.3 contain an example on how the Roulette Wheel works on a population of 10 individuals when calculating the k-error linear complexity for a sequence of size 16 and when $k_0 = 5$. In table 5.3 it can be seen that even though both individuals 6 and 7 give the same fitness value, individual 6 has a greater probability since its weight is zero, while the weight of individual 7 is 3, therefore there are greater expectations from it in reducing the linear complexity of the input sequence. The table 5.3 shows the way the selection is made on the basis of the random values r generated. We notice that the best individual is kept and the ones with high fitness are duplicated.

For each element of the population (PS elements) a random value is generated and a search for the right slot is performed (at most PS slots). Therefore, the

Table 5.2: Example of Roulette Wheel Selection on sequence s = 1011110011010110 and $k_0 = 5$ (part 1)

No	Chromosomes before selection	Fitness	relative prob.	cumulative prob.
0	0000000000000000	-8	0.190627	0.190627
1	0000000000000000	-8	0.190627	0.381255
2	0000100011001000	-8	0.0381255	0.41938
3	0010010001010000	-11	0.0285941	0.447975
4	0000000000100001	-9	0.0571882	0.505163
5	0000001000000000	-9	0.0857824	0.590945
6	0000000000000000	-8	0.190627	0.781573
7	0000100010000100	-8	0.0476569	0.82923
8	0000000000000100	-7	0.107228	0.936458
9	0000000100010000	-8	0.0635425	1

Table 5.3: Example of Roulette Wheel Selection on sequence s = 1011110011010110 and $k_0 = 5$ (part 2)

Random val. $\in (0, 1)$	Source	Chromosomes after selection	Fitness
0.449	4	0000000000100001	-9
0.786	7	0000100010000100	-8
0.734	6	0000000000000000	-8
0.897	8	0000000000000100	-7
0.189	1	0000000000000000	-8
0.913	8	0000000000000100	-8
0.335	1	0000000000000000	-8
0.109	0	0000000000000000	-8
0.518	5	0000001000000000	-9
0.954	9	0000000100010000	-9

computational complexity is polynomial, $\mathcal{O}(ck_0^2\lceil t \ln q \rceil^2) \approx \mathcal{O}(t^4)$.

Tournament Selection (TRSEL)

In a two order tournament selection model, random pairs of individuals from the current population are picked and the best one out of the two is selected to survive in the next population. Intuitively, this method would be particularly suitable as the fitness values for this problem are very close which makes the Roulette Wheel selection (even when using the scaled formulas) to give close probabilities of selection to most of the individuals. However, we notice that using this method the diversity of the population decreases very quickly after each generation.

Formally, the following two steps are repeated PS times:

1. Generate two random values pos_1 and pos_2, such that $0 \leq pos_1 < pos_2 \leq$

Table 5.4: Example of Tournament Selection on sequence $s = 1011110011010110$ and $k_0 = 5$

No	Chromosomes before selection	Fitness	Tournament between	Chromosomes after selection	Fitness
0	0000000000000000	-8	9 and 6	0000000100010000	-8
1	0000000000000000	-8	4 and 7	0000100010000100	-8
2	0000100011001000	-8	9 and 3	0000000100010000	-8
3	0010010001010000	-11	5 and 9	0000000100010000	-8
4	0000000000100001	-9	8 and 4	0000000000000100	-7
5	0000001000000000	-9	4 and 7	0000100010000100	-8
6	0000000000000000	-8	8 and 2	0000000000000100	-7
7	0000100010000100	-8	4 and 9	0000000100010000	-8
8	0000000000000100	-7	4 and 5	0000000000100001	-9
9	0000000100010000	-8	9 and 7	0000000100010000	-8

$PS - 1$.

2. If $f(e^{(pos_1)}) < f(e^{(pos_2)})$ then select $e^{(pos_2)}$, otherwise select $e^{(pos_1)}$.

See table 5.4 for an example of tournament selection applied to a population of 10 sequences of size 16 when $k_0 = 5$.

The computational complexity of this selection method is linear, $\mathcal{O}(t)$.

5.1.2.4 Crossover

The crossover involves choosing two parents, with the probability of crossover denoted p_X, $p_X \in [0, 1]$ and combining these parents to obtain new (possibly better) solutions. The following is the general crossover algorithm.

1. $i = 0$.
2. If $i \geq PS$ then STOP.
3. Generate a random value r, $r \in [0, 1]$.
4. If $r < p_X$ and no parent yet selected then choose first parent $p^{(1)} \leftarrow p^{(i)}$. Go to Step 4.
5. If $r < p_X$ and first parent $p^{(1)}$ has been selected, then choose the second parent $p^{(2)} \leftarrow p^{(j)}$ such that $p^{(i)} \neq p^{(j)}$, combine parents $p^{(i)}$ and $p^{(j)}$ to obtain one or two children and reset parents to be the best two out of the set of parents and children.
6. $i \leftarrow i + 1$. Go to Step 2.

In the following, we will denote the parent chromosomes $p^{(1)}$ and $p^{(2)}$. Having the two parents $p^{(1)}$ and $p^{(2)}$, the following standard crossover schemes are considered (Goldberg [20], Michalewicz [54]).

- Single point crossover (SPX). Generate a random natural number pos, with $pos \in \{0, 1, \ldots, t-2\}$.

  $p^{(1)} = (\underbrace{p_0^{(1)}, p_1^{(1)}, \ldots, p_{pos-1}^{(1)}}, p_{pos}^{(1)}, \ldots, p_{t-1}^{(1)})$

  $p^{(2)} = (\underline{p_0^{(2)}, p_1^{(2)}, \ldots, p_{pos-1}^{(2)}}, p_{pos}^{(2)}, \ldots, p_{t-1}^{(2)})$

  The resulting offspring are:

  $c^{(1)} = (\underbrace{p_0^{(1)}, p_1^{(1)}, \ldots, p_{pos-1}^{(1)}}, \underline{p_{pos}^{(2)}, \ldots, p_{t-1}^{(2)}})$

  $c^{(2)} = (\underline{p_0^{(2)}, p_1^{(2)}, \ldots, p_{pos-1}^{(2)}}, \underbrace{p_{pos}^{(1)}, \ldots, p_{t-1}^{(1)}})$

  This strategy provides some diversity without disrupting building blocks[1].

- Two point crossover (TPX). Generate two random natural numbers pos_1 and pos_2, such that $0 \leq pos_1 < pos_2 \leq t-2$.

  $p^{(1)} = (\underbrace{p_0^{(1)}, \ldots, p_{pos_1}^{(1)}, \ldots, p_{pos_2}^{(1)}}, \ldots, p_{t-1}^{(1)})$

  $p^{(2)} = (\underline{p_0^{(2)}, \ldots, p_{pos_1}^{(2)}, \ldots, p_{pos_2}^{(2)}}, \ldots, p_{t-1}^{(2)})$

  The resulting offspring are:

  $c^{(1)} = (\underbrace{p_0^{(1)}, \ldots}, \underline{p_{pos_1}^{(2)}, \ldots, p_{pos_2}^{(2)}}, \underbrace{\ldots, p_{t-1}^{(1)}})$

  $c^{(2)} = (\underline{p_0^{(2)}, \ldots}, \underbrace{p_{pos_1}^{(1)}, \ldots, p_{pos_2}^{(1)}}, \underline{\ldots, p_{t-1}^{(2)}})$

- Uniform random crossover (URX). Using this crossover technique, only one child, c, is obtained from each pair of two parents $p^{(1)}$ and $p^{(2)}$.

  $p^{(1)} = (p_0^{(1)}, p_1^{(1)}, \ldots, p_{t-1}^{(1)})$

  $p^{(2)} = (p_0^{(2)}, p_1^{(2)}, \ldots, p_{t-1}^{(2)})$

  The method involves generating t random real numbers, $r_i \in [0, 1]$, $i = 0, 1, \ldots, t-1$ where t is the length of the input sequence. For each i, if $r_i < 0.5$ then $c_i \leftarrow p_i^{(1)}$, otherwise $c_i \leftarrow p_i^{(2)}$ (Sywerda [81]).

We can devise crossover schemes which use some knowledge of the problem at hand, i.e. the problem of calculating the k-error linear complexity.

In order to achieve this, for each chromosome e, when calculating the linear complexity of $s + e$ (with the Berlekamp-Massey Algorithm) all the intermediary

[1]Building blocks are short sequences of good genes which appear within the chromosomes (in terms of our problem short subsequences of small linear complexity in s+e). It is desirable not to disrupt them if possible in order to promote them to the following generations.

linear complexities, i.e. the full linear complexity profile can be stored. That is, for each $e^{(i)}$, $i = 0, 1, \ldots, PS - 1$, the linear complexity profile $lcp^{(i)}$ is known, where $lcp^{(i)} = (lcp_0^{(i)}, \ldots, lcp_{t-1}^{(i)})$, a vector of size t such that $lcp_j^{(i)}$ represents the linear complexity of the sequence $s + e^{(i)}$ up to term j, i.e. linear complexity of the sequence $(s_0 + e_0^{(i)}, s_1 + e_1^{(i)}, \ldots, s_j + e_j^{(i)})$. The intermediary discrepancies can be also stored in an array $dis^{(i)}$, where $dis_j^{(i)}$ is the intermediary discrepancy at term j for sequence $s + e^{(i)}$, i.e. the difference between the element $s_j + e_j^{(i)}$ and the j^{th} element generated by the characteristic polynomial of the sequence $(s_0 + e_0^{(i)}, s_1 + e_1^{(i)}, \ldots, s_{j-1} + e_{j-1}^{(i)})$.

When processing term j of sequence $s+e^{(i)}$, only the case when the discrepancy $dis_j^{(i)} \neq 0$ and $2lcp_j^{(i)} \leq n$ (case (1b) in Section 2.2.6) yields an increase in the current complexity of the sequence. We are interested in minimising the linear complexity of $s + e$, i.e. the fitness function of e. It is therefore natural to change the current term part of the crossover in such a way to make the discrepancy zero and therefore make an increase in complexity unnecessary at position j.

The following two crossover techniques use the previous remark and the information given by the linear complexity profile as well as the intermediary discrepancies held against each chromosome.

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 $dis^{(2)}$.

- 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, i, such that the linear complexity for the sequence $s + p^{(1)}$ up to term i increases when processing term $i + 1$ and such that $p_{i+1}^{(1)}$ and $p_{i+1}^{(2)}$ differ. In other words, 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 by applying the following recombination it is possible for the linear complexity corresponding to the first child $L(s + c^{(1)})$ to be reduced.

  $p^{(1)} = (\underbrace{p_0^{(1)}, p_1^{(1)}, \ldots, p_{pos}^{(1)}, \ldots, p_i^{(1)}, p_{i+1}^{(1)}, \ldots, p_{t-1}^{(1)}})$

  $p^{(2)} = (\underline{p_0^{(2)}, p_1^{(2)}, \ldots, p_{pos}^{(2)}, \ldots, p_i^{(2)}, p_{i+1}^{(2)}, \ldots, p_{t-1}^{(2)}})$

  The resulting offspring are:

  $c^{(1)} = (\underbrace{p_0^{(1)}, p_1^{(1)}, \ldots, p_{pos}^{(1)}, \ldots, p_i^{(1)}}, \underline{p_{i+1}^{(2)}, \ldots, p_{t-1}^{(2)}})$

  $c^{(2)} = (\underline{p_0^{(2)}, p_1^{(2)}, \ldots, p_{pos}^{(2)}, \ldots, p_i^{(2)}}, \underbrace{p_{i+1}^{(1)}, \ldots, p_{t-1}^{(1)}})$

  This strategy provides good diversity without disrupting long building blocks.

- Two point crossover using the linear complexity profile (LCPTPX). One thing which can be extended from the LCPSPX crossover presented above, is to use the linear complexity information from 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 by applying the following recombination it is possible that in some of the cases the linear complexity corresponding to one or both children reduces, and therefore their fitness improves. Note that in the above it is assumed that $i < j$. This is not a restriction as if it does not happen $p^{(1)}$ and $p^{(2)}$, and also 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 resulting offspring 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 towards the end of sequences but the ones at the beginning of the sequence remain untouched.

If any of the previous schemes does not succeed in finding the right positions of crossover i or j with the required properties, then that crossover can be simply ignored.

The computational complexity of any of these crossover schemes is polynomial, i.e. $\mathcal{O}(t^4)$. For a certain percentage p_X of the population, approximately $p_X \cdot PS$ individuals, one or two children are generated and these need to be evaluated using Berlekamp-Massey Algorithm of computational complexity $\mathcal{O}(t^2)$.

5.1.2.5 Mutation

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

We consider two types of mutation, the standard random mutation and one which uses the linear complexity information similarly with the crossover schemes LCPSPX and LCPTPX presented in section 5.1.2.4. We define a parameter called the probability of mutation, $p_M \in [0, 1]$.

- Simple random mutation (SRM). This type of mutation iterates through all PS individuals in the population and for each of them, for all t terms, it generates a random value r, $r \in [0, 1]$ and if $r < p_M$ then it perturbates the current term with a random value from the field.

  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 random value val, $val \in GF(q)$ and $e_j^{(i)} = e_j^{(i)} + val$.

- Random mutation using the linear complexity profile (LCPRM). This mutation process tries to obtain individuals with a higher fitness by using the linear complexity profile 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, $i = 0, 1, \ldots, PS - 1$ 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 $lcp_m^{(i)} < lcp_{m+1}^{(i)}$. Make $e_{m+1}^{(i)} \leftarrow e_{m+1}^{(i)} - dis_{m+1}^{(i)}$. We remind that $dis_{m+1}^{(i)}$ represents the discrepancy at step $m + 1$ in the sequence $s + e^{(i)}$, namely the difference between term $s_{m+1} + e_{m+1}^{(i)}$ and the $(m + 1)^{th}$ term generated using the characteristic polynomial of the sequence $(s_0 + e_0^{(i)}, \ldots, s_m + e_m^{(i)})$. We choose this particular mutation since this subtraction will make the discrepancy of $s + e^{(i)}$ at position $m + 1$ to be zero, making it likely for the linear complexity of $s + e^{(i)}$ to be lower (certainly at index $m + 1$ the linear complexity will be lower) and therefore the fitness value of the error pattern $e^{(i)}$ to be larger.

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

The computational complexity of the mutation step is polynomial, i.e. $\mathcal{O}(t^4)$. A certain percentage p_M of the population PS is mutated and at most PS elements are evaluated using Berlekamp-Massey Algorithm of computational complexity $\mathcal{O}(t^2)$.

5.1.2.6 Summary

For both crossover and mutation, additional postprocessing is needed in order to check if the resulting offspring have a higher weight than the input value k_0 and if so, randomly switch some of the non zero terms to 0 until the weight is at most k_0. Whenever fitness values of individuals are evaluated, the global solution is updated if necessary.

Since the genetic algorithm depends on the input parameters as well as on a choice of a set of parameters, in the following we will refer to the genetic algorithm as: $kGA(t, k, s, PS, NOGEN, ST, XT, MT, p_X, p_M)$ where t, k and 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 5.1.2.3),
- XT is the crossover scheme used, it can be SPX, $LCPSPX$, TPX, $LCPTPX$, URX (see section 5.1.2.4),
- MT is the mutation scheme used, which can be SRM or $LCPRM$ (see section 5.1.2.5),
- 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.

Adding all the computational complexities of the different components and multiplying by the number of generations NOGEN which is a constant it follows that the algorithm has polynomial complexity, namely $\mathcal{O}(t^4)$.

5.1.3 Experiments 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 k-Error Genetic Algorithm we have set up a series of tests.

In the first experiment we consider 5 randomly chosen binary sequences of length 32 and $k_0 = 5$, i.e. approx. 15% of the length of the sequences (each bit of the sequences is generated with the C `rand()` linear congruential generator function). The search space size in this case is $SS = \sum_{i=0}^{5} \binom{32}{i} = 204469$ (see equation (5.1)). The population size according to relation (5.2) is $PS = ck_0\lceil t \ln q \rceil$. For $k_0 = 5$, $t = 32$ and $q = 2$, the value is $PS = 110c$. We try three different values for

the coefficient c, $c = 0.1$, $c = 1$ and $c = 10$ corresponding to small, medium and large sized populations, respectively. In order to have comparable running times and comparable results, we choose the number of generations, $NOGEN$ such that the product $PS \cdot NOGEN$ (which is a broad approximation for the number of patterns that the algorithm evaluates) is constant. Note though that the exact number of error patterns processed by the algorithm depends on the number of new individuals appearing from one generation to another. The following combinations for the pair $(PS, NOGEN)$ will be considered: $(11, 1000)$, $(110, 100)$ and $(1100, 10)$.

Each possible combination of selection (Elitist Selection with level 25% - ELSEL, Roulette Wheel - RWSEL, Tournament - TRSEL), crossover (Single Point Crossover - SPX, Two Point Crossover - TPX, Uniform Random Crossover - URX, Single Point Crossover using Linear Complexity Profile - LCPSPX, Two Point Crossover using Linear Complexity Profile - LCPTPX) and mutation (Simple Random Mutation - SRM, Random Mutation using Linear Complexity Profile - LCPRM) is considered with the different values for population size and numbers of generations as described above. The algorithm is run 5 times for each sequence and each combination of parameters with a different random seed.

In total, having 3 combinations of population size, 3 selection types, 5 crossover types, 2 mutation techniques and 5 repeated tests, we obtain 450 different runs for each sequence and store the approximate k_0-error linear complexity returned by each configuration and seed.

The probability of crossover is $p_X = 0.6$ and the probability of mutation is $p_M = 0.05$.

In the following, the evaluation of the algorithm is done by calculating the ratio between the approximate k_0-error linear complexity profile obtained by the k-Error Genetic Algorithm and the exact k_0-linear complexity profile (calculated using the Efficient Exhaustive Search Algorithm). We call this indicator the accuracy of the k-Error Genetic Algorithm. The accuracy of the results of the algorithm for each set of parameters (PS, NOGEN, Selection, Crossover, Mutation) is averaged over the 5 runs.

Table 5.5 contains the top 10 configurations of parameters which returned the best accuracy over the 5 input sequences. The results are summarised in figure 5.4. We omit the accuracy of the 0-error linear complexity in the tables as this is trivially the linear complexity of the input sequence and it will always be correctly calculated by the k-Error Genetic Algorithm by simply applying the Berlekamp-Massey Algorithm.

We notice that there is not a big difference in accuracy for the different configurations which give the best results.

Table 5.5: The accuracy of the results of $kGA(32, 5, s, \text{PS}, \text{NOGEN}, \text{ST}, \text{XT}, \text{MT}, 0.6, 0.05)$ - Top 10 best configurations

No.	Pop. Size	No. gen.	Selection Type	Crossover Type	Mutation Type	Accuracy	Acc_1	Acc_2	Acc_3	Acc_4	Acc_5
1	1100	10	ELSEL	LCPTPX	LCPRM	1.072	1.00	1.00	1.01	1.08	1.27
2	11	1000	ELSEL	LCPTPX	SRM	1.074	1.00	1.00	1.01	1.08	1.28
3	1100	10	ELSEL	LCPSPX	LCPRM	1.074	1.00	1.00	1.01	1.09	1.27
4	110	100	ELSEL	SPX	LCPRM	1.076	1.00	1.00	1.02	1.09	1.27
5	1100	10	ELSEL	TPX	LCPRM	1.078	1.00	1.00	1.00	1.09	1.30
6	1100	10	ELSEL	URX	LCPRM	1.078	1.00	1.00	1.03	1.08	1.28
7	11	1000	ELSEL	LCPSPX	LCPRM	1.08	1.00	1.00	1.03	1.08	1.29
8	110	100	ELSEL	SPX	SRM	1.082	1.00	1.00	1.04	1.11	1.26
9	11	1000	ELSEL	TPX	LCPRM	1.082	1.00	1.00	1.02	1.09	1.30
10	1100	10	ELSEL	URX	SRM	1.084	1.00	1.00	1.01	1.10	1.31

Table 5.6: The accuracy of the results of $kGA(32, 5, s, \text{PS}, \text{NOGEN}, \text{ST}, \text{XT}, \text{MT}, 0.6, 0.05)$ - Top 10 worst configurations

No.	Pop. Size	No. gen.	Selection Type	Crossover Type	Mutation Type	Accuracy	Acc_1	Acc_2	Acc_3	Acc_4	Acc_5
441	11	1000	TRSEL	URX	SRM	1.214	1.08	1.16	1.27	1.23	1.33
442	11	1000	RWSEL	SPX	LCPRM	1.220	1.06	1.10	1.22	1.30	1.42
443	11	1000	RWSEL	LCPTPX	LCPRM	1.222	1.05	1.13	1.25	1.30	1.38
444	11	1000	RWSEL	LCPSPX	LCPRM	1.228	1.06	1.12	1.25	1.30	1.41
445	11	1000	RWSEL	URX	LCPRM	1.230	1.08	1.13	1.27	1.30	1.37
446	11	1000	TRSEL	TPX	LCPRM	1.260	1.07	1.14	1.30	1.33	1.46
447	11	1000	TRSEL	LCPTPX	LCPRM	1.270	1.07	1.17	1.30	1.34	1.47
448	11	1000	TRSEL	URX	LCPRM	1.276	1.09	1.17	1.35	1.33	1.44
449	11	1000	TRSEL	LCPSPX	LCPRM	1.286	1.07	1.20	1.37	1.34	1.45
450	11	1000	TRSEL	SPX	LCPRM	1.304	1.09	1.17	1.35	1.40	1.51

For the top 10 worst configurations see table 5.6 and figure 5.5. Between the results of the worst configurations there are not big differences in accuracy either. We try therefore to identify what differentiates good configurations from bad ones.

The accuracies for different values of k, Acc_k are defined as $Acc_k = \frac{L_{kGA,k}(s)}{L_k(s)}$, where $L_{kGA,k}(s)$ is the approximate k-error linear complexity returned by the k-Error Genetic Algorithm. The accuracy column in the two tables represents the average accuracy over the 5 values of the k-error linear complexities calculated for $k = 1, 2, 3, 4, 5$. Namely, Accuracy $= \frac{\sum_{i=1}^{5} Acc_i}{5}$.

Looking at the table of top 10 best configurations, it is difficult to say which choice is better for each of the parameters as the configurations which give similarly accurate results can be quite different. For example, in table 5.5 the larger population size / small number of generations configuration is better since it gives top results five times out of ten, whereas the other configurations give top results in two or three out of ten cases, for medium population size/medium number of generations and for small population size / large number of generations, respec-

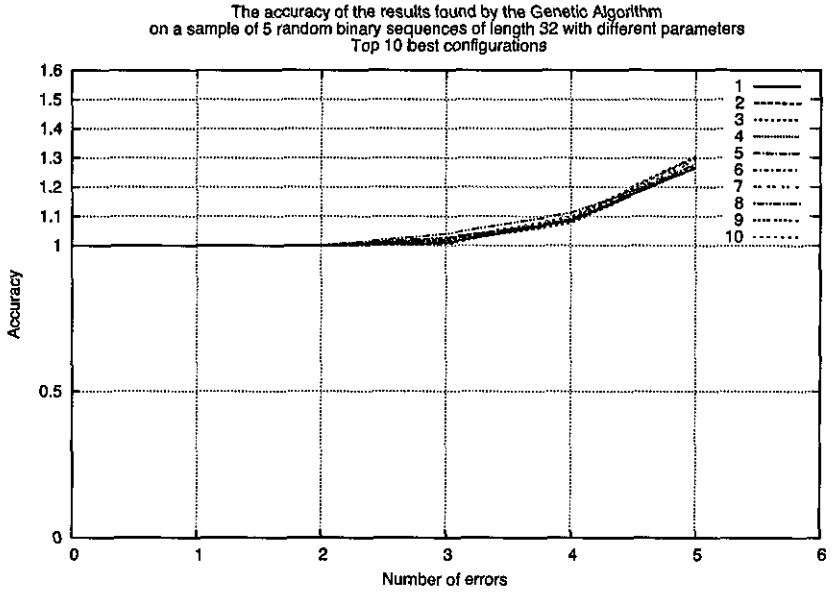

Figure 5.4: The accuracy of the results found by the Genetic Algorithm on a sample of 5 random binary sequences of length 32 with different parameters - Top 10 best configurations

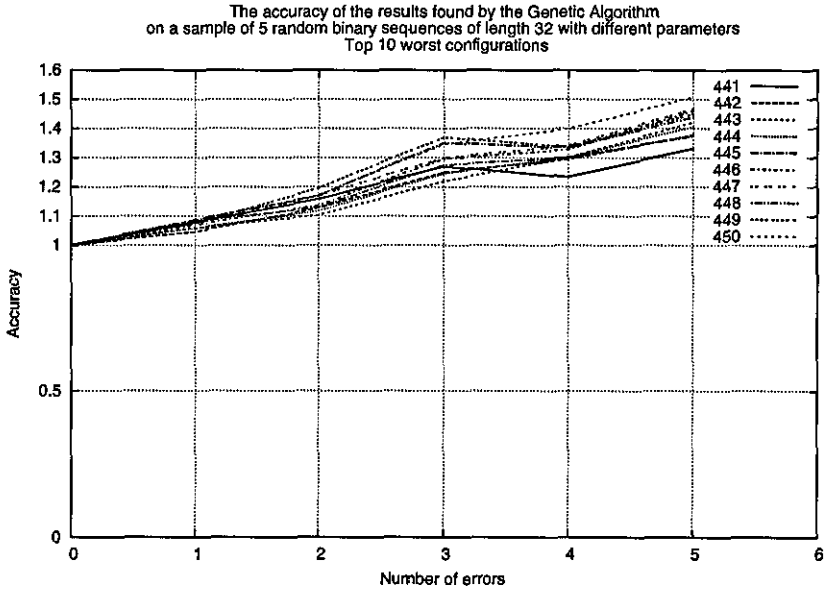

Figure 5.5: The accuracy of the results found by the Genetic Algorithm on a sample of 5 random binary sequences of length 32 with different parameters - Top 10 worst configurations

tively. Tables 5.5 and 5.6 show that the ELSEL selection leads to the best results. It is difficult though to choose the best crossover or mutation scheme.

This is the reason to analyse further the accuracy results when they are grouped by each of the parameters. We summarise these results in the following graphs.

- Figure 5.6 shows the results grouped by population size / number of generations. We notice that the best combination over the sequences included in this experiment is large population size / small number of generations. This choice matches the results in the top 10 best configurations table 5.5.

- Figure 5.7 shows the results grouped by selection scheme. It is very well delimited that the best selection scheme for this experiment is the elitist selection of level 25% and this matches the results shown in the top 10 best and worst configurations tables 5.5 and 5.6.

- Figure 5.8 shows the results grouped by crossover scheme. For the crossover schemes, it looks like in this experiment any scheme is as good as the other with the exception of the uniform random crossover (URX).

- Figure 5.9 shows the results grouped by mutation scheme. When grouping by mutation technique the simple random mutation (SRM) seems to give the best results. This is not reflected clearly by the top 10 best configurations table 5.5 but it is supported by the results summarised in the top 10 worst configurations table 5.6.

There are a few conclusions which we could draw out of the previous experimental analysis. A large population size / low number of generations, the ELSEL selection type, any type of crossover with the exception of the uniform random crossover (URX) and the simple random mutation (SRM) are the parameters which lead to the best results.

This shows that, even if we devised operators which use some knowledge of the problem in hand (like LCPSPX, LCPTPX or LCPRM), the benefit of using those operators is minimal. Moreover, the clear advantage of using the elitist selection scheme of order 25% (ELSEL) suggests that there is a higher benefit in refreshing the current population with new individuals which are randomly generated (keep the diversity of the population high), than keeping in the population the fittest individuals (keep the selective pressure high).

In order to have more confidence in the conclusions above we investigate how the algorithm scales with the length of the input sequence and if the choices in this case remain similar to the previous experiment. We take a binary sequence of length 64 and $k_0 = 9$ which is approximative 15% of the length of the sequence.

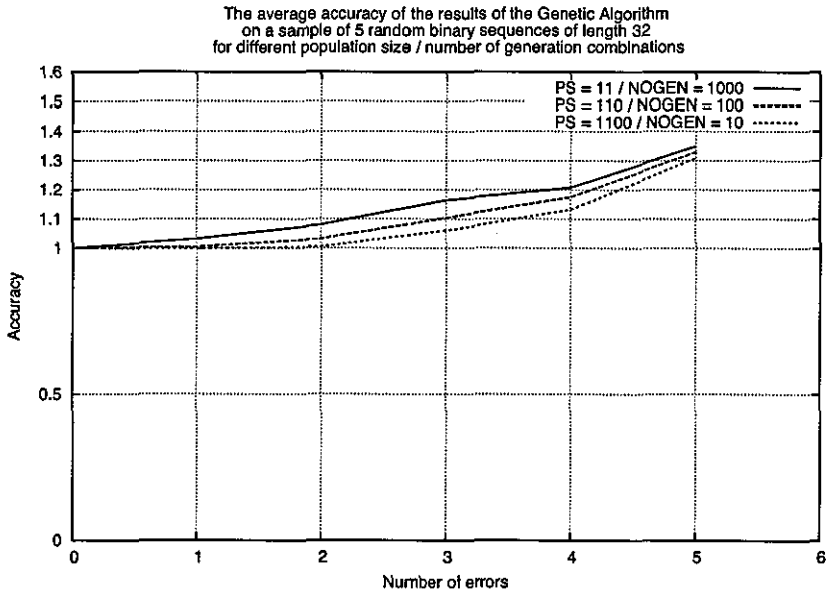

Figure 5.6: The average accuracy of the results of the Genetic Algorithm on a sample of 5 random binary sequences of length 32 for different population size / number of generations combinations

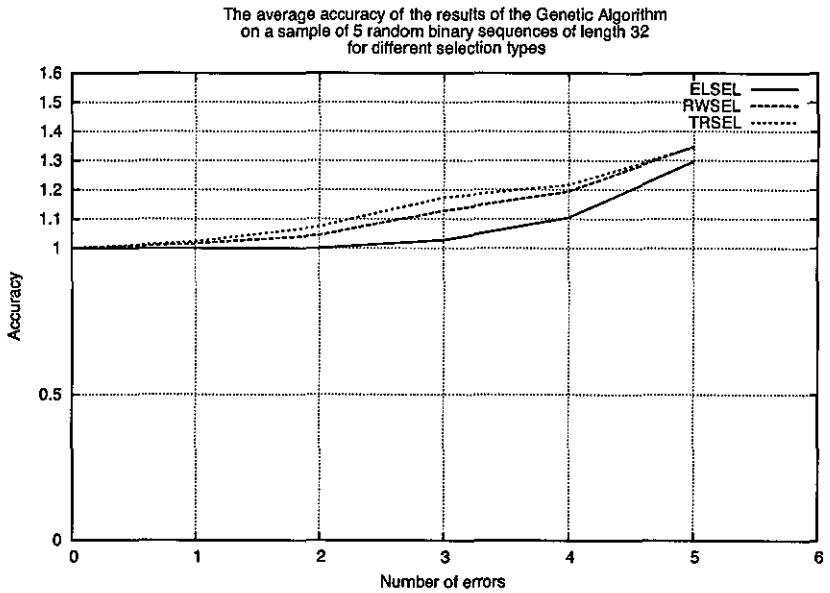

Figure 5.7: The average accuracy of the results of the Genetic Algorithm on a sample of 5 random binary sequences of length 32 for different selection types

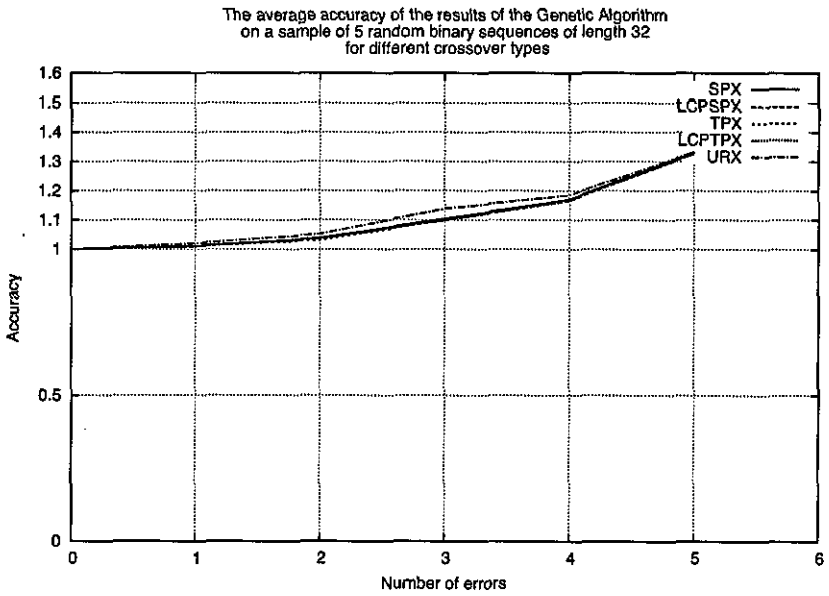

Figure 5.8: The average accuracy of the results of the Genetic Algorithm on a sample of 5 random binary sequences of length 32 for different crossover type

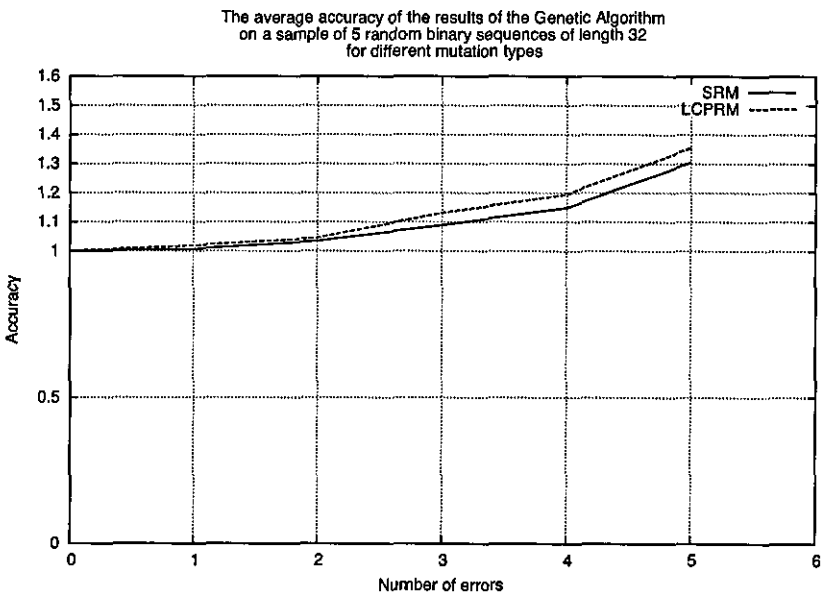

Figure 5.9: The average accuracy of the results of the Genetic Algorithm on a sample of 5 random binary sequences of length 32 for different mutation types

Table 5.7: The accuracy of the results of $kGA(64, 9, s,$ PS, NOGEN, ST, XT, MT, 0.6, 0.05) - Top 10 best configurations

No.	Pop. Size	No. gen.	Selection Type	Crossover Type	Mutation Type	Accuracy	Acc_1	Acc_2	Acc_3	Acc_4	Acc_5
1	4000	10	ELSEL	LCPTPX	SRM	1.205	1.00	1.00	1.02	1.14	1.16
2	4000	10	ELSEL	TPX	SRM	1.207	1.00	1.00	1.02	1.12	1.14
3	400	100	ELSEL	TPX	LCPRM	1.211	1.00	1.00	1.02	1.12	1.15
4	40	1000	ELSEL	TPX	LCPRM	1.214	1.00	1.00	1.04	1.14	1.19
5	400	100	ELSEL	LCPSPX	SRM	1.222	1.00	1.00	1.04	1.15	1.16
6	4000	10	ELSEL	TPX	LCPRM	1.228	1.00	1.00	1.02	1.13	1.16
7	40	1000	ELSEL	LCPTPX	LCPRM	1.229	1.00	1.00	1.04	1.15	1.19
8	40	1000	ELSEL	LCPSPX	SRM	1.229	1.00	1.00	1.04	1.16	1.20
9	400	100	ELSEL	LCPSPX	LCPRM	1.232	1.00	1.00	1.03	1.16	1.20
10	4000	10	RWSEL	SPX	LCPRM	1.233	1.00	1.00	1.07	1.18	1.16

The full search space size for a sequence of length 64 and for $k_0 = 9$ is $SS = \sum_{i=0}^{9} \binom{64}{i} = 2.430 * 10^{10}$ (see equation (5.1)). Therefore, applying the formula for the population size (relation (5.2)) we obtain $PS = ck_0\lceil t \ln q \rceil \simeq 400c$. We try three different values for the coefficient c, $c = 0.1$, $c = 1$ and $c = 10$ corresponding to a small, medium and large population size, respectively. Similarly with the previous experiment, in order to have comparable running times we choose the number of generations $NOGEN$ such that the product of the population size and number of generations is constant. The following pairs (PS, NOGEN) are considered (40, 1000), (400, 100) and (4000, 10). Each possible combination of selection, crossover and mutation is considered and, for each, the algorithm is run 5 times with different random seeds. In total, having 3 combinations of population size / number of generations, 3 selection schemes, 5 crossover types and 2 mutation types, we obtain 90 different runs. The probabilities of crossover and of mutation have the same values as in the first experiment, $p_X = 0.6$ and $p_M = 0.05$.

The tables 5.7 and 5.8 contain the top 10 best and worst configurations obtained respectively, and the figures 5.10 and 5.11 summarise the same results in a graphical representation. The accuracy in the tables is the average accuracy over the 9 non zero values of k in the approximate k-error linear complexity profile.

The difference of accuracy between the top best and worst configurations is smaller in this experiment, however the distribution of the results supports the conclusions drawn in the first experiment. A large population size / small number of generations, elitist selection type, any crossover type with the exception of the uniform random crossover are the configurations giving the best results (table 5.7). A small population size / large number of generations, tournament or roulette wheel selection and the uniform random selection all appear often at the bottom of the list as worst configurations, which confirms the conclusions of the first experiment (table 5.6 and 5.8).

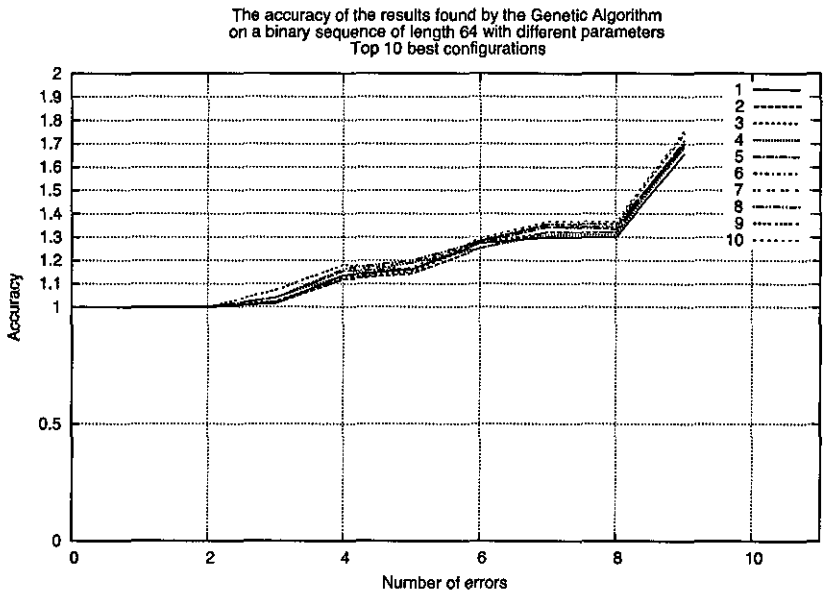

Figure 5.10: The accuracy of the results found by the Genetic Algorithm on a binary sequences of length 64 with different parameters - Top 10 worst configurations

Table 5.8: The accuracy of the results of $kGA(64, 9, s, \text{PS}, \text{NOGEN}, \text{ST}, \text{XT}, \text{MT}, 0.6, 0.05)$ - Top 10 worst configurations

No.	Pop. Size	No. gen.	Selection Type	Crossover Type	Mutation Type	Accuracy	Acc_1	Acc_2	Acc_3	Acc_4	Acc_5
81	400	100	TRSEL	LCPSPX	LCPRM	1.325	1.00	1.05	1.13	1.25	1.30
82	40	1000	TRSEL	LCPSPX	LCPRM	1.325	1.04	1.10	1.13	1.27	1.31
83	40	1000	RWSEL	LCPSPX	LCPRM	1.327	1.02	1.08	1.16	1.29	1.30
84	400	100	RWSEL	URX	LCPRM	1.329	1.01	1.07	1.11	1.25	1.30
85	400	100	TRSEL	URX	SRM	1.333	1.01	1.06	1.14	1.29	1.34
86	40	1000	TRSEL	TPX	LCPRM	1.339	1.01	1.10	1.16	1.30	1.31
87	40	1000	RWSEL	URX	LCPRM	1.352	1.03	1.10	1.18	1.34	1.37
88	40	1000	RWSEL	URX	SRM	1.368	1.06	1.11	1.19	1.34	1.38
89	40	1000	TRSEL	URX	SRM	1.371	1.00	1.06	1.13	1.19	1.34
90	40	1000	TRSEL	URX	LCPRM	1.372	1.00	1.06	1.12	1.21	1.35

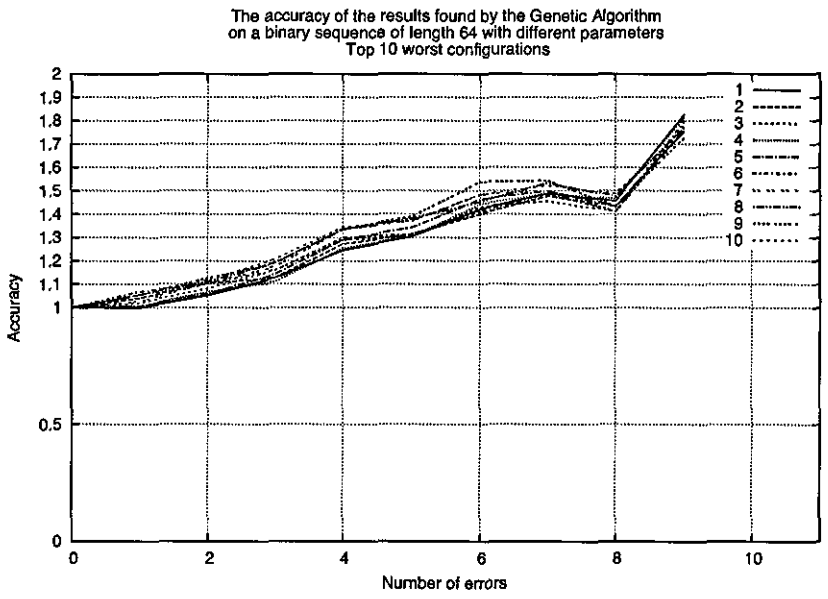

Figure 5.11: The accuracy of the results found by the Genetic Algorithm on a binary sequences of length 64 with different parameters - Top 10 worst configurations

The same conclusions are supported when the accuracy results are grouped by each of the different parameters.

- Figure 5.12 shows the results grouped by population size / number of generations.
- Figure 5.13 shows the results grouped by selection scheme.
- Figure 5.14 shows the results grouped by crossover scheme.
- Figure 5.15 shows the results grouped by mutation scheme.

We noticed a strong correlation between the number of different error patterns evaluated throughout a run of the k-Error Genetic Algorithm and the accuracy of the results. This is a natural remark since the processing of a bigger set of individuals will always return a better approximation of the solution.

We estimate the number of individuals e which are evaluated throughout the algorithm, by evaluation meaning that the algorithm calculates the linear complexity of $s + e$ in order to find its fitness value. This number of such individuals is $PS + NOGEN(n_S + n_X + n_M)$ as the algorithm first evaluates the initial population and then for each generation it evaluates a certain number of individuals

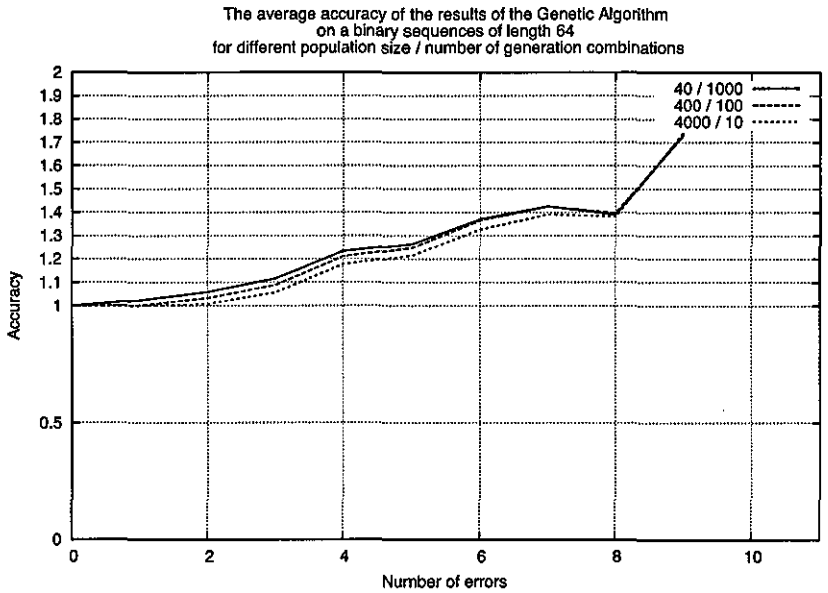

Figure 5.12: The average accuracy of the results of the Genetic Algorithm on a binary sequences of length 64 for different population size / number of generations combinations

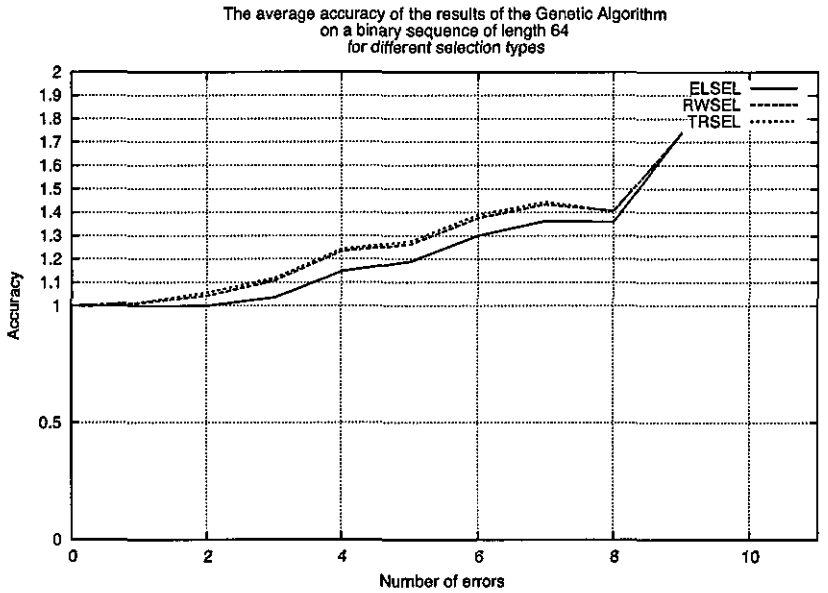

Figure 5.13: The average accuracy of the results of the Genetic Algorithm on a binary sequences of length 64 for different selection types

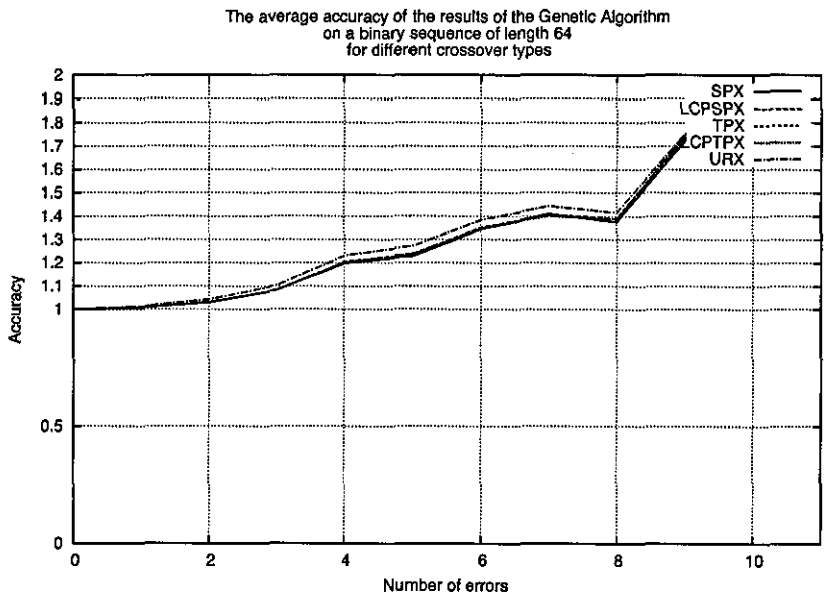

Figure 5.14: The average accuracy of the results of the Genetic Algorithm on a binary sequences of length 64 for different crossover type

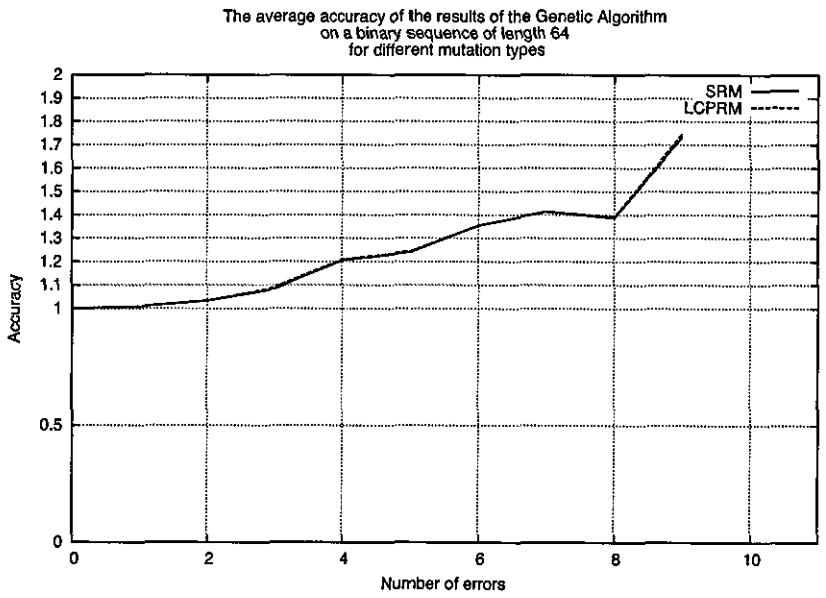

Figure 5.15: The average accuracy of the results of the Genetic Algorithm on a binary sequences of length 64 for different mutation types

Table 5.9: The 5-error linear complexity results of $kGA(32, 5, s^{(i)}, 1100, 10, ELSEL(25\%), TPX, SRM, 0.6, 0.05)$ compared to the exact values

s	$s^{(0)}$	$s^{(1)}$	$s^{(2)}$	$s^{(3)}$	$s^{(4)}$
Exact value	10	9	9	8	10
GA Value	10	11	11	9	10
Generation	10	3	1	3	7

depending on the chosen selection, crossover and mutation scheme, where we denote the number of individuals evaluated during selection, crossover and mutation, with n_S, n_X and n_M respectively.

For the roulette wheel and tournament selection no individuals are evaluated, as the selection works only on the existing population so $n_S = 0$ in this case. For the elitist scheme of level 25%, 75% of the population is replaced with new individuals which need to be evaluated. It is easy to estimate $n_X = 2 \cdot p_X \cdot PS$ and $n_M = t \cdot p_M \cdot PS$. These two values are not varying when changing the crossover or mutation type. Therefore the only element which makes up the amount of new individuals to vary is the selection scheme. We think that this is the reason why the elitist selection scheme has shown the most significant improvement in accuracy when compared to the other methods.

In order to check if there is any convergence pattern for the k-Error Genetic Algorithm, namely if after a certain number of generations the solution stabilises and converges slower to the optimum, we do the following additional analysis on the previous two experiments.

Table 5.9 shows the approximate values of the 5-error linear complexity found by the k-Error Genetic Algorithm applied to each of the test sequences $s^{(i)}$ (with $0 \leq i < 5$) of length 32, when population size is 1100, number of generations is 10, selection scheme is elitist selection of level 25%, crossover is the two point crossover with probability of crossover 60% and mutation is the simple random mutation with probability of mutation 5%. This table displays the exact values of the 5-error linear complexity and the generation number when the k-Error Genetic Algorithm has found the solution.

Table 5.10 shows the approximate value found by each of the runs for the best combination of parameters: population size 4000, number of generations 10, selection scheme elitist of level 25%, two point crossover with probability of crossover 60% and the simple random mutation with probability of mutation 5%. The table displays the approximate values of the 9-error linear complexity and the generation number when the algorithm kGA has found the solution.

With these results it is difficult to say if continuing the algorithm for a few more generations or stopping the algorithm a few generations before would have

Table 5.10: The 9-error linear complexity results of $kGA(64, 9, s^{(i)}, 4000, 10, ELSEL(25\%), TPX, SRM, 0.6, 0.05)$ compared to the exact value of $L_9(s) = 14$

Runs	1	2	3	4	5
GA Value	25	24	24	25	25
Generation	2	9	4	3	8

made any improvements.

5.1.4 Conclusion

We propose a genetic algorithm for approximating the k-error linear complexity of a sequence over a finite field. We implement and analyse various techniques for each of the evolutionary operators and investigate the best choice of parameters for the algorithm. The genetic algorithm approach seems suitable to the problem especially for inputs which generate a large search space but it is difficult to control in terms of accuracy. A good scheme would use a large sized population, an elitist type of selection with a level of 25%, two point random crossover with a probability of 0.6 and a standard random mutation with a probability of 0.05. With these choices, the algorithm outputs an approximate value of the k-error linear complexity which is on average only 7.2% higher than the exact value for sequences of length 32 when $1 \leq k_0 \leq 5$ and 20.5% higher for sequences of length 64 when $1 \leq k_0 \leq 9$.

Even though the advantage of this technique is that it is simple, quick and allows for different levels of accuracy, the algorithm can turn out to be unreliable and not scalable with the size of the search space.

5.2 Simulated Annealing Algorithm

This section presents an alternative evolutionary heuristic technique called simulated annealing, applied to the k-error linear complexity profile problem for sequences over finite fields. We design an algorithm using the simulated annealing technique and evaluate its behaviour. The algorithm proves to be efficient and has good accuracy. It outputs an approximate value of k-error linear complexity on average 12.4% higher for sequences of length 32 and 38% higher for sequences of length 64.

5.2.1 Background

Independently presented by Kirkpatrick et al. [33] and Cerny [9], the simulated annealing (SA) technique is a generic probabilistic meta-algorithm used for global optimisation of problems. The method is inspired from the metallurgic technique of annealing, where materials are heated and then slowly cooled (according to a cooling schedule) to increase the size and the order of its crystals. These principles were first incorporated into numerical calculations by Metropolis et al. [53].

Similarly to the physical process, the simulated annealing algorithm is iteratively 'cooling' or mutating an initial random solution e to a randomly close solution e'. This new solution e' replaces e with a probability that depends on the difference between the quality of the two solutions and on a global parameter T called temperature. If the new solution is better (judging by a fixed objective function or a so called energy) then it is chosen for the next step. If the new solution is worse then it can still be chosen for the next step with a probability $P(e, e', T)$ which depends on how much variance of energy the current temperature T of the system allows between the two solutions e and e'. In order to implement the above there is need to define the distance between two solutions and an objective function which reflects the quality of each possible solution. The temperature schedule allows the system to evolve initially at a high temperature and then at progressively lower temperatures until the system 'freezes' in a near optimum state when no further mutations occur (Kirkpatrick [33]). The cooling schedule is governed by a decreasing function c_s, at each step the current temperature T being replaced by $c_s(T)$.

The fact that at high temperatures, the algorithm allows the solution to mutate to elements which do not improve the objective function is one of the advantages of this method. This prevents the algorithm from getting stuck into a local minimum, one of the risks when using 'greedier' methods like the steepest descent method heuristic (Goldberg [20]).

Simulated annealing is typically suited to optimisation problems of large scale, especially where the target global optimum is hidden among many local optima, and where the space of solutions is discrete and very large therefore cannot be explored exhaustively. Furthermore, since the set is discrete, there is no notion of 'continuing downhill in a favourable direction', as the concept of direction may not have any meaning in the search space (Press et al. [64]).

This technique has been successful in a range of real life disciplines from physics to neural networks, image processing and finance, as well as classical NP-Complete problems like the Travelling Salesman Problem (Ingber [26]). Listing 21 shows a classical Simulated Annealing Algorithm.

Algorithm 21 Simulated Annealing Algorithm

```
e ← random initial solution
T ← T_0
while (T ≥ T_freeze and termination condition not met) do
    for 1, ..., number of steps at constant temperature do
        e' ← random solution in the neighbourhood of e
        if f_O(e) < f_O(e') then
            e ← e'
        else
            e ← e' with probability P(e, e', T)
        end if
    end for
    T ← c_s(T)
end while
return e
```

5.2.2 kSA Algorithm

In this section we describe the k-Error Simulated Annealing Algorithm for approximating the k-error linear complexity of sequences with elements in a finite field $GF(q)$, with q a prime power.

Similar to the k-Error Genetic Algorithm the input is a finite sequence $s = s_0, s_1, \ldots, s_{t-1}$ of size $t > 0$ with terms over a finite field $GF(q)$, where q is a prime power and an integer value k_0, with $0 < k_0 \leq w_H(s) - 1$. The expected output is an approximate k_0-error linear complexity profile of s containing for each $i = 0, 1, \ldots, k_0$, L_i^*, the approximate i-error linear complexity; e_i^*, the error pattern producing the linear complexity L_i^* on s; $C_i^*(X)$ a minimal characteristic polynomial corresponding to the sequence $s + e_i^*$.

See listing 22 for a schematic view of the algorithm.

The elements which characterise the k-Error Simulated Annealing Algorithm are:

- the population of possible solutions,
- the objective function,
- the cooling schedule,
- the mutation technique,
- the probability of acceptance.

We will describe our implementation for each of these elements in the following.

The population of possible solutions is the set of all error sequences e of weight at most k_0, $E_{k_0} = \{e \in GF(q)^t | w_H(e) \leq k_0\}$. Therefore the size of the search

space is $SS = \sum_{i=0}^{k_0} \binom{t}{i}(q-1)$ (see definition 2.13 in section 2.2, subsection 2.2.7), where t is the size of the input sequence, q is the order of the finite field and k_0 is the input value of the algorithm.

The objective function is the function which needs to be minimised by the algorithm. In our case, given the definition of the k-error linear complexity, we can choose as objective function the function which maps each error pattern in the search space to the linear complexity of the sum sequence $s+e$, $L(s+e)$. We denote the objective function f_O.

$$f_O : GF(q)^t \rightarrow \mathbf{N}, \text{ defined as } f_O(e) = L(s+e).$$

The cooling schedule is characterised by the number of cooling steps (denoted *noCoolingSteps*), the number of iterations at the each temperature (denoted *noConstTSteps*) and the cooling function (c_s).

Just as with any heuristic search technique we aim to find a way to only process a small, representative subset of all possible solutions. Due to the high variance of the search space size when t, k_0 and q vary, we need to tune the number of elements processed depending on these values.

The choice of the cooling schedule is very important. The cooling speed, similar to the physical phenomenon of metalurgic annealing, directly affects the efficiency of the algorithm. There are a few celebrated cooling schedule procedures, e.g. logarithmic, exponential or adaptive schedules (Ingber [27]). We will use a classical exponential cooling schedule which needs a cooling parameter $\alpha \in (0,1)$ and the decreasing function c_s is defined by $c_s(T,\alpha) = \alpha T$. Therefore, assuming that the initial temperature is T_0, at any step i, the temperature $T_i = \alpha^i T_0$.

The number of elements processed by the algorithm is the product of the number of cooling steps, denoted *noCoolingSteps* (in how many steps the temperature drops from the initial temperature to a freezing temperature) and the number of steps spent by the algorithm at each temperature, denoted *noConstTSteps*.

The initial temperature needs to allow very erratic movements within the search space, any mutation of the current element, whether it improves the objective function or not, should be accepted. Therefore we choose the initial temperature to be $T_0 = t$ which is a loose upper bound for the possible difference between the values of the objective function applied to any two different error sequences from the search space. Since the objective function has a discrete codomain we can choose the freezing temperature to be $T_{freeze} = 1$.

By fixing the initial temperature and the freezing temperature we can estimate the number of cooling steps using the cooling function c_s. The algorithm stops as soon as the current temperature becomes less than the freezing temperature

$T_{freeze} = 1$. The temperature at step i is $T_i = \alpha^i T_0$. Therefore we can write the following chain of relations:

$$T_{noCoolingSteps} \leq T_{freeze} \Leftrightarrow \alpha^{noCoolingSteps} T_0 \leq 1 \Leftrightarrow$$

$$noCoolingSteps + \log_\alpha T_0 \leq 0 \Leftrightarrow noCoolingSteps \leq -\log_\alpha T_0$$

We can therefore take

$$noCoolingSteps = \lfloor -\log_\alpha t \rfloor$$

When choosing the number of iterations at each temperature, there are two aspects which we find important to take into account. Firstly, we need to spend less time at each temperature if the cooling schedule is slow and more time if the cooling schedule is fast. Secondly, the total number of mutations in the algorithm needs to scale accordingly with the search space size.

With these two requirements in mind, we use the following formula:

$$noConstTSteps = \lfloor \frac{rqk_0 t}{\alpha} \rfloor,$$

where the coefficient $r > 0$ is an additional parameter which allows for more tuning if necessary. This way the number of steps at constant temperature is inversely proportional to the cooling schedule coefficient α (first requirement). Also the number of steps at constant temperature is directly proportional with the size of the field q, the number of allowed errors k_0 and with the size of the input sequence t (second requirement).

The parameter $noFailSteps$ indicates how many consecutive mutations towards solutions of lower fitness are allowed before cancelling the current temperature and cooling to the next one in the schedule. We choose the following formula for $noFailSteps = \frac{noConstTSteps}{t}$.

Figure 5.16 shows how the number of elements in the search space for a particular instance of the problem and the maximum number of error patterns processed by the algorithm varies with the length of the input sequence t, where $8 \leq t \leq 128$, $q = 2$ and $k_0 = 15\%t$. The number of error patterns processed by the k-Error Simulated Annealing Algorithm is

$$noCoolingSteps \cdot noConstTSteps = \lfloor -\log_\alpha t \rfloor \cdot \lfloor \frac{rqk_0 t}{\alpha} \rfloor$$

$$\leq \lfloor -\frac{\log t}{log\alpha} \frac{rq0.15t^2}{\alpha} \rfloor \simeq \mathcal{O}(t^2 \log t)$$

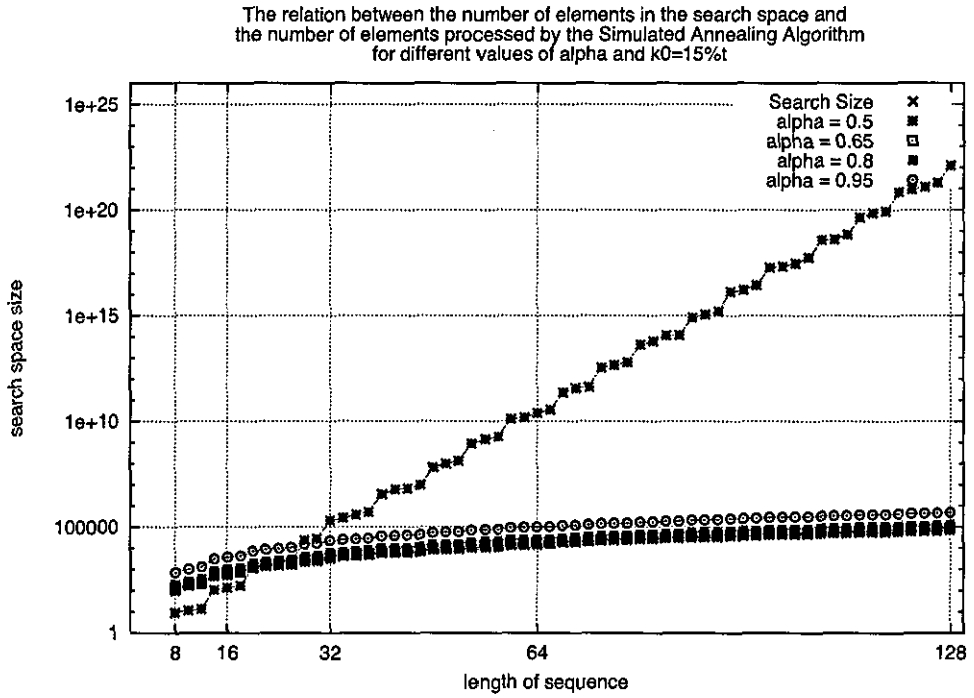

Figure 5.16: The relation between the number of elements in the search space and the number of elements processed by the Simulated Annealing Algorithm for different values of alpha.

Four values for the cooling parameter α are included in the graph, namely $\alpha = 0.5, 0.65, 0.8, 0.95$. While the search space size grows exponentially, the number of elements processed by the k-Error Simulated Annealing Algorithm grows at a much slower speed following a function upper bounded by $\mathcal{O}(t^2 \log t)$.

The probability of acceptance for a solution e', denoted $P(e, e', T)$, is usually following a Boltzmann-like distribution (Kirkpatrick et al. [33], Ingber [27], Press et al. [64]), i.e.

$$P(e, e', T) = \exp\left[-\frac{f_O(e) - f_O(e')}{T}\right].$$

The initial element e is randomly generated such that it has weight exactly k_0. That is done by generating k_0 random positions between 0 and $t-1$ (each bit is generated using C `rand()` linear congruential generator function) and assigning to the terms in these positions, random values from the current field, $GF(q)$. Formally, for each $i = 0, \ldots, k_0 - 1$, generate a position $pos_i \in \{0, 1, \ldots, t-1\}$ and a random value $val_i \in GF(q)$ and attribute $e_{pos_i} \leftarrow val_i$.

We consider two different implementations for the mutation procedure, denoted

Algorithm 22 Simulated Annealing Algorithm for the k-error linear complexity profile problem - A Schematic View

Input: A finite sequence $s = s_0, s_1, \ldots, s_{t-1}$ of size $t > 0$ with terms over a finite field $GF(q)$, where q is a prime power; an integer value k_0, with $0 < k_0 \leq w_H(s) - 1$.

Output: The approximate k_0-error linear complexity profile, L_i^*, $C_i^*(X)$ and e_i^*, for all $i = 0, 1, \ldots, k_0$.

for $i = 0, 1, \ldots, k_0$ **do**
  $L_i^* \leftarrow L(s)$
  $C_i^*(X) \leftarrow C(X)$, a minimal characteristic polynomial
  $e_i^* \leftarrow \underbrace{(0, 0, \ldots, 0)}_{t \text{ times}}$
end for
$noCoolingSteps \leftarrow \lfloor -\log_\alpha t \rfloor$
$noConstTSteps \leftarrow \lfloor \frac{rqk_0t}{\alpha} \rfloor$
$noFailSteps \leftarrow \frac{noConstTSteps}{t}$
$T \leftarrow t$
$i \leftarrow 0$
$e \leftarrow$ a random sequence over $GF(q)$ with $w_H(e) \leq k_0$
while $i < noCoolingSteps$ **do**
  $noFS \leftarrow 0$
  $j \leftarrow 0$
  while $j < noConstTSteps$ and $noFS < noFailSteps$ **do**
    $e' \leftarrow$ mutate(e)
    if $f_O(e') < f_O(e)$ **then**
      $e \leftarrow e'$
      $noFS \leftarrow 0$
    else
      $U \leftarrow$ a random variable $\in (0, 1)$
      if $U < P(e, e', T)$ **then**
        $e \leftarrow e'$
        $noFS \leftarrow 0$
      else
        $noFS \leftarrow noFS + 1$
      end if
    end if
    $j \leftarrow j + 1$
  end while
  if $L^*_{w_H(e)} > L(s + e)$ **then**
    update the global solution, L_i^*, $C_i^*(X)$ and e_i^* for $i = 0, 1, \ldots, k_0$
  end if
  $T \leftarrow c_s(T, \alpha)$
  $i \leftarrow i + 1$
end while
return L_i^*, $C_i^*(X)$ and e_i^* for $i = 0, 1, \ldots, k_0$

$mutate(e)$.

One of the implementations uses extra information provided by the linear complexity profile of each sum sequence $s+e$, in a similar way to the mutation scheme used for the k-Error Genetic Algorithm. For each error pattern e, when calculating the linear complexity of $s+e$ with the Berlekamp-Massey Algorithm we hold all the intermediary linear complexities, obtaining therefore the full linear complexity profile, lcp (see definition 2.2). We denote lcp_j, for $j = 0, 1, \ldots, t-1$, the linear complexity of the sequence $s+e$ up to term j. We denote dis_j the intermediary discrepancy at term j for the sequence $s+e$, i.e. the difference between the term $s_j + e_j$ and the j-th term generated using the characteristic polynomial of the sequence $(s+e)_0, \ldots, (s+e)_{j-1}$.

Algorithm 23 Simple Random Mutation - mutate(e)

$m \leftarrow$ random value in $\{0, 1, \ldots, 2k_0\}$
for $i = 0, 1, \ldots, m-1$ **do**
  $pos_i \leftarrow$ random value in $\{0, 1, \ldots, t-1\}$
  $val_i \leftarrow$ random value in $GF(q)$
  $e_{pos_i} \leftarrow e_{pos_i} + val_i$
end for
return e

We consider the following mutation types:

- Simple random mutation (SRM). When moving from one element $e \in GF(q)^t$ to another $e' \in GF(q)^t$, the Hamming weight needs to stay less than k_0. We modify some of the terms of the sequence such that there are high chances for the Hamming weight of the mutated sequence to be still under k_0. We first generate a random integer value m, $m \in \{0, 1, \ldots, 2k_0\}$ and then generate m positions pos_i, $i = 0, 1, \ldots, m-1$. We add randomly generated values, $val_i \in GF(q)$, on the generated positions so that $e_{pos_i} \leftarrow e_{pos_i} + val_i$ for each $i = 0, 1, \ldots, m-1$. See listing 23 for this implementation.

- Random mutation using the linear complexity profile (LCPRM). This mutation procedure attempts to mutate the current error sequence e to good elements by using the linear complexity profile and the discrepancy information for a better chance to minimise the objective function of the new element e', $L(s+e')$. Formally, generate a random position $u \in \{0, 1, \ldots, t-1\}$ and find the first position m in e such that $u < m$ and $lcp_m < lcp_{m+1}$. Once m is found, make $e'_{m+1} = e_{m+1} - dis_{m+1}$, where dis_{m+1} is the discrepancy at step $m+1$ in the sequence $s+e$. Note that, if no suitable $m \leq t-1$ is found then the last term is mutated randomly $e'_{t-1} = e'_{t-1} - dis_{t-1}$. See listing 24 for this implementation.

Algorithm 24 Random Mutation using the linear complexity profile - mutate(e)

$u \leftarrow$ random value in $\{0, 1, \ldots, t-1\}$
$m \leftarrow u$
while $m \leq t-1$ and $lcp_m = lcp_{m+1}$ **do**
  $m \leftarrow m+1$
end while
$e_{m+1} \leftarrow e_{m+1} - dis_{m+1}$
return e

In both cases, if the resulting sequence e' has the Hamming weight more than k_0 then we randomly switch to zero some of the non zero terms until the weight becomes less then k_0.

The computational complexity of the algorithm is determined by the number of the error patterns processed, as for each error pattern e the Berlekamp-Massey Algorithm is used to calculate the linear complexity of $s+e$ for the objective function, $f_O(e) = L(s+e)$.

Therefore the computational complexity is:

$$noCoolingSteps \cdot noConstTSteps \cdot t^2 = \lfloor -\log_\alpha t \rfloor \cdot \lfloor \frac{rgk_0t}{\alpha} \rfloor t^2$$
$$\leq \lfloor -\frac{\log t}{log\alpha} \frac{rq0.15t^2}{\alpha} t^2 \rfloor \simeq \mathcal{O}(t^4 \log t)$$

5.2.3 Experiments and results

We set several experiments in order to assess the accuracy and efficiency of the algorithm. The parameters considered in our tests are summarised below:

$T_0 = t$, $noCoolingSteps = \lfloor -\log_\alpha t \rfloor$
$\alpha \in (0,1)$, $noConstTSteps = \lfloor \frac{rgk_0t}{\alpha} \rfloor$
$c_s(T) = \alpha T$, $noFailedSteps = \lfloor \frac{noConstTSteps}{t} \rfloor$
$r \in (0,1)$, $mutationType \in \{SRM, RMLCP\}$

In order to compare with the exact result (obtained using the Efficient Exhaustive Search Algorithm) and also to compare with the results of the k-Error Genetic Algorithm (section 5.1), we consider the same set of 5 random binary sequences of length 32 and $k_0 = 5$ (each bit generated using the C `rand()` linear congruential generator function). The full search space size for an exhaustive search is $\sum_{i=0}^{5} \binom{32}{i} = 204469$. This is quite a small search space to justify a heuristic method, but it allows us to compare with the exhaustive search and to assess the efficiency and accuracy of the proposed algorithm.

We try four different values for the cooling coefficient $\alpha = 0.5, 0.65, 0.8, 0.95,$

Table 5.11: The accuracy for the k-Error Simulated Annealing Algorithm applied to 5 binary sequences of length 32 and $k_0 = 5$.

Mutation type	α	Accuracy	Acc_1	Acc_2	Acc_3	Acc_4	Acc_5	Error patterns
LCPRM	0.95	1.124	1.12	1.11	1.14	1.12	1.13	19036
LCPRM	0.8	1.164	1.12	1.15	1.23	1.12	1.20	5360
SRM	0.95	1.208	1.12	1.18	1.26	1.21	1.27	19976
LCPRM	0.65	1.222	1.12	1.17	1.28	1.23	1.31	3692
LCPRM	0.5	1.224	1.12	1.19	1.27	1.23	1.31	2713
SRM	0.8	1.262	1.11	1.20	1.31	1.32	1.37	5670
SRM	0.65	1.278	1.12	1.21	1.35	1.35	1.36	3970
SRM	0.5	1.296	1.12	1.22	1.36	1.35	1.43	2882

corresponding to very fast, fast, slow and very slow cooling schedules, respectively, and inherently to less or more error patterns processed by the algorithm, respectively. We ran the algorithm with each of these values for the cooling coefficient α and each mutation type, SRM and $LCPRM$. The algorithm is run 5 times for each input sequence and each combination of parameters with a different random seed.

Having 4 values for the cooling coefficient, 2 mutation types and 5 repeated tests we obtain 40 different runs for each input sequence and store the k_0-error linear complexity returned by each run. The accuracy of the results of the algorithm for each combination of parameters is averaged over the 5 runs.

Table 5.11 shows the average accuracy for each of the runs with different combinations of α and mutation types in increasing order of the accuracy. The accuracy of the result is the ratio between the approximate k_0-error linear complexity value obtained by the k-Error Simulated Annealing Algorithm, $L_{kSA,k}(s)$ and the exact value of the k_0-error linear complexity, calculated by the Efficient Exhaustive Search Algorithm, $L_k(s)$. We omit the accuracy for the 0-error linear complexity as this is trivially the linear complexity of the input sequence, and therefore the algorithm will calculate it in polynomial time. The accuracies for the different values of k, Acc_k are defined as $Acc_k = \frac{L_{kSA,k}(s)}{L_k(s)}$. The column accuracy in the table shows the average accuracy of the k-error linear complexity for all values of k, $1 \leq k \leq 5$, namely $Accuracy = \frac{\sum_{i=1}^{5} Acc_i}{5}$. Figure 5.17 shows these results in a graph.

The difference in accuracy between the eight configurations is not very significant. It is easy though to notice that the best configuration uses the slowest cooling schedule ($\alpha = 0.95$) and the mutation type which uses knowledge about the linear complexity of the solutions ($LCPRM$), giving an average result only 12.4% higher than the exact value.

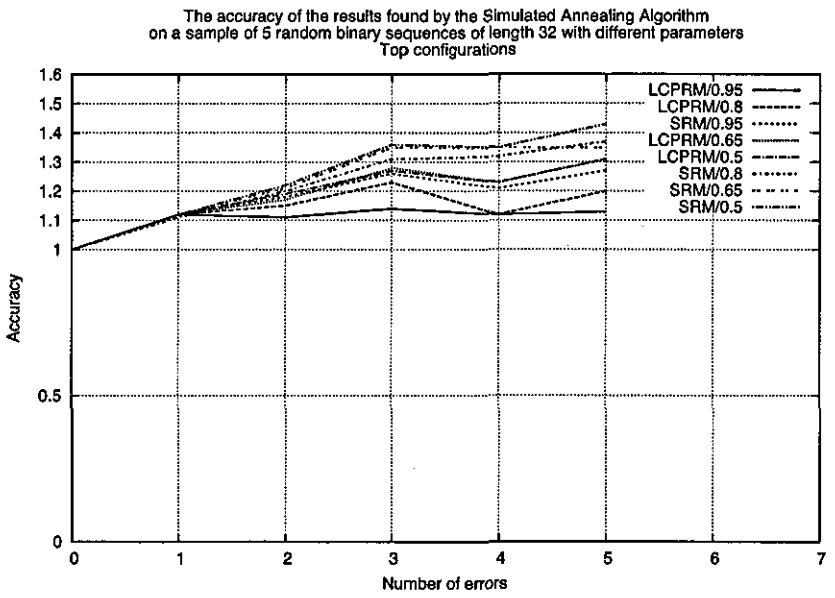

Figure 5.17: The accuracy of the results found by the Simulated Annealing Algorithm on a sample of 5 random binary sequences of length 32 with different parameters

The number of elements processed varies with the value of the cooling coefficient, α. On average, for the five sequences considered, the algorithm processes approximatively 19506, 5515, 3831, 2797 elements, corresponding to the different values of the cooling coefficient $\alpha = 0.95, 0.8, 0.65, 0.5$. The runs corresponding to a slower cooling schedule (large α) process more of the solutions in the search space. This gives them a better chance to find a more accurate approximation but makes them less efficient for large search space problems.

Figures 5.18 and 5.19 show the average accuracy of the results for the k-Error Simulated Annealing Algorithm when grouping by mutation type or by the different values of the cooling coefficient α, respectively. It is noticeable that the best results are given by the mutation scheme which uses the extra information about the linear complexity profile (figure 5.18). Also, we can conclude that the accuracy improves when the cooling coefficient increases, a fact which can be intuitively inferred since the number of error patterns processed by the algorithm increases too.

We are interested to check if the results of the algorithm are scaling with the size of the search space, therefore we tested it on sequences of higher length. The results of the algorithm on a sequence of length 64 are shown in figure 5.20. When grouping by mutation type or α respectively, similar conclusions to the ones above

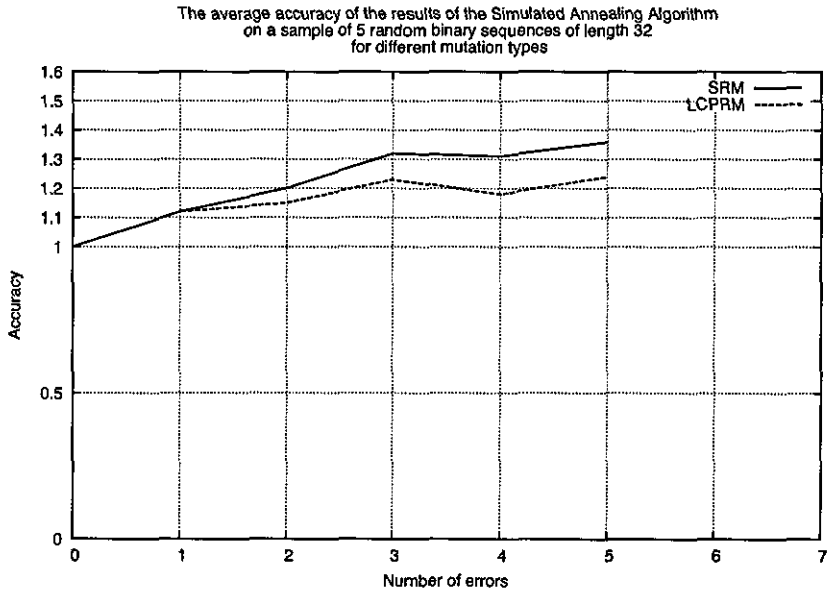

Figure 5.18: The average accuracy of the results of the Simulated Annealing Algorithm on a sample of 5 random binary sequences of length 32 for different mutation types

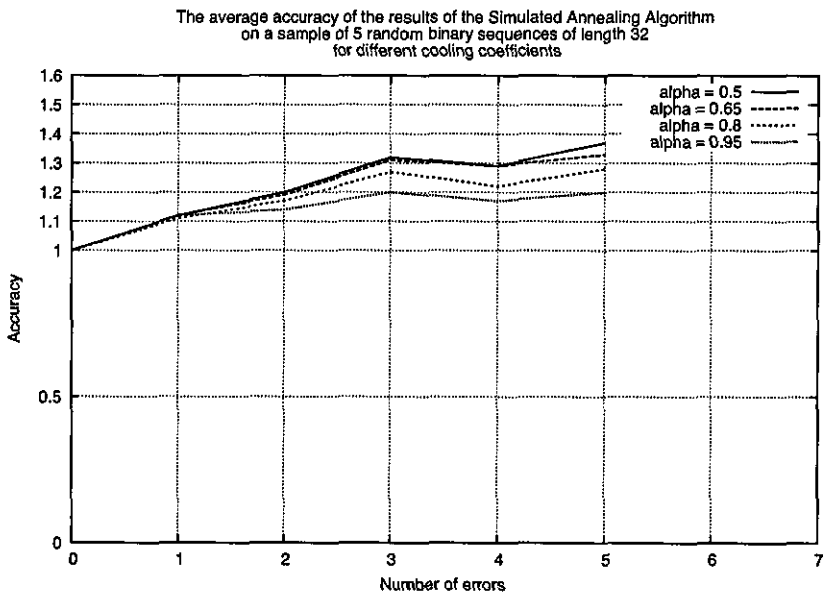

Figure 5.19: The average accuracy of the results of the Simulated Annealing Algorithm on a sample of 5 random binary sequences of length 32 for different cooling coefficients

Table 5.12: The accuracy for the k-Error Simulated Annealing Algorithm applied to a binary sequence of length 64 and $k_0 = 9$.

Mutation type	α	Accuracy	Acc_1	Acc_2	Acc_3	Acc_4	Acc_5	Acc_6	Acc_7	Acc_8	Acc_9	Error patterns
LCPRM	0.95	1.380	1.10	1.19	1.28	1.45	1.45	1.46	1.40	1.38	1.71	92039
LCPRM	0.8	1.405	1.10	1.19	1.28	1.45	1.49	1.56	1.49	1.40	1.69	25566
SRM	0.95	1.407	1.10	1.19	1.28	1.45	1.45	1.52	1.54	1.43	1.71	89239
LCPRM	0.5	1.430	1.10	1.19	1.28	1.45	1.50	1.62	1.56	1.41	1.76	13043
LCPRM	0.65	1.430	1.10	1.19	1.28	1.45	1.50	1.61	1.50	1.43	1.81	16720
SRM	0.8	1.445	1.10	1.19	1.28	1.45	1.50	1.59	1.59	1.51	1.80	25249
SRM	0.5	1.454	1.10	1.19	1.28	1.45	1.48	1.58	1.62	1.58	1.81	13077
SRM	0.65	1.463	1.10	1.19	1.28	1.45	1.52	1.62	1.62	1.56	1.83	16736

can be drawn from the results, see figures 5.21 and 5.22.

5.2.4 Conclusion

We propose a new algorithm for approximating the k-error linear complexity of sequences over finite fields. The algorithm uses simulated annealing techniques and we assess it experimentally by comparing its result to the exact value for accuracy and with the previous proposed heuristic algorithms for both accuracy and efficiency. The algorithm proves to be robust and like the k-Error Genetic Algorithm allows fine tuning of the parameters depending on the size of the input sequence and the number of errors k, so that the result is close to the exact value while still running in reasonable time. The algorithm outputs an approximate value of the k-error linear complexity which is on average only 12.4% higher than the exact value for sequences of length 32 when $1 \leq k_0 \leq 5$ and 38% higher for sequences of length 64 when $1 \leq k_0 \leq 9$.

5.3 Conclusions

This chapter includes two types of algorithms which implement evolutionary computation techniques, genetic algorithm and simulated annealing.

The results of these algorithms have a good accuracy however the number of the error patterns processed is much higher than the Modified Berlekamp-Massey Algorithm presented in chapter 4.

While k-Error Simulated Annealing Algorithm gives an average accuracy of 1.124 for sequences of length 32, the k-Error Genetic Algorithm presented in section 5.1 has the best average accuracy of 1.072 on the same set of sequences when $1 \leq k_0 \leq 5$. It is interesting to look at the heuristics presented so far and compare the accuracies and the average number of error patterns processed by each to see which one is the most powerful. Table 5.13 summarizes the heuristic algorithms presented so far and the difference between accuracy and efficiency

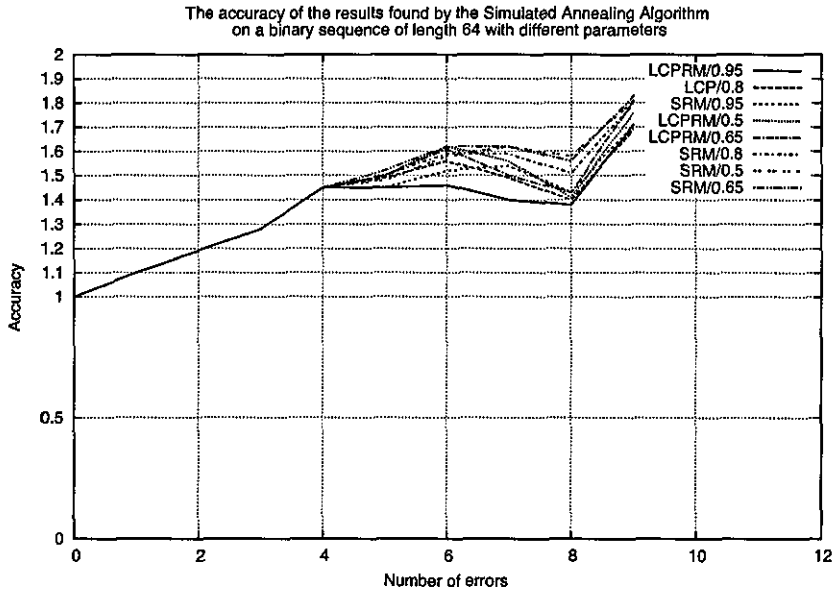

Figure 5.20: The accuracy of the results found by the Simulated Annealing Algorithm on a binary sequence of length 64 with different parameters

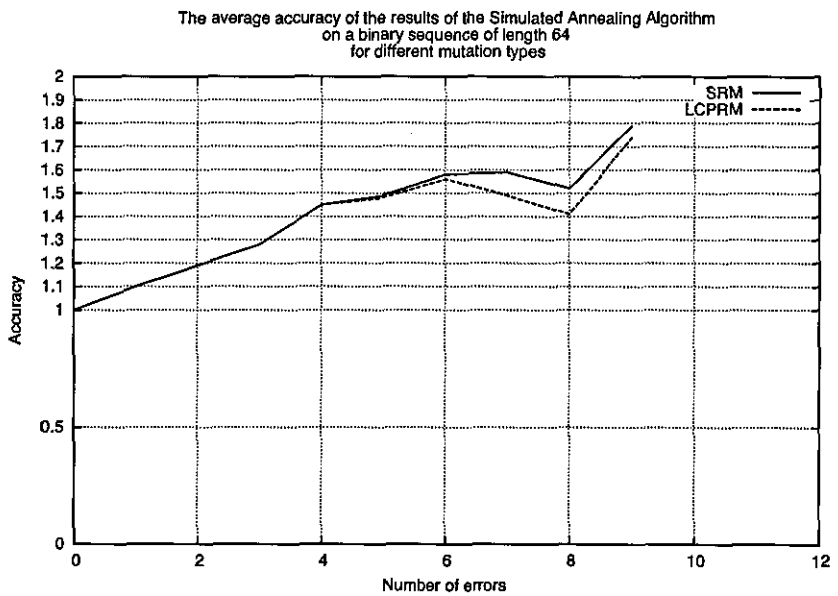

Figure 5.21: The average accuracy of the results of the Simulated Annealing Algorithm on a binary sequence of length 64 for different mutation types

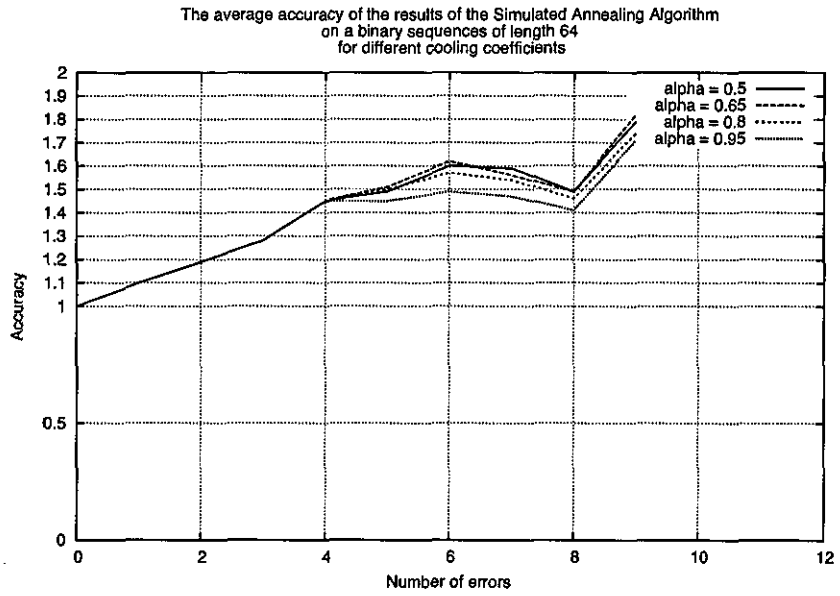

Figure 5.22: The average accuracy of the results of the Simulated Annealing Algorithm on a binary sequence of length 32 for different cooling coefficients

Table 5.13: The accuracy of the different heuristic algorithms on binary sequences of length 32 and when $k_0 = 5$.

Algorithm	Accuracy	Acc_1	Acc_2	Acc_3	Acc_4	Acc_5	Error patterns
kSA(LCPRM, 0.95)	1.124	1.12	1.11	1.14	1.12	1.13	19036
MBM	1.26	1.11	1.19	1.27	1.35	1.38	210
GA(1100, 10, ELSEL, LCPTCP, LCPRM)	1.072	1.00	1.00	1.01	1.08	1.27	26661

(number of error patterns visited) for the same set of sequences of length 32. We believe that the best compromise accuracy/efficiency is obtained by the Modified Berlekamp-Massey Algorithm (chapter 4) which has a consistently good accuracy for all values of k and for a very low number of error patterns visited. For the same set of sequences the Efficient Exhaustive Search Algorithm (see chapter 3) processes on average 35327 error patterns and the Modified Berlekamp-Massey Algorithm (see chapter 4) processes on average only 210 error patterns. The best configuration of the k-Error Genetic Algorithm processes on average 26661 error patterns and produces an accuracy of 1.072. The Modified Berlekamp-Massey Algorithm produces an average accuracy of 1.26 for sequences of length 32. Table 5.13 summarises these results. See table 5.14 for a similar comparison of the three heuristic algorithms on the sequences of length 64.

One advantage of the evolutionary algorithms is that they can easily be par-

Table 5.14: The accuracy of the different heuristic algorithms on binary sequences of length 64 and when $k_0 = 9$.

Algorithm	Accuracy	Acc_1	Acc_2	Acc_3	Acc_4	Acc_5	Acc_6	Acc_7	Acc_8	Acc_9	Error patterns
kSA(LCPRM, 0.95)	1.380	1.10	1.19	1.28	1.45	1.45	1.46	1.40	1.38	1.71	92039
MBM	1.14	1.01	1.04	1.07	1.10	1.14	1.18	1.21	1.24	1.27	36592
GA(4000, 10, ELSEL, LCPTCP, SRM)	1.205	1.00	1.00	1.02	1.14	1.16	1.27	1.30	1.30	1.66	105763

allelised, in such a scenario the accuracy could be improved by using more than one processors with no additional time costs. Also they allow for fine tuning of the parameters in order to cover more or less of the search space and therefore get more or less accurate approximation of the exact result. The running time of such algorithms is also easy to tune so they can constitute an easy, quick way of identifying sequences of low k-error linear complexity with a certain degree of confidence. On the other hand the compromise accuracy/efficiency is worse than the one for the Modified Berlekamp-Massey Algorithm.

One disadvantage of the algorithms defined in this section is that the size of the chromosomes increases at a one to one rate with the size of the input sequence. This makes the algorithm difficult to scale and the memory used increases greatly with the size of the input sequence. In trying to address this problem, we think it would be interesting as future work to investigate the possibility of devising an evolutionary algorithm for which the individuals are recurrences or characteristic polynomials. The fitness of each individual would reflect how well they generate the input sequence. For example, the fitness of each recurrence or polynomial could be the Hamming weight of the difference between the input sequence and the sequence generated using that individual. A challenge in this design is how to create efficient recombination techniques which would combine two recurrences or polynomials such that the child is fitter. We leave this as future work.

These algorithms can be used as an alternative to the existing algorithms for an approximate value of the k-error linear complexity of sequences.

Chapter 6

Discrete Fourier Transform

In this chapter we consider the k-error linear complexity of infinite periodic sequences of period N over a field K.

The existing exact algorithms to compute the k-error linear complexity apply to periodic sequences over finite fields where period is a power of the characteristic of the field (see Stamp and Martin [79], Lauder and Paterson [38] for $p = 2$ and Kaida, Uehara and Imamura [31], Kaida [29] for arbitrary p). These algorithms assume a full period of the sequence known a priori, i.e. the whole sequence is known. Such algorithms are useful in the design phase of the cypher, in order to identify vulnerable sequences with low k-error linear complexity. We attempt in this chapter to find algorithms for computing the k-error linear complexity of periodic sequences where the period is not restricted to being a power of the characteristic of the field.

We introduce a generalisation of the notion of k-error linear complexity, which we call the extension field k-error linear complexity, defined as being the k-error linear complexity of s when working in the smallest extension field of K which contains an N-th root of unity, assuming N is not divisible by the characteristic of K (published in Alecu and Sălăgean [3]).

Blahut's theorem shows that the linear complexity of a periodic sequence is equal to the weight of the Discrete Fourier Transform of that sequence. We show that using Blahut's theorem, the optimisation problem of finding the extension field k-error linear complexity can be firstly transformed into an optimisation problem in the DFT domain. Namely, in the transform domain we have to find an error pattern E which has linear complexity at most k and minimises the weight of $DFT(s) + E$.

While we do not know of an efficient exact algorithm for this problem, we propose an approximation algorithm which searches among error patterns in the DFT domain which have period up to k (and therefore complexity up to k). The approximation algorithm has quadratic complexity, $\mathcal{O}(N^2)$ operations in the

extension field.

The algorithms presented in this chapter were implemented using GAP (Groups, Algebra and Programming) [18] and the results on a series of sequences are discussed in section 6.4.

The results included in this chapter extend the work presented in Alecu and Sălăgean [3].

6.1 Background

The discrete Fourier transform (DFT) is a transform for Fourier analysis of finite-domain discrete-time signals (Rueppel [70, 77]). We recall the definition of the Discrete Fourier Transform. Note that the basic algebraic concepts necessary to the understanding of this section are included in Appendix A.

We identify an infinite periodic sequence s by one period $(s_0, s_1, \ldots, s_{N-1})$, where N is the period of the sequence.

Definition 6.1. *Let F be a field containing a primitive N-th root of the unity, denoted α, and let s be a sequence of period N over F. The* discrete Fourier transform *(DFT) of s is the periodic sequence $S = (S_0, S_1, \ldots, S_{N-1})$ over F, where*

$$S_i = \sum_{j=0}^{N-1} s_j \alpha^{ij}, \text{ for all } i = 0, 1, \ldots, N-1.$$

Reciprocally, the inverse discrete Fourier transform *of S is the sequence s defined by*

$$s_j = N^{-1} \sum_{i=0}^{N-1} S_i \alpha^{-ij}, \text{ for all } j = 0, 1, \ldots, N-1.$$

Note that in order to compute the DFT of a periodic sequence s of period N over a finite field of characteristic p, usually the following constraint is imposed, $gcd(N, p) = 1$. However there are generalisations of the Discrete Fourier Transform (see Massey and Serconek [45]) which allow applying the Discrete Fourier Transform technique to sequences of arbitrary period. This case is not considered in our current work, and can be a topic of future research.

It is well known that the Discrete Fourier Transform is linear:

Property 6.2. *Let N be a positive integer and F be an arbitrary field which contains a primitive N-th root of unity. The Discrete Fourier Transform and the inverse Discrete Fourier Transform are linear operators on the vector space F^N.*

We also present the shifting property (Property 6.3) of the Discrete Fourier Transform.

Property 6.3. *Let N be a positive integer and F be an arbitrary field which contains a primitive N-th root of unity, α. Let $s' = (s'_0, s'_1, \ldots, s'_{N-1})$ be the periodic infinite sequence over F obtained by cyclically shifting all periods of $s = (s_0, s_1, \ldots, s_{N-1})$ to the left by h positions, i.e.*

$$s'_i = s_{i+h}, \text{ for all } 0 \leq i < N$$

where indices are taken modulo N. Then if $S' = (S'_0, S'_1, \ldots, S'_{N-1})$ is the Discrete Fourier Transform of s' and $S = (S_0, S_1, \ldots, S_{N-1})$ is the Discrete Fourier Transform of s, the following relation stands

$$S'_i = \alpha^{-hi} S_i, \text{ for all } 0 \leq i < N.$$

Remark 6.4. *With the notations in property 6.3, let $s'' = (s''_0, s''_1, \ldots, s''_{N-1})$ be the periodic infinite sequence over F obtained by cyclically shifting all periods of $s = (s_0, s_1, \ldots, s_{N-1})$ to the right with h positions, i.e.*

$$s''_i = s_{i-h}, \text{ for all } 0 \leq i < N$$

where indices are taken modulo N. Then if $S'' = (S''_0, S''_1, \ldots, S''_{N-1})$ is the Discrete Fourier Transform of s'' and $S = (S_0, S_1, \ldots, S_{N-1})$ is the Discrete Fourier Transform of s, the following relation stands

$$S''_i = \alpha^{hi} S_i, \text{ for all } 0 \leq i < N.$$

Note that this follows immediately from property 6.3 concerning the cyclical left shift, since for any periodic sequence of period N, a cyclical right shift with h positions is equivalent to a cyclical left shift of $N - h$ positions. Therefore, all properties corresponding to cyclical right shifts of a sequence can be written in terms of cyclical left shifts and reciprocally.

For computing the DFT of a sequence of period N, a straightforward approach takes $\mathcal{O}(N^2)$ operations in the field F. Faster computation approaches, usually called fast Fourier transform (FFT), are available and they can give a complexity of $\mathcal{O}(N \log N)$ operations in the field F. See Pollard [62] or Preparata and Sarwate [63] for the computational complexity of computing the Discrete Fourier Transform of sequences over finite fields or for more general considerations related to the fast Fourier transform see Cormen et al [10].

There is a close connection between the linear complexity and the Discrete Fourier Transform of a sequence. In 1979, Blahut [7] used the link between the linear complexity of a periodic sequence and its discrete Fourier transform, setting

this way the fundaments of the spectral theory of error correcting codes. The theorem was established in an explicit form by Massey [41] and given the name of Blahut's theorem in Massey and Schaub [44]. The theorem gives a simple way of computing the linear complexity of a periodic sequence when the whole period is known.

Theorem 6.5 (Blahut's theorem (Simmons [77])). *The linear complexity of a periodic sequence* $s = (s_0, s_1, \ldots, s_{N-1})$ *of period* N *equals the Hamming weight of* $DFT(s)$. *Reciprocally, the linear complexity of the periodic sequence* $S = (S_0, S_1, \ldots, S_{N-1})$ *equals the Hamming weight of* $DFT^{-1}(S)$.

It has been proven that Blahut's theorem also holds over commutative rings (Massey [43]) but we will only use it for sequences over finite fields.

Paterson shows that the well established method of root counting for calculating linear complexity of sequences is equivalent to using the Discrete Fourier Transform and applies this technique to the theory of filtering m-sequences (Paterson [60]).

Using the generalised Discrete Fourier Transform defined by Massey and Serconek [45], Meidl and Niederreiter [49, 50] build a statistical theory for the linear complexity and k-error linear complexity, determining the number of periodic sequences with a certain given period and linear complexity, as well as the expected value of the linear complexity of periodic sequences. For some values of k, a formula for the number of periodic sequences with given period and given k-error linear complexity is obtained. Formulas are given for the expected value of the 1-error linear complexity as well as lower and upper bounds for the expected value of the k-error linear complexity when $k \geq 2$ (Meidl [48]). The statistical theory of linear complexity and k-error linear complexity is extended by the authors in Niederreiter [59], Meidl and Niederreiter [51], Meidl [47, 48].

Blackburn [5] generalises both Games-Chan Algorithm (Games and Chan [17]) (the case when $gcd(p, N) \neq 1$) and the Discrete Fourier Transform (the case when $gcd(p, N) = 1$) for computing the k-error linear complexity of periodic sequences over finite fields.

For more details about the Discrete Fourier Transform applied to linear recurrent sequences see also Ding, Xiao, Shan [12, Section 5.7]. A concise reference for the use of the Discrete Fourier Transform in coding and cryptography is included in Massey [43] along with the generalisation of Blahut's theorem from finite fields to commutative rings.

6.2 k-error linear complexity computation using DFT

The generalisation of the definition of k-error linear complexity and problem transformation proofs have been stated in our paper [3] and will be described in the following two sections.

6.2.1 Extension field k-error linear complexity

Note that the k-error linear complexity of a periodic sequence s over a field K could decrease if we consider the same sequence to be over some extension field F of K (and therefore allow the error sequence to be over F). A natural choice of the extension field F would be the smallest extension field of K which contains an N-th root of unity, where N is the period of s. This extension field F is the splitting field of the polynomial $x^N - 1$. Note that since s has period N, the minimal characteristic polynomial of s is a factor of $x^N - 1$.

Definition 6.6 (Extension field k-error linear complexity). *Let s be a sequence of period N over a field K, such that N is not divisible by the characteristic of K. Let F be the smallest extension field of K which contains an N-th root of unity. Let $k \leq N$. The* extension field k-error linear complexity *of s, denoted $EL_{k,N}(s)$ is defined as the k-error linear complexity of s when s is viewed as a sequence over the extension field F:*

$$EL_{k,N}(s) = \min\{L(s+e) | e \in F^N, w_H(e) \leq k\}$$

Note that since the extension field F includes the original field K the relation $L_{k,N}(s) \geq EL_{k,N}(s)$ is immediate.

Also, note that $EL_{k,N}(s)$ shares the properties of $L_k(s)$, namely it is decreasing when k increases (see property 2.14) and also $EL_{w_H(s),N}(s) = 0$ (property 2.15).

Let us examine the cryptanalytic significance of this notion. Assume a cryptanalyst knows a few terms of the sequence s over K and is able to either find a sequence s' over K which coincides with s in all but up to k positions (the scenario considered in k-error linear complexity), or is able to find a sequence s'' over an extension field F which coincides with s in all but up to k positions (the scenario considered in the extension field k-error linear complexity). Finding s'' will be at least as useful to the cryptanalyst as finding s', and likely more useful, as the positions of s'' which are from $F \setminus K$ can be immediately identified as not being from the original sequence s, whereas given s' it may not be known which of the positions do not coincide with s.

6.2.2 Problem transformation

The extension field k-error linear complexity problem can be expressed as an optimisation problem: given integers N and k with $k \leq N$ and a sequence s of period N over a field K, find e, a sequence of period N over the extension field F such that $w_H(e) \leq k$ and $L(s+e)$ is minimal (F is the smallest extension field of K containing an N-th root of unity). This problem can be transformed into an optimisation problem in the DFT domain. Since the DFT and the inverse DFT are linear transforms (Property 6.2), in order to find error patterns e of weight at most k which minimise the linear complexity of $s+e$, we can search for sequences E of linear complexity at most k and which minimise the weight of $DFT(s)+E$. The resulting sequence E is then transformed back using the inverse DFT, obtaining an optimal error pattern e, $e = DFT^{-1}(E)$. Note that this error pattern can be in the extension field used for computing the DFT, F, rather than in the original field over which s was defined, K.

Theorem 6.7. *Let s, e be sequences over a field K such that s has period N, N is not divisible by the characteristic of K. Let F be the smallest extension field of K which contains an N-th root of unity. Let $S = DFT(s)$, $E = DFT(e)$ and let $k \leq N$. The following equivalence stands:*

$$\begin{aligned}
&e \text{ is such that } w_H(e) \leq k \text{ and } L(s+e) \text{ is minimal} \\
&\text{if and only if} \\
&E \text{ is such that } L(E) \leq k \text{ and } w_H(S+E) \text{ is minimal}
\end{aligned}$$

Proof. Theorem 6.5 implies that $w_H(e) = L(E)$ and therefore $w_H(e) \leq k$ iff $L(E) \leq k$. The linearity of the DFT (property 6.2) implies that $S+E = DFT(s)+DFT(e) = DFT(s+e)$ and using Theorem 6.5 again, we obtain $w_H(S+E) = w_H(DFT(s+e)) = L(s+e)$. □

Corollary 6.8. *With the notations in theorem 6.7, the extension field k-error linear complexity of a periodic sequence s can be expressed as*

$$EL_{k,N}(s) = \min\{w_H(DFT(s)+E) | E \in F^N, L(E) \leq k\}$$

An algorithm for computing the k-error linear complexity of a sequence follows immediately based on Theorem 6.7 and Corollary 6.8. The algorithm is formulated for finite fields (with the usual notation $GF(p^m)$ for the Galois field with p^m elements) but it can easily be formulated for any field in which algorithms exist for the required operations.

ALGORITHM *kDFT*
INPUT: p prime, $m \geq 1$, N a positive integer not divisible by p, s a sequence of period N over $GF(p^m)$, $k \leq N$.
OUTPUT: $EL_{k,N}(s)$ and a sequence e of period N over $GF(p^r)$ (where $GF(p^r)$ is the smallest extension of $GF(p^m)$ containing an N-th root of unity) such that $w_H(e) \leq k$ and $L(s+e)$ is minimal.
STEP 1. Determine r, α such that $GF(p^r)$ is the smallest extension of $GF(p^m)$ which contains a primitive N-th root of unity, α.
STEP 2. Calculate the sequence $S = DFT(s)$ of period N over $GF(p^r)$.
STEP 3. Find E a sequence over $GF(p^r)$ of period N and linear complexity $L(E) \leq k$ such that $w_H(S+E)$ is minimal.
STEP 4. Return $EL_{k,N}(s) = w_H(S+E)$ and $e = DFT^{-1}(E)$ as the error pattern.

The Algorithm *kDFT* is of theoretical interest only, because we do not know of an efficient algorithm for STEP 3 other than exhaustive search (for finite fields). In the next section we therefore suggest an approximation algorithm.

6.2.3 Approximation algorithm for the extension field k-error linear complexity

An approximation algorithm has been devised for the extension field k-error linear complexity of a sequence by finding an approximate solution to the optimisation problem in STEP 3 of algorithm *kDFT* described in the previous section. Namely, we aim to find a sequence E of linear complexity at most k such that $w_H(S+E) \leq w_H(S)$, but $w_H(S+E)$ is not necessarily minimal. To this end, the search is limited to sequences E which have minimal period at most k besides having period N. Since $k \leq N$, the period must therefore be a divisor of N. Obviously any sequence of period at most k will also have linear complexity at most k. In order to decrease $w_H(S+E)$ as much as possible we choose the elements of E so that they cancel out as many elements of S as possible.

Theorem 6.9. *Let S be a sequence of period N over a field F. Suppose N is not prime and d is a proper divisor of N. For each $i = 0, 1, \ldots, d-1$ denote by β_i the element which occurs most frequently among $S_i, S_{d+i}, \ldots, S_{(\frac{N}{d}-1)d+i}$. Let E be the sequence of period d defined as $E = (-\beta_0, -\beta_1, \ldots, -\beta_{d-1})$. Then E achieves the minimal value of $w_H(S+E)$ (where E is viewed as a sequence of period N for the purpose of computing this weight) among all sequences E over F of period d.*

Proof. Let $G = (G_0, G_1, \ldots, G_{d-1})$ be an arbitrary sequence over F of period d.

We split S in N/d parts of size d as follows:

$$\begin{array}{llll} S_0 & S_1 & \dots & S_{d-1} \\ S_d & S_{d+1} & \dots & S_{2d-1} \\ \dots & \dots & \dots & \dots \\ S_{(\frac{N}{d}-1)d} & S_{(\frac{N}{d}-1)d+1} & \dots & S_{\frac{N}{d}d-1} \end{array}$$

and we consider each column i consisting of $S_i, S_{d+i}, \dots, S_{(\frac{N}{d}-1)d+i}$. Let us denote by ζ_i the number of zero values on column i and by ν_i the number of elements equal to G_i on column i. The weight $w_H(S+G) = w_H(S) - \sum_{i=0}^{d-1}(\nu_i - \zeta_i)$ is minimised by maximising the values of all the ν_i, i.e. by choosing G_i such that a maximum number of elements of S in column i are equal to G_i. □

In theorem 6.9, in order to get uniqueness in the choice of the sequence E which lowers the Hamming weight of S, if there is more than one element among $S_i, S_{d+i}, \dots, S_{(\frac{N}{d}-1)d+i}$ which have the same maximum occurrence then choose zero if any of those elements are zero, or the element of smallest index otherwise (zero is the identity element with respect to addition). Note that this assumption does not restrict the generality of the theorem or of the corresponding algorithm.

The following routine for finding a candidate sequence E (corresponding to STEP 3 in $kDFT$ Algorithm from previous section) can be based on theorem 6.9. We call the routine $GetError(S, N, d)$. The input of the procedure includes a periodic sequence S, an integer not prime N representing the period of the sequence S and a proper divisor of N, $d < N$. The sequence E returned by the procedure achieves the minimal value of $w_H(S+E)$ among all sequences over F of period d (where E is viewed as a sequence of period N for the purpose of computing this weight). Note that in the worst case, the procedure $GetError$ returns a sequence E such that $w_H(S+E) = w_H(S)$, in which case $E = \underbrace{(0, 0, \dots, 0)}_{N \text{ times}}$.

ALGORITHM $GetError(S, N, d)$
INPUT: p prime, $m \geq 1$, N a positive integer, not prime and not divisible by p, S a sequence of period N over $GF(p^m)$, $d < N$ a proper divisor of N.
OUTPUT. A sequence E of period d such that E achieves the minimal value of $w_H(S+E)$.
STEP 1. For each $i = 0, 1, \dots, d-1$ determine β_i as the most frequent value that appears among $S_i, S_{d+i}, \dots S_{(\frac{N}{d}-1)d+i}$, and set $E_{i+jd} = -\beta_i$ for $j = 0, 1, \dots, \frac{N}{d} - 1$. If there is more than one value with the same maximum occurrence then choose zero value (i.e. identity element with respect to addition) if it is one of them or choose the one with smallest index otherwise.

STEP 2. Return E.

Based on theorem 6.9, we propose the following approximation algorithm for the extension field k-error linear complexity of a sequence s over a finite field.

ALGORITHM $kDFT$-Approximation
INPUT: p prime, $m \geq 1$, N a positive integer, not prime and not divisible by p, s a sequence of period N over $GF(p^m)$, $k \leq N$.
OUTPUT: a sequence e of period N over $GF(p^r)$ (where $GF(p^r)$ is the smallest extension of $GF(p^m)$ containing a primitive N-th root of unity) such that $w_H(e) \leq k$ and $L(s+e)$ is minimal among all sequences e for which $DFT(e)$ has period smaller than or equal to k; the second output is $L(s+e)$.
STEP 1. Determine r, α such that $GF(p^r)$ is the smallest extension of $GF(p^m)$ which contains a primitive N-th root of unity, α.
STEP 2. Calculate the sequence $S = DFT(s)$ over $GF(p^r)$. Set $L_{best} = w_H(S)$ and set E_{best} to the all-zero sequence.
STEP 3. For all $d \leq k$ with d a proper divisor of N, execute steps 4-5.
STEP 4. Set $E = GetError(S, N, d)$.
STEP 5. If $w_H(S+E) < L_{best}$ then set $E_{best} = E$ and $L_{best} = w_H(S+E)$.
STEP 6. Return $DFT^{-1}(E_{best})$ and L_{best}.

Example 6.10. *Let $p = 5$, $N = 15$. Let $s = (0,0,1,0,1,1,0,0,1,1,1,0,0,0,1)$ be a sequence over $GF(2)$ of period 15. Let α be a primitive element of the Galois field $GF(2^4)$ defined by the equation $\alpha^4 + \alpha + 1 = 0$.*

The DFT transform of s in $GF(2^4)$ is:

$$S = (1, 0, 0, \alpha^{14}, 0, 1, \alpha^{13}, \alpha^5, 0, \alpha^7, 1, \alpha^{10}, \alpha^{11}, \alpha^5, \alpha^{10}).$$

The linear complexity of s, $L(s) = w_H(S) = 11$. The two proper divisors of 15 are 3 and 5 so two ways of splitting of the sequence are considered, as shown in table 6.1.

For $d = 3$ (table 6.1(A)), choosing one of the most frequent entries in each column gives us $\beta_0 = 1$, $\beta_1 = 0$ and $\beta_2 = 0$ (other choices are possible but these follow our uniqueness requirement). We can therefore put

$$E = (1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0)$$

and

$$e = DFT^{-1}(E) = (1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0).$$

Table 6.1: The splitting of the sequence S

(A) IN 5 PARTS OF LENGTH 3

1	0	0
α^{14}	0	1
α^{13}	α^5	0
α^7	1	α^{10}
α^{11}	α^5	α^{10}

(B) IN 3 PARTS OF LENGTH 5.

1	0	0	α^{14}	0
1	α^{13}	α^5	0	α^7
1	α^{10}	α^{11}	α^5	α^{10}

This error pattern e of weight 3 decreases the linear complexity of s to 10.

Table 6.1(B) shows that when splitting s into 3 parts of 5 elements we can choose $E = (1,0,0,0,0,1,0,0,0,0,1,0,0,0,0)$.

We compute $e = DFT^{-1}(E) = (1,0,0,1,0,0,1,0,0,1,0,0,1,0,0)$. *The sequence E applied onto S reduces its weight from 11 to 8, therefore* $L(s+e) = 8$. *Since* $w_H(e) = 5$, *it follows that the extension field 5-error linear complexity* $EL_{5,15}(s) \leq 8$.

Theorem 6.11. *The algorithm kDFT-Approximation has a computational complexity of $\mathcal{O}(N^2)$ operations in $GF(p^r)$. This can be improved to $\mathcal{O}(N^{1.5}\log N)$ when a Fast Fourier Transform approach is used.*

Proof. Steps 2 and 6, corresponding to the computation of the DFT and inverse DFT of the sequences s and E, respectively, need $\mathcal{O}(N^2)$ operations (or $\mathcal{O}(N\log N)$ if a fast Fourier transform approach is applied).

Steps 4-5 are executed for each of the proper divisors of N. There are several upper bounds for the number of divisors. We will use the rather coarse upper bound of $2\sqrt{N}$, see Ramanujan [65].

For each proper divisor d, determining the most frequent element on each of the d columns takes at most $\mathcal{O}(\frac{N}{d}\log\frac{N}{d})$ operations (this can be achieved by expressing all non-zero elements as powers of a primitive element of $GF(p^r)$ and sorting each column according to the exponents), so $GetError(S,N,d)$ procedure from STEP 4 takes a total of $\mathcal{O}(N\log\frac{N}{d})$ operations. This is executed for each of the proper divisors of N, i.e. at most $2\sqrt{N}$ times, giving a total computational complexity of the **for** loop in *GetError* of $\mathcal{O}(N^{1.5}\log N)$.

Therefore, the computational complexity of the algorithm kDFT-Approximation is $\mathcal{O}(N^2 + N^{1.5}\log N) = \mathcal{O}(N^2)$. If a Fast Fourier Transform is used, this can be improved to $\mathcal{O}(N\log N + N^{1.5}\log N) = \mathcal{O}(N^{1.5}\log N)$. □

It would be interesting to obtain bounds on the quality of the approximation, i.e. on the ratio between the maximum decrease in complexity and the decrease in complexity achieved by the *kDFT*-Approximation algorithm,

$$(L(s) - EL_{k,N}(s))/(L(s) - L(s+e)),$$

where e is the error sequence produced by the algorithm $kDFT$-Approximation. It is likely though that such a ratio does not always make sense, as there would probably be sequences for which the approximation algorithm will not find any non-zero error sequence e with $L(s+e) < L(s)$ although $EL_{k,N}(s) < L(s)$.

6.3 An improved approximation algorithm

An improved approximation algorithm which relies on using the shifting property of the Discrete Fourier Transform can be devised. Namely, firstly the input sequence s is cyclically shifted to the right by a number of positions $h > 0$, the Discrete Fourier Transform is applied to the shifted version s' and then the same procedure as in $kDFT$-Approximation Algorithm, *GetError*, is carried out in the transform domain in order to find a sequence E' of period at most k and which minimises the value of $w_H(S'+E')$ where $S' = DFT(s')$. Finally, by transforming the sequence E' back to the initial domain and applying a cyclical shift to the left by h positions, we aim to find an additional error sequence candidate which would lower the linear complexity of the input sequence s.

The following theorem shows that if a sequence of period N has a smaller period d, $1 < d < N$ (note that d is a proper divisor of N) then the Discrete Fourier Transform of that sequence when considered of period N will have a certain form, namely it will be zero except for the terms on the positions $0, \frac{N}{d}, \ldots, \frac{(d-1)N}{d}$.

Theorem 6.12. *Let e be a sequence of period N and of smaller period $1 < d < N$ over a field F.*

Let $e = (\underbrace{\underbrace{e_0, e_1, \ldots, e_{d-1}}_{d \text{ times}}, \underbrace{e_0, e_1, \ldots, e_{d-1}}_{d \text{ times}}, \ldots, \underbrace{e_0, e_1, \ldots, e_{d-1}}_{d \text{ times}}}_{\frac{N}{d} \text{ times}})$.

If the periodic sequence $E = DFT(e)$ of period N denotes the Discrete Fourier Transform of e then E is periodic with smallest period at least $\frac{N}{d}$ and it has the following form: $E = (E_0, \underbrace{0, \ldots, 0}_{\frac{N}{d}-1 \text{ times}}, E_{\frac{N}{d}}, \underbrace{0, \ldots, 0}_{\frac{N}{d}-1 \text{ times}}, \ldots E_{\frac{(d-1)N}{d}}, \underbrace{0, \ldots, 0}_{\frac{N}{d}-1 \text{ times}})$.
Moreover, for all $0 \leq i < N$, the terms of the transform sequence are

$$E_i = \begin{cases} 0, & \text{if } i \neq 0, \frac{N}{d}, \ldots, \frac{(d-1)N}{d}, \\ \frac{N}{d}(e_0 + e_1\alpha^i + \ldots + e_{d-1}\alpha^{(d-1)i}), & \text{otherwise.} \end{cases} \tag{6.1}$$

Proof. Note that since E has period N and d, and since $1 < d < N$, then d is a

proper divisor of N. For each $i = 0, 1, \ldots, N-1$

$$\begin{aligned} E_i &= \sum_{j=0}^{N-1} e_j \alpha^{ji} = \\ &= e_0\alpha^{0i} + e_1\alpha^{1i} + \ldots + e_{d-1}\alpha^{(d-1)i} + \\ &\quad e_0\alpha^{di} + e_1\alpha^{(d+1)i} + \ldots + e_{d-1}\alpha^{(2d-1)i} + \\ &\quad \ldots + \\ &\quad e_0\alpha^{(\frac{N}{d}-1)di} + e_1\alpha^{((\frac{N}{d}-1)d+1)i} + \ldots + e_{d-1}\alpha^{(\frac{N}{d}d-1)i} = \\ &= e_0(1 + \alpha^{di} + \ldots + \alpha^{(\frac{N}{d}-1)di}) + \\ &\quad e_1\alpha^i(1 + \alpha^{di} + \ldots + \alpha^{(\frac{N}{d}-1)di}) + \\ &\quad \ldots + \\ &\quad e_{d-1}\alpha^{(d-1)i}(1 + \alpha^{di} + \ldots + \alpha^{(\frac{N}{d}-1)di}) \end{aligned}$$

When i is such that $\alpha^{di} \neq 1$ the sum $1 + \alpha^{di} + \ldots + \alpha^{(\frac{N}{d}-1)di}$ represents a geometric series of $\frac{N}{d}$ terms with initial value 1 and common ratio α^{di}. It follows that the sum equals $\frac{(\alpha^{di})^{\frac{N}{d}}-1}{\alpha^{di}-1} = \frac{\alpha^{Ni}-1}{\alpha^{di}-1} = 0$ for all $i \neq 0, \frac{N}{d}, \frac{2N}{d}, \ldots, \frac{(d-1)N}{d}$.

When i is such that $\alpha^{di} = 1$, i.e. $i = 0, \frac{N}{d}, \frac{2N}{d}, \ldots, \frac{(d-1)N}{d}$, the sum $1 + \alpha^{di} + \ldots + \alpha^{(\frac{N}{d}-1)di} = \underbrace{1 + 1 + \ldots + 1}_{\frac{N}{d} \text{ times}} = \frac{N}{d}$. Relation (6.1) follows immediately. □

A similar result can be proved for the inverse Discrete Fourier Transform. We just state it in theorem 6.13.

Theorem 6.13. *Let E be a sequence of period N and smaller period $1 < d < N$ over a field F.*

Let $E = \underbrace{(\underbrace{E_0, E_1, \ldots, E_{d-1}}_{d \text{ times}}, \underbrace{E_0, E_1, \ldots, E_{d-1}}_{d \text{ times}}, \ldots, \underbrace{E_0, E_1, \ldots, E_{d-1}}_{d \text{ times}})}_{\frac{N}{d} \text{ times}}$.

If the periodic sequence $e = DFT^{-1}(E)$ of period N denotes the inverse Discrete Fourier Transform of E then it follows that e is periodic with smallest period at least $\frac{N}{d}$ and has the following form:

$$e = (e_0, \underbrace{0, \ldots, 0}_{\frac{N}{d}-1 \text{ times}}, e_{\frac{N}{d}}, \underbrace{0, \ldots, 0}_{\frac{N}{d}-1 \text{ times}}, \ldots e_{\frac{(d-1)N}{d}}, \underbrace{0, \ldots, 0}_{\frac{N}{d}-1 \text{ times}}).$$

Moreover, for all $0 \leq i < N$, the terms of the transform sequence are

$$e_i = \begin{cases} 0, & \text{if } i \neq 0, \frac{N}{d}, \ldots, \frac{(d-1)N}{d}, \\ \frac{1}{d}(E_0 + E_1\alpha^{-i} + \ldots + E_{d-1}\alpha^{-(d-1)i}), & \text{otherwise.} \end{cases}$$

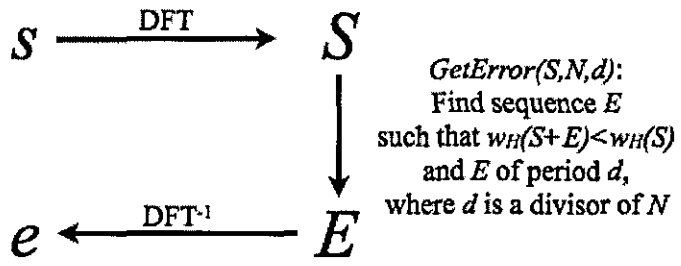

Figure 6.1: The kDFT-Approximation Algorithm

Figure 6.1 briefly describes the way the $kDFT$-Approximation Algorithm presented in section 6.2.2 works. For an input sequence s of period N, the Discrete Fourier Transform of that sequence, S, is calculated and then for each proper divisor d of N it tries to find an error sequence E of period d such that the Hamming weight of $S+E$ is less than the Hamming weight of S, i.e. $w_H(S+E) < w_H(S)$. If such sequence is found then the inverse Discrete Fourier Transform of E, denoted e, is a good error pattern for s, i.e. one which lowers the linear complexity of s while also having the Hamming weight at most d.

Since the error patterns found by the $kDFT$-Approximation Algorithm presented in the previous section are inverse Discrete Fourier Transform of sequences of period $1 < d < N$, where d is a proper divisor of N, it follows from theorem 6.13 that the aforementioned approximation algorithm can only 'catch' error patterns from the initial domain, with the non zero terms only at several specific indices, namely positions $0, \frac{N}{d}, \ldots, \frac{(d-1)N}{d}$.

We want to extend the approximation algorithm in order to find alternative good error patterns without a significant increase in the computational complexity. In the following we will investigate the scenario introduced at the beginning of the section and presented in figure 6.2, by applying the Discrete Fourier Transform to shifted versions of the input sequence. We will treat the two additional transformations which appear in this variation of the algorithm (Lemma 6.14 for the cyclical right shift of the input sequence s and Lemma 6.15 for the cyclical left shift of the error pattern e') and identify if the error sequence e produced in such a scenario can contribute to the calculation of the extended field k-error linear complexity problem.

Lemma 6.14 shows that the linear complexity of a sequence is invariant under the operation of cyclically shifting it to the left or right (figure 6.3).

Lemma 6.14. *Let s and s' be two sequences of period N over a field F. Suppose that $s' = (s'_0, s'_1, \ldots, s'_{N-1})$ is obtained by cyclically shifting all periods of sequence*

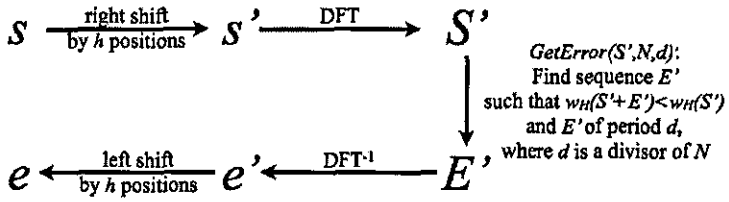

Figure 6.2: The $kDFT$-Approximation Algorithm where the input sequence is firstly cyclically shifted to the right.

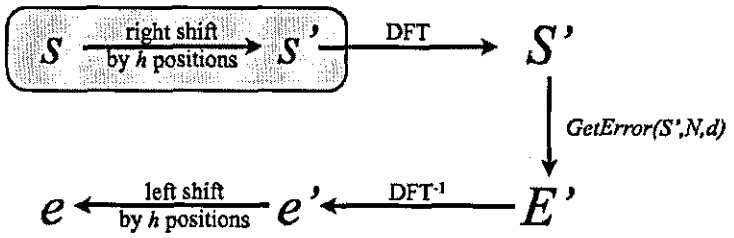

Figure 6.3: The $kDFT$-Approximation Algorithm where the input sequence is firstly cyclically shifted to the right ($L(s) = L(s')$).

s to the left (or right) by h positions, i.e.

$$s'_i = s_{i+h}, 0 \leq i < N (or\ s'_i = s_{i-h}, 0 \leq i < N)$$

where subscripts are taken modulo N. Then $L(s) = L(s')$.

Proof. When s' is a cyclical left shift of s, Property 6.3 implies that if $S' = DFT(s)$, $S = DFT(s)$ and α is an N-th root of unity in F then $S'_i = \alpha^{-hi} S_i$. Therefore $w_H(S) = w_H(S')$. Since $L(s) = w_H(S)$ and $L(s') = w_H(S')$, it follows that $L(s) = L(s')$. Since a cyclical right shift of s by h positions is equivalent with a cyclical left shift of s by $N - h$ positions (remark 6.4), the property concerning the right shift follows immediately from the above considerations. □

Lemma 6.15 deals with the other proposed transformation (see figure 6.4) and it shows that the error sequence e' which lowers the linear complexity of the right cyclically shifted input sequence s' can be used to obtain an error sequence e for the input sequence s itself. Moreover, the error sequence e obtained in such a manner will lower the linear complexity of the input sequence s with the same amount as e' lowers the linear complexity of s'.

Lemma 6.15. *Let s and s' be two sequences of period N over a field F. Suppose that* $s' = (s'_0, s'_1, \ldots, s'_{N-1})$ *is obtained by cyclically shifting all periods of sequence*

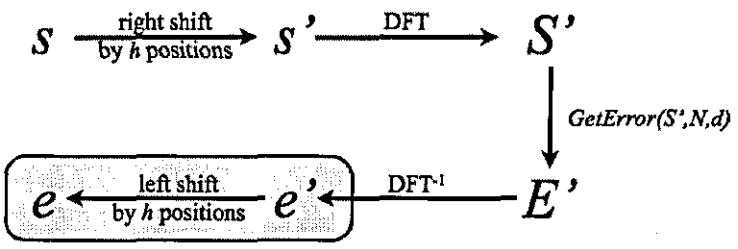

Figure 6.4: The $kDFT$-Approximation Algorithm where the input sequence is firstly cyclically shifted to the right ($L(s+e) = L(s'+e')$).

$s = (s_0, s_1, \ldots, s_{N-1})$ *to the right by* h *positions, i.e.*

$$s'_i = s_{i+h}, 0 \leq i < N$$

where subscripts are taken modulo N. *If* $e' = (e'_0, e'_1, \ldots, e'_{N-1})$ *is an error pattern which reduces the linear complexity of* s' *then* $e = (e_0, e_1, \ldots, e_{N-1})$ *obtained by cyclically shifting all periods of sequence* e' *to the left by* h *positions is an error pattern of the same Hamming weight which reduces the linear complexity of* s *by the same amount as* e', *i.e.* $L(s'+e') = L(s+e) < L(s)$.

Proof. Since s' is obtained by cyclically shifting s, Lemma 6.14 implies $L(s) = L(s')$. Therefore we need to prove that $L(s+e) = L(s'+e')$. Assume $S = DFT(s)$, $S' = DFT(s')$, $E = DFT(e')$ and $E' = DFT(e)$. Using the linearity and the shift property of the Discrete Fourier Transform (DFT), it follows that

$$DFT(s'+e') = DFT(s') + DFT(e') = S' + E' = (\alpha^{-ij}(S_i + E_i))_{0 \leq i < N}.$$

Note that saying that e is obtained from e' by cyclically shifting all periods of sequence e to the left is equivalent to saying that e' is obtained from e by cyclically shifting all periods of e' to the right. Therefore $E' = (\alpha^{-ij}E_i)_{0 \leq i < N}$.

Therefore $L(s'+e') = w_H(S'+E') = w_H((\alpha^{-ij}(S_i+E_i))_{0 \leq i < N}) = w_H(S+E) = L(s+e)$. The result can be obtained similarly when s and e are shifted to the left and right, respectively, to obtain s' and e'. □

Since the error patterns e' (see figure 6.4) are inverse Discrete Fourier Transform of sequences of period $d < N$, with d a proper divisor of N, from theorem 6.13 it follows that they have non zero values only on terms $0, \frac{N}{d}, \ldots, \frac{(d-1)N}{d}$. It follows that taking such a sequence e' and cyclically shifting it to the left by h positions, for all h such that $0 \leq h < \frac{N}{d}$, produces a set of $\frac{N}{d}$ distinct sequences denoted e.

Along with the previous comment, lemmas 6.14 and 6.15 imply that finding a good error pattern using routine *GetError* for the Discrete Fourier Transform of

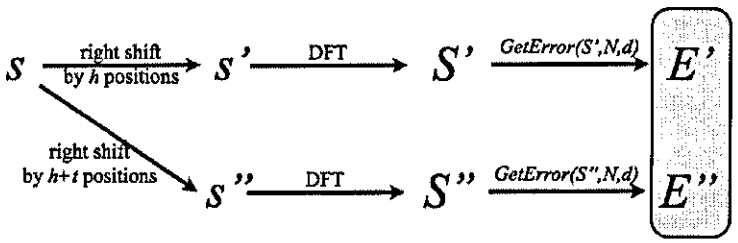

Figure 6.5: The Relation between the error sequences found for the Discrete Fourier Transform of two different shifts of the initial sequence.

a cyclically shifted version of an infinite periodic sequence means finding a good error pattern for the sequence itself, one which is different from the one produced by $kDFT$-Approximation and therefore could improve the accuracy result.

Lemma 6.16 clarifies the relation between the sequences E found using the *GetError* procedure for the Discrete Fourier Transform of two different cyclical shifts of the input sequence s (see figure 6.5).

Lemma 6.16. *Let s be a sequence of period N over a field F which contains an N-th root of unity, denoted α. Suppose N is not prime and d is a proper divisor of N.*

Suppose that $s' = (s'_0, s'_1, \ldots, s'_{N-1})$ and $s'' = (s''_0, s''_1, \ldots, s''_{N-1})$ are obtained by cyclically shifting all periods of sequence $s = (s_0, s_1, \ldots, s_{N-1})$ to the right by h and by $h + t$ positions, respectively i.e.

$$s'_i = s_{i-h}, 0 \leq i < N \text{ and } s''_i = s_{i-h-t}, 0 \leq i < N$$

where subscripts are taken modulo N and $t > 0$.

Denote $S = DFT(s)$ and $S' = DFT(s')$. For each $i = 0, 1, \ldots, d-1$ denote β'_i and β''_i the elements which occur most frequently among $S'_i, S'_{d+i}, \ldots, S'_{(\frac{N}{d}-1)d+i}$ and among $S''_i, S''_{d+i}, \ldots, S''_{(\frac{N}{d}-1)d+i}$, respectively. If there is more than one value with the same maximum occurrence then choose zero (i.e. the identity element with respect to addition in F) if zero is one of them or choose the one with the lowest index otherwise.

Under these conditions, $\beta''_i = \alpha^{ti}\beta'_i$.

Proof. Using the shift property 6.3 of the DFT we obtain that

$$\{S'_i, S'_{d+i}, \ldots, S'_{((\frac{N}{d}-1)d+i)}\} = \{\alpha^{hi}S_i, \alpha^{h(d+i)}S_{d+i}, \ldots, \alpha^{h((\frac{N}{d}-1)d+i)}S_{((\frac{N}{d}-1)d+i)}\}$$

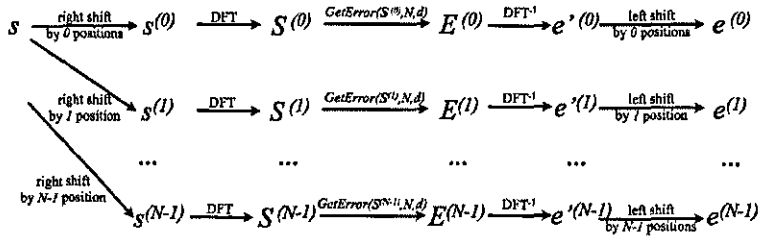

Figure 6.6: The $kDFT$-Approximation-Shift Algorithm where the input sequence is cyclically shifted to the right.

and that

$$\begin{aligned}\{S''_i, S''_{d+i}, \ldots, S''_{((\frac{N}{d}-1)d+i)}\} = \\ &= \{\alpha^{(h+t)i}S_i, \alpha^{(h+t)(d+i)}S_{d+i}, \ldots, \alpha^{(h+t)((\frac{N}{d}-1)d+i)}S_{((\frac{N}{d}-1)d+i)}\} \\ &= \alpha^{ti}\{\alpha^{hi}S_i, \alpha^{h(d+i)}S_{d+i}, \ldots, \alpha^{h((\frac{N}{d}-1)d+i)}S_{((\frac{N}{d}-1)d+i)}\}.\end{aligned}$$

The relation between the two most occurrent elements β'_i and β''_i is now immediate. □

Lemmas 6.17 and 6.18 give the characterisation of the error sequences e resulting from different initial shifts (see figure 6.6). We prove that these error patterns $e^{(i)}$ are repeating every $\frac{N}{d}$ shifts.

Lemma 6.17. *Suppose N is not prime and d is a proper divisor of N. Let F be a field which contains an N-th root of unity, denoted α.*

Let $E' = (E'_0, E'_1, \ldots, E'_{d-1})$ and $E'' = (E''_0, E''_1, \ldots, E''_{d-1})$ be two sequences of period d over F such that $E''_i = \alpha^{ti}E'_i$ for all $0 \leq i < d$, where $t > 0$. If $e' = DFT^{-1}(E')$ and $e'' = DFT^{-1}(E'')$ (where E' and E'' are considered of period N for the purpose of calculating the DFT) then

$$t \text{ is a multiple of } \frac{N}{d} \Rightarrow e'_i = e''_{i+t} \text{ for all } 0 \leq i < N.$$

Proof. Note that since d is a proper divisor of N, the sequences E' and E'' have period N. We express $e' = DFT^{-1}(E')$ and $e'' = DFT^{-1}(E'')$ using the formulas in theorem 6.13.

$$e'_i = \begin{cases} 0, & \text{if } i \neq 0, \frac{N}{d}, \ldots, \frac{N}{d}(d-1), \\ \frac{1}{d}(E'_0 + E'_1\alpha^{-i} + \ldots + E'_{d-1}\alpha^{-(d-1)i}), & \text{otherwise.} \end{cases}$$

$$e''_i = \begin{cases} 0, & \text{if } i \neq 0, \frac{N}{d}, \ldots, \frac{N}{d}(d-1), \\ \frac{1}{d}(E''_0 + E''_1\alpha^{-i} + \ldots + E''_{d-1}\alpha^{-(d-1)i}), & \text{otherwise.} \end{cases}$$

$$= \begin{cases} 0, & \text{if } i \neq 0, \frac{N}{d}, \ldots, \frac{N}{d}(d-1), \\ \frac{1}{d}(E'_0 + E'_1\alpha^{t-i} + \ldots + E'_{d-1}\alpha^{(d-1)(t-i)})s, & \text{otherwise.} \end{cases}$$

Therefore, if t is a multiple of $\frac{N}{d}$ from the expressions defining e'_i and e''_i it follows that $e'_i = e''_{i+t}$ for all $0 \leq i < N$. □

The following lemma follows immediately from the previous one, lemma 6.17.

Lemma 6.18. *Suppose N is not prime and d is a proper divisor of N. Let F be a field which contains an N-th root of unity, denoted α.*

Let $E = (E_0, E_1, \ldots, E_{d-1})$ be a sequence of period d over F and for all $h = 0, 1, \ldots N-1$, denote with $E^{(h)}$ the sequences of period d such that $E^{(h)}_j = \alpha^{hj}E_j$ for all $0 \leq j \leq d$.

For all $h = 0, 1, \ldots, N-1$, let $e^{(h)}$ be the sequence obtained from $e'^{(h)} = DFT^{-1}(E^{(h)})$ by cyclically shifting all terms in each period of size d to the left by h positions ($E^{(h)}$ is considered of period N for the purpose of calculating the DFT). Under these conditions, the following two sets are equal
$\{e^{(0)}, e^{(1)}, \ldots, e^{(N-1)}\} = \{e^{(0)}, e^{(1)}, \ldots, e^{(\frac{N}{d}-1)}\}$.

Proof. Lemma 6.17 implies that for all $h \geq 0$, and any $i > 0$, the periodic sequences $e'^{(h)} = DFT^{-1}(E^{(h)})$ and $e'^{(h+i\frac{N}{d})} = DFT^{-1}(E^{(h+i\frac{N}{d})})$ are such that $e'^{(h)}$ is equal to the sequence obtained by cyclically shifting $e'^{(h+i\frac{N}{d})}$ to the left by $i\frac{N}{d}$ positions. It follows that $e^{(h)}$ and $e^{(h+i\frac{N}{d})}$ obtained from cyclically shifting $e'^{(h)}$ and $e'^{(h+i\frac{N}{d})}$ by h and $h+i\frac{N}{d}$ positions to the left, respectively, are equal for all $h \geq 0$, and any $i > 0$. This leads to the conclusion stated in the lemma. □

We are now ready to present the following improved approximation algorithm for computing the extension field k-error linear complexity of a sequence s of period N, $EL_{k,N}(s)$.

ALGORITHM $kDFT$-Approximation-Shift
INPUT: p prime, $m \geq 1$, N a positive integer, not prime and not divisible by p, s a sequence of period N over $GF(p^m)$, $k \leq N$.
OUTPUT: a sequence e of period N over $GF(p^r)$ (where $GF(p^r)$ is the smallest extension of $GF(p^m)$ containing a primitive N-th root of unity) such that $w_H(e) \leq k$ and $L(s+e)$ is minimal among all sequences e for which there exists a sequence e' which is a circular shift of e such that $DFT(e')$ has period smaller than or equal to k; the second output is $L(s+e)$.
STEP 1. Determine r, α such that $GF(p^r)$ is the smallest extension of $GF(p^m)$ which contains a primitive N-th root of unity, α.
STEP 2. Calculate the sequence $S = DFT(s)$ over $GF(p^r)$. Set $h_{best} = 0$, $L_{best} = w_H(S)$ and set E_{best} to the all-zero sequence.

Table 6.2: $kDFT$-Approximation-Shift

d	h	$e^{(h)}$	$L(s+e^{(h)})$
3	0	$(1,0,0,0,0,1,0,0,0,0,1,0,0,0,0)$	10
3	1	$(0,0,0,0,1,0,0,0,0,1,0,0,0,0,1)$	10
3	2	$(0,0,0,\alpha^{10},0,0,0,0,\alpha^{10},0,0,0,0,\alpha^{10},0)$	9
3	3	$(0,0,1,0,0,0,0,1,0,0,0,0,1,0,0)$	9
3	4	$(0,1,0,0,0,0,1,0,0,0,0,1,0,0,0)$	10
5	0	$(\alpha^3,0,0,\alpha^{10},0,0,\alpha^{12},0,0,\alpha^8,0,0,\alpha^2,0,0)$	8
5	1	$(0,0,\alpha^2,0,0,\alpha^3,0,0,\alpha^{10},0,0,\alpha^{12},0,0,\alpha^8)$	10
5	2	$(0,\alpha^{12},0,0,\alpha^8,0,0,\alpha^2,0,0,\alpha^3,0,0,\alpha^{10},0)$	10

STEP 3. For all $d \leq k$ with d a proper divisor of N, execute steps 4-7.
STEP 4. For all $h = 0, 1, \ldots, \frac{N}{d} - 1$ execute steps 5-7.
STEP 5. Let s' be the sequence obtained from s by cyclically shifting the terms of each period to the right by h positions. Calculate $S' = DFT(s')$.
STEP 6. $E' = GetError(S', N, d)$.
STEP 7. If $w_H(S'+E') < L_{best}$ then $E_{best} = E'$, $h_{best} = h$ and $L_{best} = w_H(S'+E')$.
STEP 8. Return e, the sequence obtained from $e' = DFT^{-1}(E_{best})$ by cyclically shifting all terms of each period to the left by h_{best} positions. Also, return L_{best}.

Example 6.19. *Let $N = 15$. Let $s = (0,0,1,0,1,1,0,0,1,1,1,0,0,0,1)$ be a sequence over $GF(2)$ of period 15, as in the previous example 6.10 in section 6.2.3. Let α be a primitive element of the Galois Field $GF(2^4)$ defined by the equation $\alpha^4 + \alpha + 1 = 0$.*

Table 6.2 contains the error patterns obtained from the different shifts corresponding to the two proper divisors of 15, 3, and 5.

Note that the addition of this shifting technique has improved the accuracy of the approximation for this example sequence. For divisor 3 and shift 2 the resulting error pattern lowers the linear complexity of the initial sequence s also giving indications on where the errors are, i.e. terms 3, 7 and 11.

Theorem 6.20. *The algorithm $kDFT - Approximation - Shift$ is correct.*

Proof. Let p be a prime, m a positive integer such that $m \geq 1$, N a positive integer, not prime and not divisible by p, s a sequence of period N over $K = GF(p^m)$ and $k \leq N$.

We prove that the $kDFT - Approximation - Shift$ algorithm returns a sequence e of period N over $F = GF(p^r)$ (F is the smallest extension of $GF(p^m)$ containing a primitive N-th root of unity), such that $w_H(e) \leq k$ and $L(s+e)$

is minimal among all sequences e for which there exists a sequence e' which is a circular shift of e and such that $DFT(e')$ has period smaller than or equal to k.

For all h, where $0 \leq h < N$ and for any d proper divisor of N we make the following notations:

- let $s^{(h)}$ be the sequence obtained from s by cyclically shifting all terms of each period to the right by h positions (Step 4).
- denote $S^{(h)} = DFT(s^{(h)})$ and $E^{(h,d)}$ the sequence chosen to reduce the weight of $S^{(h)}$ of period N using the proper divisor d as described in theorem 6.9, i.e. $E^{(h,d)} = GetError(S^{(h)}, N, d)$ (Step 6).
- let $e^{(h,d)}$ be the sequence obtained by shifting all terms of $DFT^{-1}(E^{(h,d)})$ to the left by h positions.

Lemma 6.18 implies that for a fixed value of d the following two sets are equal $\{e^{(0,d)}, e^{(1,d)}, \ldots, e^{(N-1,d)}\}$ and $\{e^{(0,d)}, e^{(1,d)}, \ldots, e^{(\frac{N}{d}-1,d)}\}$. This justifies the range of the loop in step 3.

Theorem 6.9 and corollary 6.8 justify the choice of the sequences $E^{(h,d)}$ in Step 6 for all values $h = 0, \ldots, \frac{N}{d} - 1$ and all proper divisors of N, d, i.e. by using the procedure $GetError(S^{(n)}, N, d)$. Moreover, it follows that $E^{(h,d)}$ achieves minimum $w_H(S^{(h)} + E^{(h,d)})$ among all sequences E over F of period d, for each value of h and d.

With the above notations, the error pattern e returned by the algorithm is the sequence from the following set

$$\{e^{(h,d)} | 0 \leq h < \frac{N}{d}, d \text{ proper divisor of } N\}$$

such that $L(s + e)$ is minimal. It follows that $L(s + e)$ is minimal among all sequences for which there exists a sequence e' which is a circular shift of e such that $DFT(e')$ has period smaller or equal to k. □

Theorem 6.21. *The algorithm kDFT-Approximation-Shift has computational complexity $\mathcal{O}(N^3)$ operations in $GF(p^r)$. This can be improved to $\mathcal{O}(N^{2.5} \log N)$ if a Fast Fourier Transform approach is used.*

Proof. From theorem 6.11, the computational complexity of the algorithm $kDFT$-Approximation is $\mathcal{O}(N^2)$. If a Fast Fourier Transform is used, this is improved to $\mathcal{O}(N^{1.5} \log N)$.

The change from $kDFT$-Approximation consists of the additional **for** loop in STEP 4, the STEP 5 and the shifting necessary in STEP 8.

The cyclical shift of a sequence to left or right by a certain amount of positions can be done in linear time. Therefore, STEP 5 takes $\mathcal{O}(N + N^2) = \mathcal{O}(N^2)$ or $\mathcal{O}(N + N \log N) = \mathcal{O}(N \log N)$, if Fast Fourier Transform is used.

It follows that the complexity of one step of the **for** loop in STEP 4 is $\mathcal{O}(N^2 + N \log \frac{N}{d}) = \mathcal{O}(N^2)$ or $\mathcal{O}(N \log N + N \log \frac{N}{d}) = \mathcal{O}(N \log N)$ if a Fast Fourier Transform is used.

Steps 5-7 are executed $\frac{N}{d}$ times for each d, a proper divisor of N. So the total complexity of the **for** loop in STEP 3 is $\mathcal{O}(N^3)$ or $\mathcal{O}(N^{2.5} \log N)$ if a Fast Fourier Transform is used (we just need to multiply the computational complexity of steps 5-7 with $d\frac{N}{d} = N$).

Adding in the complexity of STEP 2, the computational complexity of the whole algorithm remains $\mathcal{O}(N^3)$ or $\mathcal{O}(N^{2.5} \log N)$ when a Fast Fourier Transform is used. □

6.4 Experimental results

We implemented the algorithms kDFT-Approximation and kDFT-Approximation-Shift presented in section 6.2 using GAP [18].

The results of several experiments are shown in Tables 6.3, 6.4 and 6.5. We tested binary sequences of all odd lengths N up to 300 excluding those that are prime and those for which the extension field $GF(p^r)$ in which the primitive N-th root of unity lies has $r \geq 20$ (the latter restriction is only for efficiency reasons). We also tested a couple of higher lengths of the form $N = 2^r - 1$, e.g. $N = 1023, 2047$.

For each length we generated 100 sequences using the standard pseudorandom number generator (linear congruential generator). The tables 6.3, 6.4 and 6.5 show for each length N, the extension field for the primitive N-th root of unity, and, for each of the proper divisors of N, the percentage of sequences for which an error sequence which decreases the complexity was found (success rate) and the average decrease of complexity which has been thus obtained (average decrease of linear complexity, where the decrease in complexity is computed as $(L(s)-L(s+e))/L(s)$, where e is the error sequence returned by the algorithm), for each of the algorithms $kDFT$-Approximation and $kDFT$-Approximation-Shift proposed in sections 6.2.3 and 6.3, respectively.

For practical applications the cases of interest are those where k represents some small percentage (e.g. 5% or 10%) of the length N. These can be identified in the tables on the rows where the ratio d/N is below such a percentage.

One can notice that the approximation algorithm was successful in finding good error patterns for the great majority of the sequences, the success rate being 100% or close to 100% for most of the sequences and for most of the divisors.

Table 6.3: Experimental results

			kDFT-Approx		kDFT-Approx-Shift	
Length N	Extension field	Divisor d	Success rate	Av. decr. of lin. compl.	Success rate	Av. decr. of lin. compl.
15	$GF(2^4)$	3	74%	28.05%	90%	41.21%
		5	88%	39.34%	92%	49.09%
21	$GF(2^6)$	3	79%	18.98%	94%	32.46%
		7	89%	37.05%	90%	45.57%
27	$GF(2^{18})$	3	100%	11.01%	100%	15.36%
		9	100%	34.44%	100%	38.05%
33	$GF(2^{10})$	3	69%	8.14%	74%	12.79%
		11	100%	33.28%	100%	35.74%
35	$GF(2^{12})$	5	96%	15.67%	97%	20.03%
		7	88%	19.15%	88%	23.68%
39	$GF(2^{12})$	3	78%	6.91%	86%	10.17%
		13	100%	33.69%	100%	35.63%
45	$GF(2^{12})$	3	100%	9.13%	100%	16.62%
		5	83%	12.63%	87%	17.72%
		9	77%	22.61%	89%	28.30%
		15	100%	34.18%	100%	38.91%
51	$GF(2^8)$	3	82%	7.76%	99%	15.62%
		17	97%	33.73%	97%	35.89%
57	$GF(2^{18})$	3	77%	4.33%	77%	5.02%
		19	100%	33.28%	100%	34.73 %
63	$GF(2^6)$	3	87%	9.45%	89%	17.84%
		7	87%	15.98%	94%	22.24%
		9	79%	17.44%	95%	24.22%
		21	96%	34.46%	96%	38.52%
65	$GF(2^{12})$	5	90%	7.15%	91%	9.24%
		13	100%	20.06%	100%	21.83%
85	$GF(2^8)$	5	91%	7.41%	100%	13.84%
		17	97%	20%	98%	24.19%
91	$GF(2^{12})$	7	83%	7.16%	86%	9.87%
		13	100%	14.89%	100%	16.51%
93	$GF(2^{10})$	3	80%	4.87%	100%	10.43%
		31	97%	34.23%	97%	36.09%

Table 6.4: Experimental results (continued)

			kDFT-Approx		kDFT-Approx-Shift	
Length N	Extension field	Divisor d	Success rate	Av. decr. of lin. compl.	Success rate	Av. decr. of lin. compl.
105	$GF(2^{12})$	3	70%	4.70%	92%	9.21%
		5	79%	6.46%	87%	10.47%
		7	91%	7.30%	95%	11.69%
		15	90%	14.46%	90%	18.06%
		21	98%	20.02%	98%	23.43%
		35	100%	33.49%	100%	35.88%
117	$GF(2^{12})$	3	96%	3.09%	97%	6.22%
		9	96%	7.79%	100%	10.83%
		13	100%	11.33%	100%	12.87%
		39	100%	33.73%	100%	34.70%
129	$GF(2^{14})$	3	76%	1.96%	85%	3.27%
		43	100%	33.27%	100%	33.92%
133	$GF(2^{18})$	7	91%	4.65%	91%	5.35%
		19	100%	14.56%	100%	15.99%
171	$GF(2^{18})$	3	96%	1.86%	96%	2.47%
		9	96%	4.68%	96%	4.91%
		19	100%	11.20%	100%	11.82%
		57	100%	33.62%	100%	34.14%
189	$GF(2^{18})$	3	100%	2.36%	100%	6.16%
		7	89%	5.24%	99%	7.47%
		9	93%	6.16%	96%	9.21%
		21	100%	12.42%	100%	14.94%
		27	100%	15.29%	100%	18.15%
		63	100%	32.39%	100%	33.60%
195	$GF(2^{12})$	3	81%	2.32%	97%	5%
		5	82%	3.07%	98%	5.96%
		13	100%	7.15%	100%	9.96%
		15	96%	7.16%	98%	9.73%
		39	100%	20.33%	100%	21.66%
		65	100%	33.36%	100%	34.26%
217	$GF(2^{15})$	7	86%	3.54%	88%	5.82%
		31	98%	14.14%	98%	15.30%
219	$GF(2^{18})$	3	76%	1.42%	79%	4.06%
		73	100%	33.35%	100%	33.93%

Table 6.5: Experimental results (continued)

Length N	Extension field	Divisor d	kDFT-Approx Success rate	kDFT-Approx Av. decr. of lin. compl.	kDFT-Approx-Shift Success rate	kDFT-Approx-Shift Av. decr. of lin. compl.
255	$GF(2^8)$	3	82%	2.96%	100%	6.42%
		5	89%	3.92%	99%	7.20%
		15	92%	8.19%	98%	11.29%
		17	95%	8.71%	99%	12.92%
		51	99%	20.67%	99%	22.48%
		85	99%	33.35%	99%	34.72%
273	$GF(2^{12})$	3	77%	1.70%	100%	4.22%
		7	93%	2.95%	98%	5.57%
		13	100%	5.18%	100%	8.15%
		21	98%	7.53%	100%	10.00%
		39	100%	14.50%	100%	15.96%
		91	100%	33.40%	100%	34.12%
1023	$GF(2^{10})$	3	93%	0.77%	100%	2.36%
		11	100%	2.18%	100%	3.49%
		31	95%	4.31%	98%	5.82%
		33	98%	4.27%	100%	6.14%
		93	100%	9.71%	100%	10.96%
		341	100%	33.37%	100%	33.77%
2047	$GF(2^{11})$	23	98%	0.02%	100%	2.36%
		89	100%	0.05%	100%	0.07%

It would be useful to compare the decrease in complexity with the decrease in complexity obtained by an optimal solution. However, it is infeasible to compute the optimal solution by exhaustive search except for very small lengths.

We implemented this test only for sequences of length 15 and for the approximation algorithm when the divisor 3 is considered, which gives an approximation for $EL_3(s)$. For a number of 100 sequences generated in a similar fashion as for the previous test, the average accuracy of the approximation is 2.58 for $kDFT$-Approximation and 2.03 for $kDFT$-Approximation-Shift, where the accuracy is the ratio between the approximate value of $EL_3(s)$ returned by each of the algorithms and the exact value calculated using an exhaustive technique.

6.5 Conclusion

We extended the classical notion of k-error linear complexity of periodic sequences, named extension field k-error linear complexity, and we present a general algorithm for its computation. This algorithm has theoretical value but it is not practical since one of its steps has exponential computational complexity.

For the above reason, we describe a practical implementation which efficiently approximates the extension field k-error linear complexity.

While our approximation algorithms will not always find error patterns that decrease the complexity of the sequence, experimental results were promising and such error patterns were found in the vast majority of cases. The proposed algorithms have the advantage that they are very fast, polynomial computational complexity, so they could be used to discard many of the weak sequences before other, more costly tests are performed.

Further work would include the following: testing the proposed algorithms

on sequences generated by some of the classical pseudorandom number generators used in cryptographical applications; designing alternative approximation algorithms; further investigation of the relationship between the extension field k-error linear complexity and the k-error linear complexity.

Chapter 7

Conclusions

7.1 Concluding remarks

Pseudorandom sequences used in cryptography need to have a high linear complexity and a high k-error linear complexity. While the linear complexity can be efficiently computed using the Berlekamp-Massey Algorithm, there are no efficient algorithms for computing the k-error linear complexity, except for sequences whose period is a power of the characteristic of the field.

We investigated ways of computing or approximating the k-error linear complexity for sequences of arbitrary length over finite fields. The advantage of the heuristic algorithms we propose is that they work on arbitrary sequences and even if they only approximate the exact result, the approximation is accurate and the computational time complexity is manageable.

Our work could be used for building quick tests in the design and analysis stage for cryptographic sequences used in stream ciphers.

To summarise, the research findings of this thesis are as follows:

- An improved exhaustive algorithm for computing the exact value of the k-error linear complexity of a sequence.
- A Modified Berlekamp-Massey Algorithm to efficiently approximate the k-error linear complexity of cryptographic sequences over finite fields.
- An implementation of two evolutionary techniques, genetic algorithms and simulated annealing, for approximating k-error linear complexity of cryptographic sequences.
- A new concept which extends the k-error linear complexity, named the extension field k-error linear complexity.

- A general algorithm for computing the extension field k-error linear complexity.
- An efficient algorithm for approximating the extension field k-error linear complexity by using the Discrete Fourier Transform.

7.2 Suggestions for future work

Our work has produced some interesting results but also left a few problems open for future research.

For example, it would be interesting to design an evolutionary algorithm for the k-error linear complexity problem where the individuals are not the error patterns like in chapter 5 since this is prohibitive in terms of space. One possibility would be for the individuals to be recurrences or characteristic polynomials. In this case, efficient and meaningful recombination techniques should be devised.

It is interesting to look closer at the concept of extension field k-error linear complexity defined in chapter 6 and the relation between this concept and the k-error linear complexity. Also, it is of interest to find an optimal solution or alternative approximation techniques for Step 3 of the general algorithm $kDFT$ presented in the same chapter.

We hold as future work to investigate if the Fleischmann Algorithm (Fleischmann [16]) can be used to calculate the k-error linear complexity. The Fleischmann Algorithm is an extension of the Berlekamp-Massey Algorithm which computes a two-sided linear complexity for sequences which are not fully known apriori, a situation which happens in cryptanalysis applications.

We do not exclude the possibility and leave as future work to investigate if the k-error linear complexity problem is NP-hard, in which case our heuristic techniques could be the best methods that one could apply for this problem.

References

[1] A. Alecu and A. Sălăgean. A genetic algorithm for computing the k-error linear complexity of cryptographic sequences. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 3569–3576, 2007.

[2] A. Alecu and A. Sălăgean. Modified Berlekamp-Massey Algorithm for Approximating the k-Error Linear Complexity of Binary Sequences. In Steven D. Galbraith, editor, *Cryptography and Coding*, volume 4887 of *Lecture Notes in Computer Science*, pages 220–232. Springer, 2007.

[3] A. Alecu and A. Sălăgean. An Approximation Algorithm for Computing the k-error Linear Complexity of Sequences Using the Discrete Fourier Transform. In *IEEE International Symposium on Information Theory*. IEEE Computer Society, 2008.

[4] E. R. Berlekamp. *Algebraic Coding Theory*. McGraw-Hill, NY, 1968.

[5] S. R. Blackburn. A Generalisation of the Discrete Fourier Transform: Determining the Minimal Polynomial of a Periodic Sequence. *IEEE Transactions on Information Theory*, 40(5):1702–1704, 1994.

[6] R. E. Blahut. *Theory and Practice of Error Control Codes*. Addison-Wesley, Reading, 1983.

[7] R. E. Blahut. Transform Techniques for Error Control Codes. *IBM J. Res. Develop.*, 23(3):299–315, May 1979.

[8] S. Boztas. A Fast Algorithm for Linear Complexity (mod p). Research Report 9, Department of Mathematics, Royal Melbourne Institute of Technology, 1997.

[9] V. Cerny. A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. *J. of Optimization Theory and Applications*, 45:41–51, 1985.

[10] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* MIT Press and McGraw-Hill, 2nd edition, 2001. Chapter 30. Polynomials and the FFT.

[11] Z.D. Dai. Proof of Rueppel's Linear Complexity. *IEEE Transactions on Information Theory*, 32(3):440–443, May 1986.

[12] C. Ding, G. Xiao, and W. Shan. *The Stability Theory of Stream Ciphers.* Springer-Verlag, Heidelberg, 1992.

[13] ECRYPT. eSTREAM - The ECRYPT Stream Cipher Project, accessed in November 2008. http://www.ecrypt.eu.org/stream/index.html.

[14] H. J. Fell. Linear complexity of transformed sequences. In Gérard D. Cohen and Pascale Charpin, editors, *EUROCODE*, volume 514 of *Lecture Notes in Computer Science*, pages 205–214. Springer, 1990.

[15] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. Wiley, NY, 1968.

[16] M. Fleischmann. Modified Berlekamp-Massey algorithm for two-sided shift-register synthesis. *IEEE Electronics Letters*, 31(8):605–606, 1995.

[17] 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.

[18] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.4.9*, 2006.

[19] M. R. Garey and D. S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences).* W. H. Freeman, January 1979.

[20] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, USA, 1989.

[21] J. Dj. Golic. On the linear complexity of functions of periodic $GF(q)$ sequences. *IEEE Transactions on Information Theory*, 35(1):69–75, 1989.

[22] S. W. Golomb. *Shift Register Sequences.* AEGEAN PARK PRESS, California, 1982.

[23] E. J. Groth. Generation of Binary Sequences With Controllable Complexity. *IEEE Transactions on Information Theory*, 17(3):288–296, 1971.

[24] F. G. Gustavson. Analysis of the Berlekamp-Massey Linear Feedback Shift-Register Synthesis Algorithm. *IBM Journal Research Development*, 1(1):204–212, 1976. Apparently a simple efficiency analysis.

[25] T. Herlestam. On functions of linear shift register sequences. In *Proceedings EUROCRYPT'85*, Lecture Notes in Computer Science 219: Advances in Cryptology, pages 119–129. Spinger-Verlag, April 1985.

[26] L. Ingber. Simulated annealing: Practice and theory. *Journal of Mathematical Computation Modelling*, 18:29–57, 1993.

[27] L. Ingber. Adaptive simulated annealing (asa): Lessons learned. *Control and Cybernetics, Special Issue on "Simulated Annealing Applied to Combinatorial Optimization*, 25:33–54, 1996.

[28] S. Jiang, Z. Dai, and K. Imamura. Linear Complexity of a Sequence Obtained from a Periodic Sequence by Either Substituting, Inserting or Deleting k Symbols Within One Period. *IEEE Trans. in Information Theory*, 46(3):1174–1177, May 2000.

[29] 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.

[30] T. Kaida, S. Uehara, and K. Imamura. Computation of the k-error linear complexity of binary sequences with period 2^n. 1996.

[31] 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.

[32] 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.

[33] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1986.

[34] A. Klapper. The vulnerability of geometric sequences based on fields of odd characteristic. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 7(1):33–51, Winter 1994.

[35] N. Kolokotronis, P. Rizomiliotis, and N. Kalouptsidis. Minimum linear span approximation of binary sequences. *IEEE Trans. on Information Theory*, 48(10):2758–2764, October 2002.

[36] K Kurosawa, T. Sakata, and W. Kishimoto. A relationship between linear complexity and k-error linear complexity. *IEEE Trans. on Information Theory*, 46(10):694–698, October 2000.

[37] S. Lang. *Algebra.* Springer, 3rd edition, 2002. ISBN-13 9780387953854.

[38] A. G. B. Lauder and K. G. Paterson. Computing the Error Linear Complexity Spectrum of a Binary Sequence of Period 2^n. *IEEE Trans. Inf. Theory*, 49(1):273–2803, 2003.

[39] R. Lidl and H. Niederreiter. *Introduction to finite fields and their applications.* Cambridge University Press, 1994.

[40] R. Lidl and H. Niederreiter. *Finite fields.* Cambridge University Press, second edition, 1997.

[41] J. L. Massey. Cryptography and Systems Theory. In *Proceedings of Allerton Conference on Comm., Control, and Computing*, Oct. 1-3, 1986.

[42] J. L. Massey. Shift-Register Synthesis and BCH Decoding. *IEEE Trans. Information Theory*, 15(1):122–127, 1969.

[43] J. L. Massey. The discrete fourier transform in coding and cryptography. In *Proc. IEEE Information Theory Workshop*, pages 1–2, February 8-11 1998.

[44] J. L. Massey and T. Schaub. Linear Complexity in Coding Theory. In *Coding Theory and Applications*, Lecture Notes in Computer Science, No. 311, pages 19–32, Heildelberg and New York, 1988. Springer.

[45] J. L. Massey and S. Serconek. Linear complexity of periodic sequences: A general theory. *Lecture Notes in Computer Science*, Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology:358–371, 1996.

[46] R. J. McEliece. *Finite Fields for Computer Scientists and Engineers.* Kluwer Academic Publishers, Massachusetts, first edition, 1987.

[47] W. Meidl. How many bits have to be changed to decrease the linear complexity? *Designs, Codes and Cryptography*, 33:109–122, 2004.

[48] W. Meidl. On the Stability of 2^n-Periodic Binary Sequences. *IEEE Trans. on Information Theory*, 51(3):1151–1155, March 2005.

[49] W. Meidl and H. Niederreiter. Linear Complexity, k-Error Linear Complexity, and the Discrete Fourier Transform. *Journal of Complexity*, 18:87–103, 2002.

[50] W. Meidl and H. Niederreiter. On the Expected Value of the Linear Complexity and the k-Error Linear Complexity of Periodic Sequences. *IEEE Trans. on Information Theory*, 48(11):2817–2825, 2002.

[51] W. Meidl and H. Niederreiter. Periodic Sequences with Maximal Linear Complexity and Large k-Error Linear Complexity. *Applicable Algebra in Engineering, Communication and Computing*, 14(4):273–286, November 2003.

[52] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press LLC, Florida, 1997.

[53] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *J. of Chemical Physics*, 21:1087–1092, 1953.

[54] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, Berlin, 1999.

[55] H. Niederreiter. Sequences with almost perfect linear complexity profile. In *Proceedings EUROCRYPT'87*, Lecture Notes in Computer Science 304: Advances in Cryptology, pages 37–51. Spinger-Verlag, 1987.

[56] H. Niederreiter. The probabilistic theory of linear complexity. In *Proceedings EUROCRYPT'88*, Lecture Notes in Computer Science 330: Advances in Cryptology, pages 191–209. Spinger-Verlag, 1988.

[57] H. Niederreiter. Keystream sequences with a good linear complexity profile for every starting point. In *Proceedings EUROCRYPT'89*, Lecture Notes in Computer Science 434: Advances in Cryptology, pages 523–532. Spinger-Verlag, 1989.

[58] H. Niederreiter. Linear complexity and related complexity measures for sequences. In *Progress in Cryptology - INDOCRYPT 2003*, volume Volume 2904/2003 of *Lecture Notes in Computer Science*, pages 161–245, Berlin / Heidelberg, 2003. Springer.

[59] H. Niederreiter. Periodic Sequences with Large k-Error Linear Complexity. *IEEE Trans. on Information Theory*, 49(2):501–505, 2003.

[60] K. G. Paterson. Root counting, the dft and the linear complexity of nonlinear filtering. *Designs, Codes and Cryptography*, 14:247–259, 1998.

[61] F. Piper. Stream ciphers. *Elektrotechnik und Maschinenbau*, 104(12):564–568, 1987.

[62] J. M. Pollard. The fast fourier transform in a finite field. *Mathematics of Computation*, 25:365–374, 1971.

[63] F. P. Preparata and D.V. Sarwate. Computational complexity of fourier transforms over finite fields. *Mathematics of Computation*, 31:740–751, 1977.

[64] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C++: The Art of Scientific Computing.* Cambridge University Press, USA, 2002.

[65] S. Ramanujan. On the Number of Divisors of a Number. *Journal of the Indian Mathematical Society*, VII:131–133, 1915.

[66] C. 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.

[67] M. J. B. Robshaw. Stream Ciphers. Technical Report TR-701, RSA Laboratories, 1995.

[68] M. J. B. Robshaw. On Evaluating the Linear Complexity of a Sequence of Least Period 2^n. *Designs, Codes and Cryptography*, 4(4):263–269, October 1994.

[69] R. A. Rueppel. Linear complexity and random sequences. In *Advances in Cryptology: Proceedings EUROCRYPT'85, Linz, Austria*, volume Lecture Notes in Computer Science: Advances in Cryptology, pages 167–188. Spinger-Verlag, 1985.

[70] R. A. Rueppel. *Analysis and Design of Stream Ciphers.* Springer-Verlag, New York, 1986.

[71] R. A. Rueppel and O. J. Staffelbach. Products of Linear Recurring Sequences with Maximum Complexity. *IEEE Trans. on Information Theory*, 33(1):124–131, 1987.

[72] R. S. Safavi-Naini and J. R. Seberry. Pseudo-random sequence generators using structure noise. In J. H. Loxton, editor, *Number Theory and Cryptography (London Mathematical Society Lecture Note Series 154)*, pages 129–136. Cambridge University Press, 1990.

[73] A. Salagean. On the computation of the linear complexity and the k-error linear complexity of binary sequences with period a power of two. *IEEE Trans. Information Theory*, 51(3):1145–1150, 2005.

[74] A. Salagean. An algorithm for computing minimal bidirectional linear recurrence relations. In *IEEE International Symposium on Information Theory.* IEEE Computer Society, 2008.

[75] C. E. Shannon. Communication Theory of Secrecy Systems. *Bell System Technical Journal*, 28(4):656–715, 1949.

[76] T. Siegenthaler. Correlation-Immunity of Nonlinear Combining Functions for Cryptographic Applications. *IEEE Transactions on Information Theory*, 30(5):776–780, 1984.

[77] G. J. Simmons. *Contemporary Cryptology: The Science of Information Integrity.* IEEE Press, Piscataway, NJ, 1992. Rainer Rueppel, Stream Ciphers, chapter 2, pages 65-134.

[78] W. Stallings. *Cryptography and Network Security - principles and practices.* Pearson Education, Inc., New Jersey, 2003.

[79] 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.

[80] D. R. Stinson. *Cryptography Theory and Practice.* CRC Press LLC, United States of America, 1995.

[81] G. Sywerda. 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.

[82] G. S. Vernam. Cipher printing telegraph systems for secret wire and radio telegraphic communications. *Journal of the IEEE*, 55(1):109–115, 1926.

[83] M. Z. Wang and J. L. Massey. The Characterization of All Binary Sequences with Perfect Linear Complexity Profiles. In *Proceedings of Eurocrypt'86, Linkoping, Sweden*, 1986.

[84] Jianqin Zhou and Xirong Xu. An algorithm for the k-error linear complexity of a sequence with period 2pn over gf(q). *CoRR*, abs/cs/0512039, 2005. http://arxiv.org/abs/cs/0512039.

Appendix A

Rings, Ideals and Finite Fields

This appendix summarises a few essential algebraic concepts which are used throughout the Thesis. These along with more details can be found in algebra or finite fields textbooks like Lang [37], Lidl and Niederreiter [40, 39] or McEliece [46].

Definition A.1. *A set of elements G with a binary operation $\cdot$, denoted $(G, \cdot)$ is a* ***group*** *if it satisfies the following properties:*

(i) $\cdot$ *is associative. For any* $a, b, c \in G$, $a \cdot (b \cdot c) = (a \cdot b) \cdot c$.

(ii) There is a identity element $e \in G$ *such that* $e \cdot a = a \cdot e = a$ *for all* $a \in G$.

(iii) For each $a \in G$, *there exists an inverse element* $a^{-1} \in G$ *such that* $a \cdot a^{-1} = a^{-1} \cdot a = e$.

If for all $a, b \in G, a \cdot b = b \cdot a$ *then the group is called* **commutative** *or* **abelian**. *If G is a finite set then $(G, \cdot)$ is called a finite group.*

In the following we will only consider commutative groups.

Definition A.2. *Suppose S is a set. A subset R of a* $S \times S = \{(a, b) | a \in S, b \in S\}$ *is a* ***equivalence relation*** *on the set S if it satisfies the following properties:*

(i) For any $a \in S$, $(a, a) \in R$ *(reflexivity).*

(ii) If $(a, b) \in R$ *then* $(b, a) \in R$ *for any* $a, b \in S$ *(symmetry).*

(ii) If $(a, b) \in R$ *and* $(b, c) \in R$ *then* $(a, c) \in R$ *for any* $a, b, c \in S$ *(transitivity).*

For a fixed $a \in S$, *the set* $[a] = \{b | (a, b) \in R\}$ *is called the* **equivalence class** *of a. The equivalence classes are inducing a partition of S into nonempty, mutually disjoint subsets.*

Definition A.3. *A subset H of a group G is a* ***subgroup*** *of G if H is itself a group with respect to the operation of the group G.*

The subgroup H of the group G is called a ***normal subgroup*** *of G if $aha^{-1} \in H$ for all $a \in G$ and all $h \in H$.*

Remark A.4. *All subgroups of an abelian group are normal.*

Theorem A.5. *If H is a subgroup of G, then the relation R_H on G defined by $(a,b) \in R_H$ if and only if $a = bh$ for some $h \in H$, is an equivalence relation.*

Definition A.6. *The equivalence relation R_H is called* ***congruence modulo*** *H. It induces a partition of G into nonempty, mutually disjoint subsets corresponding to the congruence classes. These sets are called* ***cosets*** *of G modulo H and they are denoted by $aH = \{ah | h \in H\}$.*

Theorem A.7. *If H is a normal subgroup of G, then the set of cosets of G modulo H forms a group with respect to the operation defined by $(aH)(bH) = (ab)H$.*

Definition A.8. *For a normal subgroup H of G, the group formed by the cosets of G modulo H is called the* ***factor group*** *of G modulo H and it is denoted G/H.*

Definition A.9. *A set of elements R with two binary operations, $+$ (addition) and $\cdot$ (multiplication), denoted as $(R, +, \cdot)$, is a* ***ring*** *if it satisfies the following properties:*

(i) (R, +) is a commutative group (stable, associative, commutative, has an identity element called 0, all $a \in R$ have an additive inverse denoted $-a$).

(ii) Distributive law. For all $a, b, c \in F, (a+b) \cdot c = a \cdot c + b \cdot c$ and $a \cdot (b+c) = a \cdot b + a \cdot c$.

Definition A.10. *A ring $(R, +, \cdot)$ is called a* ***cancellation ring*** *or a ring without zero-divisors if for all $a, b \in R, a \cdot b = 0 \Rightarrow a = 0$ or $b = 0$.*

Definition A.11. *Let $(R, +, \cdot)$ be a ring and $I \subseteq R$, a subset. I is called an* ***ideal*** *of R if it satisfies the following properties:*

(i) $(I, +)$ is a subgroup of $(R, +)$

(ii) for all $i \in I$ and $a \in R \Rightarrow i \cdot a \in I$ and $a \cdot i \in I$.

If there exists $i_k \in I, k = 1, \ldots, t$ so that $I = \{\sum_{k=1}^{t} i_k r_k | r_k \in R\}$ then I is called finite generated and $\{i_k | k = 1, \ldots, t\}$ is a base for that ideal.

Definition A.12. *If the set of generators of an ideal I is a singleton $\{i\}$ then the ideal is called* ***principal ideal*** *and we denote it $(i) = \{i \cdot r | r \in R\}$.*

Note that a singleton is a set containing one element.

Definition A.13. *An **integral domain** is a cancellation ring with an identity element.*

Definition A.14. *A **principal ideal domain** (PID) is an integral domain in which every ideal is principal.*

Definition A.15. *An ideal J of a ring R defines a partition of R into disjoint cosets, called **residue classes modulo** J.*

The default operations on the residue classes modulo J are for any $a, b \in R$ defined by $(a+J)+(b+J) = (a+b)+J$ and $(a+J)(b+J) = ab+J$.

*The ring of residue classes of the ring R modulo the ideal J under the default operations $+$ and $\cdot$ is called the **residue class ring** or **factor ring** of R modulo J and it is denoted R/J.*

Definition A.16. *A set of elements F with two operations, $+$ (addition) and $\cdot$ (multiplication), denoted as $(F, +, \cdot)$, is a **field** if it satisfies the following properties:*

(i) $(F, +, \cdot)$ *is a ring.*

(ii) $(F^*, \cdot)$ *is a commutative group (stable, associative, commutative, has an identity element called 1, all elements $b \in F$ have a multiplicative inverse denoted b^{-1}).*

Definition A.17. *If F is a field, the ring of residue classes of F modulo an ideal J with the default operations $+$ and $\cdot$ is called the **factor field** of R modulo J and it is denoted R/J.*

Definition A.18. *If R is an arbitrary ring and there exists a positive integer k such that $kr = 0$ for every $r \in R$, then the least such positive integer k is called the **characteristic** of R. If no such positive integer exists then R is said to have characteristic 0.*

Definition A.19. *A field F with a finite number of elements is called a **finite field**. The number of the elements in a field is called the **order** of the field. A field of order q is also called a **Galois field** of order q and is denoted $GF(q)$.*

Theorem A.20. *A finite field has prime characteristic.*

Example A.21. *1.* $(\mathbb{Z}, +, \cdot)$ *is an infinite ring.*

2. $(\mathbb{Z}_4 = \{0, 1, 2, 3\}, +, \cdot)$ *where $+$ and $\cdot$ are operations modulo 4 is not a cancellation ring since* $2 \cdot 2 = 0(\mod 4)$.

3. $(\mathbb{R}, +, \cdot), (\mathbb{Q}, +, \cdot), (\mathbb{C}, +, \cdot)$ *are infinite fields.*

4. $(GF(p), +, \cdot)$ *or* $(\mathbb{Z}_p, +, \cdot)$ *with* p *a prime number are finite fields.*

Definition A.22. *An* ***isomorphism*** *between two fields is a bijective function between the two fields, where both the function and its inverse are preserving the structure between the two fields. In this case we say that the two fields are isomorphic. If* F *and* G *are two isomorphic fields we say that* $F \simeq G$.

The most important result for the characterization of finite fields is the following.

Theorem A.23. *There exists a field of order* q *if and only if* q *is a prime power* $(q = p^m, m \geq 1$, p *a prime integer). Furthermore, if* q *is a prime power, then, up to an isomorphism, there is only one field of that order, denoted* $GF(q)$.

Definition A.24. *Let* R *be an arbitrary ring. A* ***polynomial over*** R *is an expression of the form*

$$f(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + \ldots + a_n x^n,$$

where n *is a nonnegative integer, the coefficients* a_i *are elements of* R *for all* $i = 0, 1, \ldots, n$, $a_n \neq 0$, *and* x *is a symbol not belonging to* R, *called an indeterminate over* R. a_n *is called the leading coefficient of* $f(x)$ *and* a_0 *the constant term.* n *is called the degree of the polynomial and we say that* $n = \deg(f)$.

We say that $f(X)$ *has a free term if the coefficient* a_0 *is not zero, i.e.* $a_0 \neq 0$.

If R *has an identity element denoted with* 1 *and the leading coefficient of* $f(x)$ *is* 1 *then* $f(x)$ *is called a monic polynomial.*

The ring formed by the polynomials over R *with the above operations is called the polynomial ring over* R *and is denoted* $R[x]$.

Theorem A.25. *If* F *is a field then the polynomial ring* $F[x]$ *is a principal ideal domain. Moreover, for every ideal* $J \neq \{0\}$ *where* 0 *is the identity, there exists a uniquely determined monic polynomial* $g \in F[x]$ *such that* g *is the generator of* J, *i.e.* $J = (g)$.

Definition A.26. *An element* r *in a ring* R *is* ***irreducible*** *if there is no pair of elements* $p, q \in R$ *such that* $p, q \notin \{0_R, 1_R, r\}$ *and* $r = pq$.

A polynomial $p \in F[x]$ *is said to be* ***irreducible*** *over* F *(or irreducible in* $F[x]$, *or prime in* $F[x]$*) if* p *has positive degree and* $p = bc$ *with* $b, c \in F[x]$ *implies that either* b *or* c *is a constant polynomial.*

Theorem A.27. *For $f \in F[x]$, the residue class ring $F[x]/(f)$ is a field if and only if f is irreducible over F.*

Remark A.28. *The finite field $GF(p^m)$ can be built as the factor field $\mathbb{Z}_p[X]/(f)$, where f is a polynomial of degree m, irreducible in $\mathbb{Z}_p[X]$ and (f) is the ideal generated by f in $\mathbb{Z}_p[X]$. Note that the structure of the residue class ring $\mathbb{Z}_p[X]/(f)$ can be characterized as being the set of all p^m polynomials of degree strictly smaller than f and with coefficients in $\mathbb{Z}_p[X]$.*

$$\mathbb{Z}_p[X]/(f) = \{g + (f) | g \in \mathbb{Z}_p[X]\} = \{r + (f) | r \in \mathbb{Z}_p[X], \deg r < \deg f\}$$

Example A.29. *Denote $f(x) = x^2 + x + 1 \in \mathbb{Z}_2[X]$, an irreducible polynomial. In these conditions, $\mathbb{Z}_2[X]/(f) \simeq GF(2^2)$ and we can consider as the elements of $GF(2^2)$ the residue classes represented by the four polynomials of degree strictly smaller than three, i.e. $0, 1, x, 1 + x$.*

Definition A.30. *An element b in a field F, is called a **root** of a polynomial $f \in F[x]$ if $f(b) = 0$.*

Theorem A.31. *An element $b \in F$ is a root of a polynomial $f \in F[x]$ if and only if $x - b$ divides $f(x)$.*

Definition A.32. *If F is a field, a subset K of F that it is itself a field under the operations of F is called a **subfield** of F.*

*If $K \neq F$ than K is called a **proper subfield** of F.*

*In these conditions F is called an **extension (field) of** K.*

Definition A.33. *A field containing no proper subfields is called a **prime** field.*

Theorem A.34. *The prime subfield of a field is isomorphic to either $GF(p)$ or $\mathbf{Q}$, according as the characteristic of F is a prime p or 0.*

Definition A.35. *Let K be a subfield of the field F and M a subset of F. Then the field $K(M)$ is defined as the intersection of all subfields of F containing both K and M and is called the **extension (field)** of K obtained by adjoining the elements in M.*

For a finite subset $M = \{\theta_1, \ldots, \theta_n\}$ we write $K(M) = K(\theta_1, \ldots, \theta_n)$.

If M is a singleton $\{\theta\}$, then we write $K(\theta)$ and say it is a simple extension of K and θ is called the defining element of $K(\theta)$ over K.

Definition A.36. *Let $f \in K[x]$ be of positive degree and F an extension field of K. Then f is said to **split in F** if f can be written as a product of linear factors in $F[x]$, i.e. there exist elements $\alpha_1, \alpha_2, \ldots, \alpha_n \in F$ such that $f(x) = a(x - \alpha_1) \ldots (x - \alpha_n)$, where a is the leading coefficient of f.*

The field F is a splitting field of f over K if f splits in F and if, moreover, $K = K(\alpha_1, \alpha_2, \ldots, \alpha_n)$.

Note that a splitting field F of f over K is the smallest field containing all the roots of f, i.e. no proper subfield of F that is an extension of K contains all the roots of f. Due to the fact that we can prove an existence and uniqueness up to an isomorphism theorem with regards to splitting fields of f over K we refer to the splitting field F of f over K.

Definition A.37. *Let n be a positive integer. The splitting field of $x^n - 1$ over a field K is called the n-th cyclotomic field over K and it is denoted $K^{(n)}$. The roots of $x^n - 1$ in $K^{(n)}$ are called the **nth roots of unity** over K and the set of all these roots is denoted $E^{(n)}$.*

Definition A.38. *A multiplicative group G is said to be **cyclic** if there is an element $a \in G$ such that for any $b \in G$ there is some integer j such that $b = a^j$. Such an element is called a generator of the cyclic group and we write $G = \langle a \rangle$.*

Theorem A.39. *Let n be a positive integer and K a field of characteristic p. Then:*

(i) If p does not divide n, then $E^{(n)}$ is cyclic group of order n with respect to the multiplication in $K^{(n)}$.

(ii) If p divides n, write $n = mp^e$ with positive integers m and e and with m not divisible with p. Then $K^{(n)} = K^{(m)}$, $E^{(n)} = E^{(m)}$, and the roots of $x^n - 1$ in $K^{(n)}$ are the m elements of $E^{(m)}$, each attained with multiplicity p^e.

Definition A.40. *Let K be a field of characteristic p and n a positive integer not divisible by p. Then a generator of the cyclic group $E^{(n)}$ is called a **primitive nth root of unity** over K.*

Definition A.41. *Let F be a field. A **vector space** over the field F is a set V together with two binary operations:*

(i) vector addition: $V \times V \to V$ denoted $\mathbf{v} + \mathbf{w}$, where$\mathbf{v}, \mathbf{w} \in V$

(ii) scalar multiplication: $F \times V \to V$ denoted $a\mathbf{v}$, where $a \in F$ and $\mathbf{v} \in V$.

satisfying the following axioms:

(i) closure under vector addition and scalar multiplication, i.e. $\mathbf{v} + \mathbf{w}$, $a\mathbf{v} \in V$, for any $\mathbf{v}, \mathbf{w} \in V$ and any $a \in F$.

(ii) $(V,+)$ group, where $+$ is the vector addition.

(iii) Distributivity of scalar multiplication with respect to vector addition, i.e. $a(\mathbf{v} + \mathbf{w}) = a\mathbf{v} + b\mathbf{w}$, for any $\mathbf{v}, \mathbf{w} \in V$ and any $a \in F$.

(iv) Distributivity of scalar multiplication with respect to field addition, i.e. $(a+b)\mathbf{v} = a\mathbf{v} + b\mathbf{v}$, *for any* $\mathbf{v} \in V$ *and any* $a, b \in F$.

(v) Compatibility of scalar and field multiplication, $a(b\mathbf{v}) = (ab)\mathbf{v}$, *for any* $\mathbf{v} \in V$ *and any* $a, b \in F$.

(vi) Identity element of scalar multiplication, i.e. if the multiplicative identity of F *is denoted* 1 *then* $1\mathbf{v} = \mathbf{v}$.

We denote a vector space V *over a field* F *with* $(V/F, +, \cdot)$.

Definition A.42. *A map from a vector space to another* $L : V_1/F \to V_2/F$ *is called a* ***linear operator*** *if it satisfies the following:*

(i) $L(\mathbf{v} + \mathbf{w}) = L(\mathbf{v}) + L(\mathbf{w})$, *for all* $\mathbf{v}, \mathbf{w} \in V_1$.

(ii) $L(a\mathbf{v}) = aL(\mathbf{v})$ *for all* $a \in F$ *and* $\mathbf{v} \in V_1$.