

M.Tech. (Computer Science) Dissertation Series

**Finding The Approximate Generator Polynomial And
Corresponding Initial Sequence Of A Given Binary
String Using Simulated Annealing**

*A dissertation submitted in partial fulfillment of the requirement for the
M.Tech. (Computer Science) degree of the Indian Statistical Institute*

By

Deepak Kumar Dalai

Under the supervision of

Dr. Subhamoy Maitra



INDIAN STATISTICAL INSTITUTE

203, Barrackpore Trunk Road

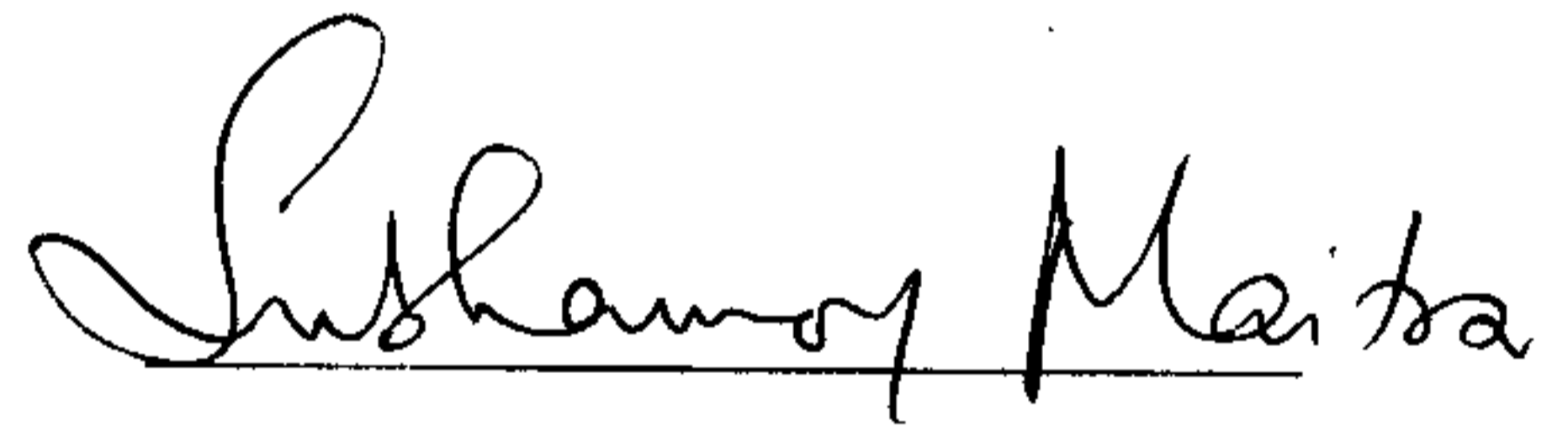
Kolkata-700108

2003

CERTIFICATE OF APPROVAL

This is to certify the thesis entitled '**Finding The Approximate Generator Polynomial And Corresponding Initial Sequence Of A Given Binary String Using Simulated Annealing**' submitted by **Deepak Kumar Dalai** towards partial fulfillment of requirements for the degree of M.Tech. in Computer Science at Indian Statistical Institute, Kolkata embodies the work done under my supervision.

Dated: The 10th of July, 2003.



Signed:

(Dr. Subhamoy Maitra)

Supervisor

Sugata Gangopadhyay
10/7/03

Countersigned:

External Examiner

Lecturer, BITS Pilani.

ACKNOWLEDGEMENT

I take this opportunity to express my deep sense of gratitude and indebtedness to my guide **Dr. Subhamoy Maitra** of Applied Statistical Unit of Indian Statistical Institute, Kolkata for this generous and whole hearted support to me in completing this project.

I wish to place on record my sincere thanks to all members of crypto group of Indian Statistical Institute, Kolkata for their huge help and cooperation for my project.

Lastly, I would like to express my profound gratitude to all other teachers of Indian Statistical Institute, Kolkata and sincere thanks to all of my classmates who were the source of inspiration during the M.Tech (Comp. Sc.) programme.

Deepak Kumar Dalai

Contents

1. Introduction	2
2. Comparison between the binary strings generated by two primitive polynomials	3
3. Implementation of Simulated Annealing	6
4. Finding the initial sequence using a fast technique	9
5. Experimental results	13
References	15
Appendix	
A. The basics of simulated Annealing	i
B. Berlekamp-massey Algorithm	ii

Finding the approximate generator polynomial and the corresponding initial sequence of given binary string using simulated annealing

Deepak Kumar Dalai
e-mail: *dk_dalai@rediffmail.com*

Abstract

In LFSR based cryptosystem binary sequence generated by a generator polynomial. Since our real channel is not noise free, at the receiver end the receiver gets corrupted data i.e. some bits altered due to noise. so, to decode the message we need the actual polynomial and the initial sequence. From which we will get back the generated sequence. Here, we try to obtain the original generator polynomial and the initial sequence using simulated annealing.

1 Introduction

When information is sent through channel, it is sent by encoding the information into binary strings. For security and correctness point of view the encoder uses LFSR(Linear Feedback Shift Register), in which the primitive polynomials are being used for generating linear codes and large non repeating binary string. But, in real life, we do not have noiseless channel. so, some data are being corrupted i.e. some bits are reversed(0 changes to 1 and vice versa). Since the primitive polynomials are being used to encode the data, there must be some information in corrupted data. The probability of alteration of a bit is less than or equal $\frac{1}{2}$ (if greater than $\frac{1}{2}$ then we alter all bits then it will be $\leq \frac{1}{2}$). Also, the channel is not much noisy. So, the probability of alteration of a bit is low. So, some bits will be altered in the sequence of bits generated by primitive polynomial. So, the hamming distance of corrupted data and actual sequence will be low. Whatever, to get the original message, we must have to get back the binary strings used to encryption, generated by the LFSR. Hence, to get original binary string we have to find out the primitive polynomial used in LFSR and the initial sequence(length of initial sequence is the degree of polynomial) used to generate the sequence of binary bits. Since, we expect the noise introduced is very low, the primitive polynomial which will give the sequence of binary bits which is minimum distance from the received binary string will be the required polynomial. The natural and convenient way to achieve it is to search on every primitive polynomial of that degree running on every nonzero initial sequence. This process is easy to feel, but is it feasible in our present system ? Since the length initial sequence is n for the polynomial of degree n , to get the minimum initial sequence we have to run on every nonzero sequence of length n which are $2^n - 1$ numbers. And there are $2^{n+1} - 1$ nos. nonzero polynomials and $\phi(\frac{2^n-1}{n})$ nos. of primitive polynomials. So, the complexity of such process will be around $O(n \times 2^{2n})$, Which is not feasible for the present system.

In case correlation attack, the cryptanalysts use some fast correlation algorithm [1] to get back the initial sequence of known generator polynomial. But, in this case we do not have any knowledge about the generator polynomial. So, we have to use some technique to get the generator polynomial which will generate the string which is minimum distance from the received string. We can use simulated annealing [2] for optimization purpose. By which, the generator polynomial and the initial sequence can be approximated in polynomial time complexity. So, we should have the algorithm to get the generate polynomial of a given binary sequence. For that we can use the technique for solving of a system of equations. But, we have a beautiful algorithm, Berlekamp-Massey algorithm [3], which gives the minimum degree polynomial.

In section 2 some experiment before using the simulated annealing is described. Implementation of simulated annealing on finding both generator polynomial and the initial sequence to the generator polynomial is discussed in section 3. To find the initial sequence, a fast technique is discussed in section 4. The experimental results obtained by computer simulation are presented in section 5. Reader unfamiliar with simulated annealing and Berlekamp-Massey algorithm can find a brief introduction in Appendix A and appendix B respectively.

2 The comparison between the binary strings generated by two primitive polynomials

Before going to implement the simulated annealing, it could be tested how the sequence generated by primitive polynomials differ to each other. So, in this section, the experimental results on comparison of the sequences generated by primitive polynomials and the initial sequence gives the minimum distance is discussed. At first, we can test for comparing the primitive polynomials of same degree then comparing a smaller one with a larger. So, we test the primitive polynomials of degree 5, 6 and 7. There are $\Phi\left(\frac{2^5-1}{5}\right) = 6$ primitive polynomials of degree 5. And these are

- i. $x^5 + 1 = 0$
- ii. $x^5 + x^2 + x + 1 = 0$
- iii. $x^5 + x^4 + x^2 + x + 1 = 0$
- iv. $x^5 + x^3 + 1 = 0$
- v. $x^5 + x^4 + x^3 + x^2 + 1 = 0$
- & vi. $x^5 + x^4 + x^3 + x + 1 = 0$

Where the polynomials (i), (ii), (iii) are reciprocals of (iv), (v), (vi) respectively and vice versa. Pair of reciprocal polynomials give the same sequence but in reverse order of same initial sequence. So, it is required to test first 3 polynomials. The minimum distance between sequences generated by any two polynomials from first three polynomials gives 12. Also, notice that the distance between any two sequences generated by two polynomials (except the reciprocal pairs) is always divisible by 4.

Similarly, there are $\Phi\left(\frac{2^6-1}{8}\right) = 6$ primitive polynomials of degree 6. And these are

- i. $x^6 + x + 1 = 0$
- ii. $x^6 + x^4 + x^3 + x + 1 = 0$
- iii. $x^6 + x^5 + x^2 + x + 1 = 0$
- iv. $x^6 + x^5 + 1 = 0$
- v. $x^6 + x^5 + x^3 + x^2 + 1 = 0$
- & vi. $x^6 + x^5 + x^4 + x + 1 = 0$

Where the polynomials (i), (ii), (iii) are reciprocals of (iv), (v), (vi) respectively and vice versa. So, we need to find the minimum distance between sequences generated by any pair of first 3 polynomials. Here, pair of polynomials (i), (ii) and (i), (iii) give minimum distance 24 each and the pair (ii), (iii) gives minimum distance 20.

Here, also notice that the distance between any two sequences generated by any two polynomials (except the reciprocal pairs) is multiple of 4.

It is sure that it must be multiple of 2, because of all sequence generated by them have equal number of 0s and 1s. But, why is it multiple of 4?

Some results testing on some primitive polynomial of degree 7 are below. The pair $x^7 + x + 1$ and $x^7 + x^3 + 1$ gives minimum distance 56.

The pair $x^7 + x + 1$ and $x^7 + x^3 + x^2 + x + 1$ gives minimum distance 56.

The pair $x^7 + x^3 + x^2 + x + 1$ and $x^7 + x^6 + x^5 + x^2 + 1$ gives minimum distance 52.

So, from these experiment on small polynomial we got that there is big difference between the sequences generated by two primitive polynomials of same degree. It will be higher for primitive polynomials of bigger degree. So, if some noise is introduced on a binary string generated by a primitive polynomial and to get the original polynomial we have find out the primitive polynomial which generate the closest binary string, because from above experiment we expect that no two primitive polynomial generate closer sequence. But, it could be possible the sequence generated by a large primitive polynomial will be near to the sequence generated by a small polynomial. So, it could be tested by a smaller primitive polynomial against the larger one i.e. a part of lager sequence is compared with smaller sequence. So, here we can take the large primitive polynomial(degree 20). Taking consecutive $2^5 - 1$, $2^6 - 1$ or $2^7 - 1$ bits of binary sequence randomly from the sequence generated by the large polynomial and test to find the minimum distance from the sequence generated by a primitive polynomial of degree 5 ,6 or 7 respectively.

Here, some primitive polynomial of degree 20 and these are

- i. $x^{20} + x^{15} + x^{13} + x^{12} + x^{11} + x^{10} + x^9 + x^8 + x^4 + x^2 + 1 = 0$
- ii. $x^{20} + x^{19} + x^{17} + x^{15} + x^{13} + x^{12} + x^{10} + x^9 + x^8 + x^5 + 1 = 0$
- iii. $x^{20} + x^{15} + x^{13} + x^{10} + x^6 + x^2 + 1 = 0$
- & iv. $x^{20} + x^{19} + x^{15} + x^{14} + x^{13} + x^{10} + x^7 + x^6 + 1 = 0$

Generating the sequence from a polynomial of above and taking a consecutive $2^5 - 1$ bits, the minimum distance from the sequences generated by a primitive polynomial of degree 5 give the maximum time the minimum distance is 8 or 10.

Similarly, taking consecutive $2^6 - 1$ bits and finding minimum distance from the sequences generated by a primitive polynomial of degree 6,observed minimum distance 22, 23, 24 in maximum time.

Similarly, taking consecutive $2^7 - 1$ bits and finding minimum distance from the sequences generated by a primitive polynomial of degree 7, observed minimum distance 46, 48, 50 in maximum time. So, it is observed that the smaller one also does not give fairly closer sequence to bigger one. But, it is better than the first experiment of same degree. Sometimes the smaller one gives very close sequence. So, we can use smaller polynomial for our purpose.

3 Implementation of Simulated Annealing

To find the approximate polynomial and the initial sequence in a polynomial time we can use a discrete optimization technique such as simulated annealing [2][4] with an appropriate optimization heuristic. In this section we outline a simple approach for applying simulated annealing to find the optimal polynomial and the optimal initial sequence. This approach is intended to enhance the basic exhaustive search as outlined in Section 1. Readers unfamiliar with the simulated annealing algorithm should study the Appendix A for a brief description.

First, a cost function (fitness measure or optimization heuristic) is required. Of course, the simple and ideal cost function would measure the Hamming distance from the received sequence to the proposed solution. We can use some other cost function like error rate, number of unsatisfied parity check bits [4]. But, here we use the simple cost function as hamming distance.

Since we have to find out both the optimum polynomial and the corresponding initial sequence to this optimization, we can implement simulated annealing twice. First simulation annealing is implemented on polynomials to find approximate polynomial. But to find approximate polynomial we must have to find out the appropriate initial sequence which will give the smallest hamming distance. So, to find the appropriate initial sequence we can use simulated annealing again to find the initial sequence. That is, using first simulated annealing we get a polynomial in each step, then finding the minimal initial sequence, the simulated annealing can be implemented again on initial sequence.

To use simulated annealing, it is necessary to initiate the configuration. Then implement the simulated annealing to get another configuration and repeat it on the present configuration till to get satisfied configuration or no. of loops. So, in this case we have to give initial configuration for both case. Then we have to implement simulated annealing on both.

At first, we consider how to initiate the configuration and implement the simulated annealing to find the initial sequence for a given polynomial of degree n . we require n bits for initial sequence. The hamming distance of first n bits in received string from the original initial sequence is less. We take first n bits of received sequence to initial configuration for the initial sequence. Now we consider about implementation of simulated annealing. Let P be the polynomial and $C_i, cost_i$ be configuration for initial sequence for polynomial P and cost for this configuration C_i after i^{th} iteration respectively, where cost is the hamming distance of generated sequence using the initial sequence from received sequence. In $i + 1^{st}$ iteration choose a bit in C_i randomly. Then alter that bit (i.e. if it is 0 make it 1 or if it is 1 make it 0) and assign to the configuration C_{i+1} . Use this

configuration C_{i+1} to generate the sequence of the same length of received sequence. Find out the $cost_{i+1}$ for configuration C_{i+1} . If this $cost_{i+1}$ is lesser than the previous cost $cost_i$ then go for next iteration. If $cost_{i+1}$ is greater than previous cost $cost_i$ then we will accept it randomly using some condition. This possibility of satisfaction of this condition should be very low, because the noise is very less and the exact configuration must be in the small neighbour of initial configuration. If this condition satisfied then go for next iteration else reassign to C_{i+1} as C_i i.e. $C_{i+1} \leftarrow C_i$. Repeat this till a polynomial time loop and take the configuration C_k which gives the minimum cost among the configurations during this process.

Then we consider how to initiate the configuration and implement the simulated annealing to find the polynomial. Since we require the polynomial of degree n or less than n , we implement the Berlekamp-Massey algorithm [3] on received sequence till get higher polynomial of degree n . To get polynomial of degree of n , we need at most $2n$ bits as a property of Berlekamp-Massey algorithm and the time complexity is $O(n^2)$. If there is no error in first $2n$ bits then we will get the original polynomial. So, we take this polynomial as the initial configuration. Here, we get initial configuration P_0 , now we consider about implementation of simulated annealing to find the next configuration. Let P_i be the i^{th} configuration after i iteration. And cost after i^{th} iteration be the $cost'_i$ is the cost returned implementing simulation annealing to get initial sequence for P_i as described above. Choose a number from $\{0, 1, \dots, n\}$ let it be k then alter the coefficient of x^k in configuration P_i (i.e. if coefficient is 0 make it 1 or if coefficient is 1 make it 0) and assign it to P_{i+1} . Then find out the cost $cost'_{i+1}$ for it implementing simulation annealing for finding initial sequence. If $cost'_{i+1} < cost'_i$ then go for next iteration. If $cost'_{i+1}$ is greater than previous cost $cost'_i$ then we will accept it randomly using some condition. This possibility of satisfaction of this condition should be very low. If this condition satisfied then go for next iteration else reassign to P_{i+1} as P_i i.e. $P_{i+1} \leftarrow P_i$. Repeat this till a polynomial time loop and take the configuration P_k which gives the minimum cost among the configurations during this process.

Hence, the required polynomial which gives minimum cost and the corresponding initial sequence will be the initial sequence. But, this process does give good result. The returned minimum cost is not low every time.

Algorithm .

Data: A binary sequence of length N.

Target: The polynomial of degree n and corresponding initial sequence.

Procedure(to find polynomial and initial sequence):

```
{
  step 1: Initialize configuration for polynomial using Berlekamp-Massey
  algorithm to get a polynomial of degree n.
  step 2: Implement simulated annealing to find the minimal initial
  sequence {
    step 2.1: Initialize configuration for initial sequence taking first
    n bits.
    step 2.2: Choose a random position in between 1 and n then
    alter the bit at that position.
    step 2.3: If that new configuration gives the lesser cost(hamming
    distance from received sequence) than previous cost go to step
    2.2 with new configuration.
    else generate a random number in between 0 and 1 if it is very
    low ( $< 0.1 \times e^{-(\text{distancediff}/\text{cost})}$ ) then go to step 2.2 with new
    configuration else go to step 2.2 with previous configuration .
    step 2.4: Repeat it till a polynomial time loop or satisfy a cost.
  }
  step 3: Find a position randomly in between 1 and n say t and alter
  the coefficient of  $x^t$  of polynomial.
  step 4: If that new configuration gives the lesser cost(hamming
  distance from received sequence) than previous cost go to step 2 with
  new configuration.
  else generate a random number in between 0 and 1 if it is very low
  ( $< 0.2 \times e^{-(\text{distancediff}/\text{cost})}$ ) then go to step 2 with new configuration
  else go to step 2 with previous configuration .
  step 5: Repeat it till a polynomial time loop or satisfy a cost.
}
```

4 Finding initial sequence using a fast technique

Implementing the above technique (simulated annealing on both polynomial and initial sequence) it is observed that it does not give good result (in section 5). Because the error occurs twice, since simulated annealing implemented twice. The error occurred in finding initial sequence is believed as errorless for cost calculation for finding polynomial. So, more error occurs. If we will be able to feed actual initial sequence (i.e. the initial sequence which gives the minimum distance) then implementing the simulated annealing only for finding the polynomial, it must be better. So, here we purpose to implement simulating annealing to find polynomial only. The initial sequence for the polynomial at any iteration we will use a fast technique, which is used for fast correlation attack [1]. How to get the initial sequence of a given polynomial from a erroneous sequence is described below.

Let P be the polynomial of degree l and the received sequence be z_1, z_2, \dots, z_N of length N . Now to find out the initial sequence x_1, x_2, \dots, x_l which will give the minimum distance for P from the z_1, z_2, \dots, z_N .

Here an LFSR output sequence having some fixed length N can be regarded as a binary linear $[N, l]$ -code, where l is the degree of the feedback polynomial of the target LFSR. The number of codewords equals the number of initial states of LFSR, that is 2^l . Thus, this problem can be reformulated as a decoding problem of this particular code in the presence of noise. The problem is how to decode this linear $[N, l]$ -code with as low decoding complexity as possible.

Associate with the target LFSR another binary linear $[n_2, k]$ -code with $k < l$. The k information symbols of this code may coincide with the first k symbols of the initial state of the LFSR we want to recover. In the decoding step, the symbols of the observed sequence are combined according to the parity checks, and the probability of each codeword in the code is calculated, i.e., ML-decoding (Maximum Likelihood decoding). Since the new code has dimension k the decoding complexity decreases from $O(l \times 2^l)$ to $O(k \times 2^k)$.

Let the target LFSR have length l and let the set of possible LFSR sequences be denoted by \mathcal{L} . Clearly, $|\mathcal{L}| = 2^l$ and for a fixed length N the truncated sequences from \mathcal{L} form a linear $[N, l]$ block code referred to as \mathcal{C} . Furthermore, the observed keystream sequence $z = z_1, z_2, \dots, z_N$ is regarded as the received channel output and the LFSR sequence $x = x_1, x_2, \dots, x_N$ is regarded as a codeword from an $[N, l]$ linear block code. we can describe each z_i as the output of the binary symmetric channel, BSC, when x_i was transmitted.

Let us briefly recall some results from coding theory. Since each symbol x_i is a linear combination of the l initial values we see that the set of words (x_1, x_2, \dots, x_N) forms the linear code C and we call these words *codewords*. The word (x_1, x_2, \dots, x_l) is called the *information word* and the symbols x_1, x_2, \dots, x_l are called *information symbols*. We have that

$$\begin{cases} c_0x_1 + c_1x_2 + \dots + c_lx_{l+1} = 0 \\ c_0x_2 + c_1x_3 + \dots + c_lx_{l+2} = 0 \\ \dots \\ c_0x_{N-l+1} + c_1x_{N-l+2} + \dots + c_lx_N = 0 \end{cases} \quad (1)$$

where c_i are coefficients of the generator polynomial $g(D) = c_0 + c_1D + \dots + c_lD^l$ ($c_0 = c_l = 1$). We define the $(N-l) \times N$ matrix

$$H = \begin{pmatrix} c_0 & c_1 & c_2 & \dots & c_l & 0 & \dots & 0 \\ 0 & c_0 & c_1 & \dots & c_l & c_0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & c_l & c_{l-1} & \dots & c_1 & c_0 \end{pmatrix}.$$

Thus, our code consists of all codewords such that $H(x_1, x_2, \dots, x_N)^T = 0$. The matrix H is called the parity check matrix of the LFSR code. It follows from (1) that

$$\begin{cases} x_{l+1} = h_{l+1}^1x_1 + h_{l+1}^2x_2 + \dots + h_{l+1}^lx_l \\ x_{l+2} = h_{l+2}^1x_1 + h_{l+2}^2x_2 + \dots + h_{l+2}^lx_l \\ \dots \\ x_i = h_i^1x_1 + h_i^2x_2 + \dots + h_i^lx_l \\ \dots \\ x_N = h_N^1x_1 + h_N^2x_2 + \dots + h_N^lx_l = 0 \end{cases} \quad (2)$$

If we denote $h_i(D) = h_i^1 + h_i^2D + \dots + h_i^lD^{l-1}$, then clearly $h_i(D) = D^{i-1} \bmod g(D)$ for $i = 1, 2, \dots, N$.

Therefore the code C has the $(l \times N)$ -generator matrix

$$G = \begin{pmatrix} h_1^1 & h_1^2 & \dots & h_1^l \\ h_2^1 & h_2^2 & \dots & h_2^l \\ \dots & \dots & \dots & \dots \\ h_l^1 & h_l^2 & \dots & h_l^l \end{pmatrix} \quad (3)$$

The observed output sequence, which we call the received word and denote $z = (z_1, z_2, \dots, z_N)$, is regarded as a noisy version of the unknown codeword (x_1, x_2, \dots, x_N) . Now, we have to find out the initial sequence (x_1, x_2, \dots, x_l) from which we will be able to get (x_1, x_2, \dots, x_N) . We use ML-decoding (Maximum Likelihood decoding) for decoding.

Let $\mathcal{C} = \{x\}$ be an (n, k) -code and let \bar{x} be the transmitted codeword and let z be the received word. Now let x_0 be the code such that

$$\text{dist}(x_0, z) = \min_{x \in \mathcal{C}} \text{dist}(x, z).$$

Here $\text{dist}(x, y)$ denotes the Hamming distance between x and y , i.e., the number of ones in the binary vector $x + y$. This x_0 is the decoded code using ML-decoding. Now we describe the fast technique to get initial sequence (x_1, x_2, \dots, x_l) . Let $k \leq l$ be fixed. We shall describe a procedure that recovers the symbols x_1, x_2, \dots, x_k when observing z_1, z_2, \dots, z_N . In the system (2) we look for pairs of equations such that,

$$h_i^{k+1} = h_j^{k+1}, h_i^{k+2} = h_j^{k+2}, \dots, h_i^l = h_j^l, 1 \leq i \neq j \leq N. \quad (4)$$

We find all such pairs of equations. Let the number of such distinct pairs be n_2 . Denote the indices of all such pairs as $\{i_1, j_1\}, \{i_2, j_2\}, \dots, \{i_{n_2}, j_{n_2}\}$.

If i and j satisfies (4), then the sum $x_i + x_j$ is a linear combination of information symbols x_1, x_2, \dots, x_k only, and is independent of the remaining information symbols $x_{k+1}, x_{k+2}, \dots, x_l$,

$$x_i + x_j = (h_i^1 + h_j^1)x_1 + (h_i^2 + h_j^2)x_2 + \dots + (h_i^k + h_j^k)x_k. \quad (5)$$

This means that the sequence

$$(X_1, X_2, \dots, X_{n_2}) = (x_{i_1} + x_{j_1}, x_{i_2} + x_{j_2}, \dots, x_{i_{n_2}} + x_{j_{n_2}})$$

forms an (n_2, k) -code, referred to as \mathcal{C}_2 , whose information symbols are (x_1, x_2, \dots, x_k) , i.e., it has dimension k . The generator matrix of the (n_2, k) -code \mathcal{C}_2 is

$$G_2 = \begin{pmatrix} h_{i_1}^1 + h_{j_1}^1 & h_{i_2}^1 + h_{j_2}^1 & \dots & h_{i_{n_2}}^1 + h_{j_{n_2}}^1 \\ h_{i_1}^2 + h_{j_1}^2 & h_{i_2}^2 + h_{j_2}^2 & \dots & h_{i_{n_2}}^2 + h_{j_{n_2}}^2 \\ \dots & \dots & \dots & \dots \\ h_{i_1}^k + h_{j_1}^k & h_{i_2}^k + h_{j_2}^k & \dots & h_{i_{n_2}}^k + h_{j_{n_2}}^k \end{pmatrix}. \quad (6)$$

We denote

$$Z_1 = z_{i_1} + z_{j_1}, Z_2 = z_{i_2} + z_{j_2}, \dots, Z_{n_2} = z_{i_{n_2}} + z_{j_{n_2}}. \quad (7)$$

Since we observe the output symbols $(z_{i_1}, z_{j_1}, z_{i_2}, z_{j_2}, \dots, z_{i_{n_2}}, z_{j_{n_2}})$ we can calculate also $(Z_1, Z_2, \dots, Z_{n_2})$, that is, a word acting as a received word for \mathcal{C}_2 . If (e_1, e_2, \dots, e_N) is the noise sequence of the code \mathcal{C} , $(e_i = x_i + y_i)$, then clearly the noise sequence $(E_1, E_2, \dots, E_{n_2})$ for \mathcal{C}_2 is

$$E_1 = e_{i_1} + e_{j_1}, E_2 = e_{i_2} + e_{j_2}, \dots, E_{n_2} = e_{i_{n_2}} + e_{j_{n_2}}.$$

Since our model is the BSC, all the e_i 's are independent random binary random variables with error probability p . It is then clear that $E_m = e_{i_m} + e_{j_m}$ for $i_m \neq j_m$ are also independent binary random variables for all $1 \leq m \leq n_2$ with error probability

$$p_2 = Pr(e_{i_m} + e_{j_m} = 1) = 2p(1 - p) > p.$$

Thus, we have created a new code \mathcal{C}_2 with smaller dimension but the BSC over which the codeword is transmitted has a stronger noise p_2 , i.e., $p_2 > p$. To restore the symbols x_1, x_2, \dots, x_k we have to decode the $[n_2, k]$ -code \mathcal{C}_2 in the BSC with the stronger noise p_2 . However, as long as the length of the new code \mathcal{C}_2 guarantees unique decoding, this new code will be decoded significantly faster than the LFSR code \mathcal{C} . As before, we apply a simple ML-decoding procedure when decoding \mathcal{C}_2 .

Algorithm .

Data: the length l of target LFSR, and the generator polynomial $g(D)$.

Precomputation.

Fix a computational complexity level by choosing a $k < l$ (for example $k = l/2$). Construct the generator matrix G (see (3)). Using a sorting algorithm, sort the columns of G with respect to

$$(h_i^{k+1}, h_i^{k+2}, \dots, h_i^l), i = 1, 2, \dots, N.$$

Find all pairs $\{i, j\}$ that satisfy (5). For each pair, store the indices i, j together with the value of $(h_i^1 + h_j^1, h_i^2 + h_j^2, \dots, h_i^k + h_j^k)$. Hence, we have constructed the code \mathcal{C}_2 with generator matrix G_2 (see (6)).

Decoding.

Input: The received (observed) vector (z_1, z_2, \dots, z_N) .

Step 1. Compute $(Z_1, Z_2, \dots, Z_{n_2})$ (see (7)).

Step 2. Decode the code C_2 with the generator matrix (6) using exhaustive search through all the 2^k codewords of C_2 , and select the information word (x_1, x_2, \dots, x_k) with highest probability.

After having restored (x_1, x_2, \dots, x_k) we need to restore the remaining part of the initial state, $(x_{k+1}, x_{k+2}, \dots, x_l)$. An obvious way would be to repeat the proposed procedure for some other information bits, say $(x_{k+1}, x_{k+2}, \dots, x_{2k})$. We can use the same parity checks, since the code is cyclic.

However, with knowledge of the first k information symbols, the remaining problem is much simplified compared to the original problem. Hence we can discard the complexity and the error probability of this step. (E.g. if we restore the 20 first information bits of a length 80 LFSR, we use the obtained values of these 20 first information bits and get a new decoding problem but now only for a length 60 LFSR.)

5 Experimental results

In this section we discuss some implementations simulated annealing discussed in section (3). At first we generate sequence of some length feeding an initial sequence to a primitive polynomial. Then we introduce some noise randomly on some bits i.e. choosing some places randomly and alter the bits. Then we get corrupted sequence. Now we have to implement the simulated annealing to find out the approximate polynomial and corresponding initial sequence.

We tested for some polynomials of primitive polynomials degree 20 and the results are written below.

Experiment 1.

Here, we took the primitive polynomial

$$x^{20} + x^{15} + x^{13} + x^{12} + x^{11} + x^{10} + x^9 + x^8 + x^4 + x^2 + 1 = 0$$

then we generate a sequence of length 100 giving initial sequence (10011100001101100010). Randomly we change 12 bits in the sequence. Then we try to find a polynomial of degree 20. we got a polynomial $x^{20} + x^{19} + x^{15} + x^{13} + x^{11} + x^6 + x^4 + x^3 + 1$ and initial sequence (11011100001101110000) which is 28 distance apart from received sequence. Then I try to approximate by a polynomial of degree $\lceil \log_2 100 \rceil = 7$. we got a polynomial $x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + 1$ and initial sequence (0000110) which gives cost 31.

Experiment 2.

Here, we took the primitive polynomial

$$x^{20} + x^{19} + x^{17} + x^{15} + x^{13} + x^{12} + x^{10} + x^9 + x^8 + x^5 + 1 = 0$$

then we generate a sequence of length 100 giving initial sequence (11001101000010111010). Randomly we change 9 bits in the sequence. Then we try to find a polynomial of degree 20. we got a polynomial $x^{20} + x^{15} + x^{14} + x^{11} + x^9 + x^7 + x^5 + x^2 + 1$ and initial sequence (11001101000010111100) which is 29 distance apart from received sequence. Then I try to approximate by a polynomial of degree $\lceil \log_2 100 \rceil = 7$. we got a polynomial $x^7 + x^4 + x^3 + x^2 + 1$ and initial sequence (0110100) which gives cost 33.

Experiment 3.

Here, we took the primitive polynomial

$$x^{20} + x^{15} + x^{13} + x^{10} + x^6 + x^2 + 1 = 0$$

then we generate a sequence of length 100 giving initial sequence (00111011000011100100). Randomly we change 14 bits in the sequence. Then we try to find a polynomial of degree 20. we got a polynomial $x^{20} + x^{18} + x^{17} + x^{16} + x^{14} + x^{12} + x^9 + x^8 + x^7 + x^6 + x^4 + x + 1$ and initial sequence (00100010000101101000) which is 29 distance apart from received sequence. Then I try to approximate by a polynomial of degree $\lceil \log_2 100 \rceil = 7$. we got a polynomial $x^7 + x^6 + x^5 + x^4 + x^2 + x + 1$ and initial sequence (1010001) which gives cost 34.

Experiment 4.

Now we test for bigger polynomial of degree 25. Here, we took the primitive polynomial

$$x^{25} + x^{22} + x^{19} + x^{16} + x^{14} + x^{11} + x^{10} + x^7 + x^6 + x + 1 = 0$$

then we generate a sequence of length 200 giving initial sequence (1110010011000010111011011). Randomly we change 32 bits in the sequence. Then we try to find a polynomial of degree 25. we got a polynomial which is 69 distance apart from received sequence. Then I try to approximate by a polynomial of degree $\lceil \log_2 200 \rceil = 8$. we got a polynomial $x^8 + x^3 + 1$ and initial sequence (11001000) which gives cost 77.

So, we are not getting good approximate implementing simulated annealing.

References

- [1] Vladimir V. Chepyzhov, Thomas Johansson, Ben Smeets, "A Simple Algorithm for Fast Correlation Attacks on Stream Ciphers", *Electronic Edition (Springer LINK)-181-195*.
- [2] Van Laarhoven P.J.M., Aarts E.H.L., "Simulated Annealing: Theory and Applications", D. Reidel Publishing Company, 1987. A text book for Simulated Annealing.
- [3] Guang Gong, "Sequence Analysis", *Lecture Notes for CO739X, Winter 1999*.
- [4] Jovan Dj. Golic, Mahmoud Salmasizadeh, Andrew Clark, Abdollah Khodkar and Ed Dawson, "Discrete Optimization and Fast Correlation Attacks".

Appendix

A. The Basics of Simulated Annealing

Simulated Annealing is a combinatorial optimization technique first introduced in 1983 by Kirkpatrick, Gelatt and Vecchi. They used the technique for deciding the optimal placement of components on an integrated circuit (IC) chip. The number of variables in such a problem can be huge, making determination of an optimal solution impossible. Simulated annealing scans a small area of the solution space in the search for the global minimum.

The Metropolis Algorithm

Simulated Annealing utilizes a process known as the Metropolis Algorithm which is based on the equations governing the movement of particles in a gas or liquid between different energy states. Equation (2) describes the probability of a particle moving between two energy levels, E_1 and E_2 , $\Delta E = E_2 - E_1$,

$$P(E) = e^{-\frac{\Delta E}{kT}} \quad (1)$$

where $\Delta E = E_2 - E_1$, k is Boltzmann's constant and T is the temperature. Equation (2) is used to make a decision as to whether or not a transition between different states should be accepted. The Metropolis Algorithm is based on

$$P(E) = \begin{cases} 1 & \text{if } \Delta E \geq 0 \\ e^{-\frac{\Delta E}{kT}} & \text{if } \Delta E < 0 \end{cases} \quad (2)$$

By allowing a configuration to move to a higher energy level (or higher cost), the search for the global minimum of the cost function is aided, since it is possible to move out of the regions of local minima. This is not the case for the so-called "iterative improvement" techniques which will only update the solution if a better (more optimal) one is found, or, in other words, if $\Delta E > 0$.

The Simulated Annealing Algorithm

1. Generate an initial solution to the problem (usually random).
2. Calculate the cost of the initial solution.
3. Set the initial temperature $T = T^{(0)}$.
4. For temperature, T , do many times:

- Generate a new solution - this involves modifying the current solution in some manner.
 - Calculate the cost of the modified solution.
 - Determine the difference in cost between the current solution and the proposed solution.
 - Consult the Metropolis Algorithm to decide if the proposed solution should be accepted.
 - If the proposed solution is accepted, the required changes are made to the current solution.
5. If the stopping criterion is satisfied the algorithm ceases with the current solution, otherwise the temperature is decreased and the algorithm returns to step 4.

B. Berlekamp-Massey algorithm

The *Berlekamp-Massey algorithm* is an algorithm for determining the linear complexity of a finite sequence and the feedback polynomial of a linear feedback shift register (LFSR) of minimal length which generates this sequence. This algorithm is due to Massey [Mas69], who showed that the iterative algorithm proposed in 1967 by Berlekamp [Ber67] for decoding BCH codes can be used for finding the shortest LFSR that generates a given sequence.

For a given sequence s^n of length n , the BerlekampMassey performs n iterations. The t^{th} iteration determines an LFSR of minimal length which generates the first t digits of s^n . The algorithm can be described as follows.

Input. $s^n = s_0s_1 \cdots s_{n-1}$, a sequence of n elements of F_q .

Output. Λ , the linear complexity of s^n and P , the feedback polynomial of an LFSR of length Λ which generates s^n .

Initialization.

$$P(X) \leftarrow 1, P'(X) \leftarrow 1, \Lambda \leftarrow 0, m \leftarrow 1, d' \leftarrow 1.$$

For t **from** 0 **to** $n - 1$ **do**

$$d \leftarrow s_t + \sum_{i=1}^{\Lambda} p_i s_{t-i}.$$

If $d \neq 0$ **then**

$$T(X) \leftarrow P(X).$$

$$P(X) \leftarrow P(X) - d(d')^{-1}P'(X)X^{t-m}.$$

if $2\Lambda \leq t$ then
 $\Lambda \leftarrow t + 1 - \Lambda.$
 $m \leftarrow t.$
 $P'(X) \leftarrow T(X).$
 $d' \leftarrow d.$

Return Λ and P .

In the particular case of a binary sequence, the quantity d' does not need to be stored since it is always equal to 1. Moreover, the feedback polynomial is simply updated by $P(X) \leftarrow P(X) + P'(X)X^{t-m}$.

The number of operations performed for computing the linear complexity of a sequence of length n is $O(n^2)$.

It is worth noticing that the LFSR of minimal length that generates a sequence s^n of length n is unique if and only if $n \geq 2\Lambda(s^n)$, where $\Lambda(s^n)$ is the linear complexity of s^n .

Example: The following table describes the successive steps of the Berlekamp-Massey algorithm applied to the binary sequence of length 7, $s_0 \cdots s_6 = 0111010$. The

t	s_t	d	Λ	$P(X)$	m	$P'(X)$
			0	1	-1	1
0	0	0	0	1	-1	1
1	1	1	2	$1 + X^2$	1	1
2	1	1	2	$1 + X + X^2$	1	1
3	1	1	2	$1 + X$	1	1
4	1	0	2	$1 + X$	1	1
5	0	1	4	$1 + X + X^4$	5	$1 + X$
6	0	0	4	$1 + X + X^4$	5	$1 + X$

values of Λ and P obtained at the end of step t correspond to the linear complexity of the sequence $s_0 \cdots s_t$ and to the feedback polynomial of an LFSR of minimal length that generates it.

The linear complexity $\Lambda(s)$ of a linear recurring sequence $s = (s_t)_{t \geq 0}$ is equal to the linear complexity of the finite sequence composed of the first n terms of s for any $n \geq \Lambda(s)$. Thus, the Berlekamp-Massey algorithm determines the shortest LFSR that generates an infinite linear recurring sequence s from the knowledge of any $2\Lambda(s)$ consecutive digits of s .