

GENERATING FIBONACCI NLFSRS WITH OPTIMAL PERIODS

BY

MAHDI SALEH HASSAN AL-IBRAHIM

A Thesis Presented to the

DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

INFORMATION ASSURANCE AND SECURITY

MAY 2022

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **MAHDI SALEH HASSAN AL-IBRAHIM** under the direction of his thesis adviser and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN INFORMATION ASSURANCE AND SECURITY**.

Thesis Committee



Dr. Sultan Almuhammadi (Adviser)

 25/12/2022

Dr. Mohammad Alshayeb (Member)



Dr. Farag Azzedin (Member)



Dec 25, 2022

Dr. Hamoud Aljamaan
Department Chairman



Dr. Suliman Al-Homidan
Dean of Graduate Studies

Date



©Mahdi Saleh Hassan Al-Ibrahim
2022

Dedication

To you dear reader, without whom, this work has little to no value.

ACKNOWLEDGMENTS

First and foremost, thanks to Allah the creator of all.

Then, thanks to my advisor Dr.Sultan and to my committee members, Dr. AlShayeb, and Dr. Farag for their constant support and understanding. I'm lucky to have them in my team. Thanks to KFUPM, I'll always be a KFUPMer... But more importantly, sorry to my wife, daughter and family, for that I kept sacrificing my time with them. Perhaps these sacrifices, could lead to something great one day...

TABLE OF CONTENTS

ACKNOWLEDGMENTS	v
LIST OF TABLES	x
LIST OF FIGURES	xi
LIST OF SYMBOLS	xiii
LIST OF ABBREVIATIONS	xiv
ABSTRACT (ENGLISH)	xv
ABSTRACT (ARABIC)	xvii
CHAPTER 1 INTRODUCTION	1
1.1 Feedback Shift Registers	1
1.2 Properties of NLFSR	4
1.2.1 Feedback Function Properties	4
1.2.2 NLFSR's Output Properties	8
1.3 NLFSR's Period	8
1.3.1 Optimal Period	9
1.3.2 Optimal Feedback Function	11
1.4 Feedback Function Types and Families	16
1.5 Thesis Structure	19

CHAPTER 2	LITERATURE REVIEW AND MOTIVATION	20
2.1	Literature Review	20
2.1.1	Finding All Optimal Functions For an FFT	20
2.1.2	Utilizing Hardware Acceleration	22
2.1.3	SAT Short Period Exploit	24
2.1.4	Galois Configured NLFSRs	26
2.2	Motivation	28
CHAPTER 3	THE PERIOD-TESTING SUIT	29
3.1	Suit Use Cases	29
3.2	Programming Tools	30
3.3	Suit Data Objects	31
3.3.1	Fib-Family	31
3.3.2	Fib-Function	32
3.3.3	Run-Options	35
3.4	How it Works	36
3.4.1	Period Tester	37
3.4.2	Next Function Mapping	38
3.4.3	X Wires	39
3.5	Proof of Correctness	40
CHAPTER 4	THE EXPERIMENT	43
4.1	Experiment's Objectives	43
4.2	Experiment Methodologies	44
4.2.1	Programming Methodology	44
4.2.2	Data Presentation Methodology	45
4.2.3	Data Collection Methodology	47
4.2.4	Data Analysis Methodology	47
4.3	Experiment Plan	48
4.4	Experiments Description	49
4.4.1	Series 10-wires	50

4.4.2	Series min-max	50
4.4.3	Series max-min	52
4.4.4	Series max-max	53
4.4.5	Series min-mins	53
4.4.6	Series full-stages	54
4.4.7	Series Remarks	55
CHAPTER 5 RESULTS AND FINDINGS		56
5.1	List of Findings	56
5.1.1	Finding 1: Optimal Type Hypothesis	57
5.1.2	Finding 2: O-Slide Hypothesis	57
5.1.3	Finding 3: X-Gab Hypothesis	59
5.1.4	Finding 4: Normal Distribution Pattern	60
5.1.5	Finding 5: Exponential Decay Pattern	61
5.1.6	Finding 6: Almost-O-Slide Pattern	62
5.1.7	Finding 7: Empty x Closer to Edges Pattern	64
5.1.8	Finding 8: Increasing Degree Pattern	65
5.1.9	Finding 9: Increasing Product Terms Pattern	65
5.1.10	Finding 10: Dense FFFs	66
5.2	Limitations	68
5.3	RQs & RAs	69
CHAPTER 6 THE CONCLUSION		71
6.1	Thesis Work	71
6.2	The Work's Impact	72
6.3	Future Work	72
6.4	Final Words	73
APPENDIX A PSEUDO CODES		75
APPENDIX B TABLES		80

APPENDIX C FIGURES	85
REFERENCES	88
VITAE	91

LIST OF TABLES

1.1	Feedback functions that are members of $FFF = (1, [2]), 4.$	18
2.1	Comparison between this work and [1] [2] [3] with regard to scope and results.	22
2.2	Comparison between this work and [4] [5] [6] [7] with regard to period testing methodology, feedback function features, and the count of optimal functions found. Table 2.3 better compares the count of optimal functions with respect to registers of sizes 22 to 31.	23
2.3	Comparison between this work and [4] [5] [6] [7] with regard to optimal NLFSRs found in each study for n -bit registers with sizes $(22 \leq n \leq 31)$.	25
5.1	First 20 elements of the density list for FFFs with register size $n = 18$.	67
B.1	Series min-max: (1-max-max) & (2-max).	80
B.2	Series max-min.	81
B.3	Series max-max.	82
B.4	Series min-mins.	82
B.5	Series full-stages.	83
B.6	Empty xs closer to edges pattern.	84

LIST OF FIGURES

1.1	An n -bit Fibonacci type FSR.	2
1.2	A complete logic gate representation of an NLFSR with feedback function equal to equation 1.1.	6
1.3	Illustration of a branching in NLFSR states.	12
3.1	Detailed NLFSR representation with feedback function equal to equation 3.1.	34
3.2	A macro level perspective of the period testing algorithm highlighting the major steps of the algorithm.	38
4.1	An illustration of the steps and components clarifying the methodology used in this research.	45
5.1	A group of feedback functions forming an O-slide.	58
5.2	A group of feedback functions highlighting the X-gab pattern.	61
5.3	Almost-O-slide pattern. Visual form analysis.	63
5.4	Almost-O-slide pattern. Compact form analysis.	64
C.1	Count of optimal functions in FFF: $(1, [2, 2]), n$; where $3 \leq n \leq 24$. Forms a rough <i>normally distributed</i> graph.	85
C.2	Count of optimal functions in FFF: $(2, [3]), n$; where $4 \leq n \leq 24$. Forms a rough <i>normally distributed</i> graph.	86
C.3	Probability percentage of finding optimal period in FFF: $(1, [2, 2]), n$; where $3 \leq n \leq 24$. Forms a rough <i>exponentially decaying</i> graph.	86

C.4	Probability percentage of finding optimal period in FFF: $(2, [3]), n$; where $4 \leq n \leq 24$. Forms a rough <i>exponentially decaying</i> graph.	87
-----	---	----

LIST OF SYMBOLS

$\oplus$	XOR or Addition operation: returns 1 if only one operand is 1 else 0
$\cdot$	AND or multiplication operation: returns 1 if all operands are 1 else 0
ρ	The minimum possible register size for a given feedback function type

LIST OF ABBREVIATIONS

CSPRN	Cryptographically secure pseudorandom number
CSPRNG	Cryptographically secure pseudorandom number generator
FFF	Feedback Function Family
FFT	Feedback Function Type
FSR	Feedback Shift Register
LFSR	Linear Feedback Shift Register
NLFSR	Non-Linear Feedback Shift Register

THESIS ABSTRACT

NAME: Mahdi Saleh Hassan Al-Ibrahim
TITLE OF STUDY: Generating Fibonacci NLFSRs with Optimal Periods
MAJOR FIELD: Information Assurance and Security
DATE OF DEGREE: May 2022

Recent studies introduced a new exploitation technique that affect some popular stream ciphers. Affected ciphers can be compromised via carrying an active attack targeting the cipher's Non-Linear Feedback Shift Register (NLFSR), forcing it into a short period. Grain-80, Grain-128, Bivium and Trivium are some of those affected stream ciphers [8] [9]. These findings opened new ways to crypt-analyse NLFSRs. Till this day, there is a weakness in the established mathematical foundation of NLFSR. Specifically, there is insufficient research for how to build an NLFSR that generate an optimal period [1] [7]. This knowledge deficiency contributed to the recent discovery of the short NLFSR period exploit in stream ciphers. A notable effort attempting to improve our understanding of NLFSRs period optimality is the work in [1] [2] [3]. The goal of this work is to push these efforts forward by expanding the scope of those

studies, attempting to formulate a mathematical model. This work adds valuable knowledge to the field of cryptography, especially valuable to stream cipher designers.

Presented in this work, a hypothesis to verify and generating all possible non-linear Fibonacci configured feedback function *families* that can have optimal period functions. Specifically, it's built a flexible and highly efficient NLFSR Period-Testing suit. It allowed running a large experiment which resulted in discovering over 90 million optimal feedback functions. The database of optimal functions were analyzed, formulating a hypothesis that can identify all possible optimal feedback function *families*. Furthermore, it's identified multiple rules and patterns that can significantly help in searching for feedback functions with optimal periods. Finally, The Period-Testing suit as well as the experiment data is made available publicly.

ملخص الرسالة

الاسم: مهدي صالح حسن آل إبراهيم

عنوان الدراسة: إنتاج أنواع لسجلات الإزاحة ذات التغذية غير الخطية من نمط فيوناتشي التي يمكنها توليد مخارج ذات فترات مثلى

التخصص: أمن المعلومات

تاريخ الدرجة العلمية: ماي 2022

أظهرت عدة دراسات حديثة طريقة جديدة لاختراق شفرات التعمية التدفقية. يمكن اختراق شفرات التعمية التدفقية المتضررة من خلال إجبار (سجلات الإزاحة ذات التغذية غير الخطية) المستخدمة في الشفرة بالوقوع في بذرة سيئة. شفرات (Grain-80) و (Grain-128) و (Bivium) و (Trivium) من ضمن الشفرات المتضررة [8] [9]. هذه الدراسات تعيد بتذكيرنا بنقص مفهومنا الرياضي لما يجعل (سجلات الإزاحة ذات التغذية غير الخطية) تنتج تسلسل ذو فترة مثلى. هذا النقص في المفهوم الرياضي ساعد في ولادة طريقة الاختراق الجديدة المذكورة. الأعمال البحثية في [1] [2] [3] ساهمت في سد هذا الفراغ المعرفي. هذا العمل يهدف لاستكمال هذه الجهود المبذولة من خلال توسيع نطاق البحث عن الدوال ذات الفترات المثلى بالإضافة إلى محاولة إثراء مفهومنا الرياضي في هذا المجال. يضيف هذا العمل كم معرفي لما يجعل (سجلات الإزاحة ذات التغذية غير الخطية) تنتج تسلسل ذو فترة مثلى. بالخصوص، كمخارج لهذا البحث، تم تطوير برنامج يمكنه أن يوجد الفترة المثلى لـ (سجلات الإزاحة ذات التغذية غير الخطية) بسرعة فائقة. مكن هذا البرنامج ذو الكفاءة العالية من إنجاز تجربة ضخمة. دراسة مخرجات التجربة أنتج عنه ملاحظة بعض الخصائص الرياضية. هذه الملاحظات يشكلون جزء كبيراً من مساهماتي في هذه الأطروحة. كما أنه من ضمن مساهماتي، مشاركة البرنامج و البيانات التي مكنت من إيجاد الخصائص الرياضية للعامة.

CHAPTER 1

INTRODUCTIONS

Cryptographically secure pseudorandom numbers (CSPRNs) play a significant role in cryptography. A cryptographic application that uses CSPRN needs to have a Cryptographically secure pseudorandom number generator (CSPRNG). A CSPRN is a number picked randomly from a pool of numbers with equally likely probability. A fair dice roll or a coin flip are examples for a CSPRNG. To an adversary, all possible values of a CSPRN are equally likely, regardless of context. NLFSRs can be used to generates a bit sequence that is suitable to be used in cryptographic applications. NLFSRs are cost effective, has good randomness properties, and are hard to be reverse engineered [2]. The main limitation of a NLFSR is that its usefulness as a CSPRNG is bound by its period. This work studies how to build NLFSRs with optimal periods.

1.1 Feedback Shift Registers

It is imperative to be familiar with the Feedback Shift Register (FSR) technology to follow up with this work. In this section, FSRs, the concept of an FSR's period along

with other definitions, terminologies, theorems and proofs are presented.

An FSR is a simple machine used to generate a bit sequence that has good randomness properties. FSRs consist of two main components: an n -bit *register* and a *feedback function*. Figure 1.1 illustrates FSR components, where the top part represents the n -bit register.

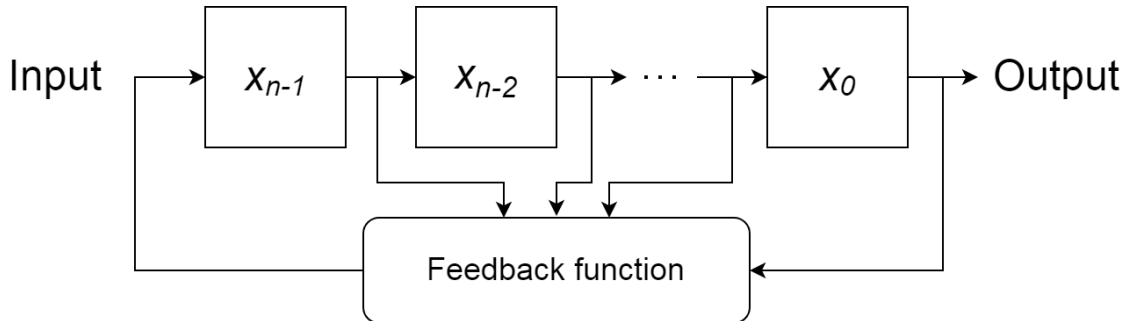


Figure 1.1: An n -bit Fibonacci type FSR.

The first main component, the n -bit register, is a collection of n storage cells called *stages*. Each stage holds a bit value. In Figure 1.1, the stages are represented as the linked squares, labeled x_0 to x_{n-1} . The stages labeled x_0 and x_{n-1} are stages with special roles. The stage labeled x_{n-1} is the *input stage*, it is responsible to store the values coming from the feedback function. The stage labeled x_0 is the *output stage*, it is responsible to output its stored value in each *FSR cycle*. FSR cycle is defined later in 1.1.1. The n -bit register is used to hold the state of the FSR and to perform a *shift operation* in the direction of the output stage. The state of the FSR is determined by the bit value of all stages in the n -bit register in their respective order, from x_{n-1} to x_0 . Therefore, the state of the FSR is represented using the tuple $(x_{n-1}, \dots, x_1, x_0)$. Performing the shift operation results in moving the value in every stage to the neighboring stage in the direction of the output stage.

The second main component, the feedback function, is a function that takes the state of the FSR as input, and outputs a single bit, the *feedback bit*. The feedback bit is to be stored in the input stage. This occurs as one of the steps in each FSR cycle. FSR cycle is defined in 1.1.1. In this work, the feedback function will be the focus of study. Specifically, what feedback functions produce an *optimal period*. Optimal period definition, theory, and further feedback function discussion is presented in Section 1.3

Both components, the n -bit register and the feedback function have an important role in every FSR cycle.

Definition 1.1.1 (FSR cycle). An FSR cycle is a process in which the following occurs in order:

1. The feedback function is fed with the state of the FSR, and the feedback bit is computed.
2. The bit value in the output stage is outputted.
3. The shift operation is performed.
4. The input stage stores the feedback bit.

Two main outcomes to be highlighted when an FSR cycle is performed: First, the FSR produces a bit as result. Second, the state of the FSR changes to a new state. Performing multiple FSR cycles in succession is sufficient to produce a sequence of random bits. More details on the characteristics of the output of FSR cycles is presented in Section 1.3

FSRs can be categorized into 2 categories, Linear Feedback Shift Register (LFSR) and NLFSR. In this work, only NLFSRs are studied. The difference between the two is a property in their respective feedback functions. In NLFSRs, the feedback function is non-linear, while in LFSRs, it is linear. NLFSRs produce more secure pseudo random sequence and are more suitable to be used in cryptographic applications [2]. The improved security of NLFSRs are linked to the non-linearity of their feedback functions. Non-linearity affects the reverse engineering complexity of FSRs. NLFSRs requires a bit sequence length of 2^n , while it is sufficient to analyze a bit sequence of length $2n$ to reverse engineer a LFSR. Further discussion is presented in Section 1.3. Another categorization of FSRs is the *Fibonacci* and *Galois* configurations. In Fibonacci configuration, the feedback bit is fed to the input stage only, while in Galois configuration, no such restriction exist, as the feedback bit can be fed to any one or more stages. This study covers the Fibonacci configuration only. Therefore, all presented findings in this study are attributed to the Fibonacci configured NLFSR.

1.2 Properties of NLFSR

In this section, it is introduced multiple properties of NLFSRs. These properties shall be used in presenting further explanations and proofs. Furthermore, important constraints and conventions are presented in this section.

1.2.1 Feedback Function Properties

In this study, feedback functions are allowed two bit-wise operations only:

1. Bit-wise (XOR $\oplus$)

2. Bit-wise (AND $\cdot$)

Both operations follow their conventional definitions in the field of computer science. The (XOR $\oplus$) operation outputs 1 if both operands have different values, else 0 is outputted. The (AND $\cdot$) operation outputs 1 if all operands have the value 1, else 0 is outputted. The (XOR $\oplus$) operation is both commutative and associative, therefore, the order of execution amongst (XOR $\oplus$) operands can follow any order. The order of execution between (XOR $\oplus$) and (AND $\cdot$) operations follows that agreed upon in the field of computer science. That is, it is first evaluated the result of each (AND $\cdot$) operation, then the result of each (XOR $\oplus$) operation is evaluated. In this study we add parenthesis surrounding (AND $\cdot$) operations for added clarity. An example of a valid feedback function is given in equation 1.1:

$$f(x_0, x_1, x_2, x_3, x_4) = x_0 \oplus x_1 \oplus x_2 \oplus x_3 \oplus (x_2 \cdot x_3) \oplus (x_1 \cdot x_3 \cdot x_4) \quad (1.1)$$

Figure 1.2 showcases how equation 1.1 can fit into an NLFSR in logic gates notations.

it is imperative to establish multiple terminologies with regard to the feedback functions under study. The following terminologies will be used heavily throughout this study:

- **Variable:** is a representation of each unique argument as defined in the function, i.e. $(x_0, x_1, x_2, x_3, x_4)$ are all the variables in equation 1.1.
- **Term:** is one or more variables bound by an (AND $\cdot$) operation. For example,

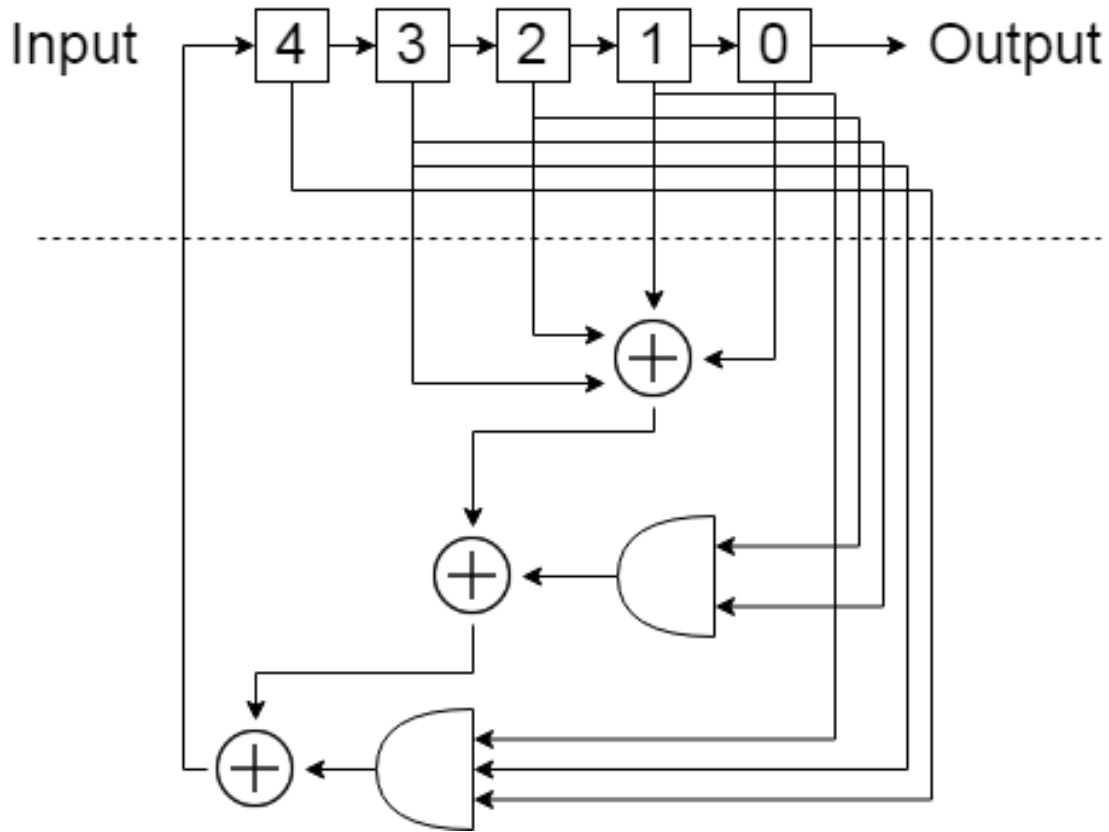


Figure 1.2: A complete logic gate representation of an NLFSR with feedback function equal to equation 1.1.

x_0 is a term and $(x_2 \cdot x_3)$ is another term in equation 1.1.

- **Term's degree:** is the variable count of each term. For example, x_0 is a term of degree 1 and $(x_2 \cdot x_3)$ is a term of degree 2 in equation 1.1.
- **Function's degree:** is the degree of the largest term in the function. In equation 1.1, the term with the largest degree is $(x_1 \cdot x_3 \cdot x_4)$ so the function's degree equals 3.
- **Product term:** is a term in which its degree is bigger than 1
- **Linear part:** is the collection of all terms that has degree equals 1.

- **Non-linear part:** is the collection of all product terms.

The following conventions are adhered to while working with feedback functions in this study:

1. The terms are ordered ascendingly based on their degree. As a result the linear part is presented first followed by the non-linear part. A valid ordering format is presented in equation 1.1
2. It is not allowed to have two identical terms in the same feedback function. Having identical terms causes the two terms to cancel out with the (XOR $\oplus$) operation, this causes unwanted noise. Further clarification for this decision is presented in section 1.4.

Feedback functions under this study have the following properties:

1. Non-linear.
2. Self contained within the NLFSR.
3. Deterministic.

An NLFSR's feedback function is non-linear by definition. For the feedback function to be non-linear, it must have a degree bigger than 1. A feedback function is considered linear if its degree equals 1. An NLFSR's feedback function is self contained. This means the feedback function depends on no external inputs outside of the NLFSR. Feedback functions being deterministic means that, given a particular input to a feedback function, the feedback function is always expected to produce the same output. This is the case because all the allowed operations in a feedback function, i.e.((XOR $\oplus$) and (AND $\cdot$)) are deterministic.

1.2.2 NLFSR's Output Properties

Recall the definition of an NLFSR cycle in section 1.1. Performing multiple NLFSR cycles in succession yields an output that has the following property: The output's consecutive n bits starting from a state χ is always equal to χ flipped. Such that, if $\chi = (x_2, x_1, x_0)$, then χ flipped $= (x_0, x_1, x_2)$.

This property can be proven via utilizing the definition of NLFSR in section 1.1. In each cycle, all stages change in one way only, which is the change due to the shift operation, except for the input stage. This means that the values of each stage, including the input stage will eventually be shifted to the output stage in a future NLFSR cycles. The values of each stage will be outputted in the following stage order $(x_0, x_1, \dots, x_{n-1})$, which is equal to the NLFSR's state flipped.

Therefore, given an NLFSR's state it can be calculated the next n bit results immediately. Then, by evaluating the feedback function in each cycle, it is possible to find all future bit results. Due to the deterministic property of feedback functions, it is possible to reliably calculate the output of an NLFSR given an initial state. Given a particular NLFSR state, all resulted bits are guaranteed to follow the same order. This observation will be used in introducing the concept of an NLFSR's period.

1.3 NLFSR's Period

One of the main goals of this work is studying NLFSR's feedback functions that generate optimal periods. In this section, *NLFSR's period* is defined and further discussed.

The period of a NLFSR is defined as the size of the longest unique sequence it outputs before the sequence starts repeating again. As it was established earlier, an NLFSR outputs the same result if it started from the same NLFSR's state. Therefore, the period of a NLFSR can also be defined as the longest sequence before a particular NLFSR's state is visited more than once. Next, it is discussed the concept of an optimal period.

1.3.1 Optimal Period

An optimal NLFSR period is a period with length equal to equation 1.2

$$2^n - 1 \tag{1.2}$$

Such that n in equation 1.2 equals the register size.

An optimal period is called *optimal* because it generates the longest possible unique sequence, minus one. First it is shown what is the longest sequence and why it exist, then it is discussed why we exclude one state from an optimal period.

The longest possible unique sequence for an NLFSR of size n is 2^n . To prove this, consider this property of NLFSRs: The output of an NLFSR repeats when given the same state. When an output repeats, it is considered no longer unique. Therefore, to maximize the unique sequence's length, it is required to visit each NLFSR's state once without repetition for as long as possible. The maximum possible unique states for a register holding n binary values is equal to 2^n . To show this, recall that an n -bit register contains n stages, each stage can store a binary value, either 0 or 1, that is,

each register can store 2 different values or have 2 different states. A 1-bit register consists of 1 stage, capable of representing 2 different states, therefore, the maximum number of unique states for a 1-bit register is 2 states. A 2-bit register consists of 2 stages, each can represent 2 different states, therefore, the maximum number of unique states for a 2-bit register is $(2) * (2) = 4$ states. An n -bit register consists of n stages, each can represent 2 different states, therefore, the maximum number of unique states for an n -bit register is $(2) * (2) * \dots * (2) = 2^n$ states.

The length of an optimal period is the maximum possible number of unique states, excluding a single special state. It is excluded the *zero state* from the states covered in an optimal period. The zero state is the state where all the register stages has the binary value 0. The zero state is excluded because it is a sink state. In other words, starting from a zero state, the next state can only be the zero state again. This is true due to the limitation of the allowed feedback function operations presented in subsection 1.2.1. All feedback functions built using the operations allowed outputs 0 if all inputs are 0, making the zero state a sink state. Due to this behavior of the zero state, it is compulsory that the zero state is excluded from the coverage of all states in an optimal period. Furthermore, if the zero state is visited, then the NLFSR's feedback function is immediately considered not optimal. Therefore, one of the properties of an optimal period is that it must not visit the zero state. This property will be used in presenting other definitions and proofs. To sum up, an optimal period is the longest possible unique sequence of outputs, where every NLFSR's states is visited once without repetition, always skipping the zero state.

Another important property for an optimal period is that it is *branch-less*.

A *branching* occurs when two different NLFSR's states lead to the same *next state*. The next state is the resulted state after executing one NLFSR cycle. To prove that optimal periods are branch-less, consider the definition of an optimal period where all states are visited, excluding the zero state. If there exist two different states leading to the same next state, then at least one of these two states must not be reachable when a period occurs. In such case, visiting all states and achieving an optimal period is impossible. To further elaborate on why one of the two states is unreachable, recall that feedback functions are deterministic. This means every state must have a single deterministic next state, however, different states could lead to the same next state. If two or more different states lead to the same future state, only one of such states will be visited again after a period's length. A visual guide can further help in clarifying this concept. Each node in Figure 1.3 represent an NLFSR state. Each state must lead to a single future state, therefore, only a single arrow can exit from each node. If a node received more than a single arrow, then by definition, a branch occurred. In Figure 1.3, nodes (a,c,d) are unreachable.

Optimal periods are a result of an optimally configured NLFSR. Specifically, an optimally configured feedback function for a given register size. Next, it is discussed the theory of optimal feedback functions found in literature prior to this study.

1.3.2 Optimal Feedback Function

An optimal feedback function is a feedback function that produces an optimal period. An optimal NLFSR is an NLFSR that contains an optimal feedback function. The knowledge of optimal NLFSR from literature can be summarized in theorem 1.3.1 [1].

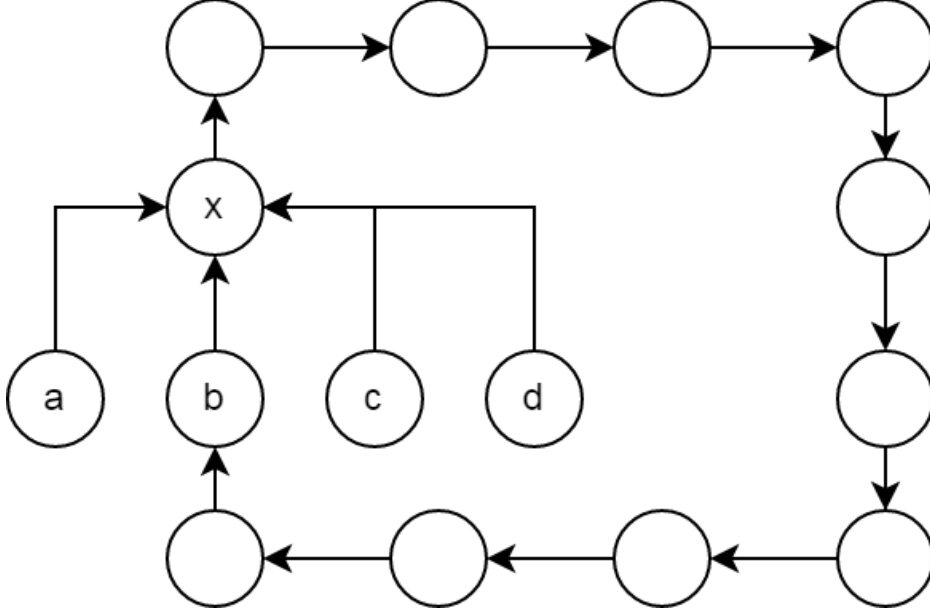


Figure 1.3: Illustration of a branching in NLFSR states.
Multiple nodes, (a,b,c,d), lead to the same future node (x). Only one of the nodes, node (b) is reachable after a period occurs.

Theorem 1.3.1 (Optimal NLFSR Format). An optimal NLFSR must be representable in the format of equation 1.3

$$f(x_0, x_1, \dots, x_{n-1}) = x_0 \oplus g(x_1, x_2, \dots, x_{n-1}) \quad (1.3)$$

An optimal NLFSR feedback function must have the variable x_0 in the linear part (XOR $\oplus$)ed with a function that does not depend on x_0 [10].

Next, a proof for theorem 1.3.1 is presented. The proof is divided in two sections. **First**, it is proven that x_0 must be present in the linear part as shown in equation 1.3. This section of the proof shall be denoted by (α) . **Second**, it is proven that function $g()$ in equation 1.3 must not depend on x_0 . In other words, x_0 must exist once in the definition of an optimal feedback function in the linear part, and must not exist again

as a variable in the rest of the function's definition. This section of the proof is split into two parts denoted by (β) and (γ) .

First, it is proven that x_0 must exist in the linear part, otherwise, the feedback function can not be optimal. Given a feedback function and a state, if there exist a non zero state, such that its next state is the zero state, then by definition, such feedback function can not be optimal. Therefore, as a proof by counter example, a state that satisfies this condition is given. The state with the binary representation to the number *one*, $(x_{n-1}, x_{n-2}, \dots, x_0) \rightarrow (0, 0, \dots, 1)$, is such state. This concludes part (α) of theorem 1.3.1 proof.

Second, it is proven that x_0 must exist once only in the feedback function, in the linear part. This section of the proof is a compound of two parts. **In part one**, it is proven that if x_0 exist more than once, then a branch must occur. This part of the proof is denoted by (β) . **In part two**, it is utilized the branch-less property of an optimal period: If a branch exist, then the function can't be optimal. This part of the proof is denoted by (γ) and it is used together with part (β) of the proof. Using both parts (β) and (γ) , it is concluded that x_0 must not exist anywhere but once in the linear part. Part (γ) is proven in subsection 1.3.1. Part (β) of the proof is given ahead.

Part (β) . Recall part (α) of the proof: x_0 must be used in the linear part, otherwise, the state *one* leads to the zero state. Using x_0 in this way has another purpose. x_0 is needed in every NLFSR cycle to diverge the feedback bit output for all pairs of *neighboring NLFSR's states*. A definition for a pair of *neighboring states* is: Any two different states that only differ by the value of the x_0 variable. If any

two different NLFSR's states happen to produce the same feedback bit, then a branch occurs, as shown in subsection 1.3.1.

Let (x_3, x_2, x_1, x_0) represent an NLFSR's state. The two state instances $(1, 0, 1, 1)$ and $(1, 0, 1, 0)$ forms a pair of neighboring states. For this given pair, the possible next state pair is $(0, 1, 0, 1)$ or $(1, 1, 0, 1)$. To prevent a branch from occurring, each state in a pair of neighboring states must lead to a different next state. To insure this behavior is met, x_0 must be used to diverge the next states of neighboring states. The divergence of neighboring states must be met in every NLFSR cycle. In other words, x_0 must successfully flip the output of the rest of the feedback function in every NLFSR cycle. x_0 can fail to diverge the next states of neighboring states if it got cancelled. x_0 gets cancelled if and only if there is at least one other x_0 variable in the feedback function. When multiple x_0 variables exist in a feedback function as operands to an $(\text{XOR } \oplus)$ operation then they no longer affects the result of the feedback function.

Next, it is shown that if the variable x_0 exist more than once in the feedback function, then there exist at least one pair of neighboring states that leads to the same next state. A pair of neighboring states that leads to the same next state is called a pairs of *colliding neighboring states*. From the constraints of building a feedback function presented in subsection 1.2.1, if x_0 is used more than once, then it is used once in the linear part and once or more in the non-linear part. In part (α) of this proof it has been proven that x_0 must exist once in the linear part. In this step, it is to be proven that if x_0 is shown once or more in the non-linear part, in addition to once in the linear part, then at least a pair of colliding neighboring states exist. The following method can reliably identify a pair of colliding neighboring states, if x_0

is used more than once in the feedback function.

Given a feedback function that has at least one x_0 variable in its non-linear part. Let δ be a non-linear term with the smallest degree that has the variable x_0 . Assign the value 1 to all variables of δ , except for x_0 . All variables of all terms except for x_0 and the variables in δ is assigned the value 0. The first element that belongs to a pair of colliding neighboring states is found via assigning x_0 the value 0, the other is found via assigning x_0 the value 1.

For example, consider the feedback function 1.4:

$$f(x_0, x_1, x_2, x_3) = x_0 \oplus x_1 \oplus (x_0 \cdot x_1) \oplus (x_0 \cdot x_1 \cdot x_3) \quad (1.4)$$

Following the method presented, we pick the term $(x_0 \cdot x_1)$, then we make x_1 equal to the value 1. All other variables but x_0 are to be assigned 0. Now the two states $(x_3, x_2, x_1, x_0) \rightarrow (0, 0, 1, 0)$ and $(0, 0, 1, 1)$ forms a pair of colliding neighboring states. To confirm the result, calculate the feedback bit of the feedback function 1.4 feeding it both of the colliding neighboring states, the feedback bit is equal to 1 in both instances. The state $(1, 0, 0, 1)$ is the shared next state for the pair of colliding neighboring states $(0, 0, 1, 0)$ and $(0, 0, 1, 1)$.

To show how the presented method always succeed in finding a pair of colliding neighboring states when there are more than one x_0 variable in the feedback function, recall that in part (β) of the proof it is discussed the importance of x_0 to diverge neighboring states in each NLFSR cycle. In the method presented, it is essentially attempted to cancel out the effect of x_0 . That is achieved via crafting a state in

which two x_0 variables are cancelled due to being operands of an (XOR $\oplus$) operation. Any extra x_0 variables are canceled via being part of an (AND $\cdot$) operation with a 0 variable. A state that is independent of x_0 is thus created, which by definition forms a pair of colliding neighboring states as discussed in part (β) of the proof. To sum up, it is shown that having more than one x_0 variable in the feedback function causes a branch to occur.

From parts (α), (β), and (γ) of this proof, it is shown that feedback functions must be of the format illustrated in equation 1.3. As a contribution, it is presented a hypothesis adding restrictions to the $g()$ function of equation 1.3 of an optimal feedback function. Such contribution is presented later in Chapter 5, subsection 5.1.1.

1.4 Feedback Function Types and Families

To achieve the goals of this work, it is desired to broaden the scope of studied feedback functions as much as possible. In this work it is covered a wider array of feedback function *types* compared to similar work. Similar work such as [1] [2] [3] has covered 4 feedback function types in total. In this work, 649 different feedback function types are studied. To describe the experiment and results of this work, it is imperative to introduce the concept of Feedback Function Type (FFT).

FFT is a collection of constraints partially describing feedback functions. Each feedback function is associated with a unique FFT. The constraints of an FFT is represented by the tuple 1.5.

$$(l, [p_1, p_2, \dots, p_k]) \tag{1.5}$$

FFT 1.5 has 2 elements. First, l , is an integer representing the variable count of the linear part of a feedback function, excluding the x_0 . The second element $[p_1, p_2, \dots, p_k]$ is an array of integers, each integer in the array describes a term in the non-linear part, such that its value represent the degree of that term. FFTs is a partial representation to an infinitely many feedback function. In this work it is beneficial to come up with a representation to a finite group of feedback functions.

Feedback Function Family (FFF) is an FFT with an extra constraint grouping a finite amount of feedback functions. FFF is represented by 1.6

$$(l, [p_1, p_2, \dots, p_k]), n \tag{1.6}$$

The added element, n , is an integer representing the register size of an NLFSR.

A final feedback functions grouping concept, the feedback function *series*, is further used in this study. A feedback function series is a group of FFFs bound by some rules. Chapter 4 formally introduces feedback function series. In this work, it is studied what FFFs include optimal functions. FFFs are made to be mutually exclusive. No feedback function is associated with multiple FFF, and each feedback function is associated with a single FFF. To preserve these properties, feedback functions are not allowed to have identical terms. Disallowing identical terms also protects the degree of the feedback function. Identical terms can cancel out each other. When all terms of the highest degree cancel each other, the feedback function degree drops. This can cause non-linear feedback functions to become linear, causing noise in the results.

Due to those aforementioned properties of FFFs, FFF makes for a good grouping

mechanism for studying feedback functions. The parameters l , $[p_1, p_2, \dots, p_k]$, and n shall be a focal point on this study's experiment, presented in Chapter 4.

An example for an FFF is given ahead. Consider $\text{FFF} = (1, [2]), 4$. That is $l = 1$, $k = 1$, $[p_1] = 2$, and $n = 4$. This FFF groups multiple feedback functions that are illustrated in Table 1.1.

$x_0 \oplus x_1 \oplus (x_1 \cdot x_2)$	$x_0 \oplus x_1 \oplus (x_1 \cdot x_3)$	$x_0 \oplus x_1 \oplus (x_2 \cdot x_3)$
$x_0 \oplus x_2 \oplus (x_1 \cdot x_2)$	$x_0 \oplus x_2 \oplus (x_1 \cdot x_3)$	$x_0 \oplus x_2 \oplus (x_2 \cdot x_3)$
$x_0 \oplus x_3 \oplus (x_1 \cdot x_2)$	$x_0 \oplus x_3 \oplus (x_1 \cdot x_3)$	$x_0 \oplus x_3 \oplus (x_2 \cdot x_3)$

Table 1.1: Feedback functions that are members of $\text{FFF} = (1, [2]), 4$.

Valid FFFs must adhere to the following rules:

1. $l > 0$
2. in $[p_1, p_2, \dots, p_k]$, $k > 0$
3. $p_1, p_2, \dots, p_k > 1$
4. $n \geq \rho$

First rule of FFF, $l > 0$, otherwise, associated feedback functions can't be optimal. To prove this, take the state with the binary representation to the number *one*, $(x_k, x_{k-1}, \dots, x_0) \rightarrow (0, 0, \dots, 1)$. If $l = 0$, starting with the state *one* always yield feedback functions with period equal to n , which is not optimal by definition 1.2. Second rule of FFF, $k > 0$, otherwise the feedback function is linear. Only non-linear feedback functions are studied in this work. Third rule of FFF, all elements of the list $p_1, p_2, \dots, p_k$ must be > 1 . Each element in $p_1, p_2, \dots, p_k$ is used to represent product terms, and the least degree for a product term is 2. Last rule of FFF, $n \geq \rho$. We define ρ as the minimum possible register size for a given feedback function. ρ is the output of a function given both parameters of an FFT l and $[p_1, p_2, \dots, p_k]$, as shown in equation 1.7.

$$f(l, [p_1, p_2, \dots, p_k]) = \max(l, p_1, p_2, \dots, p_k) + 1 = \rho \quad (1.7)$$

1.5 Thesis Structure

This is an empirical study. The main contributions are results of a conducted experiment. Chapters are organized with the focus to present the experiment and its results. This work is divided into 5 chapters.

- **Chapter 1** introduces important definitions, defines the problem, and highlights related literature work.
- **Chapter 2** covers literature review and work motivation.
- **Chapter 3** illustrates the programming work of the experiment.
- **Chapter 4** presents the experiment, its objectives and methodology.
- **Chapter 5** highlights the results and findings.
- **Chapter 6** concludes and summarizes the work.

CHAPTER 2

LITERATURE REVIEW AND MOTIVATION

In this chapter, it is presented a literature coverage of the studied topic. Multiple aspects are covered as part of the literature review, with each aspect it is elaborated how this work compares and adds value to the topic of study. Finally, the motivation behind this work is presented.

2.1 Literature Review

2.1.1 Finding All Optimal Functions For an FFT

In 2012, a work published by Dubrova [1] attempted to list all optimal feedback functions of three FFTs, all of which were of degree two. In [1], it is intended that the published results be analyzed by other work for potential mathematical discoveries that enrich our understanding of how to create an optimal NLFSRs. So far in

literature, the notation for representing FFTs is less developed than what is presented in Section 1.4 of this work. In this work FFTs are represented in a compact way that facilitates looping through types and highlighting their features. In [1], the studied FFTs are represented as follows:

1. $f(x_0, x_1, \dots, x_{n-1}) = x_0 \oplus x_a \oplus x_b \oplus x_c \cdot x_d$
2. $f(x_0, x_1, \dots, x_{n-1}) = x_0 \oplus x_a \oplus x_b \cdot x_c \oplus x_d \cdot x_e$
3. $f(x_0, x_1, \dots, x_{n-1}) = x_0 \oplus x_a \oplus x_b \oplus x_c \oplus x_d \oplus x_e \cdot x_h$

Translating those FFTs to this work's notation gives the following in order:

1. $(2, [2])$
2. $(1, [2, 2])$
3. $(4, [2])$

The FFTs in [1] were studied for NLFSRs of register sizes in the range ($4 \leq n \leq 24$). Specifically, the arrangement of FFTs together with the targeted register sizes forms the following FFF: $\{(2, [2]), n \mid 4 \leq n \leq 24\}$, $\{(1, [2, 2]), n \mid 4 \leq n \leq 24\}$, and $\{(4, [2]), n \mid 7 \leq n \leq 24\}$. That is, a total of 3 FFTs and 60 FFFs has been covered in [1]. Supporting the efforts of [1], in 2018, Almuhammadi et al [2] managed to improve the list published in [1] fixing some issues and supporting its results. In this work, as part of confirming the correctness of the developed period testing algorithm, it has been confirmed that the fixes presented in [2] are valid and complete. In 2020, Al-Hejri [3] studied a 4th type: $\text{FFT} = (2, [3])$. The studied new type targeted feedback functions of degree 3, a degree higher than what is studied in [1] and [2]. The register sizes in [3] are in the range ($5 \leq n \leq 25$). This work can be considered as a direct continuation to the efforts in [1] [2] [3], significantly increasing the scope of these efforts.

Compared to the work done in [1] [2] [3], the scope and results of this work are significantly bigger. This study covers a wider array of FFTs and FFF and explores more variation of FFT features. Table 2.1 highlights the main differences of this work vs [1] [2] [3] with regard to the scope of studied FFTs, the studied functions degree range, and optimal functions found.

Comparison subject	Mahdi (this work)	[1]	[2]	[3]
Number of FFTs Covered	649	3	3	1
Degrees Range	2 to 31	2	2	3
Optimal Feedback Functions Found	99,933,656	425	425	140

Table 2.1: Comparison between this work and [1] [2] [3] with regard to scope and results.

This expansion in scope allowed for the discovery of some noteworthy findings. Those findings are presented as contributions of this work in Chapter 5.

2.1.2 Utilizing Hardware Acceleration

Other similar work explored utilizing specialized hardware technology to accelerate the search for optimal feedback functions. While this work utilizes assembly code optimization in building its period testing algorithm rather than hardware acceleration, both share the same goal of finding optimal functions and thus covering such work is relevant. A highlight for some of the work utilizing hardware acceleration in searching for optimal feedback functions is as follows: Tomasz Rachwalik et al [4] in 2012, Dabrowski et al [5] in 2014, Poluyanenko [6] in 2017, and Paweł Augustynowicz and Krzysztof Kanciak [7] in 2019. In [4], it is used FPGAs to find NLFSRs with optimal periods in registers of sizes 25 and 27. The published results include examples of optimal NLFSRs of sizes 25 and 27. A total of 7 optimal feedback functions were

published in [4]. In [5], parallel computing techniques were used to search for optimal feedback functions. It is published 14 optimal functions with register sizes ranging from 22 to 31. A difference between this work and [5] is that feedback functions are allowed to use the negate operation in [5] unlike this work. In [6], FPGAs are used in searching for optimal feedback functions. 15 optimal feedback function for registers of sizes 26 to 29 were published. In [7], an optimized CPU and GPU implementations to find optimal feedback functions were conducted. 14 optimal feedback function for registers of sizes 26 to 30 were published. As part of this work’s efforts, all published feedback functions that can be tested in [4] [5] [6] [7] were tested for optimality. All function published in [4], [5], and [6] were found optimal. In [7], 6 of the 14 published functions were found not optimal when tested using this work’s period testing algorithm. All 6 functions failing to be optimal share a common trait. All 6 functions do not satisfy one of the hypothesis for optimal feedback functions presented as a contribution in this work, the hypothesis in question is presented in section 5.1.1. This suggests the fault is likely in the published results of [7] rather than an issue with this work’s period testing algorithm. Table 2.2 summarizes the difference between this work’s scope and technology used vs [4] [5] [6] [7].

Comparison subject	This work	[4]	[5]	[6]	[7]
Technology used	Assembly	FPGA	Parallel computing	FPGA	GPU
Degrees Range	2 to 31	25, 27	22 to 31	26 to 29	26 to 30
Optimal Functions	99,933,656	7	14	15	14

Table 2.2: Comparison between this work and [4] [5] [6] [7] with regard to period testing methodology, feedback function features, and the count of optimal functions found. Table 2.3 better compares the count of optimal functions with respect to registers of sizes 22 to 31.

Unlike [4] [5] [6] [7], this work studies each FFF thoroughly. That is, when an

FFF is studied, a full coverage of all feedback function members of the FFF is tested for optimality. Therefore, optimal functions published in this work are necessarily all the optimal functions of their associated FFF. The focus of this work is to find mathematical constructs describing optimal functions rather than increasing the pool of published optimal functions. FFFs chosen for this study were the ones yielding optimal functions while not taking too long from the period testing algorithm in execution time. This makes studying FFFs with smaller register sizes more fulfilling to this study's goals. Among the 2,848 FFFs covered in this work, less than 100 have register sizes bigger than 21. Due to this bias, this work did not manage to find any optimal feedback function for registers of sizes 27 and 30. This is not due to a weakness but rather to a difference in this study's goals vs [4] [5] [6] [7]. Regardless of the difference in the goals, this work managed to surpass other work from literature that attempts to find optimal feedback functions for NLFSRs with large register sizes. Table 2.3 compares the count of feedback functions found in this work vs [4] [5] [6] [7].

2.1.3 SAT Short Period Exploit

Recent studies introduced a new technique that allows adversaries to force NLFSRs in stream ciphers into a short period via an active attack. The new technique utilizes special SAT solvers to search for vulnerable NLFSR states. The states in question are vulnerable due to being part of a short period. An attacker able to force the initial NLFSR state into a vulnerable state is then able to accurately predict the output of the NLFSR which compromises its security.

Register size $n =$	Number of optimal feedback functions				
	Mahdi (this work)	[4]	[5]	[6]	[7]
22	62,660	-	1	-	-
23	18,582	-	1	-	-
24	19,308	-	1	-	-
25	7,256	4	1	-	-
26	3606	-	2	4	3
27	-	3	2	4	3
28	108	-	2	6	3
29	796	-	2	2	3
30	-	-	1	-	3
31	1712	-	1	-	-
Total:	114,028	7	14	15	14

Table 2.3: Comparison between this work and [4] [5] [6] [7] with regard to optimal NLFSRs found in each study for n -bit registers with sizes ($22 \leq n \leq 31$).

The work published by Dubrova and Teslenko [8] in 2018 have proposed an algorithm that work on popular stream ciphers: Bivium, Trivium, Grain-80, and Grain-128. The algorithm uncovered vulnerable states in each of the aforementioned stream ciphers. Those vulnerable states form short periods breaking the cipher's security. It is identified 39 vulnerable periods each of size no bigger than 15 cycles for all studied stream cipher. Dudzic and Kanciak [9], extended the work of [8] introducing a modified method to find short periods in NLFSRs. Bivium, Trivium, Grain-80, and Grain-128 are studied in [9]. The modified technique allowed the discovery of more vulnerable states with periods of sizes less than 66 cycles.

This work presents some mathematical constructs that can be used to strengthen the applications of stream ciphers. The vulnerability introduced in [8] [9] could be avoided in future stream cipher applications. The findings presented in this work can eventually lead to an optimal usage of Fibonacci configured NLFSR, such that stream ciphers are equipped with guaranteed optimal period Fibonacci NLFSR

neutralizing the exploit in [8] [9]

2.1.4 Galois Configured NLFSRs

Research in Galois configured NLFSRs received more attention in literature than Fibonacci configured NLFSRs. This is likely to Galois configuration being less restrictive to work with and the results of such studies can be directly used to improve stream cipher applications. The author of [1] Dubrova, followed this popular approach of studying Galois NLFSRs in [11] [12] [13] in 2012, 2014, and 2017 respectively. In [11] a method to construct Galois configured NLFSRs with optimal periods were introduced. The technique in [11] cross-joins multiple LFSR with a non-linear filter to generate an n -bit Galois NLFSR with period of size $2^n - 1$. In [12] another technique for building large period Galois NLFSR is presented. The technique in [12] merges multiple NLFSR cycles forming one larger cycle. Dubrova and Hell [13] advanced the work of [11] and [12] proposing a new stream cipher named *Espresso*. Espresso is designed for 5G systems and provides a faster performance than existing alternatives. At the time of proposing Espresso, it was tested and is resistant against known attacks. Espresso utilizes Galois based NLFSR which in later research [14] shown to be less secure than what was believed.

Another topic that received attention in literature are the efforts to find ways to convert Galois to Fibonacci configurations. Zhiqiang [15] in 2013, published a method to transform Galois NLFSRs to Fibonacci configuration. The method however, does not always succeed in transforming any Galois based NLFSR to the Fibonacci configuration. Zhiqiang in [15] suggests that finding a universal method to transform any

Galois configured NLFSR to the Fibonacci configuration seems more difficult than the factorization of polynomials. In 2014, Zhao [16] introduced a method to analyze NLFSR periods by utilizing techniques of Boolean networks. The method presented can theoretically find optimal period NLFSRs, however the method complexity is too high to be practical. The complexity is based on a transitional matrix of size $2^n * 2^n$ such that n is the register size. In 2018, Lu et al [17] proposed another method to convert Galois to Fibonacci configuration. This method converts NLFSRs into Boolean networks then by using semi-tensor product of matrices the transformation is achieved. This new method allowed transforming new types of Galois configured NLFSRs that were not possible prior to this work using the existing methods. However, there are still some types of Galois configured NLFSRs that are not yet convertible to Fibonacci configuration. Yao and Parampalli [18] in 2022 advanced the research of Galois to Fibonacci transformation algorithm. In [18] an improved version of known Galois to Fibonacci transformation algorithms are proposed. Three types of Galois NLFSRs are classified and an improved algorithm for each type is given. Furthermore, a new Galois type that can be transformed to Fibonacci is identified and a corresponding transformation algorithm for it is given. Up until [18], it is still not possible to transform all Galois NLFSRs into Fibonacci configurations.

Espresso-like stream ciphers has been shown a weakness. In Yao's and Parampalli's work [19] [14] it is shown a method to transform the Galois NLFSR in Espresso-like stream ciphers into a LFSR with a non-linear filter. This kind of FSRs can be broken via the fast algebraic attack as elaborated in [14]. This finding highlights a weakness in optimal Galois NLFSRs that are built using merging optimal LFSR

cycles. This work improves the efforts of securely incorporating NLFSRs in stream ciphers via studying optimal Fibonacci NLFSRs.

2.2 Motivation

The recent findings in literature highlights multiple weaknesses for the topic of finding and using optimal NLFSRs. Attempting to strengthening these weaknesses, two main reasons motivates this study. The first reason: the short period exploit introduced in section 2.1.3 can be seen as a result of an underdeveloped understanding for how to construct NLFSRs with optimal periods. This study improves the knowledge deficiency via introducing valuable knowledge in the shape of multiple hypothesis in Chapter 5. The second reason: the relatively unsuccessful attempt to build optimal NLFSRs with Galois configurations in stream ciphers as elaborated in section 2.1.4 incentivizes studying the other configuration, the Fibonacci configuration. In this study, the Fibonacci configuration is studied for optimality.

CHAPTER 3

THE PERIOD-TESTING SUIT

In this chapter, it's focused presenting the programming part of the work. A flexible and highly efficient Period-Testing suit has been built as part of this study, which allowed executing the experiment of this study. The suit is built using python as the main high level programming language. To achieve high efficiency, Assembly language period testing routine was used within the python program. The suit is available publicly on git hub at:

<https://github.com/m313live/NLFSR-Period-Testing-Suit>

3.1 Suit Use Cases

The ultimate use case of the suit is to compute an NLFSR's period to check whether it's optimal or not. The suit, however, is flexible. It supports multiple inputs, each descriptive a use case. The use cases are:

1. Given a fully defined NLFSR, compute it's period.
2. Given an FFF, compute the periods of all the functions associated with that

family and give a report for their periods.

3. Given a feedback function series, go over all FFFs of that series, then report the periods of all functions of each FFF.

The 3 use cases are linked with one another, programming wise. The functionality of the 3rd use case will be using the functionality of the 2nd use case, and the latter will be using the functionality of the 1st use case. From the user’s perspective, having all 3 different usecases provide important flexibility for confirming results and running experiments.

The 1st use case allows to run individual tests for a single feedback function, it is helpful to confirm results or to make quick tests. To facilitate its usage, it is possible to feed the function multiple input formats. Both, the *compact form* and *visual form* are supported as input. In this chapter, the compact and visual forms are formally introduced. The period-testing suit is built using python as the main language with 3 main data object.

3.2 Programming Tools

The suit is built with python as the main programming language. The PeachPy [20] framework is used to write high performing assembly kernels. PeachPy allows writing assembly instructions for multiple architectures including the x86-64, which is the one used for building the suit’s Period-Testing algorithm. Assembly instructions such as **ADD**, **AND**, and **SHR** are heavily utilized in building the Period-Testing algorithm, which is presented in section 3.4.1. Other benefits of using python and PeachPy is

the portability and the wide operating systems support. Thanks to those benefits, the suit works on Windows, Mac OS, and Linux; and it works with both intel and AMD CPUs. Python object oriented nature allowed for easier structure of the FFF model.

3.3 Suit Data Objects

There are 3 major data objects in the suit. The data objects are *Fib-Family*, *Fib-Function*, and *Run-Options*. Fib-Family and Fib-Function are dedicated for presenting NLFSRs. Run-Options configures how to run the test.

3.3.1 Fib-Family

As presented in section 1.4, the concept of FFF is essential in communicating the definition and results of this work's experiment. FFF also groups feedback functions that share related features and is an important entity to be programmed into the period-testing suit. The data object Fib-Family is used to represent FFF. Listing 3.1 illustrates Fib-Family.

```
1 class Fib_Family:
2     xor: int
3     ands: []
4     register_size: int
```

Listing 3.1: Fib_Family.py

3.3.2 Fib-Function

A Fib-Function is a presentation of a fully defined feedback function. Fib-Function extends Fib-Family and adds mapping to each variable. The mapping links feedback function variables to stages in the NLFSR's register. This mapping transforms an FFF to a fully defined feedback function. Finally, since FFFs also defines the register size, Fib-Function is not just a fully defined feedback function, it also represents a fully defined NLFSR. In the suit, a Fib-function is an object extending Fib-Family adding a list of integers to represent the mapping of variables to stages. listing 3.2 illustrates the construction of feedback functions in the suit.

```
1 class Fib_Function(Fib_Family):  
2     configurations: []
```

Listing 3.2: Fib_Function.py

Other than using the default constructors to initialize a feedback function, 2 alternative formats can be used. Those alternative formats are the compact form and the visual form, which are also used to present the results of the experiments of this work. Those representations hold significance in this work as some important findings were results of analyzing the output represented in both formats.

The compact form is very close in structure to how a feedback function is represented mathematically, while requiring less space. The mathematical form is a direct representation of the feedback function's variables and the operations between them. An example of a mathematical representation of a feedback function is given in equation 3.1. The compact form is a jagged list. First, the linear part is added

such that the index of each linear term is added to the list. Then, the non-linear part is added a term at a time starting from the lower degree terms. Each product term is added as a list containing the variables of the term. It is omitted adding x_0 in the linear part of the compact form, to save space and because it has been proven that it is needed for all optimal periods in Chapter 1. For example, the feedback function represented mathematically in equation 3.1 gives a compact representation equal to 3.2

The visual form of a feedback function is used to visualize the wiring of stages in NLFSR. The format of the visual form is a string over the alphabet $\Sigma = \{x, o, -, |\}$. The vertical bar symbol $|$ is used as a separator between the stages. Each variable is highlighted in the visual form using the x and o symbols to determine whether the stage is linked to an (XOR) or (AND) gate, respectively. If the stage is not linked to an (XOR) gate, the x symbol is replaced with a dash $-$, the same applies for (AND) gates. Therefore, a stage not linked with any logic gate looks like $(| - -|)$. Similar to the action taken in the compact representation, the x_0 stage is omitted from being represented in the visual representation for the same reason. A stage's linkage with an (XOR) gate is first represented with the x symbol, then, the (AND) gate linkage is represented with an o for every product term. For example, the feedback function represented mathematically in equation 3.1 gives a visual representation equal to 3.3

To clarify the relationship between the different formats used to represent feedback functions in this study, consider the following feedback function in mathematical

form 3.1:

$$f(x_0, x_1, x_2, x_3, x_4) = x_0 \oplus x_1 \oplus x_2 \oplus x_3 \oplus (x_2 \cdot x_3) \oplus (x_1 \cdot x_3 \cdot x_4) \quad (3.1)$$

The compact form of function 3.1 is presented in 3.2:

$$[1, 2, 3, [2, 3], [1, 3, 4]] \quad (3.2)$$

The logic gate representation of the function 3.1 is presented in Figure 3.1:

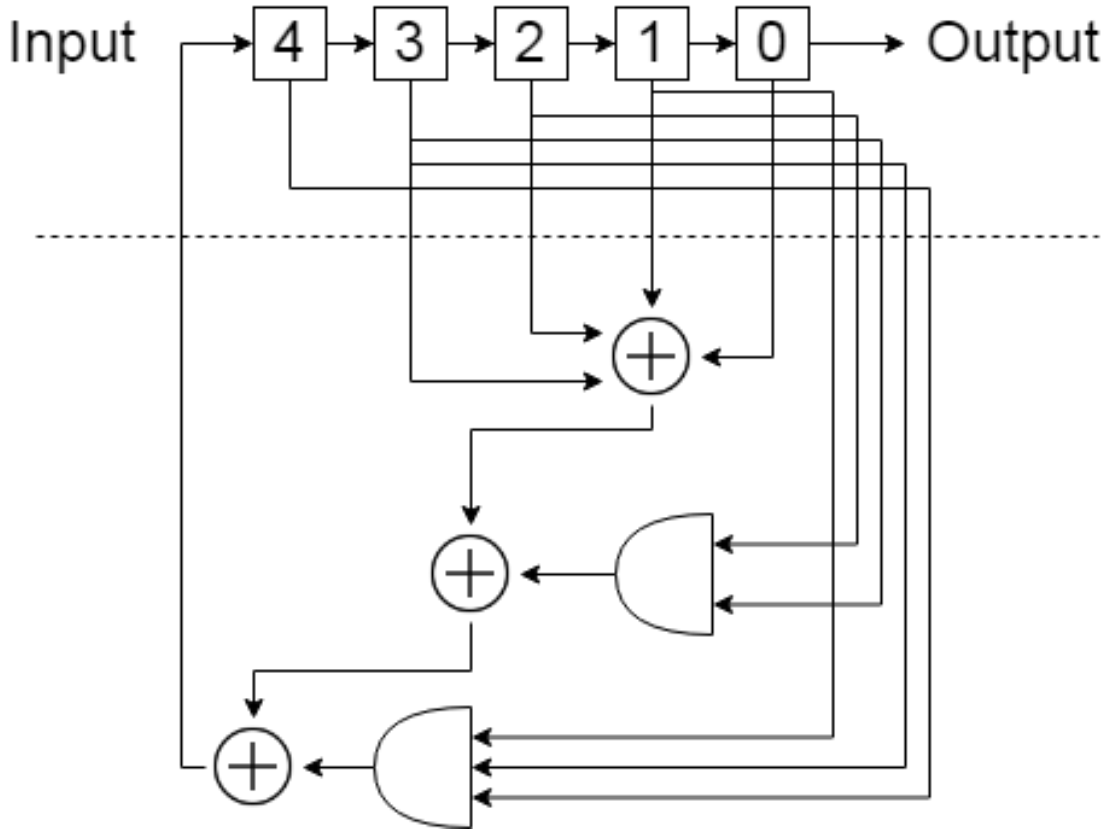


Figure 3.1: Detailed NLFSR representation with feedback function equal to equation 3.1.

The visual representation of function 3.1 is presented in 3.3:

$$|x - o| |xo - |xoo| - -o| \quad (3.3)$$

The FFF of function 3.1 is presented in 3.4:

$$(3, [2, 3]), 5 \quad (3.4)$$

3.3.3 Run-Options

Run-Options represents a list of options that affect the operation and output of the algorithm. listing 3.3 illustrates Run-Options.

```
1 class Run_Options():
2     go_fast: bool = True
3     add_x0: bool = True
4     add_x0_as_active: bool = False
5     catch_loops: bool = False
6     print_sequence: bool = False
7     print_all_periods: bool = False
8     initial_register_state: int = 1
```

Listing 3.3: Run_Options.py

The suit is highly flexible. The suit includes two different period testing algorithms. One that is highly optimized and uses Assembly level optimization. The other is less optimal, uses python level, however allows for power options, such as discovering

branching. Here is a list of the Run-Options and what they mean:

- **go_fast**: Use assembly or python version.
- **add_x0**: Always add stage x_0 in the linear part.
- **add_x0_as_active**: Include x_0 in the non-linear part while searching for optimal periods.
- **catch_loops**: Use memory to catch possible branching.
- **print_sequence**: Print the full NLFSR output.
- **print_all_periods**: Print the count of all periods searched.
- **initial_register_state**: Represents the initial register state.

3.4 How it Works

There are 3 main working algorithms:

1. **period_tester()**: Input is feedback function and initial state. Output is the period length. This maps to the 1st use case.
2. **next_function_mapping()**: Input is feedback function. Output is next function in the same family. This together with **period_tester()** powers the 2nd use case.
3. **x_wires()**: Input is a threshold value. Output is a list of all FFFs satisfying the threshold value. This together with **next_function_mapping()** fulfils the 3rd use case.

3.4.1 Period Tester

The period tester algorithm is the heart of the suit and the algorithm fulfilling the 1st use case. There are 2 different period testing algorithms in the suit. The first is the optimized Assembly version. The second is a less optimized, but more flexible, python version. Both algorithms simulate the actual work of the NLFSR. To simulate the NLFSR, we need to have a way to simulate the shift operation and the feedback function.

The assembly version uses the CPU register to hold the NLFSR's state. The logical shift operation, for example (logical shift right **SHR**), is a standard Assembly level instruction, therefore, relying on it gives us optimal performance and achieves the functionality of the shift operation. To achieve the functionality of the feedback function efficiently, it's utilized an x86 instruction called (pop count **popcnt**). **popcnt** counts all occurrences of 1s in a register. Using **popcnt** and masking techniques appropriately, we can compute the feedback function using the following tricks. A mask is to be used for the linear part. The mask filters (zeroes out) all stages that are not part of the linear part. The **popcnt** is used on the filtered result, if the result of the **popcnt** operation is even then the linear part of the feedback function result is 0, else the result is 1. Then a list of masks, one for each product term is used. For each masked instance, **popcnt** is used to count the number of 1s, this time, for each term, the result is 1, if and only if **popcnt** value is equal to the degree of that term. A loop takes all masked product terms and perform a final **popcnt** on them, if the result is even, 0 is returned, else 1. Finally, the linear part is (XOR $\oplus$) with the non-linear

part calculating the feedback bit.

The **python** version uses direct bit wise manipulation API from python standard library to achieve both the shift operation and simulating the feedback function. The python version also has an option to use memory to detect branches, as well as an option to print the results, and debug the register state in each cycle. Figure 3.2 highlights the top level inter-working of the period testing algorithm.

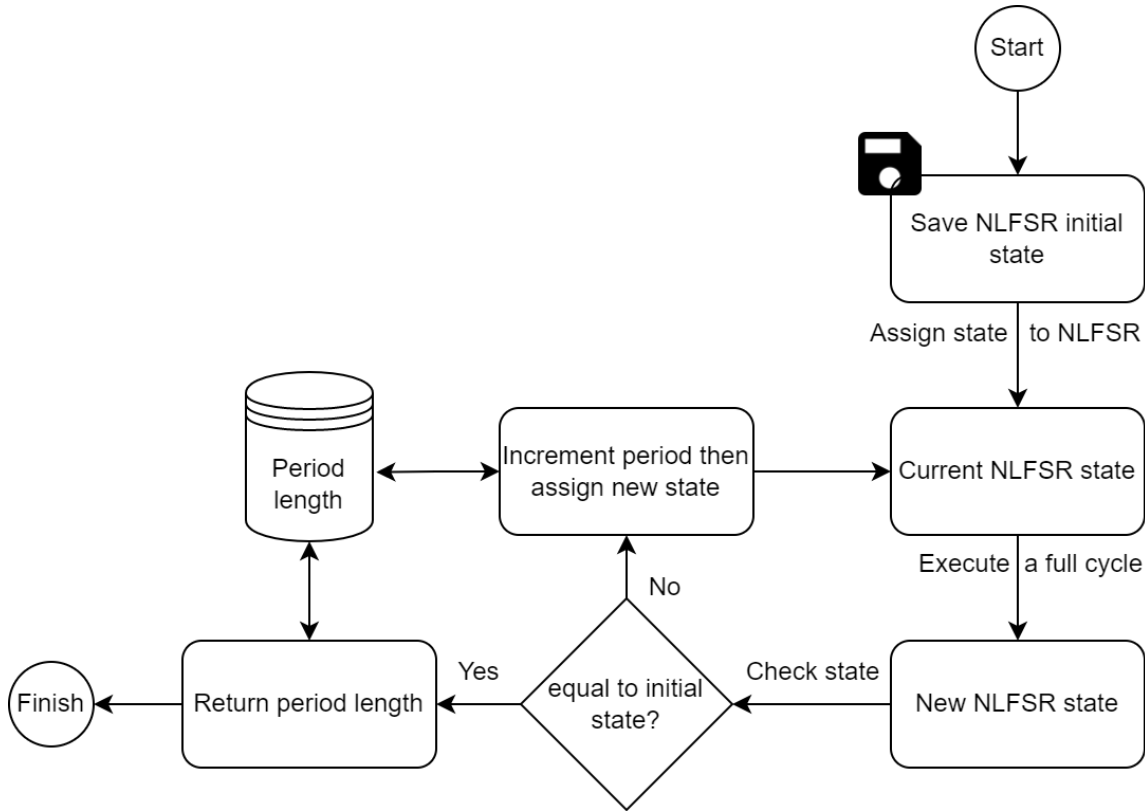


Figure 3.2: A macro level perspective of the period testing algorithm highlighting the major steps of the algorithm.

3.4.2 Next Function Mapping

Given an FFF, it's desired to be able to loop over all associated feedback functions.

In this section, such method is proposed. The proposed method additionally has the

following 2 desired properties: It skips isomorphic results, it skips results in which term cancellation can occur. Given an FFF $(l, [P_1, p_2, \dots, p_k]), n$, the approach is to create groups and move them forward in a certain way. The linear part constitute the first group, then each product term is made into a group. Starting with groups from p_k then p_{k-1} , until group l , **move** the first group that hasn't reached it's limit, then **reset** all groups ahead appropriately. Pseudo code presented in listing A.1 illustrates the process, explaining the **move** and **reset** processes in more details.

3.4.3 X Wires

This algorithm is built specifically to run the main part of the experiment. This function loops over all families bound by the following constraints: Given an FFF, $(l, [p_1, \dots, p_k]), n$.

- $l \leq 10$
- $\sum_{p_1}^{p_n} \leq 10$
- $n \leq 11$

To iterate over all configurations satisfying these constraints, it's constructed two nested loops. The inner loop iterates over valid sizes, while the outer loop iterates over valid families. The inner loop is constructed such that the values in the range $(\rho$ to 11) is being enumerated, such that ρ is as defined in 1.7. Each value in the range represents a register size. The second outer loop iterates over all families using another two nested loops. The second outer loop is a simple enumeration over the linear part. The second inner loop construction is less trivial and a brief summary of it is given ahead.

A macro view of the approach is to consider every product term as an element

in a list and perform operations on the list. Start with a minimum sized list. Then, enumerate over all valid combinations before increasing the list's size. After iterating over all combinations, the list size is increased by one and the process is repeated. A detailed description is as follows: Start with a list of size 1 containing the element 2, i.e.([2]). The minimum value for each element is 2 because each element in the list represent the degree of a product term. Then a *cascade increment* is performed until it is no longer possible. If a cascade increment is not possible without exceeding the constraints, the list size is increased by one. If it is not possible to increase the list size without exceeding the constraints, stop. A pseudo code for the *X Wires* algorithm further clarifying the steps and formally defining cascade increment is provided in listing A.2.

3.5 Proof of Correctness

Proving the correctness of a modern software such as the suit built in this chapter is quite challenging to write and read. Presenting a mathematically styled rigorous proof is extremely difficult and can be considered inefficient to be used as argument for the correctness of the suit, due to being long and hard to follow. Instead, via following the heuristics below, it's achieved some level of confidence for the suit's correctness.

1. Reproducing similar work.
2. Having two algorithms, python and assembly.
3. Following a testing driven development.

4. Making the work available publicly.

In reproducing similar work, the work done in [1] [2] [3] has been fully recreated and confirmed. Further confirmations of similar done work was to confirm all published results in [4] [5] [6] [7]. Interestingly in [7] it's found some invalid results. All invalid results in [7] were due to the FFF belonging to a non-optimal setting. This suggests the mistake is most likely from the work in [7]. This is discussed further in the Chapter 5.

The existence of two independent period testing algorithms (python and Assembly) that produces the same results makes it less likely that the same error is made in both versions. While it is not impossible that the same error exist in both versions, it is highly unlikely, due to how technically different the implementation of each algorithm is.

Following a test driven development can increase the confidence of the suit's correctness. Tests were developed before methods and were used heavily throughout the development to assert the correctness of the code. The suit was first built with an easy to test functionality. The output of the simple functions is later used to build complex functions while satisfying the test driven methodology of building tests first and testing often after each addition. Source control was used to store stable versions as commits.

Making the work available publicly is an ongoing proof of correctness. Publishing programming work to the public landscape encourages experts to try and find bugs in them. Critical cryptographic libraries follow this approach of making the source code public for an added confidence in correctness. Consequently, the longer

this published suit stays in the public domain without issues being discovered, the more confidence of its correctness is established.

CHAPTER 4

THE EXPERIMENT

This chapter introduces the experiment and explains its methodology and objectives. The experiment is run using the suit introduced in Chapter 3. A large amount of optimal feedback functions was discovered as a direct output from the experiment. Those optimal functions constitute all the data used in this study. Data analysis is presented in Chapter 5. All experiment data is available publicly. Link to get the data can be found in the git hub page of this work at:

<https://github.com/m313live/NLFSR-Period-Testing-Suit>

4.1 Experiment's Objectives

The general goal is to find some rules or patterns that can help us advance our mathematical understanding of what make an NLFSR generate an optimal period. Experiment objectives are structured as follows:

- **Objective 1:** Find a formula that can determine if a given FFT can contain optimal functions.

- **Objective 2:** Find some patterns and rules that can be used to accelerate the search of optimal configurations.
- **Objective 3:** Find if increasing the degree of the feedback function makes it less likely to find optimal functions.
- **Objective 4:** Test out if the inclusion of the stage x_0 in the non-linear part can generate any optimal period.

4.2 Experiment Methodologies

In this section it is introduced the methodology followed throughout the experiment. The experiment is mostly self contained. That is, the experiment data, execution, and analysis are performed as part of the study with minimum reliance on external resources. The general methodological approach of this work is to follow an empirical format. This work introduces a problem then uses experiments to come up with and analyze solutions. The problem being learning patterns about optimal feedback functions, while the solution of finding patterns is done through analyzing the results of experiments. Figure 4.1 gives a visual representation to the methodology used in this research.

4.2.1 Programming Methodology

The Period-Testing Suit, introduced in Chapter 3, is the main tool used for all programming work. The Suit is used for discovering optimal feedback functions and

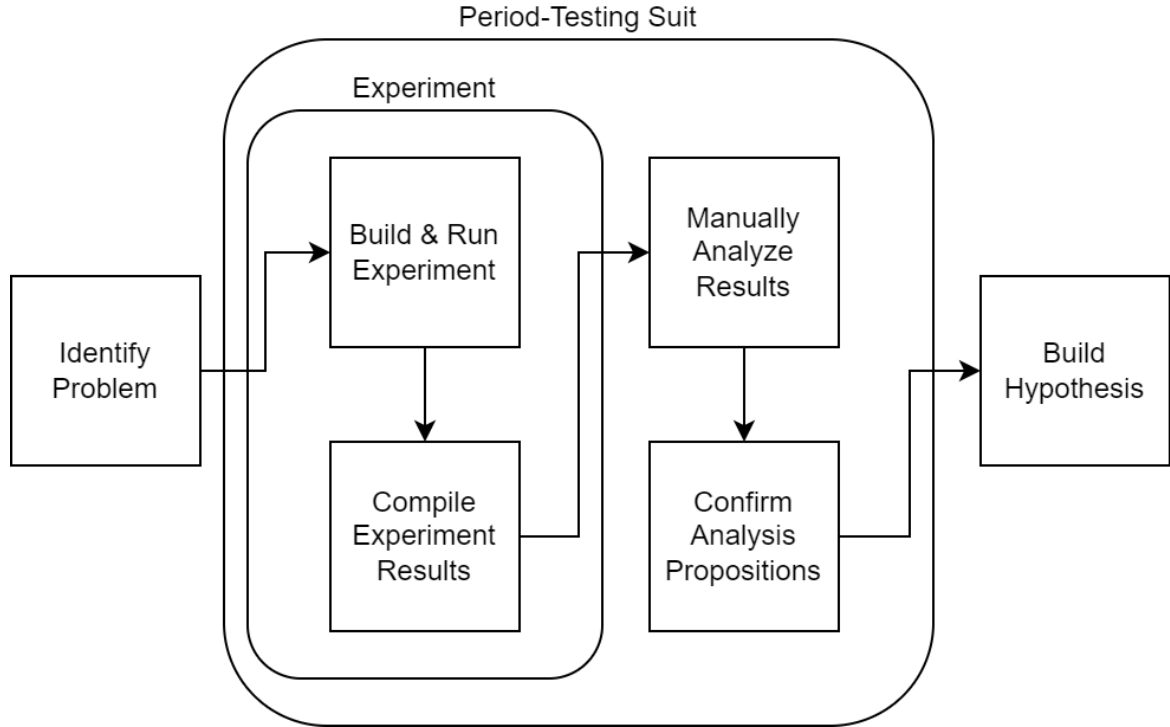


Figure 4.1: An illustration of the steps and components clarifying the methodology used in this research.

running queries against the results. The suit is built using python and the framework PeachPy [20] for writing assembly level optimized functions. Python is among the best choices for this type of projects, for the abundance of relevant useful tools and for the language features. Python’s Object Oriented Programming (OOP) and list comprehension features were specially helpful in developing both the period testing algorithm and for querying the results. The programming work is considered a big part of this study and it is already discussed in Chapter 3.

4.2.2 Data Presentation Methodology

Data presentation is a critical subject in this experiment. This experiment incorporates a manual analysis of the data as a tool to identify patterns. Data presentation

formats could make some patterns obvious while others vague when manually analyzed. In this experiment we use two different data presentation formats: the compact form and the visual form. Those formats has been formally presented in chapter 3. Each form highlights a different set of features. The compact form highlights the structure of the feedback function's terms, in both degree and content. The linear and non-linear parts are easy to be interpret and the terms' degrees are easy to be calculated and compared with neighboring terms. For example, given a feedback function represented in a mathematical form 4.1, its compact form given in equation 4.2 is easier to read and to identify the function's features.

$$f(x_0, x_1, x_2, x_3, x_4, x_5) = x_0 \oplus x_1 \oplus (x_2 \cdot x_3) \oplus (x_1 \cdot x_2 \cdot x_4) \quad (4.1)$$

$$[1, [2, 3], [1, 2, 4]] \quad (4.2)$$

The visual form, on the other hand, highlights the stages that are used in the feedback function with respect to their location in the register. The visual form uniquely highlighted some features which proved valuable in revealing multiple findings, later introduced in chapter 5. Using the compact form given in equation 4.3, which visually represent the function 4.1, it is easy to visualize the active stages and distribution of (XOR) and (AND) operations.

$$|x - o| - oo| - o - | - -o| - - - | \quad (4.3)$$

4.2.3 Data Collection Methodology

Data in this experiment are feedback functions with optimal periods. All data are self produced via utilizing the suit. Over 99 million optimal feedback functions were discovered as part of running this work's experiments. This large database of optimal feedback functions were subject to study. Furthermore, all publicly available optimal feedback functions from similar studies, [1] [2] [3] [4] [5] [6] [7], has been tested for optimality and were considered for observation in this study. It is important to mention that all optimal feedback functions with the constraints given in this study come in pairs. The second element of the pair is equal to the first, but reversed horizontally. Specifically, given a feedback function represented by equation 1.3. If $g(x_1, x_2, \dots, x_{k-1})$ is optimal, then $g(x_{k-1}, x_{k-2}, \dots, x_1)$ is also optimal [1]. This work counts each element of the pair as a unique function, because said pair produce different output. What and how data are collected are further explained in sections 4.4.

4.2.4 Data Analysis Methodology

Data analysis phase is split into two parts. The first part is manual data analysis. In this part, optimal feedback functions are grouped and manually analyzed to construct propositions. The second part is asserting propositions. In this second part, the output of the first part is verified for correctness using the Period-Testing Suit. The manual analysis step relies on observing patterns as highlighted by the two representation formats of feedback functions, the compact and the visual forms. The verification step is conducted via writing a program asserting the proposition's correctness utilizing the

database of optimal feedback functions created in this work. Having multiple data presentation formats helped in formulating multiple hypothesis, later introduced in chapter 5. Since the method of discovering patterns were largely relying on manual observation, it would make a grate extension to this study applying data science tools to the data published by this work. More on how the manual analysis was approached is explained in this section 4.4.

4.3 Experiment Plan

To conduct a good experiment that fulfills the target objectives while being accurate, a plan is prepared. The experiment aims to cover a comprehensive set of feedback function features bound by some rules. The rules are designed such that all feedback function features are explored uniformly. For that, all possible factors that could affect the NLFSR output is identified. The two main factors are the register size and the feedback function. The factors need to be simplified such that changing one factor should be as simple as incrementing or decrements its value. The register size is already a simple factor represented by an integer. Iterating through different register sizes is done simply via incrementing its integer representation. The feedback function is a complex structure, nevertheless, in chapter 1 it is presented the concept of FFT and FFF and in chapter 3 it is explained how to iterate over all feedback functions given an FFF. Using both established concepts, we can use FFTs and FFFs as our guide on how to group and iterate over all possible feedback functions bound by some rules.

Since our objectives vary, and since **Objective 2** is open ended, it is decided that the experiment shall be split into multiple parts. Each part will study a group of related FFT, each called a *series*. Every series will be focused on testing the effect of a particular set of factors in producing optimal periods. Every part of the experiment contributes toward fulfilling **Objective 2**.

4.4 Experiments Description

First, it is desired to construct a series that is made with no assumptions on what affects NLFSR's optimality. The first series studies every factor that can affect optimality assuming each factor is equally likely responsible for affecting optimality. The first and largest series is built such that it uniformly studies the effects of all parameters of FFFs. The first series is called *10-wires*. This series uses the *X wires* algorithm presented in Chapter 3, with $X = 10$ as an input. Thanks to the 10-wires series, it was established **The Optimal Type Hypothesis**, later presented in Chapter 5. Other series was constructed after observing the output of the 10-wires series. Unlike the 10-wires series which was testing all factors of optimality uniformly, other series targeted a particular factor for study. Finally a special series named *full-stages* is presented, which highlights an interesting pattern itself.

It is worth noting that in all series including series 10-wires, the established theories presented and proven in Chapter 1 which describes how to construct an optimal function are followed. With that being said, those theories has also been relaxed for a portion of the experiment. Relaxing these assumptions produced no

optimal results, further supporting the established theories. Both, presented theories in Chapter 1 as well as the relaxed section of this experiment can be used to fulfill **Objective 4**. All series, which constitute this work's experiment, are presented ahead.

4.4.1 Series 10-wires

Series 10-wires is defines as follows. Given FFF: $(l, [p_1, p_2, \dots, p_k]), n$ such that:

- $l \leq 10$
- $\sum_{p_1}^{p_k} \leq 10$
- $n \leq 11$

The 10-wires series is the largest series covering 410 FFTs and 1827 FFFs. Over 24 billion different functions were simulated, resulting the discovery of over 64 million optimal functions. This series treats all feedback function features uniformly. The linear terms count, the non-linear terms count and degree, and the register size were all bound by a uniform limit, 10, or 11 in case of the register size. This series is the direct output of the *3rd* algorithm of the suit, passing 10 as an argument. The name 10-wires comes from the physical analogy of trying all possible feedback functions in a physically implemented NLFSR such that up to 10 wires is used for both the linear and non linear parts, 10 wires each.

4.4.2 Series min-max

Series min-max is defines as follows. Given FFF: $(l, [p_1, p_2, \dots, p_k]), n$ such that:

First sub-series (1-max-max):

- $l = 1$
- $k = 2$
- $(p_1 = p_2) \geq n - 2$
- $n \leq \rho + 1 \leq 25$

Second sub-series (2-max):

- $l = 2$
- $k = 1$
- $p_1 \geq n - 2$

- $n \leq \rho + 1 \leq 25$

This series studies feedback functions where the functions' degrees are as large as possible while other factors are as low as possible. The name min-max is reflecting how in this series the linear part is minimized while the non-linear part, which determines the degree, is maximized. This series holds significance towards fulfilling **Objective 3**. It is expected that this series should yield relatively lower number of optimal feedback functions compared to other series that lowers the function's degree, however, it is found that this is not the case. The simple reasoning behind the belief that increasing the degree lowers the optimal functions count is that larger degrees terms are more likely to result in 0s in the feedback function which lowers the entropy and consequently lowers the chances of having an optimal period. The min-max series is split into two parts, the (1-max-max) and the (2-max). These series is constructed after observing the results of the 10-wires series. It is observed in the 10-wires series experiment an important distinction between even and odd numbered terms in a feedback function in determining optimality. Due to that observation, this series is split into two parts to study both even and odd variants. That observation has been made into a hypothesis and is presented in Chapter 5. Table B.1 presents a summary for the count of optimal functions per FFF in this series.

4.4.3 Series max-min

Series max-min is defines as follows. Given FFF: $(l, [p_1, p_2, \dots, p_k]), n$ such that:

- $l \leq 24$
 - if l is even
 - $k = 1$
 - $p_1 = 2$
 - $n \leq \rho + 1 \leq 26$
 - if l is odd
 - $k = 2$
 - $p_1 = p_2 = 2$
 - $n \leq \rho + 1 \leq 26$

This series studies feedback functions where the functions' linear terms are as large as possible while other factors are as low as possible. The name max-min refers to the linear part being maximized while the other factors: the non-linear part terms count, the degree, and the register size being minimized. This series holds significance towards fulfilling **Objective 3**. It is expected that this series should yield more optimal feedback functions than the (min-max) series, however, the results showed that it is the other way around. Furthermore, this series is the worst performing series in terms of the count of optimal functions. The simple reasoning behind the belief that lowering the degree increases the optimal functions count is that lower degrees has better truth table distribution of 0s and 1s. This better distribution lowers the bias of product terms resulting 0s which lowers the feedback function's entropy, hence lower the optimal feedback functions count. Table B.2 presents a summary for the count of optimal functions per FFF in this series.

4.4.4 Series max-max

Series max-max is defines as follows. Given FFF: $(l, [p_1, p_2, \dots, p_k]), n$ such that:

- $l \leq 24$
 - if l is even
 - $k = 1$
 - $p_1 = l$
 - $n \leq \rho + 1 \leq 26$
 - if l is odd
 - $k = 2$
 - $p_1 = p_2 = l$
 - $n \leq \rho + 1 \leq 26$

This series studies feedback functions where the functions' linear parts and degrees are both as large as possible while the non-linear terms count and the register sizes are as low as possible. The series name, max-max, is inspired by its definition. This series merges both (min-max) and (max-min) series. The max-max series studies whether patterns from previous series carries over to this series. This series holds significance in defining the boundaries of each pattern discovered so far. Table B.3 presents a summary for the count of optimal functions per FFF in this series.

4.4.5 Series min-mins

Series min-mins is defines as follows. Given FFF: $(l, [p_1, p_2, \dots, p_k]), n$ such that:

- $l \leq 2$
 - if l is even
 - k is odd, $k \leq 7$
 - $(p_1 = \dots = p_k) \leq 3$
 - $n \leq \rho + 5$

- if l is odd

- k is even, $k \leq 6$
- $(p_1 = \dots = p_k) \leq 3$
- $n \leq \rho + 5$

This series studies feedback functions where the functions' non-linear terms count is as large as possible, while other factors are as low as possible. The series name min-mins refers to the count of small degree non-linear terms being maximized and studied. This series studies the final remaining factor that could affect the optimality of feedback functions, the non-linear terms count. This series turns out to have the highest number of optimal feedback functions count among the other series. This series holds significant in highlighting the factor with the most positive effect on the yield of optimal feedback functions, the non-linear terms count. The higher non-linear terms count the larger the yield of optimal feedback functions in an FFF. Table B.4 presents a summary for the count of optimal functions per FFF in this series.

4.4.6 Series full-stages

Series full-stages is defines as follows. Given FFF: $(l, [p_1, p_2, \dots, p_k]), n$ such that:

- $l = n - 1$

- if l is even

- k is odd, $k \leq 7$
- $p_1 = \dots = p_k = 2$
- $n = \rho \leq 8$

- if l is odd

- k is even, $k \leq 8$
- $p_1 = \dots = p_k = 2$
- $n = \rho \leq 8$

This series studies feedback functions where the functions' linear part is equal to the register size minus one. This means that all the stages in the register are used in the feedback function in the linear part. The non-linear part of this series has low degree and high terms count. This is deliberately chosen to increase the chances of yielding optimal feedback functions. This series is challenging the unlikely case of having any optimal feedback function when the register size is as low as possible, in other words, when $n = \rho$. To highlight how unlikely this is, consider the 10-wires series which was a comprehensive and massive coverage of feedback function, only a single FFF was found to have optimal functions at $n = \rho$. Utilizing the knowledge gained from the previous series, it is able to overcome this challenge of finding FFF where optimal functions exist at $n = \rho$, and construct this series. Table B.5 presents a summary for the count of optimal functions per FFF in this series.

4.4.7 Series Remarks

Both breadth and depth were explored further beyond the original boundaries of each series. Breadth refers to exploring new feedback functions, and depth is increasing the register size. As a result of this expansion of boundaries, new patterns are discovered and old patterns are further verified. It is used the time needed to execute a particular expansion as the factor to determine whether or not to expand in that direction. The expansion involves widening one or more bounding criteria of the series such that factors to be changed are chosen based on which has the least cost. The cost of widening a factor is determined by the amount of time it takes to finish executing the experiment after the factor is changed.

CHAPTER 5

RESULTS AND FINDINGS

In this chapter the findings of the experiment are presented. These findings represent the main contribution of this work. The findings can be categorized into *rules* and *patterns*. Rules are a set of observed characteristics that are always satisfied throughout the experiment with no exceptions. Rules can be rigorously described mathematically. Since no proof is given for any rule, those rules are presented as *hypothesis*. Patterns, on the other hand, are likely occurrences of some characteristics. Patterns will be presented in plain English. Presented rules and patterns will be cross referenced with experiment series that supports them.

5.1 List of Findings

All findings will be listed in this section. Findings will be ordered by significance, such that the most significant finding is presented first.

5.1.1 Finding 1: Optimal Type Hypothesis

The **optimal Type hypothesis** is the main and most significant finding. The optimal Type hypothesis reveals if a given FFT can contain optimal functions or not. The optimal Type hypothesis is described as follows:

Definition 5.1.1 (The optimal Type hypothesis). Given FFT: $(l, [p_1, p_2, \dots, p_k]), n$.

An FFT that has an optimal feedback function must satisfy the following constraint:

- if l is even, then k must be odd
- if l is odd, then k must be even

It is a rule that a feedback function must have an even number of terms, including the x_0 term, in order for it to be optimal. The other factors: the terms' degrees and register size have no effect in determining optimality. The entire experiment data supports this hypothesis.

Using this hypothesis, we can reliably mark FFTs that can never contain optimal functions. This bit of knowledge, therefore, adds to literature's optimal function theory. Specifically, this hypothesis further describes the $g()$ function in equation 1.3. The $g()$ function now have to satisfy the Optimal Type Hypothesis as described in definition 5.1.1. That is, the $g()$ function must be structured such that it has an odd number of terms.

5.1.2 Finding 2: O-Slide Hypothesis

The **O-slide hypothesis** applies to FFFs satisfying the constraints presented in definition 5.1.2.

Definition 5.1.2 (O-slide hypothesis FFF constraints). FFFs: $(l, [p_1, p_2, \dots, p_k]), n$ that produce an O-slide pattern are bound by these constraints:

- l is even
- $k = 1$
- $p_1 = n - 2$

All optimal feedback functions associated with an FFF satisfying the O-slide constraints exist as a member of a well defined group of similar optimal feedback functions. Each member of the group share the same linear part definition and differ in the non-linear part definition. The non-linear part of optimal functions in a single group forms a unique visual artifact, a diagonal slide of missing os as traced from the functions' visual format. This diagonal slide of missing os is what aspired the name, the O-slide. Figure 5.1 shows an example of a group of functions forming an O-slide:

$[1, 5, [1, 2, 3, 4, 5]]$	x o	- o	- o	- o	x o	- -	1
$[1, 5, [1, 2, 3, 4, 6]]$	x o	- o	- o	- o	x -	- o	2
$[1, 5, [1, 2, 3, 5, 6]]$	x o	- o	- o	- -	x o	- o	3
$[1, 5, [1, 2, 4, 5, 6]]$	x o	- o	- -	- o	x o	- o	4
$[1, 5, [1, 3, 4, 5, 6]]$	x o	- -	- o	- o	x o	- o	5
$[1, 5, [2, 3, 4, 5, 6]]$	x -	- o	- o	- o	x o	- o	6

Figure 5.1: A group of feedback functions forming an O-slide.

An alternative definition to the O-slide pattern can be given as follows: An O-slide group is a group of optimal feedback functions that share the same linear part and are collectively representing all possible non-linear part variants. As an outcome to the O-slide pattern, it is sufficient to test a single feedback function of each O-slide group for optimality. Functions of an O-slide group are either all optimal or all not optimal. Therefore, it is possible to reliably guess the optimality of feedback functions

without testing them if they belong to an O-slide group where a member of that group is optimal. This finding presents the first ever method to determine feedback function optimality without testing its period. Another application of the hypothesis is that it can be used to accelerate the search for optimal functions associated with FFFs abiding the O-slide constraints given in definition 5.1.2. The speedup is measured as a factor of n , such that n is the register size. Both *2-max* and *max-max* series supports this hypothesis.

5.1.3 Finding 3: X-Gab Hypothesis

The **X-gab hypothesis** is a special case of the **O-slide hypothesis**. Definition 5.1.3, presents the FFF constraints binding the optimal feedback functions with the X-gab characteristics.

Definition 5.1.3 (X-gab hypothesis FFF constraints). FFFs: $(l, [p_1, p_2, \dots, p_k]), n$ that undergoes the X-gab pattern are bound by these constraints:

- $l = 2$
- $k = 1$
- $p_1 = n - 2$

Since the X-gab pattern is a special case of the O-slide pattern, the O-slide pattern can be observed alongside the X-gab pattern. Using the concept of the group as defined for the O-slide, the same members of the group are subject to the X-gab pattern if their associated FFF satisfies the X-gab constraints. The X-gab pattern describes a necessary rule for the linear part in optimal functions. First, the *gab* concept is defined. The gab is defined as the number of stages between the two stages marked with *xs* in the function's visual form. The X-gab pattern states that:

Optimal functions must have a gab with matching parity to the register size. For example, in an optimal function if the register size is even, the gab size must be even too. Utilizing this hypothesis, additional speed up can be achieved over the speed up already gained from the O-slide hypothesis by a factor of two. Henceforth, the speedup gained from utilizing the X-gab hypothesis is the sum of speedups from both the X-gab and the O-slide hypotheses, which is equal to a factor of $2n$, such that n is the register size. As a proof of concept, the X-gab hypothesis was used to accelerate the search for optimal function for FFFs with high register sizes, $24 \leq n \leq 32$. Utilizing the X-gab hypothesis, it was possible to achieve a speedup of up to 64 time, allowing the experiment to finish executing in under 30 minutes. The experiment resulted in discovering 3894 optimal feedback functions with large register sizes, $24 \leq n \leq 32$. Furthermore, those optimal functions are all the optimal functions for their respective FFF. Those large optimal functions are published alongside the results of the experiment. In the Figure 5.2, a gab of size 3 is separating the two stages with the x symbol, the two stages being stage x_1 and stage x_5 . In the Figure 5.2, the register size is 7 while the gab size is 3, both have matching parity of being odd. Series *2-max* supports this hypothesis.

This concludes the list of hypotheses constructed in this study, the rest of the findings, finding 4 through 9, are presenting likely occurring patterns.

5.1.4 Finding 4: Normal Distribution Pattern

Given a series of FFFs with consecutive register sizes. The count of optimal feedback functions per FFF vs the register size form a normally distributed graph. The normal

$[1, 5, [1, 2, 3, 4, 5]]$	xo	-o	-o	-o	xo	--	1
$[1, 5, [1, 2, 3, 4, 6]]$	xo	-o	-o	-o	x-	-o	2
$[1, 5, [1, 2, 3, 5, 6]]$	xo	-o	-o	--	xo	-o	3
$[1, 5, [1, 2, 4, 5, 6]]$	xo	-o	--	-o	xo	-o	4
$[1, 5, [1, 3, 4, 5, 6]]$	xo	-	-o	-o	xo	-o	5
$[1, 5, [2, 3, 4, 5, 6]]$	x-	-o	-o	-o	xo	-o	6

Figure 5.2: A group of feedback functions highlighting the X-gab pattern.

distribution patterns suggests that it is unlikely that an FFT exist such that it produces optimal period functions for all register sizes. The other observation from this pattern is that the optimal feedback function count is at its highest in an FFT when $n = \rho + X$. From the limited results obtained in this study, X is likely to be in the range of 4 to 10. Figure C.1 & C.2 illustrate the count of optimal periods given the FFTs: $(1, [2, 2]), n$ and $(2, [3]), n$ such that $\rho \leq n \leq \rho + 20$.

5.1.5 Finding 5: Exponential Decay Pattern

Given a group of FFFs with consecutive register sizes, the percentage of optimal functions per FFF vs the register size forms an exponentially decaying graph. This pattern suggests that the probability of finding an optimal function becomes exponentially less likely for bigger register sizes. Although the normal distribution pattern states that increasing the register size is likely to yield more optimal feedback functions, the exponential decay adds that the percentage of optimal feedback functions are exponentially lower in higher register sizes. The exponential decay pattern indicates that the efficiency of discovering new optimal functions drops exponentially for higher register

sizes. Furthermore, the execution time for period testing increases exponentially for higher register sizes. Therefore, the time cost for searching for optimal functions for higher register sizes is super exponential. Figure C.3 & C.4 illustrate the described pattern given the FFTs: $(1, [2, 2]), n$ and $(2, [3]), n$ such that $\rho \leq n \leq \rho + 20$.

5.1.6 Finding 6: Almost-O-Slide Pattern

The almost-O-slide pattern is observed in FFFs satisfying the following constraints:

Definition 5.1.4 (Almost-O-Slide pattern FFF constraints). FFFs: $(l, [p_1, p_2, \dots, p_k]), n$ that produce an Almost-O-slide pattern are bound by these constraints:

- l is odd
- $k = 2$
- $p_1 = p_2 = n - 2$

The almost-O-slide pattern is similar to the O-slide pattern. When observing both patterns, the visual properties of the slide in the almost-O-slide is partially respected. While the O-slide pattern can be visually identified via the existence of a diagonal slide of missing *os*, the almost-O-slide has a similar slide of missing *os*, however, the slide is not continuous. The discontinuity of the almost-O-slide is caused by the existence of non-optimal feedback functions in the slide group. In this work, no pattern has been found to identify which feedback functions in the group are not optimal. Figure 5.3 highlights the almost-O-slide pattern visually.

In Figure 5.3, feedback functions bound by green rows are part of a group. Every function in the group has two missing *os* where the latter is fixed per group. The initial missing *o* of each feedback function in a group forms a non-continuous slide.

X	o	-	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	1
X	oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	2
X	oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	3
X	oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	4
X	oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	5
X	oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	6
X	oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	7
X	oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	8
X	oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	9
X	oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	10
X	oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	11
X	oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	12
X	oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	13
X	oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	14
X	oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	15
X	oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	16
X	oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	17
X	oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	18
X	oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	19
X	oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	20
X	oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	21
X	oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	22
X	oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	23
X	oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	24
X	oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	25
X	oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	26
X	oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	27
X	oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	28
X	oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	-oo	29

Figure 5.3: Almost-O-slide pattern. Visual form analysis.

Across groups, the latter missing *o* forms yet another slide of missing *os* that itself is non-continuous. As highlighted in Figure 5.3, you can follow the red dashes in each green rectangle to trace the discontinuity of the slide.

An alternative definition to the almost-O-slide can be devised through analyzing the compact form of functions in a single group. Figure 5.4 highlights this pattern via grouping both the linear part and first product term of a function's compact form. This pattern can be used to speedup the search for optimal functions by means of skipping groups that are unlikely to have any optimal functions. However, due to the random nature of missing *os*, it is possible to skip a group with optimal functions. The following method has a 100% success rate in this study, but it is not guaranteed

4, [1, 2, 3, 4, 5, 6, 7]	[1, 2, 3, 4, 5, 6, 8]]	-oo	-oo	-oo	xoo	-oo	-oo	-o-	--o	1
4, [1, 2, 3, 4, 5, 6, 7]	[1, 2, 3, 4, 5, 7, 8]]	-oo	-oo	-oo	xoo	-oo	-o-	-oo	--o	2
4, [1, 2, 3, 4, 5, 6, 7]	[1, 2, 3, 4, 6, 7, 8]]	-oo	-oo	-oo	xoo	-o-	-oo	-oo	--o	3
4, [1, 2, 3, 4, 5, 6, 7]	[1, 2, 3, 5, 6, 7, 8]]	-oo	-oo	-oo	xo-	-oo	-oo	-oo	--o	4
4, [1, 2, 3, 4, 5, 6, 7]	[2, 3, 4, 5, 6, 7, 8]]	-o-	-oo	-oo	xoo	-oo	-oo	-oo	--o	5
4, [1, 2, 3, 4, 5, 6, 8]	[1, 2, 3, 4, 5, 7, 8]]	-oo	-oo	-oo	xoo	-oo	-o-	--o	-oo	6
4, [1, 2, 3, 4, 5, 6, 8]	[1, 2, 4, 5, 6, 7, 8]]	-oo	-oo	-o-	xoo	-oo	-oo	--o	-oo	7
4, [1, 2, 3, 4, 5, 6, 8]	[2, 3, 4, 5, 6, 7, 8]]	-o-	-oo	-oo	xoo	-oo	-oo	--o	-oo	8
4, [1, 2, 3, 4, 5, 7, 8]	[1, 2, 4, 5, 6, 7, 8]]	-oo	-oo	-o-	xoo	-oo	--o	-oo	-oo	9
4, [1, 2, 3, 4, 5, 7, 8]	[2, 3, 4, 5, 6, 7, 8]]	-o-	-oo	-oo	xoo	-oo	--o	-oo	-oo	10
4, [1, 2, 3, 4, 6, 7, 8]	[1, 2, 3, 5, 6, 7, 8]]	-oo	-oo	-oo	xo-	-o-	-oo	-oo	-oo	11
4, [1, 2, 3, 4, 6, 7, 8]	[2, 3, 4, 5, 6, 7, 8]]	-o-	-oo	-oo	xoo	--o	-oo	-oo	-oo	12
4, [1, 2, 3, 5, 6, 7, 8]	[2, 3, 4, 5, 6, 7, 8]]	-o-	-oo	-oo	x-o	-oo	-oo	-oo	-oo	13
5, [1, 2, 3, 4, 5, 6, 7]	[1, 2, 3, 4, 6, 7, 8]]	-oo	-oo	-oo	-oo	xo-	-oo	-oo	--o	14
5, [1, 2, 3, 4, 5, 6, 7]	[1, 2, 3, 5, 6, 7, 8]]	-oo	-oo	-oo	-o-	xoo	-oo	-oo	--o	15
5, [1, 2, 3, 4, 5, 6, 7]	[1, 2, 4, 5, 6, 7, 8]]	-oo	-oo	-o-	-oo	xoo	-oo	-oo	--o	16
5, [1, 2, 3, 4, 5, 6, 7]	[1, 3, 4, 5, 6, 7, 8]]	-oo	-o-	-oo	-oo	xoo	-oo	-oo	--o	17
5, [1, 2, 3, 4, 5, 6, 7]	[2, 3, 4, 5, 6, 7, 8]]	-o-	-oo	-oo	-oo	xoo	-oo	-oo	--o	18
5, [1, 2, 3, 4, 5, 7, 8]	[1, 2, 4, 5, 6, 7, 8]]	-oo	-oo	-o-	-oo	xoo	--o	-oo	-oo	19
5, [1, 2, 3, 4, 5, 7, 8]	[1, 3, 4, 5, 6, 7, 8]]	-oo	-o-	-oo	-oo	xoo	--o	-oo	-oo	20
5, [1, 2, 3, 4, 6, 7, 8]	[1, 2, 3, 5, 6, 7, 8]]	-oo	-oo	-oo	-o-	x-o	-oo	-oo	-oo	21
5, [1, 2, 3, 4, 6, 7, 8]	[2, 3, 4, 5, 6, 7, 8]]	-o-	-oo	-oo	-oo	x-o	-oo	-oo	-oo	22
5, [1, 2, 3, 5, 6, 7, 8]	[2, 3, 4, 5, 6, 7, 8]]	-o-	-oo	-oo	-o	xoo	-oo	-oo	-oo	23
5, [1, 2, 4, 5, 6, 7, 8]	[1, 3, 4, 5, 6, 7, 8]]	-oo	-o-	--o	-oo	xoo	-oo	-oo	-oo	24
5, [1, 2, 4, 5, 6, 7, 8]	[2, 3, 4, 5, 6, 7, 8]]	-o-	-oo	--o	-oo	xoo	-oo	-oo	-oo	25
5, [1, 3, 4, 5, 6, 7, 8]	[2, 3, 4, 5, 6, 7, 8]]	-o-	--o	-oo	-oo	xoo	-oo	-oo	-oo	26

Figure 5.4: Almost-O-slide pattern. Compact form analysis.

to have counter examples in other studies: First, Utilizing the outer o slide, the one across groups, search for feedback functions via fixing both the linear part and the first product term, such that the first product term is defined as $[1, 2, \dots, n - 2]$ then test the period for all variation of the second product term. If no optimal function is found, then the group most likely has no optimal feedback functions, and the search shall continue with incrementing the linear part. If a feedback function exist, the whole group has to be explored. The speedup achieved is a factor of $((n) * (n - 1))$ if no feedback functions is found in a group, else no speedup is achieved.

5.1.7 Finding 7: Empty x Closer to Edges Pattern

In the max-max series, optimal functions are more likely to have the stage with an empty x closer to the edges. In other words, the stage missing an x is likely to be x_1 or

x_{n-1} or a stage close to either. This pattern can be used to prioritize testing feedback functions that are more likely to be optimal. Table B.6 shows a list of all FFTs studied in the max-max series, highlighting the positions of the missing x s relative to the edges.

5.1.8 Finding 8: Increasing Degree Pattern

Contrary to intuition, increasing the degree of an FFF does not negatively affect the count of optimal functions. In fact, in many situations it's the contrary. That is, increasing the degree tends to increase the count of optimal feedback functions found for each FFF. For example, FFFs: $(2, [2, 2, 7]), 9$ & $(2, [6, 5, 7]), 9$ have optimal function count of 1,492 & 16,782 respectively. On the other hand, FFFs: $(2, [4, 4, 4]), 8$ & $(2, [3, 4, 4]), 8$ have optimal function count of 8,778 & 21,868 respectively. Increasing the terms' degrees is likely to yield more optimal functions, however, there are many counter examples discovered in this study, one is given earlier. Series min-max and max-max supports this pattern.

5.1.9 Finding 9: Increasing Product Terms Pattern

Increasing the product terms has the strongest effect over optimal functions count. The more product terms added the higher the count of optimal functions. This is expected, because each added term increases the count of associated feedback functions for a given FFF. The series min-mins and full-stages supports this pattern. For example, consider FFFs: $(2, [2, 2, 2, 2, 2]), 8$ & $(2, [3, 3, 3, 3, 3]), 8$ & $(2, [2]), 8$, the optimal function count is 12,768 & 261,580 & 30 respectively. This example supports that

adding more product terms significantly boosts the optimal function count regardless of the degree used. No counter example to this pattern was found where the FFF with higher product terms count also has higher associate feedback functions.

5.1.10 Finding 10: Dense FFFs

Dense FFFs are those containing the highest percentage of optimal feedback functions for a given register size. Searching a dense FFF is the most time efficient way to find optimal feedback functions. In this work, dense FFFs are presented as list of FFFs ordered from most dense to least dense. Each FFF in the ordered list is given a density rank based on its occurrence in the list. Therefore, the FFF with density rank 1 has the highest expected density of optimal feedback functions followed by FFF with density rank 2 and so on. To construct the list of dense FFFs, multiple set of rules are followed in their respective order. The following are the rules used to construct the list of dense FFFs given a register of size n :

Definition 5.1.5 (The construction of dense FFFs). Given a register size: n . The following rules are followed in their respective order to construct the list of dense FFFs:

1. All product terms has the highest possible degree minus one.
2. The product terms count is as low as possible.
3. The linear terms count is either as high as possible minus one or as low as possible.

Expressing the aforementioned rules using FFF notation and i , such that $i \in \mathbb{N}$, gives the following ordered rules. Given FFT: $(l, [p_1, p_2, \dots, p_k]), n$:

1. $p_1 = p_2 = n - i - 1$
2. If l is even then $k = i$, else $k = 2i$
3. $l = n - i - 1$ or $l = i$

To generate the list of dense FFFs, iterate over the values of i ascendingly until no longer possible. For example, table 5.1 previews the top 20 ranked FFF for register size $n = 18$.

Density Rank	FFF	Density Rank	FFF
1	$(16, [16]), 18$	11	$(11, [16, 16]), 18$
2	$(1, [16, 16]), 18$	12	$(6, [16]), 18$
3	$(15, [16, 16]), 18$	13	$(10, [16]), 18$
4	$(2, [16]), 18$	14	$(7, [16, 16]), 18$
5	$(14, [16]), 18$	15	$(9, [16, 16]), 18$
6	$(3, [16, 16]), 18$	16	$(8, [16]), 18$
7	$(13, [16, 16]), 18$	17	$(16, [16, 16, 16]), 18$
8	$(4, [16]), 18$	18	$(1, [16, 16, 16, 16]), 18$
9	$(12, [16]), 18$	19	$(15, [16, 16, 16, 16]), 18$
10	$(5, [16, 16]), 18$	20	$(2, [16, 16, 16]), 18$

Table 5.1: First 20 elements of the density list for FFFs with register size $n = 18$.

Weaknesses: Two noteworthy weaknesses against the constructed list of dense FFFs describe in this finding. First weakness is that the rules given to construct the list of dense FFFs may result in a list with locally misplaced FFF in terms of density ranks, with a rank value error up to the register size. For example, instead of the highest density FFF being the one with rank 1, it can be anywhere in the range of (1 to n) such that n is the register size. The second weakness is that higher ranked density FFFs can produce no optimal feedback functions, contrary to other lower ranked density FFFs. However, this is only true when the number of feedback

function members of the higher ranked FFF is significantly lower than that of FFFs with neighboring density rank. For example, $\text{FFF} = (17, [17, 17]), 19$ is calculated as the highest density rank for register size $n = 19$, however, it has no optimal functions. It is the FFF with rank 3 = $(15, [17, 17]), 19$ which has the actual highest density for register size $n = 19$. The count of feedback function members of $\text{FFF} (17, [17, 17]), 19$ is 2,754, while the count of members of $\text{FFF} (15, [17, 17]), 19$ is 124,848. Unfortunately, there seem to be no clear pattern to when high density ranked FFFs contain no optimal feedback functions other than the relative members count compared to neighboring FFFs.

5.2 Limitations

Though the findings are worthy additions to the studied subject, many are limited in terms of applicability and scale. The O-slide and X-gab hypothesis only apply to a very specific series of feedback functions. And the rest of the patterns only slightly help when applying heuristical decisions for searching for optimal functions. No break through finding is found that allows constructing optimal NLFSRs without testing at least one period. The O-slide hypothesis comes close to allowing the construction of an optimal feedback function without testing its period, however, an equivalent period size has to be tested for optimality first. Therefore, most of the findings don't provide significant help for registers of sizes above 64 bits. With that being said, the optimal type hypothesis, is a strong universal finding that allows for an appropriate construction of feedback functions that can be optimal.

5.3 RQs & RAs

The *Research Questions* and their *Answers* are presented in this section. The research questions are:

- **RQ1:** Is there a formula that can be used to determine if a given FFT can contain optimal feedback functions?
- **RQ2:** Are there some rules or patterns that can be used to accelerate the search for functions with optimal periods?
- **RQ3:** Is it beneficial to increase the degree of the feedback function and is the higher the degree the less likely it is to find optimal configurations?
- **RQ4:** Is the inclusion of the stage x_0 (the output stage), as an active variable in the feedback function, or using it as a product term only, generate an optimal period?

Research answers, **RAs**, are:

- **RA1:** Yes, the formula is presented In 5.1.1.
- **RA2:** Yes, the rules and patterns and how they can be used to accelerate the search are presented in the list of findings, in section 5.1.
- **RA3:** Contrary to intuition, increasing the degree and or the product terms count actually help in increasing the count of optimal functions. A more detailed discussion is presented in the findings 8 and 9, in subsections 5.1.8 & 5.1.9.

- **RA4:** No, it can't. It has already been shown that there are no optimal results from including the stage x_0 as an active variable. It is presented a proof, in Chapter 1, that this can never produce optimal results. The experiment rules were relaxed to include x_0 as an active variable and that showed no optimal results, further supporting the proof and established theories.

CHAPTER 6

THE CONCLUSION

In this last chapter, a brief summary highlighting the most important aspects of this work is given. First, it is re-introduced this work's purpose and goals. Then, it is summarized the accomplishments and their impact. Finally, it is highlighted potential areas of improvements and future work.

6.1 Thesis Work

This work is a continuation to [1] [2] [3]. This work managed to add valuable knowledge to the subject of NLFSR's optimal period. The findings of this work are made possible thanks to the Period-Testing suit built as part of this work; the suit is introduced in Chapter 3. The suit is flexible and efficient, it made running a large experiment feasible; the experiment is introduced in Chapter 4. Analysing the results of this work's experiment, multiple findings were observed and are presented as the main contributions of this thesis; the findings are introduced in Chapter 5. The most notable finding is a hypothesis that presents a necessary but not sufficient rule that

FFTs has to abide to contain feedback functions that are optimal. Multiple other hypothesis and patterns are presented, for which they can help in accelerating the search for optimal feedback functions. Finally, the suit as well as the experiment data are made available publicly. This allows easy access to the results and smoother continuation of this work by the research community.

6.2 The Work's Impact

The contributions presented advances our understanding and knowledge on the subject of this work. The literature benefit such that future research can use the hypothesis presented in this work to better construct optimal feedback functions and to accelerate the search for optimal functions. In cryptography, stream cipher designers can benefit from the outputs of this work to strengthen their implementation of NLFSR. The published Period-Testing suit and experiment data makes for an excellent foundation for future work on this subject.

6.3 Future Work

3 main routs can be pursued as a continuation to this study. Different disciplinary knowledge and skill are required for each rout.

The first rout is: Using data science tools, further explore other rules and patterns. The methodology used in identifying patterns and rules in this study is largely manual. Employing data science techniques is likely to result in further discoveries. This rout of fetching more patterns could yield the ultimate finding: A feasible

method to deterministically construct large optimal NLFSRs. Such discovery will cause a breakthrough in stream cipher design. Many popular stream ciphers ignore the existence of short cycles in their NLFSR components, which lead to the discovery of the recent short period exploit published in [8] [9]. The ability to construct large optimal NLFSR obsoletes this new exploit.

The second rout is: Prove or disprove the hypothesis presented in this study. Doing so elevates them to become theorems, giving us further confidence in them. There are 3 hypothesis, the **Optimal Type Hypothesis**, the **O-Slide Hypothesis**, and the **X-Gab Hypothesis**. Among them, the **O-Slide Hypothesis**, and the **X-Gab Hypothesis** could be proven via applying a step by step analysis of the NLFSR's state (which the suit provides). Researchers with good mathematical background on the subject could find this an appealing topic for their next research work.

The third rout is: Tweak the goal of this study, from finding optimal functions to finding functions with some optimal threshold. That is, for example, study all functions that are more than 80% optimal. Then attempt to formulate a mathematical model to deterministically construct those functions. This is a more open ended rout, and can be considered as an alternative take on this study. Being able to achieve this rout's goal, causes a similar breakthrough in stream cipher design as the first rout.

6.4 Final Words

That's it dear reader, it's your investment in reading my work that gives it its value. And keep in mind that (also quote of the thesis):

"You may not know, but you can try to compute the answer" - Mahdi, 2022

APPENDIX A

PSEUDO CODES

```
1 # if there is a next function, it's returned, else the same
   function is returned
2 # ff is the feedback_function
3 def nextFunction(ff):
4     for ff.terms descending = e1
5         if canMoveTermAhead(e1)
6             moveAhead(e1)
7             for termsInFrontOf(e1) = e12
8                 reset(e12)
9             break
10    return ff
11 # ff.terms: all terms in ff, in the format used in this work
12
13 def canMoveTermAhead(term):
14     for term.vars descending = e1
```

```

15         if canMoveVariableAhead(el)
16             return true
17     return false
18 # term.vars: all variable in this term as they show in ff
19
20 def canMoveVariableAhead(var):
21     limit = ff.RegSize - var.term.invOrder - var.invOrder
22     return var < limit
23 # ff.RegSize: the register size used alongside this ff
24 # var.term.invOrder: the variable's term inverse order vs all
    other terms. for example, [a,b,(c,d),(e,f)], the order of (e,f)
    is 0, (c,d) is 1, the order reset in linear part, henceforth,
    the order of b is 0, a is 1.
25 # var.invOrder: the variable's inverse order vs other variables in
    this term. for example, [a,b,(c,d),(e,f)], the order of f,d,b,
    a is 0, e,c is 1.
26
27 def moveAhead(term):
28     for term.vars descending = el
29         if canMoveVariableAhead(el)
30             el = el + 1
31     return
32
33 def termsInFrontOf(term):

```

```

34     return ff.terms where each.order > term.order
35 # term.order: the variable's term order vs all other terms. for
    example, [a,b,(c,d),(e,f)], a is 0, b is 1, (c,d) is 2, and (e,
    f) is 3.
36
37 def reset(term):
38     if term.previousTerm.degree != term.degree
39         defaultReset(term)
40     else
41         resetWithMoveAhead(term)
42 # term.previousTerm: the other term which has order = (this.term.
    order - 1)
43 # term.degree: this terms degree as defined in this work
44
45 def defaultReset(term):
46     term = list.ascending(term.varCount)
47 # list = [1,2,3,...,term.varCount]
48
49 def resetWithMoveAhead(term):
50     duplicate term.previousTerm = tmp
51     term = tmp.moveAhead()
52 # duplicate to not affect term.previousTerm

```

Listing A.1: Next function given a feedback function

```

1 # loop over / visit all FTs linked with some constraints, defined
   via the integer 'limit'.
2 def wires(limit):
3     maxRegSize = limit + 1
4     for a in range(1, limit):
5         for nonLinearPart in getNonLinearLists()
6             for c in range(alpha, maxRegSize):
7                 visit FT(a, nonLinearPart, c)
8 # alpha as defined in this study is the max of 'a' and all
   nonLinearPart.elements
9
10 def getNonLinearLists():
11     result = [2]
12     while true:
13         if result.cascadeIncrement() is not overTheLimit:
14             result.add(result.cascadeIncrement())
15         else if result.increaseSize() <= limit:
16             result.add(result.increaseSize())
17         else
18             return result
19
20 def list.cascadeIncrement():
21     for list.elements descending = el
22         if list.cascadeIncrement(el).sum <= limit:

```

```

23         return list.cascadeIncrement(el)
24     return overTheLimit
25 # list.sum: the sum of all elements of list
26 # overTheLimit: constant used for logic flow
27
28 def list.cascadeIncrement(el):
29     el = el + 1
30     for i in range (el.index + 1, list.size)
31         list[i] = el
32     return new list # the original list is unchanged
33
34 def list.increaseSize():
35     newList = createNewList(size= (list.size + 1))
36     newList.setAllElements(2)
37     return newList

```

Listing A.2: X Wires algorithm

APPENDIX B

TABLES

1-max-max	$n = \rho$	$n = \rho + 1$	2-max	$n = \rho$	$n = \rho + 1$
$(1, [2, 2]), n$	0	4	$(2, [2]), n$	0	6
$(1, [3, 3]), n$	0	22	$(2, [3]), n$	0	8
$(1, [4, 4]), n$	0	8	$(2, [4]), n$	0	20
$(1, [5, 5]), n$	0	26	$(2, [5]), n$	0	24
$(1, [6, 6]), n$	0	0	$(2, [6]), n$	0	56
$(1, [7, 7]), n$	0	26	$(2, [7]), n$	0	32
$(1, [8, 8]), n$	0	32	$(2, [8]), n$	0	54
$(1, [9, 9]), n$	0	44	$(2, [9]), n$	0	40
$(1, [10, 10]), n$	0	0	$(2, [10]), n$	0	22
$(1, [11, 11]), n$	0	0	$(2, [11]), n$	0	0
$(1, [12, 12]), n$	0	66	$(2, [12]), n$	0	182
$(1, [13, 13]), n$	0	190	$(2, [13]), n$	0	84
$(1, [14, 14]), n$	0	0	$(2, [14]), n$	0	300
$(1, [15, 15]), n$	0	228	$(2, [15]), n$	0	64
$(1, [16, 16]), n$	0	84	$(2, [16]), n$	0	272
$(1, [17, 17]), n$	0	0	$(2, [17]), n$	0	180
$(1, [18, 18]), n$	0	116	$(2, [18]), n$	0	114
$(1, [19, 19]), n$	0	114	$(2, [19]), n$	0	320
$(1, [20, 20]), n$	0	134	$(2, [20]), n$	0	252
$(1, [21, 21]), n$	0	358	$(2, [21]), n$	0	176
$(1, [22, 22]), n$	0	0	$(2, [22]), n$	0	736
$(1, [23, 23]), n$	0	340	$(2, [23]), n$	0	192

Table B.1: Series min-max: (1-max-max) & (2-max).

max-min	$n = \rho$	$n = \rho + 1$
$(1, [2, 2]), n$	0	8
$(2, [2]), n$	0	6
$(3, [2, 2]), n$	0	16
$(4, [2]), n$	0	8
$(5, [2, 2]), n$	0	40
$(6, [2]), n$	0	12
$(7, [2, 2]), n$	0	64
$(8, [2]), n$	0	4
$(9, [2, 2]), n$	0	74
$(10, [2]), n$	0	0
$(11, [2, 2]), n$	0	22
$(12, [2]), n$	0	2
$(13, [2, 2]), n$	0	22
$(14, [2]), n$	0	0
$(15, [2, 2]), n$	0	10
$(16, [2]), n$	0	0
$(17, [2, 2]), n$	0	2
$(18, [2]), n$	0	0
$(19, [2, 2]), n$	0	0
$(20, [2]), n$	0	0
$(21, [2, 2]), n$	0	0
$(22, [2]), n$	0	0
$(23, [2, 2]), n$	0	0
$(24, [2]), n$	0	0

Table B.2: Series max-min.

max-max	$n = \rho$	$n = \rho + 1$
$(2, [2]), n$	0	6
$(3, [3, 3]), n$	0	22
$(4, [4]), n$	0	10
$(5, [5, 5]), n$	0	44
$(6, [6]), n$	0	14
$(7, [7, 7]), n$	0	42
$(8, [8]), n$	0	18
$(9, [9, 9]), n$	0	0
$(10, [10]), n$	0	44
$(11, [11, 11]), n$	0	162
$(12, [12]), n$	0	52
$(13, [13, 13]), n$	0	352
$(14, [14]), n$	0	30
$(15, [15, 15]), n$	0	244
$(16, [16]), n$	0	34
$(17, [17, 17]), n$	0	0
$(18, [18]), n$	0	38
$(19, [19, 19]), n$	0	0
$(20, [20]), n$	0	42
$(21, [21, 21]), n$	0	1040
$(22, [22]), n$	0	0
$(23, [23, 23]), n$	0	606
$(24, [24]), n$	0	50

Table B.3: Series max-max.

min-mins	$n = \rho$	$n = \rho + 1$	$n = \rho + 2$	$n = \rho + 3$	$n = \rho + 4$	$n = \rho + 5$
$(2, [2]), n$	0	6	8	16	14	30
$(2, [2, 2, 2]), n$	0	2	30	138	350	956
$(2, [2, 2, 2, 2, 2]), n$	0	0	6	290	2660	12768
$(2, [2, 2, 2, 2, 2, 2, 2]), n$	0	0	0	126	5710	70884
$(2, [3]), n$	0	8	16	22	44	28
$(2, [3, 3, 3]), n$	0	8	182	1276	6032	15124
$(2, [3, 3, 3, 3, 3]), n$	0	0	362	17594	261580	1947020
$(2, [3, 3, 3, 3, 3, 3, 3]), n$	0	0	198	87256	5055650	115054732
$(1, [2, 2]), n$	0	4	18	26	48	36
$(1, [2, 2, 2, 2]), n$	0	0	20	86	566	1230
$(1, [2, 2, 2, 2, 2, 2]), n$	0	0	4	134	2082	10796
$(1, [3, 3]), n$	0	0	22	28	186	196
$(1, [3, 3, 3, 3]), n$	0	0	4	134	3778	14810
$(1, [3, 3, 3, 3, 3, 3]), n$	0	0	0	126	26270	419062

Table B.4: Series min-mins.

full-stages	$n = \rho$
$(3, [2, 2]), n$	2
$(3, [2, 2, 2, 2]), n$	0
$(3, [2, 2, 2, 2, 2, 2]), n$	0
$(3, [2, 2, 2, 2, 2, 2, 2]), n$	0
$(4, [2]), n$	0
$(4, [2, 2]), n$	2
$(4, [2, 2, 2, 2, 2]), n$	2
$(4, [2, 2, 2, 2, 2, 2]), n$	0
$(5, [2, 2]), n$	0
$(5, [2, 2, 2, 2]), n$	14
$(5, [2, 2, 2, 2, 2, 2]), n$	40
$(5, [2, 2, 2, 2, 2, 2, 2]), n$	14
$(6, [2]), n$	0
$(6, [2, 2]), n$	0
$(6, [2, 2, 2, 2, 2]), n$	62
$(6, [2, 2, 2, 2, 2, 2]), n$	304
$(7, [2, 2]), n$	0
$(7, [2, 2, 2, 2]), n$	44
$(7, [2, 2, 2, 2, 2, 2]), n$	1086
$(7, [2, 2, 2, 2, 2, 2, 2]), n$	5840
$(8, [2]), n$	0
$(8, [2, 2]), n$	0
$(8, [2, 2, 2, 2, 2]), n$	54
$(8, [2, 2, 2, 2, 2, 2]), n$	4594

Table B.5: Series full-stages.

max-max	from last
$(2, [2])$, 4	0
$(3, [3, 3])$, 5	0, 1
$(4, [4])$, 6	1
$(5, [5, 5])$, 7	0, 2
$(6, [6])$, 8	0
$(7, [7, 7])$, 9	1
$(8, [8])$, 10	3
$(9, [9, 9])$, 11	-
$(10, [10])$, 12	0, 4
$(11, [11, 11])$, 13	0, 2
$(12, [12])$, 14	1, 3
$(13, [13, 13])$, 15	0, 3, 6
$(14, [14])$, 16	2
$(15, [15, 15])$, 17	3, 4
$(16, [16])$, 18	1
$(17, [17, 17])$, 19	-
$(18, [18])$, 20	0
$(19, [19, 19])$, 21	-
$(20, [20])$, 22	5
$(21, [21, 21])$, 23	0, 5, 7, 9
$(22, [22])$, 24	-
$(23, [23, 23])$, 25	3, 8
$(24, [24])$, 26	1

Table B.6: Empty xs closer to edges pattern.

APPENDIX C

FIGURES

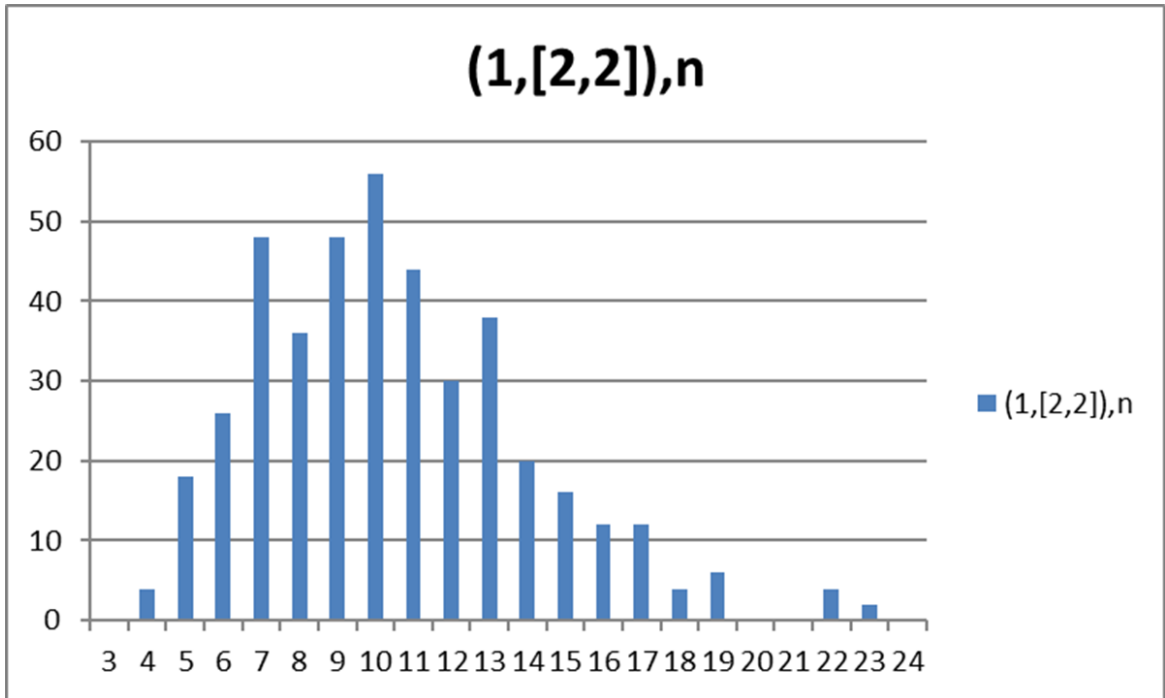


Figure C.1: Count of optimal functions in FFF: $(1, [2, 2]), n$; where $3 \leq n \leq 24$. Forms a rough *normally distributed* graph.

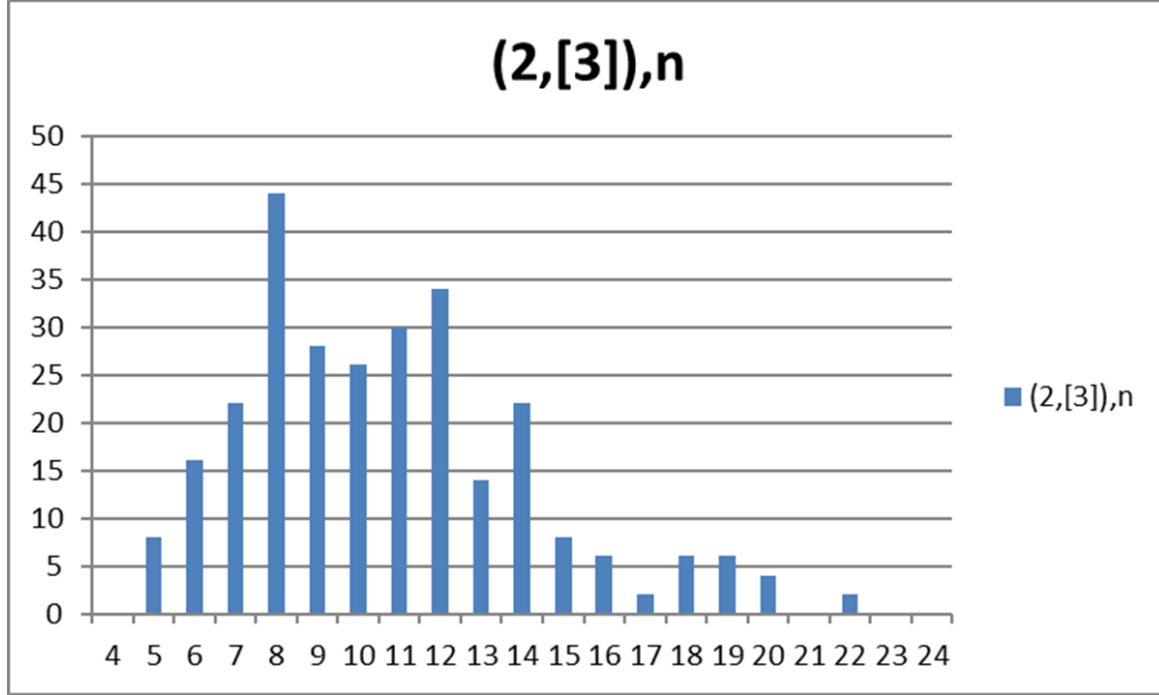


Figure C.2: Count of optimal functions in FFF: $(2,[3]),n$; where $4 \leq n \leq 24$. Forms a rough *normally distributed* graph.

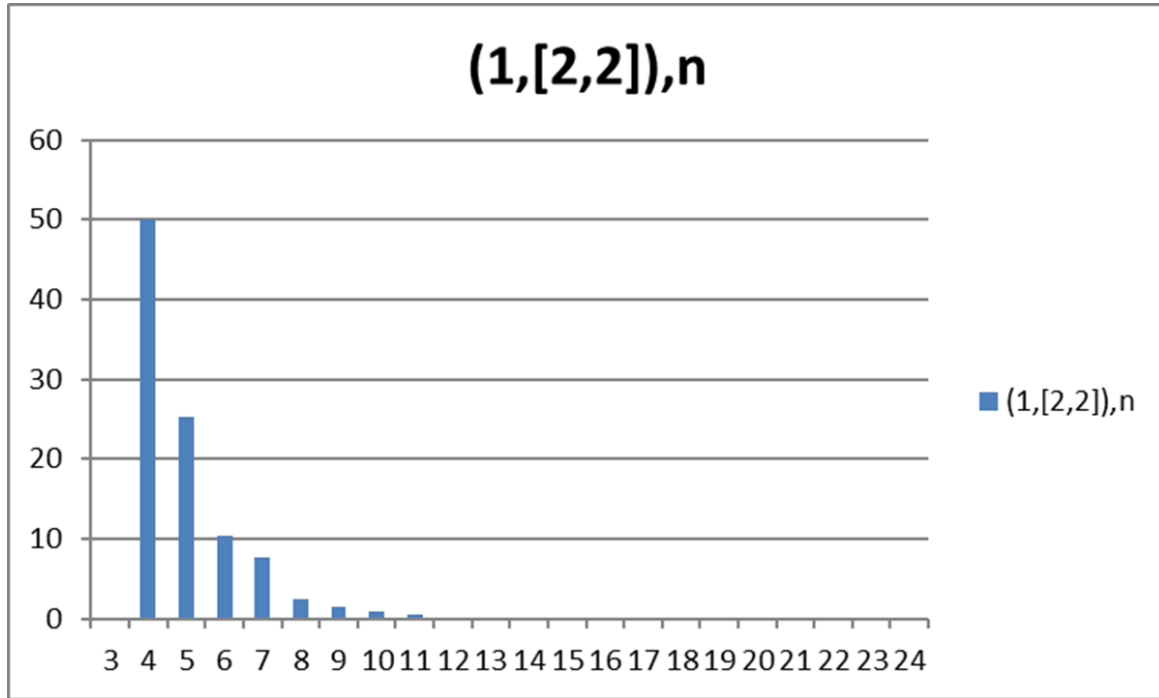


Figure C.3: Probability percentage of finding optimal period in FFF: $(1,[2,2]),n$; where $3 \leq n \leq 24$. Forms a rough *exponentially decaying* graph.

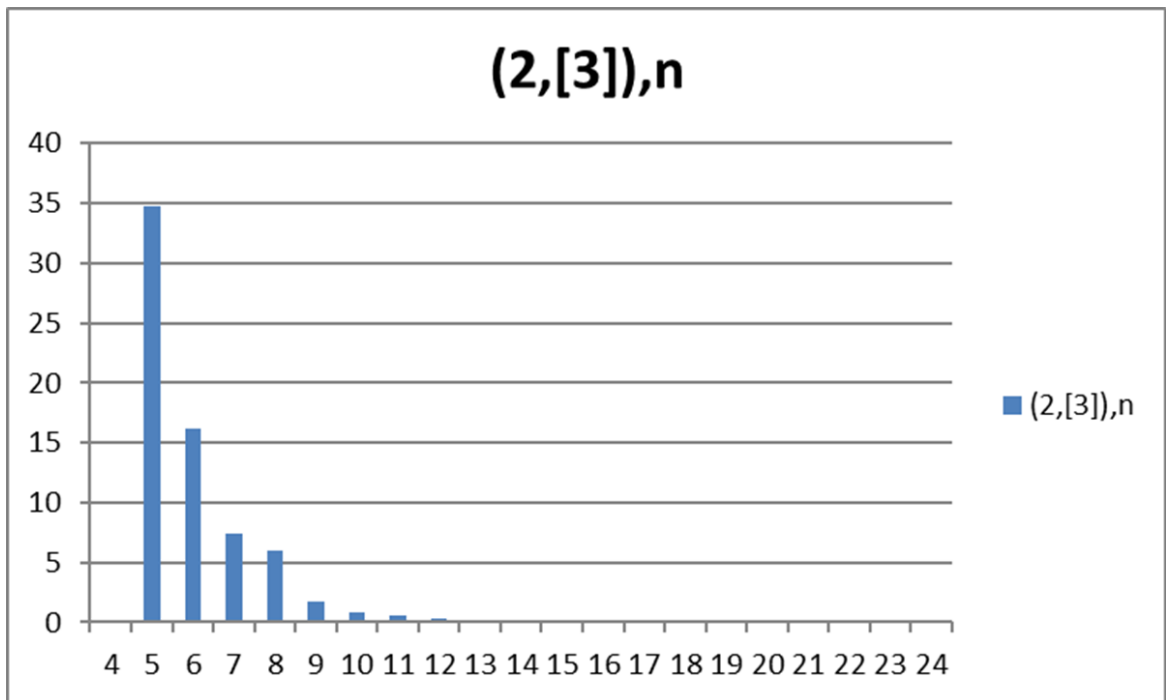


Figure C.4: Probability percentage of finding optimal period in FFF: $(2, [3]), n$; where $4 \leq n \leq 24$. Forms a rough *exponentially decaying* graph.

REFERENCES

- [1] E. Dubrova, *A List of Maximum-Period NLFSRs*. Springer, 2012. [Online]. Available: <https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A812512&dswid=-6162>
- [2] S. Almuhammadi, I. Al-Hejri, G. B. Talib, and A. Gaamel, “NLFSR Functions with Optimal Periods,” in *Computational Science and Its Applications – ICCSA 2018*. Cham, Switzerland: Springer, Jul 2018, pp. 67–79.
- [3] I. Al-Hejri and S. Almuhammadi, “Constructing New NLFSR Functions with Optimal Periods,” *IJITN*, vol. 12, no. 2, pp. 71–80, Apr 2020.
- [4] T. Rachwalik, J. Szmidt, R. Wicik, and J. Zabłocki, “Generation of nonlinear feedback shift registers with special-purpose hardware,” in *2012 Military Communications and Information Systems Conference (MCC)*. IEEE, Oct 2012, pp. 1–4. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/6387909>
- [5] P. Dąbrowski, G. Łabuzek, T. Rachwalik, and J. Szmidt, “Searching for nonlinear feedback shift registers with parallel computing,” *Inf. Process. Lett.*, vol. 114, no. 5, pp. 268–272, May 2014.

- [6] N. Poluyanenko, “Development of the search method for non-linear shift registers using hardware, implemented on field programmable gate arrays,” *EUREKA: Physics and Engineering*, no. 1, pp. 53–60, 2017.
- [7] P. Augustynowicz and K. Kanciak, “The search of square m-sequences with maximum period via GPU and CPU,” 2019, [Online; accessed 21. Nov. 2021]. [Online]. Available: <http://real.mtak.hu/131189>
- [8] E. Dubrova and M. Teslenko, “An efficient SAT-based algorithm for finding short cycles in cryptographic algorithms,” in *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, Apr 2018, pp. 65–72.
- [9] W. Dudzic and K. Kanciak, “Using SAT solvers to finding short cycles in cryptographic algorithms,” *International Journal of Electronics and Telecommunications*, vol. 66, no. 3, pp. 443–448, Sep 2020. [Online]. Available: <http://ijet.ise.pw.edu.pl/index.php/ijet/article/view/10.24425-ijet.2020.131897>
- [10] S. W. Golomb, “Shift Register Sequences – A Retrospective Account,” in *Sequences and Their Applications – SETA 2006*. Berlin, Germany: Springer, 2006, pp. 1–4.
- [11] E. Dubrova, “A Scalable Method for Constructing Galois NLFSRs With Period $2^n - 1$ Using Cross-Join Pairs,” *IEEE Trans. Inf. Theory*, vol. 59, no. 1, pp. 703–709, Aug. 2012.
- [12] —, “Generation of full cycles by a composition of NLFSRs,” *Des. Codes Cryptogr.*, vol. 73, no. 2, pp. 469–486, Nov. 2014.

- [13] E. Dubrova and M. Hell, “Espresso: A stream cipher for 5G wireless communication systems,” *Cryptogr. Commun.*, vol. 9, no. 2, pp. 273–289, Mar. 2017.
- [14] G. Yao and U. Parampalli, “Cryptanalysis of the class of maximum period galois NLFSR-based stream ciphers,” *Cryptogr. Commun.*, vol. 13, no. 5, pp. 847–864, Sep. 2021.
- [15] L. Zhiqiang, “The Transformation from the Galois NLFSR to the Fibonacci Configuration,” in *2013 Fourth International Conference on Emerging Intelligent Data and Web Technologies*. IEEE, Sep. 2013, pp. 335–339.
- [16] D. Zhao, H. Peng, L. Li, S. Hui, and Y. Yang, “Novel way to research nonlinear feedback shift register,” *Sci. China Inf. Sci.*, vol. 57, no. 9, pp. 1–14, Sep. 2014.
- [17] J. Lu, M. Li, T. Huang, Y. Liu, and J. Cao, “The transformation between the Galois NLFSRs and the Fibonacci NLFSRs via semi-tensor product of matrices,” *Automatica*, vol. 96, pp. 393–397, Oct. 2018.
- [18] G. Yao and U. Parampalli, “Improved transformation algorithms for generalized Galois NLFSRs,” *Cryptogr. Commun.*, vol. 14, no. 2, pp. 229–258, Mar. 2022.
- [19] —, “Generalized NLFSR Transformation Algorithms and Cryptanalysis of the Class of Espresso-like Stream Ciphers,” *arXiv*, Nov. 2019.
- [20] Maratyszczka, “PeachPy,” Nov 2021, [Online; accessed 23. Nov. 2021]. [Online]. Available: <https://github.com/Maratyszczka/PeachPy>

VITAE

- Name: Mahdi Saleh Hassan Al-Ibrahim
- Nationality: Saudi
- Date of Birth: 7/7/1993
- Email: *m313@live.com*
- Permenant Address: Saudi Arabia, Eastern Province, Safwa, Postal Code: 32833
- Academic Background: B.Sc. in Software Engineering, taken from KFUPM, at May 2016
- Research Areas of Interest: Cryptography, Post Quantum Cryptography, Block Chain Technologies