Construction of Pseudorandom Number Generators

Thesis for the Degree of Doctor of Philosophy (PhD)

by Viktória Padányi

SUPERVISOR: DR. TAMÁS HERENDI

University of Debrecen
Doctoral Council of Natural Sciences and Information Technology
Doctoral School of Informatics
Debrecen, 2024.

Hereby I declare that I prepared this thesis within the Doctoral Council of Natural Sciences and Information Technology, Doctoral School of Informatics, University of Debrecen in order to obtain a PhD Degree in Informatics at Debrecen University.
The results published in the thesis are not reported in any other PhD theses.

Debrecen, 2024. .....

.....
Viktória Padányi
(candidate)

Hereby I confirm that Viktória Padányi candidate conducted her studies with my supervision within the Theoretical Computer Science, Data Security and Cryptography Doctoral Programme of the Doctoral School of Informatics between 2019 and 2024. The independent studies and research work of the candidate significantly contributed to the results published in the thesis.
I also declare that the results published in the thesis are not reported in any other theses.
I support the acceptance of the thesis.

Debrecen, 2024. .....

.....
Dr. Tamás Herendi
(supervisor)

Construction of Pseudorandom Number Generators

Dissertation submitted in partial fulfillment of the requirements for the doctoral (PhD) degree in Informatics

Written by Viktória Padányi certified Computer Science Engineer

Prepared in the framework of the Doctoral School of Informatics of the University of Debrecen
Theoretical Computer Science, Data Security and Cryptography programme

Dissertation advisor: Dr. Tamás Herendi

The official opponents of the dissertation:

Dr. .....
Dr. .....
Dr. .....

The evaluation committee:

chairperson:	Dr. .....
members:	Dr. .....
	Dr. .....
	Dr. .....
	Dr. .....

The date of the dissertation defence: 2024. .....

Contents

Introduction

> "*Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin.*" — John von Neumann (1951)

> "*Random numbers should not be generated with a method chosen at random*" — D. E. Knuth (1968)

Random numbers are used in various fields of science. Their usage is a must wherever random test data or random samples are required, for instance, in artificial intelligence applications, scientific simulations, randomized algorithms, or cryptography. Two methods are possible to generate a sequence of random numbers: generating with Pseudorandom Number Generators (PRNG) and generating with True Random Number Generators (TRNG). PRNGs use some recursive algorithms to compute the elements of the sequence from the previous values. There are theoretical and practical recurrences: in the theoretical case, the algorithms use operations on arbitrary precision numbers (or on an infinite set), while in the practical case, mostly operations are used on fixed precision numbers (i.e., the base structure is a finite set, usually with some algebraic structure). In both cases the natural requirement is the uniform distribution, which I defined only for the finite case. However, the sequences are completely predictable and highly losslessly compressible by definition. It is not possible for these schemes to generate true random numbers with information-theoretically provable randomness.

In terms of usage, several parameters of the generators are an interesting issue, as well as the resource requirements of the generators and the qualities of the given generators, which can be measured by statistical tests. Statistical tests play an important role in the analysis of Pseudorandom Number Generators (PRNGs).

Therefore, as a first step of my research work, I collected some theoretical and practical PRNGs with uniform distribution. I compared some of the most important properties of the generators, and I presented statistical tests of the sequences generated by the described generators. I did a more in-depth analysis with the NIST Statistical Test Suite. The relevant results were published in [43], and the improvement of these results with the experimental achievements were published in [41].

I continued my research work by analyzing PRNGs based on the Middle Square Method (MSM). I performed a detailed analysis of the generators belonging to the group of Middle Square Methods (MSMs), and I studied what important properties they have and what their beneficial properties are. I focused on John von Neumann's MSMs. This method is now primarily of historical significance, as it is no longer in use due to the limitations of the sequences it generates. These sequences have extremely short period lengths, which means that after a certain number of iterations, the method will produce either the same number repeatedly or cycle back to a previously generated number, ultimately leading to a repeating pattern or converging to zero.

One can start with an s-digit number (seed), which becomes the first element of the sequence. The generator seed is squared, and then the least and most significant digits are removed, leaving only the middle part of the digits of the squared number. This middle part is then used as the subsequent seed for the next iteration of the method.

I continued my research with generalized number systems. I personally generalized John von Neumann's MSM to canonical number systems (CNSs). Canonical number systems provide a unique representation of real numbers using infinite sequences of digits, providing a dynamic field for exploring new possibilities to generating PRNs. The number system has two digits $(0, 1)$, and the sequence is defined in a similar way: the starting point is a properly chosen m-bit initial value.

Square the seed, cut out the middle m-bit, and this is the next seed of the generator. The arithmetics in these number systems are similar to the arithmetics in rational integers with the classical digit representation. However, the calculation of the next digit requires a more difficult reduction operation. I will use an example to derive this calculation.

The exact definition of the generalized version of MSM binary canonical number system (CNS) and the analysis properties of these generators are described in the dissertation. I tested the generators with the NIST Statistical Test Suite. The collected most essential results will be presented. The results have been published in [42].

Another topic of my research was related to automata, which is summarized in the dissertation. I discussed some properties of arithmetic in canonical number systems (CNSs) over algebraic integers. The investigation gave a precise definition of the informational quantity for a polynomial A within the ring of polynomials with integer coefficients $\mathbb{Z}[x]$, in relation to the base of the canonical number system. Through comprehensive analysis, I revealed a significant correlation between this measure of information and the length of the polynomial's representation within the CNS. This observation emphasizes the relationship between polynomial structures and their symbolic representations in canonical number systems. Building upon this observation, I have proven for every CNS polynomial P, there exists a finite transducer automaton capable of executing the addition operation on polynomials within the canonical representation of base P.

I carried out an empirical analysis to observe the size – measured in terms of the number of states – of these transducer automata with CNS polynomials. By quantifying the complexity of these automata, I gained insights into the computational overhead required for executing arithmetic operations within CNSs. This empirical assessment provides valuable guidance for designing efficient algorithms and implementations tailored to CNS-based arithmetic.

The results have been under publishing in [19].

I have proven, among other things, the existence of a transducer automaton that performs the addition operation in a given canonical number system, and I have provided an algorithm to determine the minimal addition automaton if the number system is binary CNS. Based on the properties of addition, I have developed an algorithm using Gray code to count the squares of algebraic integers. Using the construction, I analyzed some properties of squaring in binary CNS. I focused on the squaring, because it was one of the most important operation. Therefore, I applied these estimates to the square of the numbers and determined in some cases analytical bounds. Henceforth, I studied with the primary aim of being able to enumerate the square numbers and investigate how their lengths actually behave, that is, how their lengths are related to the constraints. In the last Chapter of the dissertation, I am going to present the algorithm and their observations.

1 Pseudorandom Number Generators

In this Chapter, I discuss the importance of pseudorandom number generators, which appear in various applications such as different simulations, artificial intelligence, randomized algorithms, and also in cryptography. I have highlighted the key properties that such generators should possess, including appropriate element distribution, low correlation between elements, a large period length, and speed. The Sub-chapters also stress the importance of statistical tests in evaluating the quality of these generators. I have collected a list of well-known pseudorandom number generators with uniform distribution without claiming to be complete and reviewed their properties and performance using some of the most prevalent statistical tests.

1.1 General overview

A. Kolmogorov introduced a new notion of complexity, and he first published on the subject in 1963. [23] Independently and nearly at the same time, G. Chaitin and R. Solomonoff elaborated a similar complexity theory. Their basic idea was to express the entropy of a sequence of symbols by the length of the shortest algorithm which can produce it. The verification of the goodness of Pseudorandom Number Generators (PRNGs), the algorithmic characterization of random phenomena, and the universal computer learning algorithm constituted the most crucial parts of their research. [24] [53] [10] According to A. Kolmogorov, G. Chaitin, and R. Solomonoff, a sequence of symbols are considered random if the shortest algorithm describing it contains almost the same amount of information as the sequence itself. It is called the information content of the sequence and is incompressible.

Modeling of randomness can be theoretical and practical, while the latter can be approached from two directions: physical and arithmetical. However, in practice, arithmetic modeling is more convenient to use.

1. Physical model

   The physical model is based on random processes occurring in nature. Using and transforming the stochastic features of these processes enables the generation of random number sequences. These are, therefore true random numbers (TRNs). However, the construction and use of equipment to measure and transform the values of physical and chemical processes such as radioactive decay, noise source (often a resistor or a semiconductor diode), diffusion, etc., aggravate the original purpose to such an extent that this type of solution has limited use.

2. Arithmetic model

   One can use mathematical algorithms to generate sequences of numbers, but obviously, they are not true random numbers due to the nature of the approach. As a distinction, these are called pseudorandom numbers (PRNs). In the following, I distinguish the two main types: theoretical and practical PRNs.

I consider two key factors of the quality of PRNs and TRNs which I introduce in detail in this Chapter:

1. Statistical properties: They are for testing how chaotic a number sequence is and how close to the expected distribution is.

2. Period length: It is used for analyzing when a prior sequence of generated numbers repeats.

We may define PRN sequences in several ways. Some of these definitions can be used to construct practical PRNGs, and some are good only for theoretical considerations. In the theoretical case, the definitions may use operations on arbitrary precision numbers (or on an infinite set). In contrast, in the practical case, one can use operations only on fixed precision numbers (or on a finite set, usually with some algebraic structure). In both cases, the natural requirement is uniform distribution. In the practical case, the definition of a PRNG can be interpreted as a recursive algorithm to compute the members of the sequence from some finitely many previous values.

PRNGs were needed even in the early ages of computing. In the course of my work, I collected some well-known and less familiar PRNGs, studied what important properties they have, and the advantageous attributes associated with different applications. I attempted to dive into several sequences, from classical (early) generators to modern ones with more detailed or better properties. The first practical random number generator was John von Neumann's Middle Square Method (MSM) which was introduced in 1946. Although its basic properties were not proven, it was an interesting concept to construct a uniformly distributed PRNG based on simple arithmetic and fast enough to execute with ENIAC (Electronic Numerical Integrator And Computer). Later N. Metropolis adopted the idea of binary number systems, and R.R. Coveyou modified it for his simplified MSM.

One of the most significant species of PRNGs is generators based on arithmetic in residue class systems. Because of their periodicity, all PRNs can be expressed in an inefficient way by linear congruential generators. Shuffling methods can be used very efficiently to generate random values by scrambling their seed members. This usually requires only a few simple operations.

Moreover, there is a group of generators based on particular ideas. For example, the Xorshift uses a bitwise xor operation between the seed elements. Finally, new algorithms can be constructed by embedding generators into each other.

Before a PRNG is used, one should know whether it is suitable for the intended purposes. Knowing the underlying algorithm is usually not informative enough. In many cases, the users do not even care about the technical details at all since the general performance observations are more relevant. There are thousands of statistical tests which can be utilized to express properties of randomness e.g., autocorrelation and uniformity of the different patterns.

Analyzing the histogram and similar properties of sequences may show a uniform distribution. There are test batteries for comprehensive testing of sequences. The most well-known, among others, are D. E. Knuth's tests, Diehard, Dieharder, NIST, and TestU01. In this Chapter, I have collected the results of the NIST Statistical Test Suite, which is designed and recommended by the National Institute of Standards and Technology (NIST). This widely accepted test suite is renowned for its rigorous standards and consistency in evaluating the quality of random number generators.

1.2 Theoretical model

This approach of PRNs is mainly used for theoretical observations of numerical computations. The most well-known areas of application are the Monte Carlo methods. Theoretical sequences are usually defined by some function over an infinite mathematical structure (most frequently over real intervals). The quality of such sequences is measured by some numerical properties, which are determined analytically. Based on their proven attributes, in practice, an approximate sequence can be used instead, where an error term can be determined.

1.2.1 Properties of number sequences

There are a couple of ways to analyze PRN sequences, both from theoretical and practical points of view. In this Sub-chapter, I define some tools for expressing the quality of the sequences.

Definition 1.1 *Let u be a sequence in $[0,1]$. One can say that u is uniformly distributed in the interval $[0,1]$ if*

$$\lim_{N\to\infty} \frac{1}{N} \Big|\{n \le N | u_n \in [a,b)\}\Big| = b - a \ , \quad \textit{for all } 0 \le a \le b \le 1 \ .$$

Definition 1.2 *Let A be a finite nonempty set, and u be a sequence in A. It can be said that u is uniformly distributed in A if*

$$\lim_{N\to\infty} \frac{1}{N} \Big|\{n \le N \mid u_n = a\}\Big| = \frac{1}{|A|} \ , \quad \textit{for all } a \in A \ .$$

Definition 1.3 *Let A be a finite set, and u be a sequence in A. We can say that u is periodic in A with a period length $\varrho \in \mathbb{N}$, if there exists $\varrho_0 \in \mathbb{N}$ preperiod, such that*

$$u_{n+\varrho} = u_n \quad \textit{for all} \quad n \ge \varrho_0 \ .$$

Remark 1.4 *For a given sequence – according to the definition –, there are infinitely many possibilities for the values of the preperiod (ϱ_0) and the period length.*

For the sake of simplicity, I will assume that both the preperiod and the period length are the unique minimal.

There are several further measures of pseudorandomness. Among others, A. Sárközy and C. Mauduit in [36] introduced some for binary sequences. Based on their notions, they settle a condition on pseudorandomness which shows some similarity to the one defined by A. Kolmogorov.

Additionally, discrepancy may also be defined (see, e.g., in [39]), which is significant from a theoretical standpoint. Although it is an important concept in the study of theoretical sequences, it is not examined in this dissertation. I consider it a potential avenue for future research.

1.2.2 Theoretical random number sequences

Theoretical RNSs usually have the remarkable feature that their properties can be proven via mathematical instruments.
Based on these proven properties, the approximate (practical) sequences can be qualified, and their proper usage can be verified.

1. Weyl-sequence

   The general Weyl-sequence is defined in the form $w_n = \{n \cdot \alpha\}$, where $\{x\}$ denotes the fractional part of x. One can prove that w_n is uniformly distributed in $[0, 1]$, if and only if α is an irrational number. [57]

2. Van der Corput sequence

   Van der Corput in [11] defined a sequence of rational numbers based on digit reflection. For an integer $g \geq 2$, if the digit expansion of $n \in \mathbb{N}$ in base g is

   $$n = \sum_{i=0}^{s} n_i g^i,$$

   where $0 \leq n_i < g$, then the radical-inverse of n in base g is

   $$\bar{n} = \sum_{i=0}^{s} n_i g^{-i-1}.$$

3. Sequences generated from the digits of irrational numbers

   Let α be an irrational number and $g > 1$ an integer. The digit expansion of α in base g forms an infinite sequence. A number α is normal to base g if the relative frequency of each word of length s in its digit expansion is asymptotically g^{-s}. M. Borel [7] proved that almost all real numbers are normal with respect to the Lebesgue measure, though it is not known if an arbitrary number is normal in base g. Normality is not known in the case of π, but the experiments so far do not contradict its possibility.

4. de Bruijn sequence

   The de Bruijn sequence of type (n, s) (see, [8]) is the shortest sequence containing all words of length s over an alphabet A with cardinality n. Such a sequence contains each word of length s exactly once, ensuring that all s-length blocks are uniformly distributed. The length of the (n, s)-type de Bruijn sequence is $n^s + s - 1$. Any prefix of length $n^s + t - 1$ contains each word above A of length t exactly n^{s-t} times.

1.3 Classification of PRNGs

In this part of this Chapter, I am going to introduce a comprehensive, although non-exhaustive list of some of the most well-known or historically significant generators. The generators, in practice, use some recursive relation with bounded depth. In the implementations, the recursion is usually replaced by a seed. This seed is modified iteratively, and the random sample is extracted from it. Due to the above properties, such sequences are obviously periodic. Intrinsically, all of the observed PRNGs operate based on some form of recursion. In each case, there is a seed and one can calculate the random values extracted from it. Using this, I was able to classify PRNGs according to the properties of the recursion.

1. PRNGs based on recursions with modular arithmetic

A large class of PRNGs uses modular arithmetic as a base operation for computing elements in the sequences.

(a) Homogeneous linear methods

In these methods, the sequences are defined by a linear recurrence relation with no constant terms in their definition.

i. D. H. Lehmer's Multiplicative Linear Congruential Generator (also called Power Residue Method (PRM)).
The classical definition is

$$u_{n+1} = a \cdot u_n \pmod{m} \quad (n = 0, 1, \ldots) \ ,$$

which can be expressed in the form

$$u_{n+1} = a^n \cdot u_0 \pmod{m} \quad (n = 0, 1, \ldots) \ .$$

When choosing the coefficient a, one has to try to find a large order element (a long period); $\mathrm{ord}(a)$ is the smallest positive e, such that $a^e \equiv 1 \pmod{m}$. [29] [22]

ii. Generalized Feedback Shift Register Generator
This generator was introduced by T.G. Lewis and W.H. Payne in 1973. GFSR is a widely used PRNG based on the linearly recurring equation

$$u_{n+a} := u_{n+l} \oplus u_n \quad (n = 0, 1, \ldots) \ ,$$

where $a > l > 0$, $s > 0$, $u_n \in \{0, 1\}^s$ and $\oplus$ is the bitwise exclusive-or operation. With carefully chosen a and l, the period length is $2^k - 1$ (theoretical upper bound). [30]

iii. Linear Recurrence Sequence Generator
LRS is a generalization of LCG. For properly chosen coefficients $a_0, \ldots, a_{k-1}$ and initial values $u_0, \ldots, u_{k-1}$, the sequence defined by:

$$u_{n+k} \equiv a_{k-1}u_{n+k-1} + \cdots + a_0 u_n \pmod{m} \quad (n = 0, 1, \ldots)$$

is uniformly distributed. [18] describes a construction for the coefficients when $m = 2^s$ with some $s \in \mathbb{N}$. Moreover, the number of nonzero coefficients can be reduced to six, and the period length can be 2^{s+k}.

iv. Twisted Generalized Feedback Shift Register Generator
Developed by M. Matsumoto and Y. Kurita in 1992, this is an improvement of the GFSR: before the xor operation, one of the words is "twisted" [32]:

$$u_{n+a} := u_{n+l} \oplus u_n A \quad (n = 0, 1, \ldots) \ ,$$

where A is a square matrix from $\{0, 1\}^{s \times s}$. The theoretical upper bound for the period is $2^{as}-1$, while also significantly reducing memory requirements [33].

(b) Non-homogeneous linear method
In this method, the sequence is defined by a linear recurrence relation with a constant term in its definition.

i. Linear Congruential Generator
This generator is the generalization of MLCG. LCG was published in 1958 by W. E. Thomson [54] and independently by A. Rotenberg [47] in 1960. It is one of the most influential and studied generators. LCG is defined by the recurrence relation:

$$u_{n+1} = (au_n + c) \pmod{m} \quad (n = 0, 1, \ldots) \ ,$$

where a, c, m and u_0 can be chosen such that the generator has a full period of length m. [22]
If there is a non-homogeneous linear recursion, that is first-order, which means that it depends on only one next element, it can be transformed into a homogeneous second-order one. In theory, there are non-homogeneous linear generators from which the constant term (non-homogeneous

term) can be expressed and homogenized as a whole. It may not be worth looking for non-homogeneous methods, as in the final result, all of them can be equated to a homogeneous method. In this way, it will be a practically homogeneous LRS generator.

(c) Hybrid methods

In these methods, the sequences are defined by linear methods, but the outputs are modified by some additional operations.

i. Mersenne Twister

MT was developed in 1997 by M. Matsumoto and T. Nishimura. It is a further improvement of TGFSR.

$$u_{n+k} := u_{n+l} \oplus v_n A \quad (n = 0, 1, \ldots) \ ,$$

where v_n is the concatenation of the upper $s - r$ bits of u_n and the lower r bits of u_{n+1} (r is a properly chosen parameter). The period length can be arbitrary, but with the suggested – and widely used – parameters, it is $2^{19937} - 1$, which is a Mersenne prime. [35]

ii. Well Equidistributed Long-period Linear Generator

WELL was developed by F. Panneton, P. L'Ecuyer, and M. Matsumoto in 2006. Based on similar approaches as MT. The period length of the sequence can supersede the period length of MT. [44]

iii. Wichmann-Hill algorithm

Described by B. A. Wichmann and I. D. Hill in 1982. The idea is a special combination of three "independent" LCGs. [58] Let v_n, w_n, and z_n be PRN sequences generated by different LCGs. The corresponding modulus are m_v, m_w, and m_z. Then the sequence

$$u_n = \left(\frac{v_n}{m_v} + \frac{w_n}{m_w} + \frac{z_n}{m_z} \right) \pmod 1$$

is uniformly distributed, here $x \pmod 1$ yields the fractional part of x. The period length is the product of the period length of the three base sequences. [59]

iv. Additive Congruential Random Number Generator
This generator was introduced by R. S. Wikramaratna in 1988. Theoretically, one generator provides an infinite set of PRN sequences $u^{(k)}$, where $k \in \mathbb{N}$. For all k, the sequence has an initial value $u_0^{(k)}$. The recurrence is

$$u_{n+1}^{(0)} = u_n^{(0)} \quad \text{and} \quad u_{n+1}^{(k)} = \left\{ u_n^{(k)} + u_{n+1}^{(k-1)} \right\} .$$

The main advantages of ACORN are the speed of execution, long period length, and simplicity of coding. [61]

v. Permuted Congruential Generator
PCGs were introduced and observed by M. E. O'Neill in 2014. [40] The idea is to apply an output permutation on a simple but good-quality LCG.

(d) PRNGs based on non-linear recursion

In these methods, the sequences are defined by some general modular recursion.

i. Quadratic Congruential Generator
The linear congruential method can be generalized to such a quadratic congruential method:

$$u_{n+1} = (au_n^2 + bu_n + c) \pmod m$$

where b, c, m, and u_0 are chosen to ensure a full period of length m. While the restrictions are similar to the linear method, the block distribution properties are weaker compared to LCG [22].

ii. Blum-Blum-Shub
BBS was proposed in 1986 by L. Blum, M. Blum, and M. Shub. It can be regarded as a simplified QCG. The seed of the sequence satisfies the recursion

$$u_{n+1} = u_n^2 \pmod{m} \quad (n = 0, 1, \ldots) \ ,$$

where m is the product of two large primes, from the value u_n, a single bit v_n is obtained by some extraction (e.g., least significant bit). BBS was the first theoretically proven cryptographically secure PRNG. [6]

iii. Multiplicative Fibonacci sequence
A modified method of the original Fibonacci sequence, similar to the LCG, defines the next element as:

$$F_n = F_{n-1} + F_{n-2} \pmod{m}.$$

In the Multipl. Fibonacci method, the recurrence relation is:

$$F_n = F_{n-1} \cdot F_{n-2} \pmod{m} \ ,$$

using the quadratic polynomial $p(x, y) = x \cdot y$ with the previous sequence values.

iv. Power Congruential Generator
It is a modification of the standard LCG. We replace here the arithmetic operations $+$ and $\cdot$ with the corresponding next-level operations $\cdot$ and x^y.

$$u_{n+1} \equiv c \cdot (u_n)^a \pmod{m} \quad (n = 0, 1, \ldots) \ ,$$

where a, c, m and u_0 can be chosen such that the generator has a full period of length m.

v. Inverse (Inversive) Congruential Generator
This is another modification of LCG. It was the first proposed by J. Eichenauer and J. Lehn in [14]. Now, instead of the linear $a \cdot x + c$, we define the recurrence, by the function $a \cdot x^{-1} + c$.

$$u_{n+1} \equiv \left(a \cdot (u_n)^{-1} + c\right) \pmod{m} \quad (n = 0, 1, \ldots) \ ,$$

where a, c, m and u_0 can be chosen such that the generator has a full period of length $\varphi(m)$, where $\varphi(.)$ is the Euler's totient function.

2. PRNGs based on other recursion methods

PRNGs usually have their seed from a finite domain and use a particular mapping to generate the new seed from the previous ones. The best is if we can prove the uniform behavior of this mapping, but in many cases, it is hard to do so. This class details some generators where the next seed is calculated based on a special function not listed before. These functions' uniformity - or, equivalently, the randomization property - is not always proven.

(a) MSM-based generators

i. John von Neumann's Middle-Square Method
Neumann's MSM is an interesting way to construct uniformly distributed PRNG since this was the first practical random number generator. In 1946 John von Neumann introduced the method (first published in [38]). It was simple and fast to execute with ENIAC. He used a recursive definition where the initial value u_0 is some $2s$-digit decimal number. For $n > 0$ he defined $u_n = \lfloor u_{n-1}^2 / 10^s \rfloor \pmod{10^{2s}}$. The period length depends on the initial value. Practically it is a rather weak generator, if the seed becomes 0, it is 0 for all consecutive members. [22]

ii. N. Metropolis MSM in Binary Number Systems
In the early '50-s, N. Metropolis investigated the MSM in binary number systems. He showed that in the case of 20-bit numbers, there are only 13 different cycles. [22]

iii. R. R. Coveyou's Simplified Middle-Square Method
Essentially it is a degenerate double-precision MSM. The initial value u_0 is chosen such that $2 = u_0 \pmod 4$ and $u_{n+1} = u_n(u_n + 1) \pmod{2^k}$ for $n > 0$. The period length is typically around 2^{k-1}. [22]

iv. Generalized Middle-Square Method
This generator is the generalization of John von Neumann's MSM to CNSs. Let $p(x) \in \mathbb{Z}[x]$ be an irreducible polynomial of degree d, and with coefficients $1 = a_d \leq a_{d-1} \leq \cdots \leq a_0 = 2$. The CNS has only 2 digits: 0 and 1, one can call the digits bits and the digit representation of algebraic integers in $\mathbb{Z}[\alpha]$ a binary representation. In the design, one can use a seed of $s \in \mathbb{N}$ bits. Similarly, as it is done in the original construction, let u be a sequence over $\mathbb{Z}[\alpha]$ defined by the following, where the α is a root of $p(x)$:

$u_0 \in$ is a random s-bit number;

if $n > 0$, let

$$u_{n-1}^2 = \sum_{i=0}^{h} b_i \alpha^i \ , \text{ with } b_h \neq 0 \ , \ t = \left\lfloor \frac{h-s}{2} \right\rfloor \text{ and}$$

$$u_n = \sum_{i=0}^{s-1} b_{i+t+1} \alpha^i \ .$$

The value of s should be chosen to be large enough. [42] A more detailed description of this generator can be found in Sub-chapter **2.3**.

v. Middle-Square Weyl Sequence
To improve the original MSM generator, a simple modification can be applied. As we already know, we can prove that if $GCD(m, a) = 1$, then $u_n = n \cdot a \pmod{m}$ is uniformly distributed as well. Thus, the modified MSM sequence is the non-discrete version of the original sequence: $v_n = \lfloor v_{n-1}^2/2^s \rfloor + u_n \pmod{2^{2s}}$. [60]

(b) Shuffling methods

In these methods, the sequences are defined by permutations.

i. Knuth's Algorithm M randomizing by shuffling
Let x_n and y_n be two sequences generated by some algorithms and assume $0 \leq y_n < m$ with some m. The output sequence of the generator is z_n. Let k be some (not too large) positive integer, and the seed of the generator is $v = (v_0, \ldots, v_{k-1})$. Initially let $v_n = x_n$ for $n = 0, \ldots, k-1$, then for all $n \geq k$

$$\begin{aligned} j &= \lfloor k \cdot y_n/m \rfloor \\ z_{n-k} &= v_j \\ v_j &= x_n \ . \end{aligned}$$

The period length is the least common multiple of the period length of x_n and y_n. The computational time is the sum of the computational time of the two base sequences. [22]

ii. Knuth's Algorithm B randomizing by shuffling
It is similar to Algorithm M but uses only one base sequence x_n with assumption $0 \leq x_n < m$.
The initial seed v is as before.

Then the new sequence $v_0 = x_k$ and for all $n > 0$

$$\begin{aligned} j &= \lfloor k \cdot v_{n-1}/m \rfloor \\ v_n &= v_j \\ v_j &= x_{n+k} \text{ .[22]} \end{aligned}$$

iii. RC4 based generators
R. Rivest's algorithm for stream cipher key generation was designed in 1987 but kept secret until 1994. Because of its simplicity and efficiency, it is the basis for several cryptographic protocols, such as WEP (Wired Equivalent Privacy) and WPA (Wireless Protected Access), used in WiFi communications. The seed is an array S of 256 entries of 8 bit values.

$$\begin{aligned} &i \equiv (i + 1) \pmod{256} \\ &j \equiv (j + S[i]) \pmod{256} \\ &\text{swap values of } S[i] \text{ and } S[j] \\ &u = S[(S[i] + S[j]) \pmod{256}] \\ &\text{output } u\text{.[52]} \end{aligned}$$

(c) Miscellaneous methods

i. Xorshift
This is a class of PRNGs—also called shift-register generators—described by G. Marsaglia in 2003. The period length is $2^s - 1$, where $s = 32, 64, 96, 128, 160, 192$ in the original paper. It uses a 4-word seed: (x, y, z, w) and three parameters for shifting: a, b, c. The elements of the seed are shifted and xor'ed to generate new seed members and the output.

The operations are the following:

$$\begin{aligned} tmp &= (x \oplus (x \text{ shl } a)) \\ x &= y \\ y &= z \\ z &= w \\ w &= (w \oplus (w \text{ shr } b)) \oplus (tmp \oplus (tmp \text{ shr } c)) \end{aligned}$$

Here, shl and shr are the bitwise left and right shifts, respectively. [34]

ii. Legendre symbol
The Legendre symbol has become one of the most useful methods in number theory. It can have three values: $1, -1$, or 0. For a prime b and integer a, $\left(\frac{a}{b}\right)$ equals 1 if a is a quadratic residue modulo b, -1 if it is not, and 0 if a is divisible by b. [36] A number sequence generated using the Legendre symbol is a good source of PRNs from a practical point of view. Additionally, it can be considered cryptographically secure one. [12]

iii. Encryption-based methods Let $\text{Enc} : \mathcal{P} \times \mathcal{K} \to \mathcal{P}$, be a cryptographic function (encryption), where $\mathcal{P} = \{0,1\}^n$, and $\mathcal{K} = \{0,1\}^m$ with some $n, m \in \mathbb{N}$ and construct a sequence of words by the recursion $m_{n+1} = \text{Enc}(m_n, k)$, where $k \in \mathcal{K}$ is the secret key - in our case a properly chosen parameter. A typical application of such sequences is, for instance, the Output Feedback Mode of Operation of symmetric encryption functions (see, FIPS PUB 81 [1]). Assuming the expected good quality of the mapping Enc, the derived sequences usually have very good statistical and cryptographic properties. The only disadvantage of these generators is their relatively low speed.

1.4 Analytical properties of PRNGs

However, after working on the theoretical measurements of PRNGs, I saw that a periodic sequence did not perform very well on these. Since the practical generators are always periodic, they do not have good global quality scores. Nevertheless, periodic sequences may still deliver satisfactory local results on the mentioned observations – in particular, if they have a large period length.

I considered further analytical attributes of the generators, such as period length, computational complexity, and memory usage. The properties of the analyzed PRNGs are presented in Table **1**.

Since there is a wide range of platforms they are implemented on, it is hard to compare the speed of the generators. Some perform better on constrained devices, while others are more efficient on CISC processors. Owing to this duality, instead of the speed of the algorithms, I have highlighted the number of complex (slow) and simple (fast) operations. The available implementations for most of the generators can be found in [41]. Those without released implementations are either weak or very simple.

The entries of the Table **1** are expressed in terms of the parameters of the generators:

- s: the length (bit size) of the data (seed). If the generator under consideration is based on modular arithmetics, then for the used modulus $m \approx 2^s$.

- d: the decimal length of the data. Some of the early generators are defined in decimal representation.

- k: the farthermost previous seed value used to determine the next one or the dimension of the seed vector.

Generators	Period length[1]	Operations[2]		Memory (bits)
		Slow	Fast	
Lehmer's MLCG	2^s	1 mul, 1 mod		$2s$
GFSR	2^k		s-bit xor	$k \cdot s$
TGFSR	2^{ks}	$s \times s$-bit m.mul	s-bit xor	$k \cdot s + s^2$
LCG	2^s	1 mul, 1 mod	1 sum	$4s$
LRS[3]	2^{k+s}		k sum	$k \cdot s$
MT[4]	2^{ks}	$s \times s$-bit m.mul	s-bit xor	$k \cdot s + s^2$
WELL[5]	2^{ks}	8 $s \times s$-bit m.mul	9 s-bit op	
W-H algorithm	2^{3s}	3 LCG, 3 div	2 sum	$12s$
ACORN	2^{ks}	k mod	k sum	$k \cdot s$
PCG	2^{128}	mul	s-bit perm,xor	$4s$
QCG	s^s	4 mul, 1 mod	2 sum	$4s$
BBS	2^s	1 mul, 1 mod		$2s$
Multipl. Fibonacci	2^{2s}	1 mul, 1 mod		$3s$
Power Congr. Gen.	2^s	s mul, 1 mod		$4s$
ICG	2^s	1 mul, 1 inv, 1 mod	1 sum	$4s$
Neumann's MSM[6]	8^d	1 mul	1 xor, 1 shr	$4d$
Metropolis MSM[7]	2^s	1 mul	1 xor, 1 shr	s
Coveyou's SMSM	2^s	1 mul		s
GMSM	2^s	1 mul	1 xor, 1 shr	s
MS Weyl	2^{2s}	1 LCG, 1 mul	1 sum	$5s$
Algorithm M	2^{2s}	2 gen, 1 mul, 1 div		$2 \cdot * + k \cdot s$
Algorithm B	2^s	1 gen, 1 mul, 1 div		$* + k \cdot s$
RC4	2^{1024}		7 byte op	256 bytes
Xorshift	2^s		1 xor, 1 shr	s
Legendre symbol	2^s	s mod		s
Enc.-based methods[8]	2^s	*	*	$2s$

Table 1: General properties of the generators

The Table **2** explains the operations referred in Table **1**.

Slow operations		Fast operations	
mul	multiplication	sum	addition
mod	modular reduction	xor	bitwise xor
div	division	shl, shr	bitwise left and right shift
m.mul	multiplication of a vector by a matrix	op	memory operations
inv	modular inversion	perm	bitwise permutation
gen	sample from an embedded generator		
LCG	sample from an LCG generator		

Table 2: Operations

In Table **1**, I assumed the following:

1. in the column *Period length*, the best (largest) possible values are shown;
2. the column *Operations* displays the complexity of the algorithms, i.e., the number of necessary operations to calculate a single random sample. Operations use the given size of data (s bit, or d decimal digit).
3. The complexity of the LRS generator can be reduced to k additions by choosing the proper parameters.
4. With the widely used parameters, the period length of the MT is $2^{19937} - 1$.
4. With the widely used parameters, the period length of the WELL is 2^{44497}.
5. Neumann's MSM is originally defined for decimal numbers.
6. The original construction of Metropolis with 20-bit numbers has maximal period length of 142.
7. The time complexity depends on the underlying cryptographic algorithms.

1.5 Statistical tests

Before using a PRNG, we need to know whether it is suitable for the given purpose or not. Knowing the underlying algorithm is usually not informative enough. The users often do not even care about the technical details. This is why it is important to have general performance observations, which can be realized by standard statistical tests. The different tests express the properties from different points of view. But it is still a question of which properties are expected for the application concerned. Statistical tests of PRNG are usually applications of the general statistical tests for some expected properties of an imaginary perfect uniformly distributed true random sequence. In many cases, the tests count the relative frequency of the appearance of some – in general numerically expressed – property and determine the probability of the given event.

There are several statistical tests available, for instance, one of the simplest is the Frequency test. Assume that the observed pseudorandom bit sequence is a sequence of independent samples with probability 1/2 for both the 0s and 1s. The number of 1s (or 0s) of a sequence of length n is expected to have a binomial distribution, with mean $n/2$. Because of the desired distribution, the actual value of the number of 1s has a well defined probability. Setting a level of significance, we accept or reject our hypothesis (i.e., the sequence is uniformly distributed). If we have an initial test – such as the previous frequency test –, we may define an advanced one: testing t different sequence of length n, we get t probabilities. These probabilities should follow a well defined distribution, for which we may execute another statistical test. This provides a probability, as before, and depending on the preset significance level, we accept or reject our hypothesis again. We may continue the described iteration to obtain further higher-level tests. In most cases, however, the practical tests apply only first or second-level tests.

Among the various methods used, two commonly applied tests are the Chi-Square test and the Kolmogorov-Smirnov test. [22] The χ^2 test is primarily used for goodness-of-fit assessments, determining how well the observed frequency distribution matches an expected theoretical distribution. [3] The K-S test, on the other hand, is used for sequences with continuous distribution functions. In this test, the random number generators' distribution function $F_n(x)$ is compared to the theoretical cumulative distribution function $F(x)$. [22] [3]

Several well-known test batteries include D. E. Knuth's tests [22], the Diehard Test Suite from G. Marsaglia (1995), the Dieharder written by Robert G. Brown [13], TestU01 published by P. L'Ecuyer and R. Simard in 2007 [31], and the NIST Statistical Test Suite [37].

In the following, I am going to present an execution of a statistical test suite for random and pseudorandom number generators. This is designed and recommended by the National Institute of Standards and Technology (NIST). This widely accepted test suite is renowned for its rigorous standards and consistency in evaluating the quality of random number generators.

1.5.1 NIST Statistical Test Suite

The NIST Statistical Test Suite is the standard for evaluating PRNGs and consists of 15 tests developed to test the randomness of (arbitrarily long) binary sequences produced by either hardware or software-based cryptographic PRNGs (see [37], and [5]). These tests focus on a variety of different types of non-randomness that could exist in a sequence. I was able to group the 15 tests into five main classes.

1. Frequency tests (monobit, block)
   Monobit test: Observes the relative frequency of the 0s and 1s. In a true random uniformly distributed sequence, the proportion of the 0s and 1s should be close to 1/2. The test estimates the distance of the input sequence to a completely equidistributed one.

Block test: It splits the input sequence into blocks of equal length. The test observes the relative frequency of the 0s and 1s in M-bit blocks. It is a generalization of the monobit test and determines whether the average frequency of ones in an M-bit block is approximately $M/2$.
Serial Test: Evaluates the relative frequency of m-bit overlapping patterns to determine if they match expectations for a true random sequence. In an ideal random sequence, the probability of an m-bit pattern's appearance depends only on m.

2. Extremal behavior tests (runs)
   Run test: Counts the number of consecutive identical bits of length $\geq k$. The test determines whether the number of runs of different lengths is as expected for a true random sequence. If the number of runs is high, then the sequence behaves regularly.
   Runs of ones in blocks test: The test focuses on the longest run within M-bit blocks. Similar to the previous test, but it does not count all runs. In an ideal random sequence, the longest run has a well-defined distribution, and there is no need for separate tests for 1s and 0s.
3. Linear dependency and periodicity tests
   Binary Matrix Rank Test: Tests the distribution of the rank of a random matrix.
   Linear Complexity Test: Tests the shortest LFSR expression satisfied by the sequence. Every periodic sequence satisfies a linear recurrence, and the goal is to identify the shortest one. If it is large enough, the sequence can be regarded as true random.
   Discrete Fourier Transform Test: It is a spectral test of the input sequence. Applying the DFT, we transform the sequence of signals to the frequency space and observe the behavior there.
4. Pattern tests (template matching)
   Non-overlapping Template Matching Test: Evaluates the frequency of predefined patterns. The test detects irregular occurrences of an m-bit pattern by sliding an m-bit window: if the pattern is absent, the window moves one bit; if present, it shifts by m bits.
   Overlapping Template Matching Test: Same as before, but allowing overlap in the case when a pattern is found.

Maurer's Universal Statistical Test: Tests the distance between previously undetermined patterns. It expresses the possibility and ratio of compression of the sequence. According to the theory of Kolmogorov-complexity, a true random sequence can not be significantly compressed.
Approximate Entropy Test: Based on a similar argument as the serial test, but instead of counting relative frequencies, it calculates the empirical entropy of the patterns.

5. Random walk tests (cumulative sum, excursion)
Cumulative Sums (Cusum) Test: Tests the behavior of cumulative sums by replacing zeros with -1. The cumulative sum should be close to zero. If the cumulative sum is large, the sequence is regarded as non-random.
Random Excursions Test: A cycle of a random walk is when the excursion returns to 0. The test counts the number of cycles having exactly K visits in a particular state. The states are $-4, -3, -2, -1$ and $+1, +2, +3, +4$.
Random Excursions Variant Test: Similar to the previous. It tests the total number of times that a particular state is visited. The number of occurrences of the observed states should follow a particular distribution. The states are $-9, -8, \ldots, -1$ and $+1, +2, \ldots, +9$.

A more detailed description of the tests can be found in [5].

1.5.2 Experimental results

I implemented the PRNGs discussed in Chapter **1.3** in C++ and ran on an Intel(R) Core(TM) i9-9900K CPU @ 3.60GHz with 64GB RAM. I tested the obtained sequences with the NIST Test Suite in the same environment. The implementations and the detailed results are available at `http://www.prng.hu`. In the tests, 10^9-long $0, 1$-bit sequences were used for each generator. Each sequence was split into 1000 equal-length subsequences to have reasonably large independent test cases. All fifteen tests were applied for every sequence. During the test, I used the default setup of the test suite:

1. Block Frequency Test - block length(M): 128
2. NonOverlapping Template Test - block length(m): 9
3. Overlapping Template Test - block length(m): 9
4. Approximate Entropy Test - block length(m): 10
5. Serial Test - block length(m): 16
6. Linear Complexity Test - block length(M): 500

The most essential results are collected in tables **3** and **4**. Table **3** contains the p-value computed by the different tests. The significance level is set to 0.01.

The minimum pass rate for each statistical test – with the exception of the random excursion (variant) test – is approximately 0.981819 for a sample size of 1000 binary sequences.

The minimum pass rate for the random excursion (variant) test is approximately 0.979517 for a sample size of 609 binary sequences.

The generators which did not pass a particular test are marked by an * in the tables. For those tests which stand off several subtests with different parameters – cumulative sums, non-overlapping template, random excursions, random excursions variant, serial – I displayed the average of the results instead.

Abbreviations in the tables:

1. Frq: Frequency
2. BFrq: Frequency within a Block
3. CSum: Cumulative Sums
4. Runs
5. L.Run: Longest Run of Ones
6. Rank: Binary Matrix Rank
7. FFT: Fast Fourier Transform
8. NOT: Non-Overlapping Template Matching
9. OT: Overlapping Template Matching
10. Univ: Universal Statistical
11. AEnt: Approximate Entrophy
12. RE: Random Excursions
13. REV: Random Excursions Variant
14. Serial
15. Lomp: Linear Complexity

***p*-value**	Frq	BFrq	CSum	Runs	LRun	Rank	FFT	NOT	OT	Univ	AEnt	RE	REV	Serial	LComp
Lehmer	0.53	0.15	0.38	0.42	0.18	0.26	0.86	0.43	0.13	0.29	0.05	0.52	0.60	0.32	0.99
GFSR	0.70	0.13	0.91	0.15	0.49	0.71	0.76	0.48	0.50	0.86	0.22	0.40	0.38	0.65	0.51
TGFSR	0.37	0.67	0.57	0.95	0.18	0.12	0.84	0.52	0.51	0.44	0.99	0.44	0.06	0.16	0.71
LCG	0.52	0.59	0.53	0.04	0.11	0.73	0.03	0.42	0.37	0.59	0.99	0.46	0.38	0.47	0.03
LRS	0.33	0.22	0.63	0.68	0.27	0.30	0.55	0.54	0.62	0.47	0.31	0.55	0.37	0.50	0.87
MT	0.76	0.73	0.79	0.23	0.03	0.28	0.67	0.46	0.65	0.87	0.77	0.53	0.50	0.48	0.07
WELL	0.60	0.11	0.77	0.80	0.38	0.65	0.46	0.49	0.01	0.45	0.37	0.61	0.48	0.37	0.03
W-H	0.38	0.98	0.07	0.43	0.06	0.49	0.08	0.51	0.24	0.17	0.49	0.65	0.38	0.82	0.69
ACORN	0.27	0.21	0.71	0.62	0.87	0.77	0.01	0.52	0.98	0.80	0.90	0.48	0.54	0.74	0.22
PCG	0.39	0.13	0.39	0.26	0.75	0.85	0.44	0.46	0.26	0.48	0.56	0.65	0.59	0.15	0.98
QCG	0.00*	0.00*	0.00*	0.00*	0.00*	0.13	0.00*	0.00*	0.00*	0.00*	0.00*	0.20	0.09	0.00*	0.00*
BBS	0.00*	0.00*	0.00*	0.00*	0.02	0.03	0.00*	0.00*	0.00*	0.00*	0.00*	0.54	0.48	0.00*	0.90
Multi. Fib.	0.00*	0.00*	0.00*	0.10	0.02	0.63	0.20	0.05	0.40	0.00*	0.00*	0.31	0.39	0.00*	0.24
PowCG	0.00*	0.00*	0.00*	0.00*	0.00*	0.08	0.01	0.00*	0.00*	0.00*	0.00*	0.51	0.45	0.00*	0.03
ICG	0.90	0.53	0.46	0.52	0.45	0.35	0.04	0.48	0.34	0.09	0.62	0.40	0.58	0.70	0.77
Neumann	0.28	0.78	0.73	0.71	0.59	0.23	0.10	0.46	0.77	0.95	0.57	0.46	0.55	0.73	0.34
Metropolis	0.88	0.39	0.40	0.36	0.09	0.15	0.20	0.52	0.14	0.27	0.07	0.59	0.48	0.26	0.18
Coveyou	0.03	0.46	0.01	0.10	0.39	0.68	0.00*	0.49	0.22	0.98	0.49	0.50	0.43	0.20	0.74
GMSM	0.67	0.39	0.64	0.85	0.57	0.80	0.10	0.46	0.36	0.47	0.63	0.54	0.49	0.41	0.76
MS Weyl	0.21	0.93	0.35	0.72	0.52	0.46	0.56	0.50	0.33	0.13	0.63	0.44	0.46	0.50	0.61
Alg. M	0.38	0.67	0.48	0.67	0.78	0.43	0.83	0.52	0.32	0.77	0.28	0.67	0.43	0.12	0.32
Alg. B	0.34	0.37	0.99	0.54	0.47	0.75	0.95	0.56	0.36	0.02	0.53	0.39	0.56	0.16	0.25
RC4	0.29	0.03	0.80	0.95	0.43	0.55	0.17	0.50	0.55	0.87	0.42	0.40	0.46	0.35	0.16
Xorshift	0.21	0.21	0.52	0.76	0.27	0.87	0.49	0.53	0.88	0.69	0.39	0.59	0.67	0.05	0.00*
Legendre	0.04	0.64	0.07	0.05	0.86	0.64	0.06	0.52	0.09	0.27	0.59	0.47	0.39	0.28	0.12

Table 3: NIST test results: p-value

Prop.(%)	Frq	BFrq	CSum	Runs	L.Run	Rank	FFT	NOT	OT	Univ	AEnt	RE	REV	Serial	Lomp
Lehmer	99.4	99.1	99.4	99.3	98.8	98.6	99.0	99.1	98.2	98.6	98.9	98.8	99.0	99.2	99.4
GFSR	99.3	98.9	99.6	99.0	98.8	98.9	98.3	99.0	98.9	98.9	99.2	99.0	99.0	99.2	99.2
TGFSR	98.8	99.1	99.2	99.2	99.3	99.1	98.8	99.0	98.5	98.6	98.5	99.1	99.1	98.9	98.9
LCG	99.6	99.4	99.6	99.4	98.3	99.1	99.3	99.0	98.8	99.0	98.6	98.9	99.4	98.9	98.9
LRS	98.5	99.3	98.6	99.5	98.9	99.4	98.7	99.0	98.6	99.2	98.9	99.0	99.2	98.5	99.4
MT	98.8	99.2	98.8	98.6	98.8	99.1	99.0	99.0	99.0	99.0	98.9	99.0	99.0	99.2	99.0
WELL	98.8	98.8	98.8	99.0	99.2	98.9	99.1	99.0	98.2	99.4	99.2	98.8	98.9	98.7	98.7
W-H	99.0	99.2	99.0	98.6	98.7	99.0	98.5	99.0	99.3	98.0*	99.2	99.1	99.1	99.0	98.9
ACORN	99.2	99.6	99.1	99.4	98.8	99.1	99.2	99.0	98.9	98.0*	99.0	98.9	98.9	99.1	99.2
PCG	99.0	98.6	99.2	98.8	98.3	99.2	99.3	99.0	98.5	98.6	98.4	98.7	99.0	99.4	98.9
QCG	100*	99.0	100*	88.0*	0.0*	99.2	0.0*	100*	0.0*	0.0*	0.0*	99.3	99.2	0.0*	98.7
BBS	100*	98.0*	100*	100*	98.8	98.0*	95.3*	98.4*	99.0	98.0*	98.8	98.7	99.4	100*	99.2
Multi. Fib.	100*	100*	100*	100*	98.8	99.0	98.8	98.8	98.8	99.0	99.0	99.0	98.8	98.8	98.8
PowCG	100*	100*	100*	100*	99.0	99.0	99.3	99.0*	84.0*	100*	100*	97.3	98.0	100*	99.0
ICG	98.7	99.2	98.7	99.3	98.9	99.2	98.7	99.0	99.1	99.2	99.0	99.0	98.9	98.8	99.1
Neumann	98.9	98.8	99.0	99.2	99.0	98.4	98.8	99.0	98.5	98.2	99.0	99.0	99.0	99.4	99.2
Metropolis	99.1	99.3	99.3	99.0	99.0	98.2	98.4	99.0	98.7	97.0*	98.7	99.0	99.0	98.7	99.0
Coveyou	99.7	99.1	99.6	99.5	99.1	98.0*	83.0*	99.0	99.0	98.0*	99.5	98.7	99.2	99.3	99.2
GMSM	98.8	98.4	99.2	99.1	98.9	99.3	98.5	99.0	98.7	99.0	98.7	98.7	99.0	98.9	99.6
MS Weyl	99.0	99.0	99.4	99.1	99.1	99.1	98.9	99.0	98.6	99.0	98.6	98.6	98.7	98.7	98.8
Alg. M	98.9	98.0*	99.2	98.8	98.9	98.7	99.2	99.0	99.3	98.8	99.5	98.9	99.0	99.2	0.0*
Alg. B	99.1	99.1	97.9*	99.0	99.0	99.1	99.2	99.0	98.6	98.9	98.3	99.1	99.1	98.7	99.1
RC4	98.6	99.3	98.7	98.7	98.7	98.7	98.7	98.7	98.7	98.7	98.7	98.7	98.7	98.7	98.7
Xorshift	99.4	99.1	99.0	99.0	99.0	98.8	99.3	99.0	99.2	98.6	99.0	98.8	99.1	99.2	0.0*
Legendre	98.6	98.3	97.9*	99.1	99.2	98.4	98.5	99.0	98.9	98.5	99.0	99.1	99.2	99.3	99.4

Table 4: NIST test results: proportion

As a result of the observations, most of the discussed PRNGs have successfully passed all tests within the NIST Test Battery. However, in particular cases, I encountered some interesting behavior.

In this compilation, I did not find similar analyses performed with the NIST Test Suite under the same conditions and parameters for the generators mentioned in this chapter. The generators are well-known; I aimed to review various sequences from classic (early) generators to modern generators with specific applications. My goal was to conduct a comprehensive analysis to see how these generators, which operate according to their definitions, perform when tested with the NIST Test Suite.

The most surprising finding was that the LCG and Lehmer's MLCG showed good scores in the tests, while both of them have proven to be cryptographically insecure. One can find an opposite behavior for the BBS, which performed weakly on the tests, but it has proven to be a secure one.

Moreover, for the generators based on John von Neumann's MSM, I had to increase the number of digits of their seeds considerably to achieve sequences passing the tests. For instance, the seed of the original Neumann's MSM was extended to 72-digit. In contrast, the improved GMSM generator, which is defined over algebraic number fields, showed good scores with significantly shorter seed lengths. For instance, the GMSM required only a 92-bit seed with similar properties, even for a relatively simple algebraic extension. In the next Chapter, I am going to present the generator in more detail.

Overall, this part of my research emphasized the importance of empirical testing and the potential discrepancies between theoretical guarantees and practical performance in the evaluation of PRNGs, which is important from the point of view of usage.

2 Canonical Number Systems

This Chapter details a generalization of John von Neumann's Middle-Square Method (MSM) for generating pseudorandom numbers to canonical number systems (CNSs). The Chapter also includes definitions, observations, and statistical tests for the generated sequences. In this Chapter, I am going to present a new approach for generating PRNGs, that can be analyzed using statistical tests.

2.1 Introduction of CNS

In their first appearance, canonical number systems (CNSs) were natural generalizations of the positional numeral systems of rational integers to algebraic integers. D. E. Knuth, in his famous book [22, Ch. 4], studied the history of the radix representation of integers. He described various canonical number systems, and he proved that the complex number $-1 + i$ is a possible base of the number system representing Gaussian integers. His result was further generalized by I. Kátai and J. Szabó. In [21], they proved the existence of other bases, but still in the set of Gaussian integers. Later, Kátai and B. Kovács in [20], B. Kovács in [25] and B. Kovács and A. Pethő in [26], and [27] extended the concept of canonical number systems for the ring of algebraic integers and they proved their existence in some particular cases. In [45], A. Pethő first used the notion, and S. Akiyama and A. Pethő in [48] continued the development of results of CNSs over polynomials. A. Pethő and P. Varga in [2] dealt with canonical number systems over Euclidean domains.

In [25], a simple condition is given for the construction of CNSs, but in general, it is not obvious to find them. For the sake of simplicity, I focused on binary CNSs. A. Kovács [28] presented a complete description of binary canonical number systems of degrees not greater

than 8. Later, Burcsi and A. Kovács [9] extended the results for CNSs of degrees 9, 10, and 11.

The arithmetic in a canonical number system is similar to the rational integers with the classical digit representation. However, calculating the digits requires a more complex reduction operation. My discussion will focus on binary CNSs.

For instance, let α be a root of the polynomial $x^2 + x + 2$, e.g., $-\frac{1}{2} + i\frac{1}{2}\sqrt{7}$. One can prove that in the ring of algebraic integers $\mathbb{Z}[\alpha]$, every number can be written in the canonical form $\gamma = \sum_{i=0}^{h} c_i \alpha^i$, where $c_i \in \{0, 1\}$.

				1	1	0	1	0	1
			+	1	0	1	1	1	1
				2	1	1	2	1	2
			-				1	1	2
				2	1	1	1	0	0
	-	1	1	2					
		-1	-1	0	1	1	1	0	0
+	1	1	2						
	1	0	1	0	1	1	1	0	0

Example 1. Addition in a binary number system

Let $\gamma_1 = 110101$ and $\gamma_2 = 101111$ be 6-bit long binary numbers in the above representation. Then $\gamma = \gamma_1 + \gamma_2$ can be calculated according to Example **1**. Here I used the fact that 1 1 2 represents 0.

In the following, I am going to define the generalized version of the MSM in binary CNS and analyze some properties of these generators.

2.2 Definitions and basic theorems

In this Sub-chapter, I will define some necessary notions and state important results.

Definition 2.1 *Let A be a finite set, and u be a sequence over A. One can say that $u \in A^\infty$ is periodic with period length $\varrho \in \mathbb{N}$, if there exists $\varrho_0 \in \mathbb{N}$, such that*

$$u_{n+\varrho} = u_n \quad \text{for all} \quad n \geq \varrho_0 \ .$$

The smallest ϱ_0 and ϱ with the previous property will be called the preperiod and minimal period length of u, respectively.

If $n \geq \varrho_0$, then the sub-sequence $u_n, \ldots, u_{n+\varrho-1}$ is called a period of the sequence.

Remark 2.2 *Let A be a finite set, $u \in A^\infty$, $0 < k \in \mathbb{N}$ and $F : A^k \to A$. If the sequence satisfies the recurrence defined by $u_n = F(u_{n-1}, \ldots, u_{n-k})$ for all $n \geq k$, then u is periodic with period length $\varrho \leq |A|^k$.*

The following definition is the generalization of number systems for complex numbers given by I. Kátai and J. Szabó in [21].

Definition 2.3 *Let R be an integral domain, $\alpha \in R$, and $\mathcal{N} = \{n_1, \ldots, n_m\} \subseteq \mathbb{Z}$. The pair $(\alpha, \mathcal{N})$ is called a number system in R, if any $\gamma \in R$ has a unique representation in the form $\gamma = \sum_{i=0}^{h} c_i \alpha^i$, where $c_i \in \mathcal{N}$ for all $0 \leq i \leq h$ and $c_h \neq 0$, if $h \neq 0$. The number system called canonical, if $\mathcal{N} = \{0, 1, \ldots, m-1\}$.*

I will use the notation $L(\gamma, \alpha, \mathcal{N}) = h + 1$, i.e. the length of the representation of γ in the number system $(\alpha, \mathcal{N})$.

There exists a more general definition of number systems using lattices, but for the results I have achieved, this classical definition is entirely sufficient.

Theorem 2.4 *Let $p \in \mathbb{Z}[x]$ be an irreducible polynomial with $\deg(p) = d$, and $p(x) = a_d x^d + \cdots + a_0$ such that $1 = a_d \leq a_{d-1} \leq \ldots a_0$ and $2 \leq a_0$. Furthermore, let α be a root of p and $\mathcal{N} = \{0, 1, \ldots, a_0 - 1\}$. Then $(\alpha, \mathcal{N})$ is a canonical number system in $\mathbb{Z}[\alpha]$.*

B. Kovács in [25] gives a sufficient condition for constructing canonical number systems.

Let β be an algebraic number of degree $d \geqq 1$. Then $\beta^{(i)}$ denotes the i^{th} conjugates of β for all $i = 1, \ldots, d$.

Let $\alpha, \gamma \in \mathbb{Q}[\beta]$. For the sake of simplicity, I use the notation

$$|\log|_\alpha \gamma = \max_{1 \leq i \leq d} \frac{\log \left|\gamma^{(i)}\right|}{\log \left|\alpha^{(i)}\right|} \ .$$

In the following, the constants related to both the length and the square length, which will be defined in detail later.

	lower bound	upper bound
lenght	C_1	C_2
length of the squares	C_3	C_4

Theorem 2.5 *Let β be an algebraic integer of degree $d \geqq 1$, and let $(\alpha, \mathcal{N})$ be a number system in $\mathbb{Z}[\beta]$. Then there exist computable constants $C_1 = C_1(\alpha, \mathcal{N})$ and $C_2 = C_2(\alpha, \mathcal{N})$ depending only on α and $\mathcal{N}$, such that*

$$|\log|_\alpha \gamma + C_1 \leqq L(\gamma, \alpha, \mathcal{N}) \leqq |\log|_\alpha \gamma + C_2 \tag{2.1}$$

holds for every $0 \neq \gamma \in \mathbb{Z}[\beta]$.

B. Kovács and A. Pethő in [27] proved that not only the existence of the constants, but also provided a way how to determined them.

Their formula is explicit for C_1 but implicit for C_2. Based on the described method, I have calculated the values of C_1, C_2, C_3, and C_4 for some polynomials.

By the proof of the Thoerem of [27]

$$C_1 = \min_{1 \le i \le n} \frac{\log\left(\left|\alpha^{(i)}\right| - 1\right) - \log(a_0 - 1)}{\log\left|\alpha^{(i)}\right|} \ .$$

For the determination of C_2, one has to compute first some intermediate bounds

$$C_{2,i} = \frac{a_0 - 1}{\left|\alpha^{(i)}\right| - 1} \ .$$

Now, let

$$\Gamma = \left\{ \delta \ \middle| \ \delta \in \mathbb{Z}[\alpha],\ \left|\delta^{(i)}\right| \le C_{2,i} \right\} \ ,$$

and

$$C_2 = \max_{\delta \in \Gamma} L(\delta, \alpha) \ .$$

Remark 2.6 *John von Neumann's MSM uses squaring as the only arithmetic operation. I observe how the length of the numbers changes after squaring.*

I fix α and the corresponding CNS, and I use the notation $C_1 = C_1(\alpha, \mathcal{N})$, $C_2 = C_2(\alpha, \mathcal{N})$ and $L(\gamma) = L(\gamma, \alpha, \mathcal{N})$.

For example in the usual binary representation $\alpha = 2$. The length of the binary representation of an integer n can be expressed by

$$L(n, 2) = \lfloor \log_2(n) \rfloor + 1 = \left\lfloor \frac{\log n}{\log 2} \right\rfloor + 1 \ ,$$

which means that $C_1 = 0$ *and* $C_2 = 1$.

With this simplified notation, equation (3.2) is simplified to

$$|\log|_\alpha \gamma + C_1 \leq L(\gamma) \leq |\log|_\alpha \gamma + C_2 \ . \tag{2.2}$$

Let $\gamma \in \mathbb{Z}[\beta]$ *be an algebraic integer with length* $L(\gamma)$. *By (2.2),*

$$|\log|_\alpha \gamma \leq L(\gamma) - C_1 \ , \tag{2.3}$$

and

$$L(\gamma) - C_2 \leq |\log|_\alpha \gamma \ . \tag{2.4}$$

Applying (2.2), (2.3) and (2.4) to the length of γ^2, *I obtain*

$$\begin{aligned} L(\gamma^2) &\geq |\log|_\alpha \gamma^2 + C_1 = 2|\log|_\alpha \gamma + C_1 \\ &\geq 2\left(L(\gamma) - C_2\right) + C_1 = 2L(\gamma) - 2C_2 + C_1 \end{aligned}$$

and

$$\begin{aligned} L(\gamma^2) &\leq |\log|_\alpha \gamma^2 + C_2 = 2|\log|_\alpha \gamma + C_2 \\ &\leq 2\left(L(\gamma) - C_1\right) + C_2 = 2L(\gamma) - 2C_1 + C_2 \ . \end{aligned}$$

With the notations $C_3 = C_1 - 2C_2$ *and* $C_4 = C_2 - 2C_1$, *I have*

$$2L(\gamma) + C_3 \leq L(\gamma^2) \leq 2L(\gamma) + C_4 \ . \tag{2.5}$$

I should remark that C_1 *and* C_3 *may have negative values. In 2.3, I show some estimates on the values of* C_3 *and* C_4 *for different* α*'s.*

Since $L(\gamma)$ *and* $L(\gamma^2)$ *are integers, thus* C_3 *and* C_4 *can be chosen to be integers without losing precision.*

In the following, I will show the values of the constants for a different set of algebraic integers. The structures are defined by $\mathbb{Z}[\alpha]$, where α's are given by their defining polynomials $p(x)$.

In these computations, the symbol i denotes the imaginary unit (everywhere else in this Chapter, i is an integer). I can calculate the values of C_3 and C_4 for some polynomials.

$p(x) = x^2 + x + 2$:

$$\alpha_{1,2} = -\frac{1}{2} \pm \mathbf{i}\frac{1}{2}\sqrt{7} \qquad |\alpha_1| = |\alpha_2| = \sqrt{2}$$
$$C_{2,1} = C_{2,2} \approx 2.41$$
$$C_1 \approx -2.54 \qquad C_2 = 6$$
$$C_3 = -12 \qquad C_4 = 10$$

$p(x) = x^2 + 2x + 2$ (the case of Gaussian integers, considered by D. E. Knuth in [22, p. 189]):

$$\alpha_{1,2} = -1 \pm \mathbf{i} \qquad |\alpha_1| = |\alpha_2| = \sqrt{2}$$
$$C_{2,1} = C_{2,2} \approx 2.41$$
$$C_1 \approx -2.54 \qquad C_2 = 8$$
$$C_3 = -16 \qquad C_4 = 12$$

$p(x) = x^3 + x^2 + x + 2$:

$$\alpha_1 \approx -1.35 \quad \alpha_{2,3} \approx 0.18 \pm \mathbf{i} \cdot 1.20$$
$$C_{2,1} \approx 2.83 \quad C_{2,2} = C_{2,3} \approx 4.64$$
$$C_1 \approx -7.85 \qquad C_2 = 13$$
$$C_3 = -31 \qquad C_4 = 27$$

$p(x) = x^4 + x^3 + x^2 + x + 2$:

$$C_{2,1} = C_{2,2} \approx 3.97 \quad C_{2,3} = C_{2,4} \approx 7.72$$
$$C_1 \approx -16.77 \qquad C_2 = 21$$
$$C_3 = -46 \qquad C_4 = 32$$

p(x)	C_3	C_4
$p(x) = x^2 + x + 2$	-12	10
$p(x) = x^2 + 2x + 2$	-16	12
$p(x) = x^3 + x^2 + x + 2$	-31	27
$p(x) = x^4 + x^3 + x^2 + x + 2$	-46	32

Table 6: C_3, C_4

In the last two cases, I detailed only some significant steps of the above-mentioned computation.

Related to the previously computed constants, I did some experiments. In Table **7**, I have collected the results of how the sizes of the squares changed after squaring in the considered canonical number systems. The set of four CNSs is extended by the two rational binary number systems with bases 2 and -2.

I consider all the numbers of 20 to 30 digits. For a given length h, the table contains the distances of the minimal and maximal lengths of the squares from the expected $2h$. Additionally, the average lengths of the squares are presented. The last column displays the theoretical bounds for the corresponding values of distances. An important objective was to evaluate how closely the practical results align with the theoretical calculations.

Studying the results, we can conjecture that the minimal and maximal lengths of the squares are considerably closer to the expected value of $2h$ than the analytical computations show. Another suspicion is that the average length of squares is close to $2h$, but increasing the degree of the base α increases the averages.

Length of base numbers

Digits	20	21	22	23	24	25	26	27	28	29	30	T
Defining polynomial: $x - 2$												
Decrease	1	1	1	1	1	1	1	1	1	1	1	1
Increase	0	0	0	0	0	0	0	0	0	0	0	0
Average	39.6	41.6	43.6	45.6	47.6	49.6	51.6	53.6	55.6	57.6	59.6	
Defining polynomial: $x + 2$												
Decrease	3	3	3	3	3	3	3	3	3	3	3	4
Increase	1	1	1	1	1	1	1	1	1	1	1	2
Average	38.9	40.9	42.9	44.9	46.9	48.9	50.9	52.9	54.9	56.9	58.9	
Defining polynomial: $x^2 + x + 2$												
Decrease	8	8	8	8	8	8	8	8	8	8	8	12
Increase	5	5	5	5	5	5	5	5	5	5	5	10
Average	39.6	41.6	43.6	45.6	47.6	49.6	51.6	53.6	55.6	57.6	59.6	
Defining polynomial: $x^2 + 2x + 2$												
Decrease	12	12	12	12	12	12	12	12	12	12	12	16
Increase	9	9	9	9	9	9	9	9	9	9	9	12
Average	40.6	42.6	44.6	46.6	48.6	50.6	52.6	54.6	56.6	58.6	60.6	
Defining polynomial: $x^3 + x^2 + x + 2$												
Decrease	11	14	14	14	14	14	14	14	14	14	14	31
Increase	12	12	12	12	12	12	12	12	12	12	12	27
Average	41.6	43.5	45.5	47.5	49.5	51.5	53.5	55.5	57.5	59.5	61.5	
Defining polynomial: $x^4 + x^3 + x^2 + x + 2$												
Decrease	17	18	18	18	18	18	20	20	20	20	20	46
Increase	21	21	21	21	21	21	21	21	21	21	21	32
Average	43.8	45.6	47.6	49.7	51.9	53.9	55.7	57.7	59.7	61.8	63.8	

Table 7: Lengths of squares

I investigated how closely the practical results align with the theoretically computable lower and upper bounds (denoted as C_3 and C_4) for the length of squares in various canonical number systems. Additionally, I have looked at the specific lengths at which these bounds are reached. The findings indicate that even after refinement, the theoretical values differ significantly from the practical results. While the outcomes seem to approach a constant value. The aim was to assess, for instance, whether numbers of length 30, which are still practically relevant, come close to the theoretical bounds. A potential future goal would be to demonstrate, based on practical values, that these results are final. Further refinement of the theoretical bounds remains possible, particularly in light of practical measurements, which would also allow for the finalization of the practical results.

Squaring Methodology in Generalized Number Systems
When squaring numbers in generalized number systems, the process actually involves performing multiplication according to the system's defined operations and applying carry reduction. This method is consistent with the approach used for addition, where carry operations are managed and reduced as necessary. Although the results obtained during this process may not directly correspond to the canonical number system's representation, we can reduce the digits using the William J. Gilbert's digit reduction method. [15] The Clearing algorithm efficiently manages the carry reduction in operations within CNSs. It calculates carries at each arithmetic step, organizes and merges them for clarity, and finally reduces the digits to their canonical form.
Initially, I used the classical squaring methods, as demonstrated in the Example **1**. However, in later chapters (see, Chapter **4**), I am going to present a more efficient method for computing squares. This approach eliminates the need for multiplication by relying solely on addition, thereby improving computational efficiency while remaining consistent with the principles of generalized number systems.

2.3 Generalized Middle-Square Method

Let $(\alpha, \mathcal{N})$ be a CNS and $p(x) = a_d x^d + \cdots + a_0$ be the defining polynomial of α according to Theorem **3.3**. The usual arithmetic of integers can be generalized to $(\alpha, \mathcal{N})$. The modified carry computation can be derived from p, described below.
Let $\beta \in \mathbb{Z}[\alpha]$ be the result of some arithmetical operation, and $\beta = \sum_{i=0}^{h} b_i \alpha^i$ is the representation without reduction. If for all $0 \leq i \leq h$, $b_i \in \{0, \ldots, a_0 - 1\}$ then β is represented in $(\alpha, \mathcal{N})$. Assume now that there exists $0 \leq i \leq h$ such that $b_i \notin \{0, \ldots, a_0 - 1\}$ and let j be the smallest such integer. Let $c \in \mathbb{Z}$ be such that $b_j = c \cdot a_0 + b'_j$ with $0 \leq b'_j < a_0$. Since

$$0 = a_d \alpha^d + a_{d-1} \alpha^{d-1} + \cdots + a_0 \ ,$$

thus

$$\begin{aligned} \beta &= \sum_{i=0}^{h} b_i \alpha^i + c\alpha^j \cdot (a_n \alpha^d + a_{n-1} \alpha^{d-1} + \cdots + a_0) \\ &= \sum_{i=0}^{h'} b'_i \alpha^i \ , \end{aligned}$$

where $b_i = b'_i$ if $0 \leq i < j$, and $b'_j \in \{0, \ldots, a_0 - 1\}$.
In this new representation of β, either all coefficients are in $\{0, \ldots, a_0 - 1\}$ or the smallest n such that $b_n \notin \{0, \ldots, a_0 - 1\}$ satisfies $j < n$. It is proven in [26], that this iteration will terminate in finitely many steps, providing the unique, valid digit expansion of β in $(\alpha, \mathcal{N})$.
Based on the above observation, one can create an algorithm for the arithmetic operations in $(\alpha, \mathcal{N})$, similar to the usual carry computation used for rational integers.
By Theorem **3.3**, the results of arithmetic operations have finite representation, whence the carry algorithm will always terminate.

Using binary CNSs, I can generalize John von Neumann's MSM.
Let $p(x) \in \mathbb{Z}[x]$ be an irreducible polynomial of degree d, and with coefficients $1 = a_d \leq a_{d-1} \leq \cdots \leq a_0 = 2$. The corresponding CNS has only 2 digits: 0 and 1. For the sake of simplicity, I will call the digits bits and the digit representation of algebraic integers in $\mathbb{Z}[\alpha]$ as a binary representation.
In the design of the generator, I use a seed of $m \in \mathbb{N}$ bits. Similarly, as it is done in the original construction, let u be a sequence over $\mathbb{Z}[\alpha]$ defined by the following, where the α is a root of $p(x)$:

$u_0 \in \mathbb{Z}[\alpha]$ is a random m-bit number;

if $n > 0$, let

$$u_{n-1}^2 = \sum_{i=0}^{h} b_i \alpha^i \ , \text{ with } b_h \neq 0 \ , \ t = \left\lfloor \frac{h-m}{2} \right\rfloor \text{ and}$$
$$u_n = \sum_{i=0}^{m-1} b_{i+t+1} \alpha^i \ .$$

The value of m should be chosen to be large enough, in particular such that $2m + C_3 > m$, i.e. $m > -C_3$, where C_3 is as defined above.
Another approach is if $t = \lfloor \frac{m}{2} \rfloor$, but then $\frac{m}{2} > -C_3$ should hold.

2.3.1 Experimental results

In the following, I am going to provide some experimental results related to the GMSM. I observed the periodicity properties for several base polynomials, particularly those detailed in the previous Subchapters. Furthermore, some statistical tests – the distributions of moving averages, zero-crossing gaps, and frequency classes – are presented for the GMSM generators, where the arithmetics are derived from the polynomials $x^2 + x + 2$ and $x^4 + x^3 + x^2 + x + 2$.

Comparison of the data – both optically and numerically – shows that increasing the degree of the polynomials improves the properties of the generated sequences.
Figure **1** displays the distributions of the moving average of the sequences.

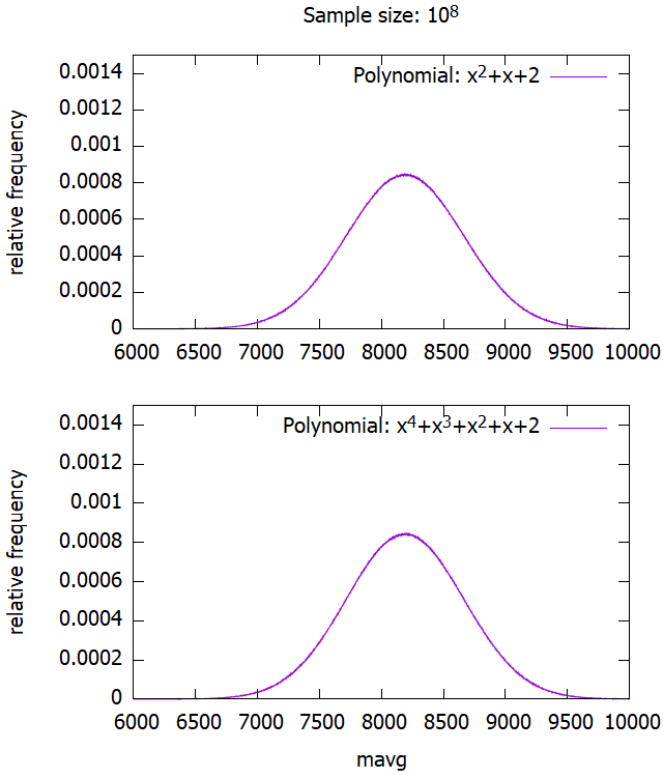

Figure 1: Moving average distribution

I initialized the sequences with randomly chosen integers. The sizes of the samples are 10^8. The seeds are 63-bit words, and the pseudo-random values are obtained by a reduction to the 14-bit prefixes (the

least significant 49 bits are eliminated). The length of the window for the summation is 100, indicating that the average is computed over 100 consecutive data points. This approach helps mitigate short-term fluctuations in the dataset, yielding a smoother moving average curve thereby simplifying the identification of underlying patterns.
I used the following simple formula to compute the sequence of moving averages:

$$e_n = \frac{1}{100} \sum_{i=n}^{n+100} u_i \ ,$$

where (u_i) is the sequence generated by the GMSM.

Next, I observed the generators' behavior under the random walk test. The generated sequences are balanced around 0 by a shift with the mean value: $v_n = u_n - E(u)$. Using the new samples, I computed the cumulative sums:

$$f_n = \sum_{i=0}^{n} v_i \ .$$

The test calculates the frequency of the lengths of the gaps between consecutive zero crossings of f. The results are presented in Figure **2**. The horizontal axis shows the distance and the vertical axis shows the frequency.

Finally, I investigated the distribution of the frequency classes. The values of the sequences are arranged into 2^{14} intervals of equal lengths (again, I reduce the random samples to the 14 most significant bits):

$$\mathrm{U}_i = \left\{ u_k \mid i = \left\lfloor \frac{u_k}{2^{49}} \right\rfloor \right\} \ , \text{ where } i \in \left\{0, \ldots, 2^{14} - 1\right\} \ .$$

My objective is to describe the probability of the event when the same (reduced) random value appears exactly t times for a given t.

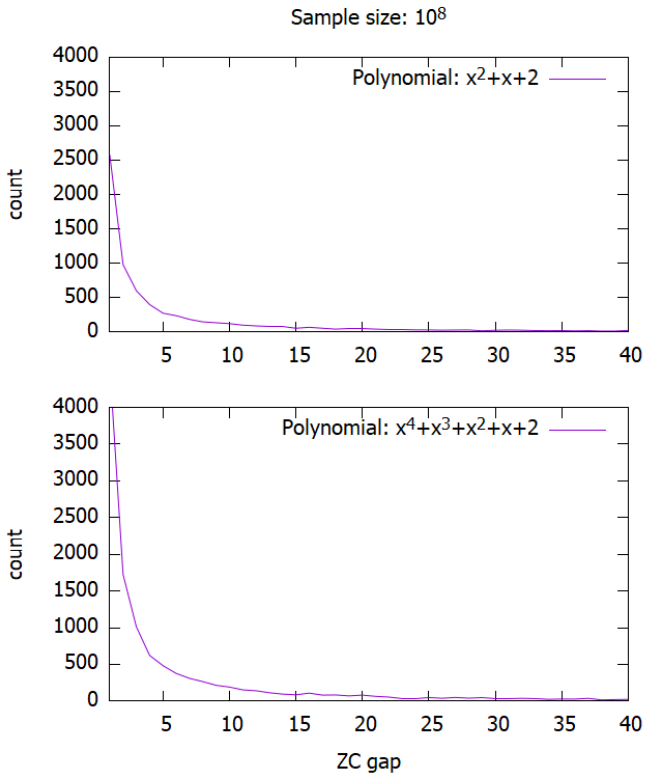

Figure 2: Random walk

For normalization reasons, the minimum and maximum of the cardinalities are computed:

$$\begin{aligned} \min &= \min\left\{|\mathrm{U}_i| \mid i = 0..2^{14} - 1\right\} \text{ and} \\ \max &= \max\left\{|\mathrm{U}_i| \mid i = 0..2^{14} - 1\right\} \ . \end{aligned}$$

Figure **3** displays the distributions of the relative frequencies of the cardinalities of U_k.

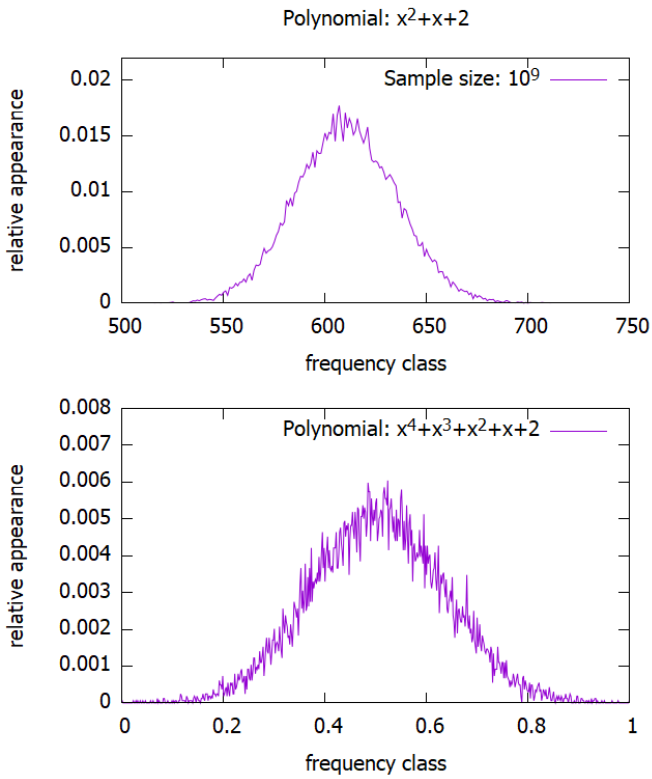

Figure 3: Frequency distribution

The horizontal axis is normalized, and the plotted values are calculated according to the following formulas:

$$x_t = \frac{t - \min}{\max - \min} \ ,$$
$$y_t = \frac{\left|\left\{i \mid |\mathrm{U}_i| = t, 0 \leq i < 2^{14}\right\}\right|}{10^8} \ .$$

Although the above-presented graphs show good properties of the regarded generators, the graphs align with expectations and do not display any extreme behavior, confirming that the generators passed the given test. However, the investigation of a detailed statistical test provides a more accurate description of the behavior of the sequences. I tested two of my generators with the NIST Statistical Test Suite (c.f. [37]). The results are summarized in Tables **9** and **10**. These two are the MSMs corresponding to the polynomials $x^2 + x + 2$ and $x^4 + x^3 + x^2 + x + 2$. I denote them by GMSM1 and GMSM2 in the tables, respectively. In both sequences, I used a 63-bit seed.
The bit sequences for the tests are produced by simply writing the blocks of seeds bit by bit consecutively.
I compared the results with two of the NIST's built-in generators, the LCG and SHA1. The comparison shows that the properties of GMSM sequences are between the two built-in ones.

I used the default parameter adjustments in Table **8**.

Both tests have the same arguments: the lengths of the sample sequences are 1000000, and the numbers of independent bitstreams are 1000. The level of acceptance is left to the default 0.01. In Table **9**, one can see that both generators have an acceptable uniformity level on average.

Test name	Block length
Block frequency	128
Non-overlapping template	9
Overlapping template	9
Approximate entropy	10
Serial	16
Linear Complexity	500

Table 8: NIST default settings

	p-value	
	GMSM1	GMSM2
Frequency	0.574903	0.142872
Block Frequency	0.936823	0.516113
Cumulative Sums	0.225069	0.484351
Runs	0.818343	0.761719
Longest Run	0.015707	0.674543
Rank	0.807412	0.552383
FFT	0.145326	0.368587
Non-Overlapping Template	0.511596	0.501944
Overlapping Template	0.248014	0.825505
Universal	0.152044	0.655854
Approximate Entropy	0.769527	0.353733
Random Excursions	0.292500	0.341976
Random Excursions Variant	0.480915	0.385875
Serial	0.145441	0.236631
Linear Complexity	0.492436	0.347257

Table 9: NIST test results: p-value

Table **10** shows the ratio of the 1000 bitstreams accepted by the tests. Referring to the final report of the NIST test suite, "the minimum pass rate for each statistical test with the exception of the random excursion (variant) test is approximately 0.981819", while "the minimum pass rate for the random excursion (variant) test is approximately 0.979456". Based on this recommendation, both generators have passed all the tests.

	Proportion	
	GMSM1	GMSM2
Frequency	0.9870	0.9890
Block Frequency	0.9890	0.9950
Cumulative Sums	0.9855	0.9890
Runs	0.9880	0.9890
Longest Run	0.9870	0.9900
Rank	0.9870	0.9860
FFT	0.9930	0.9870
Non-Overlapping Template	0.9905	0.9895
Overlapping Template	0.9860	0.9910
Universal	0.9920	0.9920
Approximate Entropy	0.9880	0.9850
Random Excursions	0.9853	0.9930
Random Excursions Variant	0.9866	0.9912
Serial	0.9905	0.9965
Linear Complexity	0.9900	0.9920

Table 10: NIST test results: Proportion

I would like to note that some of the tests are not simple but rather result from the combination of multiple tests (16), such as the Non-Overlapping Template Matching test. In these cases, the NIST performs multiple tests within a single test.

Last but not least, in Table **11**, I have collected the periodicity properties of the same GMSM sequences as in Table **7**.
Again, one block corresponds to the CNS given by the defining polynomial of its base. The entries are:

- the number of disjoint cycles;
- the maximal length of the cycles;
- the number of the length-1 cycles

for the different seed sizes. The trivial 0-cycle is excluded from the table.
The first block contains test results in the CNS with base 2, i.e., the simple binary representation of non-negative rational integers.
In the second block, the number system is the extension of the previous one to the whole set of integers with base -2.

It must be added that even if they have small period lengths, the sequences can be used for pseudorandom number generators because of the long preperiod. Increasing the size of the seed typically increases both the period length and the length of the longest period, but not in a monotonous way. This means that with a larger seed size, the period length might sometimes be shorter; however, in general, as the seed size increases, the period length also tends to increase, although there are cases where it may decrease.

Nontrivial cycles

Digits (seed)	10	11	12	13	14	15	16	17	18	19	20

Defining polynomial: $x - 2$

Cycles	4	4	6	4	9	12	12	10	11	6	12
Max period length	5	5	10	2	56	70	111	203	197	2	142
Stability points	2	3	3	3	3	3	4	4	6	5	6

Defining polynomial: $x + 2$

Cycles	2	6	7	7	11	12	16	11	13	18	18
Max period length	3	3	2	34	10	27	51	30	2	39	4
Stability points	1	3	4	3	5	4	6	5	8	9	8

Defining polynomial: $x^2 + x + 2$

Cycles	3	4	4	2	4	6	3	3	4	9	7
Max period length	2	2	10	19	10	13	34	21	13	256	476
Stability points	2	3	1	1	1	1	1	1	2	2	2

Defining polynomial: $x^2 + 2x + 2$

Cycles	2	4	6	5	5	7	5	4	7	12	13
Max period length	1	2	2	5	5	11	20	2	7	24	117
Stability points	2	3	4	2	2	2	2	3	5	9	8

Defining polynomial: $x^3 + x^2 + x + 2$

Cycles	10	13	6	6	3	1	5	6	7	11	5
Max period length	5	5	9	5	1	1	7	67	20	165	57
Stability points	8	10	4	5	3	1	3	3	3	3	1

Defining polynomial: $x^4 + x^3 + x^2 + x + 2$

Cycles	5	8	6	5	5	7	10	11	6	6	8
Max period length	13	19	4	12	83	22	57	54	270	125	258
Stability points	2	1	3	3	3	3	6	7	2	2	3

Table 11: All cycles in GMSM sequences

3 Arithmetics in CNSs

In this Chapter, I am going to discuss some properties of arithmetics in CNSs over algebraic integers. I defined the information quantity of a polynomial $A \in \mathbb{Z}[x]$ relative to the base of the CNS and proved that it has a strong relation with the length of the representation in the number system. Based on this result, I am going to show that for every CNS polynomial P, there exists a finite transducer automaton executing the addition operation of polynomials in canonical representation of base P. Finally, I will present the observed size – i.e., the number of states – of such automata.

3.1 General overview

Canonical number systems (CNSs) represent a natural extension of positional numeral systems from rational integers to algebraic integers, essentially generalizing radix representations of ordinary integers. I gave a general overview of the CNSs in Chapter **2**. As I mentioned, the arithmetic in a canonical number system is similar to the rational integers with the classical digit representation. However, calculating the digits requires a more complex reduction operation.
P. J Grabner, P. Kirschenhofer, and H. Prodinger in [16] introduced the concept of counting automata (increasing by 1) for Gaussian integers, which was generalized to quadratic number fields by J. M. Thuswardner [55]. J. Allouche, K. Scheicher, and R. F. Tichy in [4] proved the existence of addition and multiplication by a constant in algebraic number fields. Scheicher and J. M. Thuswaldner in [51] introduced a fractal approach for detailed analysis of counting automata.
Transducers were extensively studied by S. Eilenberg in Automata, Languages and Machines [50], and further developed by W. Kuich and A. Salomaa in Semirings, Automata, Languages [56].

Furthermore, I have proven that for every CNS polynomial, there exists an automaton that executes the digit-wise addition (Theorem **3.21**). The proof of the theorem requires the generalization of the result of B. Kovács and A. Pethő [27], which is done in Theorem **3.18**. The proof of Theorem **3.18** also provides an approach for calculating the constants appearing in the statement (Algorithm **1**). For better estimation of these constants, the usual upper bound of geometric sums has been improved in Theorem **3.7**.
In the following, I am going to present the algorithm I have implemented for the calculation of addition automata in binary CNSs and I will present the experiments.

3.2 Definitions and results related to CNSs

Several papers have been published about the existence and determination of CNSs in a given number field. This Sub-chapter deals with some basic notions and results related to canonical number systems. In general, it is not obvious to prove the existence or find a particular CNS over the algebraic integers $\mathbb{Z}[\alpha]$. Based on some exhausting search, A. Kovács and P. Burcsi in [9] described all irreducible polynomials of degree not greater than 11 defining a number field having binary canonical number systems.

Definition 3.1 *Let $P \in \mathbb{Z}[x]$ be a monic polynomial, $R = \mathbb{Z}[x]/P\mathbb{Z}[x]$ be the corresponding quotient ring, and let $\varphi : \mathbb{Z}[x] \to R$ be the homomorphism $\varphi(A) = A \mod P$. Furthermore, let $m > 1$, $\mathcal{N} = \{n_1, \ldots, n_m\} \subseteq \mathbb{Z}$, and let $\mathcal{N}[x] \subseteq \mathbb{Z}[x]$ be the set of polynomials with all coefficients from $\mathcal{N}$. The pair $(P, \mathcal{N})$ is called a number system, if for any $A \in \mathbb{Z}[x]$ there exists a unique $B \in \mathcal{N}[x]$, such that $\varphi(A) = \varphi(B)$.*
The number system is called canonical, and $P(x)$ is called a CNS polynomial if $\mathcal{N} = \{0, 1, \ldots, m-1\}$.

The length of the representation of A by P is $L(A) = L(A,P) = \deg(B)+1$.

Remark 3.2 *If $(P,\mathcal{N})$ is a number system, then the reduced homomorphism $\varphi : \mathcal{N}[x] \to R$ is an isomorphism.*

B. Kovács in [25] proved that if the coefficients of P are increasing, then there exists a corresponding canonical number system.

Theorem 3.3 *Let $P \in \mathbb{Z}[x]$ be a monic polynomial with $\deg(P) = d$, and $P(x) = p_d x^d + \cdots + p_0$ be such that $1 = p_d \leq p_{d-1} \leq \cdots \leq p_0$ with $2 \leq p_0$. If P is not divisible by a cyclotomic polynomial, then P is a CNS polynomial.*

The statement of the theorem is proven in [25].
A. Pethő in [46] has stated a necessary condition for CNS polynomials.

Theorem 3.4 *If $P(x)$ is a CNS polynomial, then all roots of $P(x)$ lie outside of the closed unit circle, and all real roots of $P(x)$ are less than -1.*

A kind of information quantity of a polynomial can be defined. This information quantity of the polynomial is closely related to the length of the canonical representation of it. [49]

Definition 3.5 *Let $P(x)$ be a CNS polynomial of degree d, $\alpha_1, \ldots, \alpha_d$ be the (not necessarily different) roots of P, and let $A(x) \in \mathbb{Z}[x]$. We define the relative information quantity of A by*

$$\mathbb{I}(A) = \mathbb{I}_P(A) = \max_{\substack{1 \leq i \leq d \\ A(\alpha_i) \neq 0}} \frac{\log|A(\alpha_i)|}{\log|\alpha_i|} \ .$$

In the special case $P|A$ we set $\mathbb{I}(A) = 0$.

Lemma 3.6 *Let $P(x)$ be a CNS polynomial and $A(x), B(x) \in \mathbb{Z}[x]$. Then*

i) $0 \leq \mathbb{I}(A)$

ii) If $\varphi(A) = \varphi(B)$, *then* $\mathbb{I}(A) = \mathbb{I}(B)$.

iii) $\mathbb{I}(A + B) \leq \max\{\mathbb{I}(A), \mathbb{I}(B)\} + \mathbb{I}(2)$;

iv) $\mathbb{I}(A \cdot B) \leq \mathbb{I}(A) + \mathbb{I}(B)$.

v) $\mathbb{I}(x \cdot A) = \mathbb{I}(A) + 1$.

vi) $\mathbb{I}(A^n) = n\mathbb{I}(A)$.

i) Let P' be an irreducible divisor of P, and A is not divisible by P'. Then

$$0 \neq \prod_{\substack{1 \leq i \leq d \\ P'(\alpha_i)=0}} A(\alpha_i)$$

is an integer, whence at least one of the roots α of P' is such that $|A(\alpha)| \geq 1$, and

$$0 \leq \frac{\log |A(\alpha)|}{\log |\alpha|} .$$

ii) If $\varphi(A) = \varphi(B)$, then $A(x) \equiv B(x) \mod P(x)$, i.e. $A(x) = Q(x)P(x) + B(x)$ with some $Q \in \mathbb{Z}[x]$. This yields that $A(\alpha) = B(\alpha)$ for any root of P, which implies $\mathbb{I}(A) = \mathbb{I}(B)$.

iii) Let α be a root of P and assume that $|A(\alpha)| \geq |B(\alpha)|$. Then $\log(|A(\alpha) + B(\alpha)|) \leq \log(|A(\alpha)| + |B(\alpha)|) \leq \log(2 \cdot |A(\alpha)|) = \log(2) + \log(|A(\alpha)|)$, which implies the statement.

iv)-vi) These statements can be proven by similar arguments using the additive property of the logarithm.

Theorem 3.7 *Let $\alpha \in \mathbb{C}$ with $1 < |\alpha|$, and assume that α is not a positive real number. Then there exists a constant $c < 1$ depending only on α such that for every $0 < N \in \mathbb{Z}$, and for any sequence $a_0, a_1, \ldots$ of integers satisfying $0 \leq a_i \leq N$ for all $i = 0, 1, \ldots$, we have*

$$\left| \sum_{i=0}^{k-1} a_i \alpha^i \right| \leq cN \frac{|\alpha|^k}{|\alpha| - 1} \ ,$$

for any $1 < k \in \mathbb{Z}$.

For the proof of Theorem **3.7**, we need some further lemmata.

Lemma 3.8 *Let $\alpha \in \mathbb{C}$, with $1 < |\alpha|$, $1 < N \in \mathbb{Z}$, let $0 \leq a_0, a_1, \ldots, a_{n-1} \leq N$ be integers, and*

$$M = \max \left\{ \left| \sum_{i=0}^{n-1} b_i \cdot \alpha^i \right| : 0 \leq b_0, b_1, \ldots, b_{n-1} \leq N, \textit{integers} \right\} \ .$$

If

$$\left| \sum_{i=0}^{n-1} a_i \cdot \alpha^i \right| = M \ ,$$

then $a_{n-1} \neq 0$.

Contrary, if $a_{n-1} = 0$, then

$$M = \left| \sum_{i=0}^{n-2} a_i \cdot \alpha^i \right| < |\alpha| \cdot \left| \sum_{i=0}^{n-2} a_i \cdot \alpha^i \right| = \left| \sum_{i=1}^{n-1} a_{i-1} \cdot \alpha^i \right| \leq M \ ,$$

which is a contradiction.

Lemma 3.9 *With the notations of Lemma **3.8**, let $0 < r < n$ be integers, and*

$$M_n = \max\left\{\left|\sum_{i=0}^{n-1} b_i \cdot \alpha^i\right| : 0 \le b_0, b_1, \ldots, b_{n-1} \le N, \textit{integers}\right\}$$

$$M_r = \max\left\{\left|\sum_{i=0}^{r-1} b_i \cdot \alpha^i\right| : 0 \le b_0, b_1, \ldots, b_{r-1} \le N, \textit{integers}\right\}$$

Then $|\alpha|^{n-r} \cdot M_r \le M_n$.

Let $0 \le a_0, a_1, \ldots, a_{r-1} \le N$ be integers, with

$$\left|\sum_{i=0}^{r-1} a_i \cdot \alpha^i\right| = M_r \ .$$

Then

$$|\alpha|^{n-r} \cdot M_r = \left|\sum_{i=n-r}^{n-1} a_{i-n+r} \cdot \alpha^i\right| \le M_n \ .$$

Corollary 3.10 *With the notations of Lemma **3.8**, let $1 < k, n$, with $k = q \cdot n + r$, where $0 \le r < n$, and $0 \le a_0, \ldots, a_{k-1} \le N$ are integers. Then*

$$\left|\sum_{i=0}^{k-1} b_i \cdot \alpha^i\right| \le M \frac{|\alpha|^k}{|\alpha|^n - 1} \ .$$

Assume first that $k < n$. Then by Lemma **3.9**,

$$\left|\sum_{i=0}^{k-1} b_i \cdot \alpha^i\right| \le M \frac{1}{|\alpha|^{n-k}} < M \frac{|\alpha|^k}{|\alpha|^n - 1} \ .$$

Now, let $k \geq n$. If $0 < r$, then by Lemma **3.9**,

$$\begin{aligned}
\left|\sum_{i=0}^{k-1} b_i \cdot \alpha^i\right| &\leq \left|\sum_{i=0}^{r-1} b_i \cdot \alpha^i\right| + \sum_{j=0}^{q-1} |\alpha|^{j\cdot n+r} \left|\sum_{i=0}^{n-1} b_{j\cdot n+r+i} \cdot \alpha^i\right| \\
&\leq |\alpha|^{r-n} M + |\alpha|^r \sum_{j=0}^{q-1} |\alpha|^{j\cdot n} M \\
&= M \cdot \left(|\alpha|^{r-n} + |\alpha|^r \frac{|\alpha|^{q\cdot n} - 1}{|\alpha|^n - 1}\right) \\
&= M \cdot \left(\frac{|\alpha|^{r-n} \cdot (|\alpha|^n - 1)}{|\alpha|^n - 1} + \frac{|\alpha|^{q\cdot n+r} - |\alpha|^r}{|\alpha|^n - 1}\right) \\
&= M \frac{|\alpha|^r - |\alpha|^{r-n} + |\alpha|^{q\cdot n+r} - |\alpha|^r}{|\alpha|^n - 1} \\
&= M \frac{|\alpha|^k - |\alpha|^{r-n}}{|\alpha|^n - 1} \leq M \frac{|\alpha|^k}{|\alpha|^n - 1} \ .
\end{aligned}$$

If $r = 0$, then the proof is similar but even simpler.

Lemma 3.11 *With the notations of Lemma **3.8**, if*

$$\left|\sum_{i=0}^{n-1} a_i \cdot \alpha^i\right| = M \ ,$$

then $a_0, a_1, \ldots, a_{n-1} \in \{0, N\}$.

Let $0 \leq j < n$ be such that $a_j \neq 0$, and $\mu = \sum_{i=0}^{n-1} a_i \cdot \alpha^i$. Since $M = |\mu|$, thus $\left|\mu - \alpha^j\right| \leq |\mu|$. Denote by $\bar{x}$ the complex conjugate of x. Then $(\mu - \alpha^j)(\bar{\mu} - \bar{\alpha}^j) = \left|\mu - \alpha^j\right|^2 \leq |\mu|^2 = \mu\bar{\mu}$, which implies

$$0 < \alpha^j \bar{\alpha}^j \leq \mu\bar{\alpha}^j + \bar{\mu}\alpha^j \ . \tag{3.1}$$

Assume that $a_j < N$. By the maximality of M, $|\mu + \alpha^j| \leq M$. On the other hand, $M^2 \geq \left|\mu + \alpha^j\right|^2 = (\mu + \alpha^j)(\bar{\mu} + \bar{\alpha}^j) = \mu\bar{\mu} + \alpha^j\bar{\alpha}^j + \mu\bar{\alpha}^j + \bar{\mu}\alpha^j > \mu\bar{\mu} = |\mu|^2 = M^2$, which is a contradiction.

The previous results can be summarized in the following corollary.

Corollary 3.12 *With the notations of Lemma **3.8**, there exist* $a_0, a_1, \ldots, a_{n-1} \in \{0, 1\}$ *such that*

$$N \left| \sum_{i=0}^{n-1} a_i \cdot \alpha^i \right| = M \ ,$$

where $a_{n-1} = 1$.

Lemma 3.13 *With the notations of Lemma **3.8**, let* $a_0, a_1, \ldots, a_{n-1} \in \{0, 1\}$ *such that*

$$N \left| \sum_{i=0}^{n-1} a_i \cdot \alpha^i \right| = M \, , \ \textit{and let} \ \ \mu = \sum_{i=0}^{n-1} a_i \cdot \alpha^i \ .$$

Then $a_j = 1$ *if and only if* $\dfrac{\mu}{\alpha^j} + \dfrac{\bar{\mu}}{\bar{\alpha}^j} > 0$.

Assume first, that $a_j = 1$. Then $|\mu - \alpha^j| \leq |\mu|$, whence by (3.1),

$$\frac{\mu}{\alpha^j} + \frac{\bar{\mu}}{\bar{\alpha}^j} = \frac{\mu\bar{\alpha}^j + \bar{\mu}\alpha^j}{\alpha^j\bar{\alpha}^j} \geq 1 \ .$$

Now, let $a_j = 0$. Then $|\mu + \alpha^j| \leq |\mu|$, whence $0 < \alpha^j\bar{\alpha}^j \leq -\left(\mu\bar{\alpha}^j + \bar{\mu}\alpha^j\right)$, and

$$\frac{\mu}{\alpha^j} + \frac{\bar{\mu}}{\bar{\alpha}^j} = \frac{\mu\bar{\alpha}^j + \bar{\mu}\alpha^j}{\alpha^j\bar{\alpha}^j} \leq -1 \ .$$

Remark 3.14 *By the proof of Lemma **3.13**, there is no* $0 \leq j < n$ *for which* $\dfrac{\mu}{\alpha^j} + \dfrac{\bar{\mu}}{\bar{\alpha}^j} = 0$.

Corollary 3.15 *With the notations of Lemma **3.8**, let* $a_0, a_1, \ldots, a_{n-1} \in \{0, 1\}$ *such that*

$$N\left|\sum_{i=0}^{n-1} a_i \cdot \alpha^i\right| = M\,, \qquad \mu = \sum_{i=0}^{n-1} a_i \cdot \alpha^i\,,$$

and let $\mathcal{I}(\beta)$ *be the imaginary part of* $\beta \in \mathbb{C}$. *Then there exist* $0 \leq j, k < n$ *such that* $a_i = 1$ *if and only if* $\mathcal{I}\left(\frac{\alpha^i}{\alpha^j}\right) \geq 0$ *and* $\mathcal{I}\left(\frac{\alpha^i}{\alpha^k}\right) \leq 0$, *for all* $0 \leq i < n$.

Let $\preceq$ be the binary relation on $\mathbb{C} \setminus \{0\}$, where $x \preceq y$ if and only if $\mathcal{I}\left(\frac{y}{x}\right) \geq 0$. Then for any $\beta \in \mathbb{C} \setminus \{0\}$, the relation $\preceq$ is transitive on the set

$$\mathbb{C}_\beta = \left\{\gamma : 0 \leq \frac{\beta}{\gamma} + \frac{\bar{\beta}}{\bar{\gamma}}\,, \quad \text{and} \quad \gamma \in \mathbb{C} \setminus \{0\}\right\}\,.$$

Let $\mathcal{C}_\mu = \left\{\alpha^i : 0 \leq \dfrac{\mu}{\alpha^i} + \dfrac{\bar{\mu}}{\bar{\alpha}^i}\,,\ 0 \leq i < n\right\} \subseteq \mathbb{C}_\mu$. Clearly, $a_i = 1$ if and only if $\alpha^i \in \mathcal{C}_\mu$. Since $\mathcal{C}_\mu$ is a finite set, by the transitivity property, there exists not necessarily unique $0 \leq j, k < n$ such that $\alpha^j \preceq \gamma \preceq \alpha^k$ for all $\gamma \in \mathcal{C}_\mu$. If $a_i = 0$, then $\alpha^i \notin \mathcal{C}_\mu$, and either $\alpha^i \prec \alpha^j$ or $\alpha^k \prec \alpha^i$.

[Proof of Theorem **3.7**]
Let $0 < n$ be a sufficiently large integer, and accordingly to the notation of Lemma **3.8**, let

$$N\left|\sum_{i=0}^{n-1} a_i \cdot \alpha^i\right| = M\,, \quad \text{and} \quad \mu = \sum_{i=0}^{n-1} a_i \cdot \alpha^i\,.$$

Let $c = \dfrac{|\mu|\,(|\alpha|-1)}{|\alpha|^n-1}$. Since α is not a positive real number, by Lemma **3.13**, at least for one i the coefficient $a_i \neq 1$, whence $c < 1$.
By Corollary **3.10**,

$$\left|\sum_{i=0}^{k-1} b_i \cdot \alpha^i\right| \leq M\frac{|\alpha|^k}{|\alpha|^n-1} = Nc\frac{|\alpha|^n-1}{|\alpha|-1}\frac{|\alpha|^k}{|\alpha|^n-1} = cN\frac{|\alpha|^k}{|\alpha|-1}\ .$$

Corollary 3.16 *Let $\alpha \in \mathbb{C}$ with $1 < |\alpha|$, and assume that α is not a positive real number. Let $0 < N \in \mathbb{Z}$ and*

$$M_n = \max\left\{\left|\sum_{i=0}^{n-1} b_i \cdot \alpha^i\right| : 0 \leq b_0, b_1, \ldots, b_{n-1} \leq N, \textit{integers}\right\}\ ,$$

for all $0 < n \in \mathbb{Z}$. Then there exists $c < 1$ depending only on α such that

$$\lim_{n\to\infty}\frac{M_n}{|\alpha|^n} = \frac{cN}{|\alpha|-1}\ .$$

By Theorem **3.7**,

$$\frac{M_n}{|\alpha|^n} \leq \frac{c_1 N}{|\alpha|-1}\ ,$$

for all $0 < n \in \mathbb{Z}$ with some $c_1 < 1$. By Lemma 3.9, $|\alpha|M_{n-1} \leq M_n$. Hence the sequence $\dfrac{M_n}{|\alpha|^n}$ is an increasing bounded sequence, which implies the corollary.

Remark 3.17 *In a particular case, c can be an arbitrarily small positive number. For instance, if $\alpha \in \mathbb{R}$ and $\alpha < -1$, then $c = \dfrac{1}{|\alpha|+1}$.*

Based on the previous results, a fast algorithm to find the value of c in Theorem **3.7** can be constructed. First, the definition of the relation introduced in the proof Corollary **3.10** is modified.

Now, let $\preceq$ be the binary relation on $\mathbb{C}\setminus\{0\}$, where $x \preceq y$ if and only if either $\mathcal{I}\left(\frac{y}{x}\right) > 0$ or $\mathcal{I}\left(\frac{y}{x}\right) = 0$ and $|y| \geq |x|$. This $\preceq$ is a total ordering on any half-plane $\mathbb{C}_\beta$.
Choose a sufficiently large n – there may have better constants if $\mathcal{I}(\alpha^n)$ is small –, and let

$$\mathcal{C}_1 = \left\{\alpha^i : 0 \leq \alpha^i + \bar{\alpha}^i,\ 0 \leq i < n\right\}, \quad \text{and}$$
$$\mathcal{C}_{-1} = \left\{\alpha^i : 0 > \alpha^i + \bar{\alpha}^i,\ 0 \leq i < n\right\}.$$

Sort $\mathcal{C}_1$ and $\mathcal{C}_{-1}$ by $\preceq$ and glue them at both ends. Then get a cyclically sorted set $\beta_0, \ldots, \beta_{n-1}$ of $\alpha^0, \ldots, \alpha^{n-1}$, where the next of β_{n-1} is $\beta_n = \beta_0$. To find μ and then c, Algorithm **1** with $O(n \cdot 2^n)$ time complexity can be used. It computes the values specified in the statements below. The nested `while` loops calculate the appropriate values. The inner `while` loop corresponds to Remark **3.14**.

Algorithm 1: Calculating the constant c of Theorem **3.7**

Data: $\beta_0, \ldots, \beta_{n-1}$
Result: c
$\max \leftarrow 0$, $i \leftarrow 0$, $j \leftarrow 0$, $\mu \leftarrow 0$;
while $i < n$ **do**
  while $\mathcal{I}\left(\frac{\beta_j}{\beta_i}\right) > 0$ **do**
    $\mu \leftarrow \mu + \beta_j$;
    $j \leftarrow j + 1 \mod n$;
    if $\max < |\mu|$ **then**
      $\max \leftarrow |\mu|$;
    end
  end
  $\mu \leftarrow \mu - \beta_i$;
  $i \leftarrow i + 1$;
end
$c \leftarrow \max \cdot \dfrac{|\alpha| - 1}{|\alpha|^n - 1}$;
return c;

Theorem 3.18 *Let $P(x) = x^d + p_{d-1}x^{d-1} + \cdots + p_0$ be a CNS polynomial. Then, there exist computable constants c_1, c_2 such that*

$$\mathbb{I}(A) - c_1 \leq L(A) \leq \mathbb{I}(A) + c_2 \tag{3.2}$$

holds for every $A(x) \in \mathbb{Z}[x]$.

The proof is based on the idea of [27]. The new theorem is a more generalized form of the previous result, and its proof introduces a refined method for calculating the constants c_1 and c_2. This refinement allows for more precise constants compared to earlier approaches.
Specifically, I have extended the theorem by B. Kovács and A Pethő, offering a new method for calculating these constants with greater

accuracy. The key contribution of this work is the development of an improved algorithm for determining c_1 and c_2, leading to more exact results in the context of canonical number systems.

Let $(P, \mathcal{N})$ be a CNS, with $N = |\mathcal{N}|$, and $A(x) \in \mathbb{Z}[x]$ with canonical representation $B(x) = b_{L-1}x^{L-1} + \cdots + b_0 \in \mathcal{N}[x]$, where $L = L(A) = L(B) = \deg(B) + 1$. Then $\varphi(A) = \varphi(B)$, which means $\mathbb{I}(A) = \mathbb{I}(B)$. Assume that α is a root of $P(x)$. Then

$$\begin{aligned} |B(\alpha)| &= |b_{L-1}\alpha^{L-1} + b_{L-2}\alpha^{L-2} + \cdots + b_0| \\ &\leq |b_{L-1}||\alpha|^{L-1} + |b_{L-2}||\alpha|^{L-2} + \cdots + |b_0| \\ &\leq (N-1)\frac{|\alpha|^L - 1}{|\alpha| - 1} < N\frac{|\alpha|^L}{|\alpha| - 1} \ . \end{aligned}$$

By Theorem **3.4**, $0 < |\alpha| - 1$ validates the last inequality and makes it possible to take the logarithm of both sides. Then we get

$$\log(|B(\alpha)|) < \log(N) + L\log(|\alpha|) - \log(|\alpha| - 1) \ .$$

Again by Theorem **3.4**, we may have

$$\frac{\log(|B(\alpha)|)}{\log(|\alpha|)} < L + \frac{\log(N) - \log(|\alpha| - 1)}{\log(|\alpha|)} \qquad \text{and}$$

$$\frac{\log(|B(\alpha)|)}{\log(|\alpha|)} - \frac{\log(N) - \log(|\alpha| - 1)}{\log(|\alpha|)} < L \ ,$$

which can be effectively bounded from below by some c_1:

$$\frac{\log(|B(\alpha)|)}{\log(|\alpha|)} - c_1 < L \ .$$

Taking the maximum on all roots of P, the first inequality of (**3.2**) is obtained

$$\mathbb{I}(A) - c_1 \leq L(A) \ .$$

For the proof of the second inequality, set $k = \lceil \mathbb{I}(A) \rceil$, which denotes the upper integer part of $\mathbb{I}(A)$, and let $B(x) = x^k B_1(x) + B_0(x)$, where $B_0, B_1 \in \mathcal{N}[x]$ and $\deg(B_0) < k$. Assume that α is a root of P. Then

$$B_1(\alpha) = \frac{B(\alpha) - B_0(\alpha)}{\alpha^k} \qquad \text{and}$$

$$\begin{aligned} |B_1(\alpha)| &\leq \frac{|B(\alpha)|}{|\alpha|^k} + \frac{|B_0(\alpha)|}{|\alpha|^k} \\ &< \frac{|B(\alpha)|}{|\alpha|^k} + N\frac{|\alpha|^k - 1}{|\alpha|^k(|\alpha| - 1)} \\ &< \frac{|B(\alpha)|}{|\alpha|^k} + \frac{N}{|\alpha| - 1} \ . \end{aligned}$$

The fraction $\dfrac{c_\alpha \cdot N}{|\alpha| - 1}$ can be bounded by a computable constant depending only on P, and by the definition of k, $\dfrac{|B(\alpha)|}{|\alpha|^k} = \dfrac{|A(\alpha)|}{|\alpha|^k} \leq 1$. Hence, $|B_1(\alpha)| \leq c$ where c is a constant depending only on P. Since there exist only finitely many polynomials $A^{'} \in \mathbb{Z}[x]$ with $\deg(A^{'}) < d$ satisfying $A^{'}(\alpha) < c$ and these polynomials are computable, then it can be determined

$$c'_\alpha = \max \left\{ \deg\left(B^{'}\right) : B^{'} \in \mathcal{N}[x], \varphi\left(B^{'}\right) = \right.$$

$\left. \varphi\left(A^{'}\right), \deg(A^{'}) < d, A^{'}(\alpha) < c + 1 \right\} .Let c_2 = 1 + \max\limits_{1 \leq i \leq d} c'_{\alpha_i}$.
Then

$$L(A) = \deg(B) + 1 = k + \deg(B_1) + 1 \leq k + c'_\alpha \leq \mathbb{I}(A) + c_2 \ .$$

3.3 Automata and canonical number systems

As it could be seen above, adding and multiplying by a constant are implementable with automata in CNSs. In this Sub-chapter, I am going to prove that there exists an automaton that executes the digit-wise addition. The existence of such an automaton is not obvious since the length of the sum of two algebraic integers represented in the canonical number system may have a shorter length than the summands'.

Definition 3.19 *The 6-tuple $\mathcal{A} = (k, S, X, Y, s, \delta)$ is called a finite transducer automaton with k input tape if*

$k \in \mathbb{Z}^+$;

S *is a finite, non-empty set of states;*

X *is a finite set of at least 2 elements (the input alphabet);*

Y *is a finite set of at least 2 elements (the output alphabet);*

$s \in S$ *is the initial state;*

$\delta : S \times X^k \longrightarrow S \times Y$ *is a unique mapping.*

Remark 3.20 *We follow the convention for extending the transducer automaton to operate on words of the input alphabet and having the result as a word of the output alphabet.*

In the following, if $\bar{a} = (a_0 \dots a_{n-1}) \in \mathcal{N}^*$ is a finite word, then we will use the notation $Q_a(x) = a_{n-1}x^{n-1} + \cdots + a_0$ for the corresponding polynomial.

Theorem 3.21 *Let $(P, \mathcal{N})$ be a CNS. Then there exists a finite transducer automaton $\mathcal{A}$ with two input tapes, such that $\bar{c} = \mathcal{A}(\bar{a}, \bar{b})$ for all $\bar{a}, \bar{b}, \bar{c} \in \mathcal{N}^*$ if $\varphi(Q_c) = \varphi(Q_a + Q_b)$.*

Remark 3.22 *In Theorem **3.21**, we assume that the length of the words $\bar{a}, \bar{b}, \bar{c}$ are equal. If not, we extend them with the necessary amount of 0s on the right.*

I prove the statement by constructing the addition automaton $\mathcal{A} = (2, S, X, Y, s, \delta)$, where $X = Y = \mathcal{N}$. I will define $S \subseteq \mathcal{N}[x]$ and δ in an appropriate way, and I will prove that S is finite. Assume that $N = |\mathcal{N}|$.
Let S be the smallest set such that

1. $0 \in S$, and
2. if $A \in S$, $B \in \mathcal{N}[x]$, $b \in \mathcal{N}$, and $0 \leq a < 2N - 1$ such that $\varphi\left(xB(x) + b\right) = \varphi\left(A(x) + a\right)$ then $B \in S$.

We define the transition function $\delta : S \times \mathcal{N}^2 \longrightarrow S \times \mathcal{N}$ by $\delta\left(A, a_1, a_2\right) = (B, b)$, if $\varphi\left(xB(x) + b\right) = \varphi\left(A(x) + a_1 + a_2\right)$.
Let $\bar{a}, \bar{b} \in \mathcal{N}^*$, $A_0 = 0$, and $Q_0 = Q_a + Q_b$. If $0 \leq i$, let A_{i+1} be defined by $\delta\left(A_i, a_i, b_i\right) = (A_{i+1}, c_i)$, and $Q_{i+1}(x) = Q_i(x) + (c_i - a_i - b_i)x^i$. By the definition of δ we have

$$\varphi\left(Q_{i+1}(x) + A_{i+1}(x)x^{i+1}\right) = \varphi\left(Q_i(x) + A_i(x)x^i\right) = \varphi\left(Q_a + Q_b\right) \ ,$$

for all $0 \leq i$.
Let $n = 1 + \max\left\{\deg(Q_a), \deg(Q_b)\right\}$ and $k = 1 + \deg(A_i)$. Then $\deg\left(Q_n\right) < n$, $Q_c = Q_{n+k} = Q_n + A_n x^n \in \mathcal{N}[x]$, and $\varphi(Q_c) = \varphi(Q_a + Q_b)$.
Hence, by Theorem **3.18**, and by Lemma **3.6**. iii,

$$\begin{aligned} n + k &= \deg(Q_c) + 1 = L(Q_c) \\ &\leq \mathbb{I}(Q_c) + C_2 \\ &\leq \max\left\{\mathbb{I}(Q_a), \mathbb{I}(Q_b\right\}) + \mathbb{I}(2) + C_2 \\ &= n + \mathbb{I}(2) + C_2 \ , \end{aligned}$$

where C_2 is the constant c_2 in Theorem **3.18**.
This yields that

$$\deg(A_i) < k \leq \mathbb{I}(2) + C_2 \ .$$

Since S is defined to be the smallest set satisfying (1) and (2), every $A \in S$ may appear as the last carry A_n. Thus for all $A \in S \subseteq \mathcal{N}[x]$, $\deg(A) < \mathbb{I}(2) + C_2$, which implies that S is finite.

Theorem 3.23 *The addition automaton, constructed in the proof of Theorem **3.21** is minimal.*

In the definition of $\mathcal{A}$ in the proof of the previous theorem, S is the smallest possible set satisfying the given properties. This means that when calculating the sum of two polynomials, every $A \in S$ may appear as the last carry, determining the most significant digits of the canonical representation of the sum. It implies that none of the states can be eliminated, and the automaton is minimal.
Although we can not be sure there is no linear time algorithm for multiplication, we have the following.

Theorem 3.24 *Let $(P, \mathcal{N})$ be a CNS. Then there are no finite transducer automaton S with two input tapes, such that $\bar{c} = S(\bar{a}, \bar{b})$ for all $\bar{a}, \bar{b}, \bar{c} \in \mathcal{N}^*$ if $\varphi(Q_c) = \varphi(Q_a \cdot Q_b)$.*

On the contrary, assume that $\mathcal{A} = (2, S, \mathcal{N}, \mathcal{N}, s, \delta)$ executes the multiplication, and $N = |\mathcal{N}|$.
Let $\bar{a}, \bar{c} \in \mathcal{N}^*$ satisfying $\varphi(Q_c) = \varphi\left(Q_a^2\right)$. Then by Lemma **3.6**.vi, and by Theorem **3.18**,

$$\begin{aligned} \deg(Q_c) = L(Q_c) - 1 &\geq \mathbb{I}(Q_c) - C_1 - 1 = 2\mathbb{I}(Q_a) - C_1 - 1 \\ &\geq 2\left(L(Q_a) - C_2\right) - C_1 = 2\deg(Q_a) - 2C_2 - C_1 + 1 \ . \end{aligned}$$

Set $n = 1 + \deg(Q_a)$, and $Q_i = c_i x^i + \cdots + c_0$ for all $0 \leq i$. (If $\deg(Q_c) < i$, then $Q_i = Q_c$.)

Since the automaton executes the multiplication, there exist $s_0, s_1, \cdots \in S$ such that $s_0 = s$, and $\delta(s_i, a_i, a_i) = (s_{i+1}, c_i)$, for all $0 \leq i$.

If $n \leq i$, then $a_i = 0$, and $Q_{c,n}(x) = \dfrac{Q_c(x) - Q_n(x)}{x^n}$ is determined purely by s_n. Since $\deg\left(Q_{c,n}\right) \geq n - 2C_2 - c_1 + 1$, there are infinitely many different $Q_{c,n}$, whence there are infinitely many different corresponding states. But this is a contradiction since we assumed that $\mathcal{A}$ is a finite automaton.

3.4 Estimation of the number of states of an addition automaton

As I observed before, the states of the addition automaton are the possible carries. According to the construction, these are polynomials in $\mathcal{N}[x]$. Since the degrees of the state polynomials are bounded by the constant $c = \mathbb{I}(2) + c_2$ of Theorem **3.18**, the following simple result can be established.

Theorem 3.25 *Let $(P, \mathcal{N})$ be a CNS, and $\mathcal{A}$ be the minimal automaton implementing the addition. Then the number of states of $\mathcal{A}$ is bounded by $|\mathcal{N}|^c$.*

I made an experiment on the construction of addition automata for different CNS polynomials. I determined and counted the number of states for different degree binary Kovács-type CNS polynomials. (i.e., the coefficients of P have the property $1 = p_d \leq p_{d-1} \leq \cdots \leq p_0 = 2$.) The results are collected in the following table.

deg	Min. NoS	Max. NoS
3	62	125
4	239	1195
5	1185	13037
6	6248	155765
7	33387	2041141
8	185695	27270180
9	1074944	370005950
10	6419041	$\geq$ 380278058

Table 12: Number of states

The further computations are limited unusually not by the time but by the space complexity of the applied algorithm and data representation.

In Algorithm **2**, we systematically compute all possible states and evaluate, for each input, whether the transition results in an existing state or a new state. In this context, a state essentially corresponds to a carry. Essentially, this explains how I calculated all the possible carries.

Algorithm 2: Construction of Addition Automaton

Data: p, the defining polynomial of the CNS
Result: Φ, the transition table of addition
$stateQ \leftarrow \emptyset$; `/ Empty queue for the states to process /`
$stateQ \leftarrow 0$; `/ Initial state /`
$stateS \leftarrow \emptyset$; `/ The set of used states /`
$\Phi \leftarrow [\,]$;
$a \leftarrow [0, 0, 0]$;
$b \leftarrow [0, 0, 0]$;
while $stateQ \neq \emptyset$ **do**
  $q \leftarrow stateQ$;
  $stateS \leftarrow q$;
  $(a[0], b[0]) \leftarrow (q \gg 1, q_0)$; `/ The transition and output /`
  `/ bit corresponding to` $0+0$ `/`
  $r \leftarrow$ `CR(`$p, q \oplus 1$`)` ;
  $(a[1], b[1]) \leftarrow (r \gg 1, r_0)$; `/ The transition and output /`
  `/ bit corresponding to` $0+1$ `and` $1+0$ `/`
  $r \leftarrow$ `CR(`$p, r \oplus 1$`)` ;
  $(a[2], b[2]) \leftarrow (r \gg 1, r_0)$; `/ The transition and output /`
  `/ bit corresponding to` $1+1$ `/`
  $\Phi[q] \leftarrow (a, b)$;
  if $a[0] \notin stateS$ **then**
    $stateQ \leftarrow a[0]$;
  end
  if $a[1] \notin stateS$ **then**
    $stateQ \leftarrow a[1]$;
  end
  if $a[2] \notin stateS$ **then**
    $stateQ \leftarrow a[2]$;
  end
end

4 Observation of the length of the squares

The existence of a transducer automaton that executes the addition operation in a given CNSs has been proven. Moreover, an algorithm to determine the minimal addition automaton if the number system is a binary CNS has been provided.
In this Chapter, I am going to present the algorithm enumerating all square numbers of length n of algebraic integers in $O(n \cdot 2^n)$ time. Using the construction, I analyzed some properties of squaring in binary CNSs.
The operation of squaring plays an important role in many fields, for instance, in cryptography or at pseudorandom number generators. Examples of PRNGs, on one hand, is the GMSM and, on the other hand, the Blum-Blum-Shub. I gave a more detailed description in Chapter **1** about the BBS.
It can be proven that Blum-Blum-Shub is a cryptographically secure generator. Since in many cases, squaring is a pretty particular operation, such as at the fast exponentiation. Therefore, it is worth dealing with apart.
The GMSM presented in Chapter **2**. For the observation of the properties of such sequences, it is important to know the behavior of squaring in CNSs. In the following, I am going to present the method by which all squares of fixed-length number in the given binary CNSs can be listed and analyzed. Therefore, I have investigated the general properties of arithmetics, with a particular focus on squaring.

4.1 Algorithms and their corresponding measurements

I analyzed the square number's general properties of all the square numbers. To do this we had to produce all the numbers within a given size range that are the squares of an algebraic integer. Among the rational integers, the following formula can be used $(n+1)^2 = n^2 + (2n+1)$.
In this way, we can produce the following square number from the previous one by a simple addition instead of multiplication. Thus sequential production requires less procedure. This method is not directly applicable to the general number system. Instead, we can replace the incrementation by one with another simpler procedure that allows us to list the numbers by changing only one digit in each step.
This method is the well-known Gray code. [17] I used it easily for binary numbers. In this way, the square of the number $n+a$ form can be easily determined from the square of n. This formula is accordingly $(n+a)^2 = n^2+(2an+a^2)$. In the case of binary systems, the calculation of $2an$ is essentially multiplication by 2 and a shift corresponding to a. The aim is to list the square of all numbers shorter than n in a given general number system.

I implemented an algorithm that computes the automaton of increment by one for an algebraic integer with a special form defining polynomial. This special form corresponds to the analysis. Therefore, the coefficients form monotonically increasing sequences where the main coefficient is 1, and the constant term is 2. To the faster list of the square numbers, need to list the numbers of the Gray code. To do this, I also had to implement an algorithm that solved this conversion. Finally, I implemented an automaton, which implements two numbers of addition in special CNSs.

The characteristic of the canonical number system is that it is true for the coefficients of the defining polynomial of the underlying algebraic number that they form a monotonically increasing series and its principal coefficient is 1. However, testing revealed that in some cases the program ran into an infinite loop. I analyzed these cases and I realized, that the model is can not be applicable without modification at the construction of additional automaton. However, it can be proved that the automaton must exist. Based on the constants defined in Chapter **3**, it is possible to define a notional limit on the states of the automaton. Based on these constants, I managed to change the automatic construction so that I got the automaton performing the addition in a finite step already in the cases in question.

4.1.1 Construction

The state was the value of the carry, and that was the basis of the starting point. I used it essentially as a row, we created the newer and newer states into a vector and calculated in a FIFO-manner for each state what sort of new states it generates or whether it generates new states at all. The state itself is the value of the carry, and from this we know the automaton stops. The construction terminates in finitely many steps.
The carry reduction is based on the defining polynomial $p(x) = x^n + \cdots + p_1 x + p_0$ of the CNS, where $1 \leq p_{n-1} \leq \cdots \leq p_0 = 2$ for binary number systems. I set the carry reduction word to $\overline{1p_{n-1} \dots p_1}$, where the constant term is excluded. We should remark that the carry reduction word is a representation of 0, hence if we add a multiple to any number, it will not change its value.
I used W. J. Gilbert's Clearing algorithm because it efficiently manages the carry reduction in operations within CNSs. [15] It calculates carries at each arithmetic step, organizes and merges them for clarity, and finally reduces the digits to their canonical form.

Algorithm **3** describes the carry reduction, where I have used the following operations:

$\oplus$: digitwise addition (carry is not applied);

$\odot$: digitwise multiplication (by the same number for each digit; carry is not applied);

$\gg$: right shift ($w \gg d$ means right shift by d digits).

$\ll$: left shift ($w \ll d$ means left shift by d digits).

Algorithm 3: Carry reduction

Input: p, the defining polynomial of the CNS,
a, a non reduced representation of $\gamma \in \mathbb{Z}$
Output: b, the canonical representation of γ
$\overline{p} \Leftarrow p$; / Transforming $p(x)$ to a /
/ carry reduction word /
$t \leftarrow a$;
$i \leftarrow 0$;
while $t \neq 0$ **do**
  $(q, r) \leftarrow$ **IntDiv**$(t_0, 2)$; / Solve $t_0 = q \cdot 2 + r$, /
  / where $r \in \{0, 1\}$ /
  / t_0 refers to the last digit of t /
  $b \leftarrow (b \ll 1) \oplus r$;
  $t \leftarrow (t \gg 1) \oplus (\overline{p} \odot -q)$;
end
return b

The automaton is minimal because the state means the carry. Therefore, the value of the carry shows how the number ends, so this will show how the sum will start, which will be different in each state because the state is the value of the number itself. So, if does it not get any input numbers after that, the sum is over, and we only calculate with the carry further. Consequently to this, we can be sure that the automaton is minimal because the representation of the two states is the same if it generated the same sequence of numbers. That is, if it generates a different number of sequences, the state cannot be shortened and eliminated.
Since the states of the automaton are the carry values themselves, they definitely depend on the carry. When there are no more inputs, the counting stops and since it is all different, practically all states are different. Therefore, the automaton can not be reduced.

4.1.2 Enumeration of squares

In the following, I am going to present the method to enumerate all squares without using multiplication operations. Since all applied operations have a corresponding automaton, it means that the enumeration of all squares of length n can be done in $O(n \cdot 2^n)$ time. I wanted to use it similarly as in the rational integers, but I realized that this could not be solved in that form. Enumerating square numbers is not an easy method, but we could have chosen the same order, which can be followed for rational integers in bit representation. However, it would have been challenging in that way to calculate the consecutive squares because of the back-and-forth bouncing of the carry. Instead of this, I applied the subsequent according to the Gray code. This makes it easier for only 1-bit to change two consecutive numbers that are squared. Accordingly, consecutive squares are also easy to calculate. Algorithm **4** is used to generate the n^{th} Gray code.

Algorithm 4: Gray number represented in n bits

Input: n, the binary length
a, an integer represented in n bits
Output: b, the a^{th} Gray number represented in n bits
$b \leftarrow a$; / $a = \overline{a_{n-1} \dots a_0}$ `and` $b = \overline{b_{n-1} \dots b_0}$ /
for $i \leftarrow 0 \dots n-2$ **do**
  $b_i \leftarrow b_i + a_{i+1} \mod 2$;
end
return b

The input of Algorithm **5** is the array of coefficients of the polynomial p, and k is the degree of p, and the output is the list *Squares* of the squares. I managed to enumerate the number of squares without multiplication, relying solely on addition. Enumerating the square numbers are much faster than squaring the numbers one by one.

The algorithm computes the squares of numbers using enumeration based on the previously provided explanations. It generates the next square number using a specified formula and iterates through numbers using Gray code, where only one bit changes at a time. The `GrayDiff` function calculates the difference between adjacent Gray codes by identifying where the transition occurs. It returns both the position and direction of the bit change—whether a bit has flipped from 1 to 0 or from 0 to 1. The direction indicates the type of bit change, while the position shows where the change happened.

Thus, this algorithm efficiently enumerates known square numbers for integer values by modifying the number to be squared by altering a single bit at a time, rather than incrementing by 1.

Algorithm 5: Enumerating squares

Input: $p(x)$, the defining polynomial of the CNS,
n, the common length of the base numbers
Output: LS, the list of squares

```
/ the difference of the consecutive Gray codes    /
```

Function `GrayDiff`(n, k)

- $d \leftarrow$ `Gray`$(n, k+1)$ **xor** `Gray`(n, k) ; `/ bitwise xor /`
- $pos \leftarrow \log_2 d$; `/ the position of the difference /`
- **if** d **xor** `Gray`$(n, k) \neq 0$ **then**
  - $dir \leftarrow 1$;
- **else**
  - $dir \leftarrow -1$;
- **end**
- **return** (pos, dir) ;

EndFunction

$LS \leftarrow \emptyset$;
$a \leftarrow 1(0)^{n-1}$; `/ a` n`-bit number, the first base /`
$s \leftarrow 1(0)^{2n-2}$; `/ a` $2n-1$`-bit number, the square of` a `/`
$LS \leftarrow s$;
$k \leftarrow 0$; `/ the Gray index /`
while $k < 2^n$ **do**

- $(pos, dir) \leftarrow$ `GrayDiff`$(n-1, k)$;
- $am \leftarrow a \odot (2 \cdot dir)$;
- $am \leftarrow (am \ll pos) \oplus 1$;
- $am \leftarrow s \oplus am$;
- $s \leftarrow$ `CR`$(p(x), am)$; `/ Carry reduction /`
- $LS \leftarrow s$;
- $k \leftarrow k + 1$;
- $a \leftarrow 1($ `Gray`$(n-1, k))$;

end
return LS

Polynomial	C_3	C_4
1 1 2	-8	3
1 2 2	-12	5
1 1 1 2	-14	12
1 2 2 2	-25	24
1 1 1 1 2	-20	21
1 1 1 2 2	-25	25
1 1 2 2 2	-33	39
1 2 2 2 2	-38	53
1 1 1 1 1 2	-23	32
1 2 2 2 2 2	-51	103
1 1 1 1 1 1 2	-31	46
1 1 1 1 1 2 2	-33	60
1 1 1 1 2 2 2	-42	76
1 1 1 2 2 2 2	-49	98
1 1 2 2 2 2 2	-42	124
1 2 2 2 2 2 2	-47	161
1 1 1 1 1 1 1 2	-30	68
1 1 1 1 1 2 2 2	-43	100
1 1 1 2 2 2 2 2	-45	158
1 2 2 2 2 2 2 2	-46	215
1 1 1 1 1 1 1 1 2	-32	89
1 1 1 1 1 1 1 2 2	-38	105
1 1 1 1 1 2 2 2 2	-43	159
1 1 1 1 2 2 2 2 2	-39	185
1 1 2 2 2 2 2 2 2	-49	242

Table 13: Lengths of squares for specific polynomials

I managed to list the number of squares without multiplication, relying solely on addition. I have verified how the length of squares differs from the expected $2n$, where n is the length of the base number. Table **13** represents an analytical calculation of the maximum possible change in the length of the square numbers, and it exactly calculates this for a few polynomials with limited size. Therefore, the analysis of the lengths of squares for specific polynomials was a general examination that includes the listing of practical approximations and results.

Summary

In the dissertation, I have summarized my research about Pseudo-random Number Generators and arithmetics in Canonical Number Systems.

In Chapter **1**, I have presented some significant PRNGs and a few theoretical ways to observe them. I have also analyzed the basic properties of the PRNGs, such as period length, computational speed, and memory usage. I also observed their implementations, and then I constructed my own in order to generate the proper bit sequences for comparative analysis. In the analysis, I examined properties such as the distribution of the samples produced by the generators, the low correlation between the elements, and the period length. I also examined properties considered important from the point of view of use, such as the generators' speed and memory requirements. I have found that the period length has a strong relationship with the size of the seed. Typically, it is a proportional relation.
The first relevant results were published in [43] and presented at the 11^{th} International Conference on Applied Informatics and the 23^{rd} annual Spring Wind Conference in 2020, and the first results of my research work is summarized in Table **14**. At first, there were no exact comparable numerical values for computational speed, because of the wide range of platforms they were implemented on. I have found implementations for most of the above listed generators, which is marked in the table with a reference.

Generator name	Period length	Comp. speed	Impl.
Neumann's MSM	8^n	simple	[1]
N. Metropolis MSM	142	simple	
Coveyou's SMSM	10^9	effective	
Lehmer's MLCG	5882352	depends on the mod.	
LCG	$< 2^{64}$	fast	[2]
GFSR	max. $2^k - 1$	fast	
MT	$2^{19937} - 1$	simple	[3]
BBS	m	slow	[4]
W-H algorithm	$7 \cdot 10^{12}$	slow	[5]
ACORN	m^k	simple and fast	[6]
WELL	2^{44497}	fast	[7]
Algorithm M	LCM of PL of base seq.	2x the base alg	
Algorithm B	PL of the base seq.	= base alg	
RC4	2^{1024}	fast	[8]
Xst	$2^k - 1$	very fast	[9]
PCG	2^{128}	simple and fast	[10]

Table 14: First observation of general properties of the generators

I found in the literature the behavior of some of the generators listed in the table. In my research work, I focused on the NIST test suite. At first, based on the aforementioned test package, I was only able to ascertain how many of the 15 tests the given generator passed.

	W-H	RC4	LCG	GFSR	MT	BBS	Xst	WELL
NIST	11/15	13/15	11/15	14/15	15/15	13/15	15/15	15/15

Table 15: First observation of generators with NIST Test Suite

In my research work, I have tried to improve the results I found. I expanded the observation to a wider selection of generators, and then implemented the algorithms of the discussed pseudorandom number

generators (http://www.prng.hu). Since there is a wide range of platforms they are implemented on, it is hard to compare the speed of the generators. Some perform better on constrained devices, while others are more efficient on CISC processors. In accordance with this duality, instead of the speed of the algorithms, I highlighted the number of complex (slow) and simple (fast) operations, which can be found in the Table **1**.
I have tested the generated sequences by the NIST Test Suite, the results of which are in Tables **9**, and **10**. As an outcome of my observations, most of the discussed PRNGs passed all tests of the test battery. In particular cases, however, I found interesting behavior. The most surprising is that LCG and Lehmer's MLCG show a good score, while both are proven to be cryptographically insecure. One can find an opposite behavior for the BBS, which performed weak on the tests but has proven to be a secure one. For the generators based on John von Neumann's MSM, I have to increase the number of digits of their seeds considerably to achieve sequences passing the tests. (For instance, the seed of the original John von Neumann's MSM generator was extended to 72 decimal digits.) Compared to it, the improved GMSM generator, which is defined over algebraic number fields, that I constructed, requires only a 92-bit seed with similar properties, even for a relatively simple algebraic extension.

I have presented the examination of generators based on the modified John von Neumann's MSM to CNSs in Chapter **2**. I have focused primarily on the behavior in binary canonical number systems. These are generalized number systems that only have digits $0, 1$. I defined the generalized middle-square method. One can find the precise definition in Sub-chapter **2.3**. To analyze the properties of random numbers produced by the generalized Neumann's Middle-Square method, it is necessary to analyze the properties of operations in canonical number systems.

My observation is based on the relation between the number of digits needed to represent a number and the size of the corresponding algebraic number. I have refined the calculation of the computable constants found in [27] and brought the theoretical constants closer to the values obtained during the experiments. The essence of the method used is that it does not give an upper estimate for the absolute value of the complex numbers included in the calculation individually but rather grouped them and examined which elements can be omitted or should be taken into account in the upper estimate of the absolute value of their sum. In general, the constants of the Kovács-Pethő theorem do not make use of the special properties of individual number systems. Better estimates can be used in the analysis of random number sequences. With the help of the estimates achieved, I was able to set a bound for the length of the squares of fixed-length numbers. One of the important features of the sequences generated by the generalized middle-square square method is the period length. I constructed an algorithm, and with its help, I examined the generators in some - also binary - canonical number systems from this point of view. Another important property is the statistical behavior of the generated series. I have executed some tests for this, that I presented. I also examined how the lengths of the numbers are related to their squares in canonical number systems. I accomplished an experiment to determine the properties of squaring.

Based on the theorem of B. Kovács and A. Pethő, the relation between the lengths of a number and its square is approximately a factor of 2. For this, I constructed an algorithm that lists all squares of numbers with lengths up to n without using any multiplication. I detailed all my results in Sub-chapter **2.3**. I compiled a manuscript from the results [42] and gave a presentation from it at the 1^{st} and the 2^{nd} Conference on Information Technology and Data Science.

I have presented some properties of the operation of canonical number systems over algebraic integers. I proved that there exists a finite transducer automaton that can be used to add numbers represented in number systems. I described an algorithm that can be used to determine the states of the automaton performing the addition in the given number system. The results have been under publishing in [19]. Running the program in the case of some binary canonical number systems, I found that the number of states increases very quickly with the degree of the defining polynomial of the number system. This resulted in the fact that due to memory limitations, I could only implement the calculation for polynomials of degrees up to 10. Furthermore, I have prepared an algorithm that calculates the automaton of the operation of increment by 1 for CNSs with a defining polynomial of a special form. The special form was adequate for my trials, i.e., the coefficients form a monotonically increasing series, the main coefficient is 1, and the constant term is 2. Finally, the program that solves the quick enumeration of square numbers was also presented, which uses Gray code enumeration. During testing, however, it was found that, in some cases, the program ran into an infinite loop. After examining these cases, I realized that the model can not be applied without changes to the construction of the automaton that implements addition. However, based on the article by B. Kovács and A. Pethő, it can be proven that the automaton must exist. Based on the constants defined in the article, it is possible to determine a theoretical bound for the number of states of the automaton. Based on these constants, I managed to change the automaton construction so that even in questionable cases, I can get the automaton that implements the addition in a finite step. The result of this part of my research was that, I was able to enumerate all square numbers of length n in $O(n \cdot 2^n)$ time.

Összefoglalás

Dolgozatomban összefoglaltam az álvéletlenszám-generátorokkal való kutatásomat, illetve ismertettem a műveletvégzést kanonikus számrendszerekben.

Az **1.** fejezetben bemutattam a leggyakrabban használt egyenletes eloszlású álvéletlenszám-generátorokat, és néhány elméleti módot a megfigyelésükre. Ezek implementációit is felkutattam, majd saját implementációt készítettem, ahhoz, hogy összehasonlító elemzésnek tudjam alávetni az általuk generált sorozatokat. Az elemzésben olyan tulajdonságokat vizsgáltam, mint például a generátorok által előállított elemek eloszlása, az elemek közötti alacsony korreláció, illetve a periódushossz. Vizsgáltam még a felhasználás szempontjából fontosnak tartott tulajdonságokat, úgy, mint a generátorok sebessége, valamint az erőforrásigénye. Megállapítottam, hogy a periódushossz szoros kapcsolatban áll a mag méretével. Jellemzően minél nagyobb a mag, annál nagyobb a periódushossz.
Az első releváns eredményeket [43]-ben publikáltam. 2020-ban az ICAI konferencián, illetve a DOSZ 23. Tavaszi Szél Konferenciáján is előadást tartottam belőle, kutatómunkám első eredményeit pedig a **16.** táblázatban foglaltam össze. Tanulmányoztam az álvéletlenszám-generátorok fő tulajdonságait, és, hogy melyek az egyes alkalmazásokhoz köthető előnyös tulajdonságaik. Eleinte nem voltak pontos összehasonlítható számértékek a számítási sebességre vonatkozóan, mivel a különböző számítógépes környezetben kerültek implementálásra. A táblázatban felsorolt generátorok többségéhez találtam implementációt, ezt hivatkozással jelöltem.

Generator name	Period length	Comp. speed	Impl.
Neumann's MSM	8^n	simple	[1]
N. Metropolis MSM	142	simple	
Coveyou's SMSM	10^9	effective	
Lehmer's MLCG	5882352	depends on the mod.	
LCG	$< 2^{64}$	fast	[2]
GFSR	max. $2^k - 1$	fast	
MT	$2^{19937} - 1$	simple	[3]
BBS	m	slow	[4]
W-H algorithm	$7 \cdot 10^{12}$	slow	[5]
ACORN	m^k	simple and fast	[6]
WELL	2^{44497}	fast	[7]
Algorithm M	LCM of PL of base seq.	2x the base alg	
Algorithm B	PL of the base seq.	= base alg	
RC4	2^{1024}	fast	[8]
Xst.	$2^k - 1$	very fast	[9]
PCG	2^{128}	simple and fast	[10]

Table 16: A generátorok tulajdonságainak első vizsgálata

A táblázatban felsorolt generátorok némelyikének a viselkedését kutattam. A dolgozatomban a NIST teszt csomagra fókszáltam. Eleinte az említett teszt csomagot alapul véve, annyit tudtam megmondani, hogy a 15 tesztből, mennyi teszten ment át az adott generátor.

	W-H	RC4	LCG	GFSR	MT	BBS	Xst	WELL
NIST	11/15	13/15	11/15	14/15	15/15	13/15	15/15	15/15

Table 17: Generátorok első vizsgála NIST tesztcsomaggal

A kutatásom eredményeit tovább fejlesztettem. Kibővítettem a generátorokat, majd elkészítettem a tárgyalt álvéletlenszám-generátorok implementációit (http://www.prng.hu) is. Az egyes generátorok jobban teljesítenek korlátozott utasításkészletű esz-közökön, míg mások hatékonyabbak CISC processzorokon. Ezeknek köszönhetően az algoritmusok sebessége helyett a bonyolult (lassú) és egyszerű (gyors) műveletek számát emeltem ki, ami a **1.** táblázatban található.
A generált sorozatokon végrehajtottam a NIST teszt csomagot, melynek eredményeit a **9.** és **10.** táblázatokban összegeztem. Megfigyeléseim eredményeként a tárgyalt álvéletlenszám-generátorok többsége átment a teszt csomag összes tesztjén. Néhány esetben azonban érdekes viselkedést tapasztaltam. A legmeglepőbb az, hogy az LCG és a Lehmer-féle MLCG generátorok jó eredményeket mutattak, annak ellenére, hogy mindkettő bizonyítottan kriptográfiailag nem biztonságos.
Ellentétes viselkedést találtam a BBS-nél, amely gyengén teljesített a teszteken, viszont - kriptográfiai szempontból is - biztonságos generátornak mondható. A Neumann János-féle Négyzetközép-módszerre épülő generátoroknál jelentősen növelnem kellett a magok számjegyeit, hogy a teszteken átmenő szekvenciákat elérjem. (Például az eredeti NMSM-generátor magját 72-jegyűre kellett bővítenem.) Ehhez képest a kanonikus számrendszerekre általánosított Neumann-féle négyzetközép generátor csak egy 92 bites, hasonló tulajdonságokkal rendelkező magot igényelt, még egy viszonylag egyszerű algebrai kiterjesztés esetén is.

Ezt követően az általánosított számrendszerekkel folytattam a kutatást, Neumann János négyzetközép módszerét általánosítottam kanonikus számrendszerekre, amit a **2.** fejezetben mutattam be. A rendszerben két számjegy van $(0, 1)$ a sorozat pedig hasonló módon van definiálva: egy megfelelően megválasztott m-bites kezdőérték a kiindulás. Négyzetre kell emelni a magot, ki kell vágni a középső m-bitet, és ez a generátor következő magja.
Az általánosított Neumann-féle négyzetközép módszeren alapuló véletlenszámok tulajdonságainak elemzéséhez szükséges a kanonikus számrendszerekben történő műveletvégzés tulajdonságait elemezni. Az egyik ilyen, hogy egy adott értékű szám ábrázolásához hány számjegyre van szükség.
Sikerült a [27]-ban található konstansok számításán finomítani és az elméleti konstansokat közelíteni a kísérletek során tapasztalt korlátokhoz. Az alkalmazott módszer lényege, hogy a számításban szereplő komplex számok abszolút értékére nem egyenként adunk felső becslést, hanem csoportosítottuk őket, és azt vizsgáltuk, hogy mely elemek hagyhatók el, illetve vehetők figyelembe az összegük abszolút értékének felső becslésénél. A Kovács-Pethő-tétel konstansai általános esetben nem használják ki az egyes számrendszerek speciális tulajdonságait. A jobb becslések a véletlenszám-sorozatok elemzésénél használhatók. Az elért becsélések segítségével korlátot tudtam adni a rögzített hosszúságú számok négyzetének a hosszára.
Az általánosított négyzetközép módszer által generált sorozatok egyik fontos tulajdonsága az így előállított sorozatok periódushossza. Készítettem egy programot, amely segítségével néhány - szintén bináris - kanonikus számrendszerben ebből a szempontból vizsgáltam meg a generátorokat. További fontos tulajdonság a generált sorozatok statisztikai viselkedése. Ehhez elkészítettem néhány tesztet, amit bemutattam a disszertációban.

Megvizsgáltam továbbá, hogy kanonikus számrendszerekben négyzetre emelés során hogyan alakul az eredmény hossza. Kísérletet végeztem a négyzetre emelés tulajdonságainak megállapítására.
Elsődlegesen arra voltam kíváncsi, hogy egy n hosszú számnak milyen hosszú a négyzete. Kovács B. és Pethő A. tétele alapján ez körülbelül $2n$. Ehhez készítettem egy programot, amely felsorolja a legfeljebb n hosszúságú számok négyzeteit, úgy, hogy nem használ szorzás műveletet.
A kapott eredményekből összeállítottam egy kéziratot [42] és 2021-ben, illetve 2022-ben a CITDS konferenciákon előadást tartottam belőle. Először magát a generátort mutattam be, majd pedig a teszteredményeimet.

A disszertációban bemutattam a kanonikus számrendszerek műveletének néhány tulajdonságát algebrai egész számok felett.
Sikerült bizonyítani, hogy létezik olyan véges automata, amely segítségével a számrendszerekben ábrázolt számok összeadása megvalósítható. Bemutattam az elkészült programot, amely segítségével az adott számrendszerben az összeadást végző automata állapotai meghatározhatók. Az ebből készült kéziratot közlésre benyújtottam. [19]

Néhány bináris kanonikus számrendszer esetén lefuttatva a programot, azt tapasztaltam, hogy az állapotok száma nagyon gyorsan nő a számrendszer definiáló polinomjának fokszámával. Ez azt eredményezte, hogy memóriakorlátok miatt csak legfeljebb 10 fokú polinomok esetén tudtam a számítást végrehajtani.
Továbbá, készítettem egy programot, amely egy speciális alakú definiáló polinommal rendelkező algebrai egészhez kiszámolja az 1-gyel növelés műveletének automatáját. A speciális alak vizsgálataimnak megfelelő volt, azaz az együtthatók monoton növekvő sorozatot alkotnak, a főegyüttható 1, a konstans tag pedig 2.

Végül bemutatásra került az a program is, ami megoldja a négyzetszámok gyors felsorolását, amit Gray-kódos felsorolással oldottam meg.
A tesztelés során azonban kiderült, hogy bizonyos esetekben a program végtelen ciklusba futott. Megvizsgálva ezeket az eseteket rájöttem, hogy a modell nem alkalmazható változtatás nélkül az összeadást megvalósító automata konstrukciójánál. [27] alapján azonban bizonyítható, hogy az automatának léteznie kell. A cikkben meghatározott konstansok alapján lehetséges egy elméleti korlátot meghatározni az automata állapotainak számára. Ezen a konstansok alapján az automata konstrukciót sikerült megváltoztatni úgy, hogy már a kérdéses esetekben is véges lépésben megkapjam az összeadást megvalósító automatát. Kutatásom ezen részének eredménye, hogy fel tudom sorolni az összes n hosszúságú négyzetszámokat $O(n \cdot 2^n)$ időben.

Köszönetnyilvánítás

Ezúton szeretném kifejezni hálás köszönetemet mindazoknak, akik segítették az egyetemi tanulmányaimat és ezen disszertáció elkészítését.

Mindenekelőtt nagyon szépen köszönöm Dr. Herendi Tamásnak a témavezetői iránymutatását, bizalmát és mindazt, amit az évek során tőle tanultam. Szeretnék köszönetet mondani Dr. Pethő Attilának, aki a doktoranduszi éveimet végig segítette.

Végül, de nem utolsó sorban, szeretném megköszönni a családomnak és barátaimnak a kitartó türelmüket, szeretetüket, támogatásukat és a sok bátorítást, amit a tanulmányaim során tőlük kaptam.

References

[1] FIPS 81. *DES Modes of Operation*. https://csrc.nist.gov/csrc/media/publications/fips/81/archive/1980-12-02/documents/fips81.pdf/. 1980.

[2] A. Pethő and P. Varga. *Canonical number systems over imaginary quadratic Euclidean domains*. In: Colloquium Mathematicum 146(2) (2016), pp. 1–22.

[3] N. Z. Akopov and N. H. Martirosyan. *The Optimal Approach for Kolmogorov-Smirnov Test Calculation in High Dimensional Space*. In: Mathematical Problems of Computer Science (2015), pp. 138–144.

[4] J. p. Allouche, K. Scheicher, and R. F. Tichy. *Regular Maps in Generalized Number Systems*. In: Math. Slovaca 50 (2000), pp. 41–58.

[5] L. E. Bassham et al. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. In: NIST Special Publication (2001), pp. 800–822.

[6] L. Blum, M. Blum, and M. Shub. *Random Number Generators*. In: A Simple Unpredictable Pseudo-Random Number Generator (1986), pp. 364–383.

[7] M. Émile Borel. *Les probabilités dénombrables et leurs applications arithmétiques*. In: Rendiconti del Circolo Matematico di Palermo 27 (1884-1940), pp. 247–271.

[8] N. G. de Bruijn. *A combinatorial problem*. In: Koninklijke Nederlandse Akademie v. Wetenschappen 49 (1946), pp. 758–764.

[9] P. Burcsi and A. Kovács. *Exhaustive search methods for CNS polynomials*. In: Monatshefte für Mathematik 155 (3) (2008), pp. 421–430.

[10] G. J. Chaitin. *On the Simplicity and Speed of Programs for Computing Infinite Sets of Natural Numbers*. In: Journal of the ACM 16 (3) (1969), pp. 407–422.

[11] J. G. van der Corput. *Verteilungsfunktionen (Erste Mitteilung)*. In: Proceedings of the Koninklijke Akademie van Wetenschappen te Amsterdam 38 (1935), pp. 813–821.

[12] I. B. Damgård. *On The Randomness of Legendre and Jacobi Sequences*. In: Goldwasser, S. (eds) Advances in Cryptology — CRYPTO' 88. CRYPTO 1988. Lecture Notes in Computer Science 403 (1990), pp. 163–172.

[13] Dieharder. *http://webhome.phy.duke.edu/ rgb/General/dieharder.php*.

[14] J. Eichenauer and J. Lehn. *A non-linear congruential pseudo random number generator*. In: Statistische Hefte 27 (1986), 315—326.

[15] W. J. Gilbert. *Radix Representations of Quadratic Fields*. In: Journal of Mathematical Analysis and Applications 83 (1981), pp. 264–274.

[16] Peter J. Grabner, Peter Kirschenhofer, and Helmut Prodinger. *The sum-of-digits function for complex bases*. In: Journal of the London Mathematical Society 57 (1) (1998), pp. 20–40.

[17] F. Gray. *Pulse Code Communication*. US patent #2,632,058, 1953.

[18] T. Herendi. *Construction of uniformly distributed linear recurring sequences modulo powers of 2*. In: Uniform Distribution Theory 13 1 (2018), pp. 109–129.

[19] T. Herendi and V. Padányi. *Automata and Arithmetic in Canonical Number Systems*. In: in publishing (2024).

[20] I. Kátai and B. Kovács. *Canonical number systems in algebraic number fields*. In: Acta Math. Hung. 37.1-3 (1981), pp. 159–164.

[21] I. Kátai and J. Szabó. *Canonical number-systems for complex integers*. In: Acta Sci. Math. 37 (1975), pp. 255–260. ISSN: 0001-6969.

[22] D. E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Boston: Addison-Wesley, 1981.

[23] A.N. Kolmogorov. *On Tables of Random Numbers*. In: Sankhya Ser A. 25 (1963), pp. 369–375.

[24] A.N. Kolmogorov. *Three approaches to the quantitative definition of information*. In: Problems of Information Transmission 1 (1) (1965), pp. 1–7.

[25] B. Kovács. *Integral domains with canonical number systems*. In: Publ. Math. 36.1-4 (1989), pp. 153–156. ISSN: 0033-3883.

[26] B. Kovács and A. Pethő. *Number systems in integral domains, especially in orders of algebraic number fields*. In: Acta Sci. Math. 55.3-4 (1991), pp. 287–299. ISSN: 0001-6969.

[27] B. Kovács and A. Pethő. *On a representation of algebraic integers*. In: Stud. Sci. Math. Hung. 27.1-2 (1992), pp. 169–172. ISSN: 0081-6906; 1588-2896/e.

[28] A. Kovács. *Generalized binary number systems*. In: Annales Univ. Sci. Budapest, Sect. Comp 20 (2001), pp. 195–206.

[29] D. H. Lehmer. *Mathematical methods in large-scale computing units*. In: Proceedings of a Second Symposium on Large-Scale Digital Calculating 26 (1951), pp. 141–146.

[30] T.G. Lewis and W.H. Payne. *Generalized Feedback Shift Register Pseudorandom Number Algorithm*. In: Journal of the ACM 20 (3) (1973), pp. 456–468.

[31] P. L'ecuyer and R. Simard. *TestU01: A C Library for Empirical Testing of Random Number Generators*. In: ACM Transactions on Mathematical Software 33 (4) (2007), pp. 1–40.

[32] Y. Kurita M. Matsumoto. *Twisted GFSR generators*. In: ACM Transactions on Modeling and Computer Simulation 2 (3) (1992), pp. 179–194.

[33] Y. Kurita M. Matsumoto. *Twisted GFSR generators II.* In: ACM Transactions on Modeling and Computer Simulation 4 (3) (1994), pp. 245–466.

[34] G. Marsagila. *Xorshift RNGs*. In: Journal of Statistical Software 8 (14) (2003), pp. 1–6.

[35] M. Matsumoto and T. Nishimura. *Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator*. In: ACM Transactions on Modeling and Computer Simulation 8 (1998), pp. 3–30.

[36] C. Mauduit and A. Sárközy. *On finite pseudorandom binary sequences I: Measure of pseudorandomness, the Legendre symbol*. In: ACTA ARITHMETICA 82 (4) (1997), pp. 365–377.

[37] National Institute of Standards and Technology. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications: NIST SP 800-22*. 2012.
URL: `https://www.nist.gov/publications/statistical-test-suite-random-and-pseudorandom-number-generators-cryptographic`.

[38] John von Neumann. *Various Techniques Used in Connection with Random Digits. In: A.S. Householder, G.E. Forsythe, and H.H. Germond, eds., Monte Carlo Method, National Bureau of Standards*. In: Appl. Math. 12 (1950), pp. 36–38.

[39] H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*. Philadelphia, Pennsylvania: Society for Industrial and Applied Mathematics, 1992.

[40] M. E. O'Neill. *PCG: A Family of Simple Fast Space-E cient Statistically Good Algorithms for Random Number Generation*. In: Harvey Mudd College, HMC-CS-2014-0905 (2014).

[41] V. Padányi and T. Herendi. *A Study on Comparison of Pseudo-random Number Generators*. In: International Journal of Mathematics and Computer in Engineering 1 (2023), pp. 1–20.

[42] V. Padányi and T. Herendi. *Generalized Middle-Square Method*. In: Annales Mathematicae et Informaticae 56 (2022), pp. 95–108.

[43] V. Padányi and T. Herendi. *Metaanalysis of pseudorandom number generators*. In: 23rd annual Spring Wind conference 23 (2020), pp. 474–486.

[44] F. Panneton, P. L'ecuyer, and M. Matsumoto. *Improved long-period generators based on linear recurrences modulo 2*. In: ACM Transactions on Mathematical Software 32 (1) (2006), pp. 1–16.

[45] A. Pethő. *On a polynomial transformation and its application to the construction of a public key cryptosystem*. In: Computational Number Theory: Proceedings of the Colloquium on Computational Number Theory held at Kossuth Lajos University, Debrecen (Hungary), September 4-9, 1989, edited by Attila Pethő, Michael E. Pohst, Hugh C. Williams and Horst G. Zimmer, Berlin, New York: De Gruyter (1991), pp. 31–44.

[46] A. Pethő. *On a polynomial transformation and its application to the construction of a public key cryptosystem*. In: Computational Number Theory: Proceedings of the Colloquium on Computational Number Theory held at Kossuth Lajos University, Debrecen (Hungary) September 4-9, 1989, edited by Attila Pethö, Michael E. Pohst, Hugh C. Williams and Horst G. Zimmer, Berlin, New York: De Gruyter (1991), pp. 31–44.

[47] A. Rotenberg. *A New Pseudo-Random Number Generator*. In: Journal of the ACM 7 (1) (1960), pp. 75–77.

[48] S. Akiyama and A. Pethő. *On a canonical number systems*. In: Theor. Computer. Sci. 270 (2002), pp. 921–933.

[49] H. Brunotte A. Pethő J. M. Thuswaldner S. Akiyama T. Borbély. *On a generalization of the radix representation – a survey*. In: American Math. Soc. 41 (2004), pp. 19–27. ISSN: 1069-5265.

[50] E. Samuel. *Automata, languages, and machines*. New York : Academic Press, 1974.

[51] K. Scheicher and J. M. Thuswaldner. *Canonical Number Systems, Counting Automata And Fractals*. In: Math. Proc. Cambridge Philos. Soc. 133 (2007), pp. 163–182.

[52] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. New York: John Wiley Sons, Inc., 1996.

[53] R. Solomonoff. *A Formal Theory of Inductive Inference Part I and II*. In: Information and Control 7 (1) and (2) (1964), 1–22 and 224–254.

[54] W. E. Thomson. *A Modified Congruence Method of Generating Pseudo-random Numbers*. In: The Computer Journal 1 (2) (1958), p. 83.

[55] J. M. Thuswaldner. *Elementary properties of the sum of digits function in quadratic number fields*. In: In G. E. Bergum et. al., editor, Applications of Fibonacci Numbers 7 (1998), pp. 405–414.

[56] A. Salomaa W. Kuich. *Semirings, Automata, Languages*. Springer-Verlag, 1986.

[57] H. Weyl. *Ueber die Gleichverteilung von Zahlen mod. Eins*. In: Math. Ann. 77 (3) (1916), pp. 313–352.

[58] B.A. Wichmann and I.D. Hill. *Algorithm AS 183: An Efficient and Portable Pseudo-Random Number Generator*. In: Applied Statistics 30 (1982), pp. 188–190.

[59] B.A. Wichmann and I.D. Hill. *Generating good pseudo-random numbers*. In: Computational Statistics & Data Analysis 51 (2006), pp. 1604–1622.

[60] B. Widynski. *Middle Square Weyl Sequence RNG*. In: eprint arXiv:1704.00358 [cs.CR] 8 (14) (2019).

[61] R. S. Wikramaratna. *ACORN - A new method for generating sequences of uniformly distributed Pseudo-Random Numbers*. In: Journal of Computational Physics (1989), pp. 16–31.

Links to the available implementations

[1] Neumann's MSM: https://mswsrng.wixsite.com/rand, (Downloaded: 26.01.2020.)

[2] LCG: https://gist.github.com/ekepes/8aed1310c3e7af31c99d, (Downloaded: 26.01.2020.)

[3] MT: http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html, (Downloaded: 26.01.2020.)

[4] BBS: https://github.com/OverStruck/blum-blum-shub-prbg, (Downloaded: 26.01.2020.)

[5] W-H algorithm: http://www.siafoo.net/algorithm/7, (Downloaded: 26.01.2020.)

[6] ACORN: http://acorn.wikramaratna.org/, (Downloaded: 26.01.2020.)

[7] WELL: http://www.iro.umontreal.ca/~panneton/WELLRNG.html, (Downloaded: 26.01.2020.)

[8] RC4: https://gist.github.com/rverton/a44fc8ca67ab9ec32089, (Downloaded: 26.01.2020.)

[9] Xst.: https://github.com/WebDrake/xorshift, (Downloaded: 26.01.2020.)

[10] PCG: http://www.pcg-random.org/download.html, (Downloaded: 26.01.2020.)

List of papers of the author

1. V. Padányi and T. Herendi (2020) *Metaanalysis of Pseudorandom Number Generators*, 23rd annual Spring Wind conference, vol. 23, pp. 474–486.

2. V. Padányi and T. Herendi (2022) *Generalized Middle-Square Method*, Annales Mathematicae et Informaticae, vol. 56, pp. 95-108.

3. V. Padányi and T. Herendi (2023) *A Study on Comparison of Pseudorandom Number Generators*, International Journal of Mathematics and Computer in Engineering, vol. 1, pp. 1-20.

4. T. Herendi and V. Padányi (2025) *Automata and Arithmetic in Canonical Number Systems*, Algorithms, vol. 18(3), pp. 1-22. EISSN: 1999-4893. https:// doi.org/10.3390/a18030122

List of talks of the author

1. The 11^{th} International Conference on Applied Informatics. Eger, Hungary, January 29–31, 2020.

2. The 1^{st} Conference on Information Technology and Data Science. Debrecen, Hungary, November 6–8, 2020.

3. The 23^{rd} Spring Wind Conference. Budapest, Hungary, October 16, 2020.

4. The 2^{nd} Conference on Information Technology and Data Science. Debrecen, Hungary, May 16–18, 2022.