Doktori (PhD) értekezés tézisei

# Efficiency Analysis of Some Cryptographic Primitives

Major Sándor Roland

Témavezető: Dr. Herendi Tamás


Debreceni Egyetem
Informatikai Tudományok Doktori Iskola
Debrecen, 2024

# Contents

# Chapter 1

# Introduction

The thesis focuses on two cryptographic primitives: random number generators and linear codes.

Part I of the thesis is based on an algorithm developed by Tamás Herendi in [1]. The algorithm can be used to construct linear recurrence sequences (LRS) with uniform distribution modulo powers of 2, with arbitrarily large period lengths, and arbitrarily large elements. The theoretical design of the presented results was developed during joint research between Tamás Herendi and the author. The application development, along with the implementation and analysis of the tests carried out are the results of the author. The relevant publications by the author for this part of the thesis are [2], [3] and [4].

Part II of the thesis is concerned with the problem of constructing linear codes with given, close to arbitrary parameters.

The author has developed a software package named Torch, that provides an extensible, reconfigurable solution that supports a wide variety of search algorithms, conditions and other options that allows researchers to experiment and incorporate new research results into linear code searches. The software can be used for classification problems, finding existing linear codes, or searching for currently unknown codes. The theoretical design of the presented results was developed during joint research between Carolin Hannusch and the author. The software development, implementation and analysis are the results of the author. The relevant publications by the author for this part of the thesis are [5] and [6].

## 1.1 Problem statement

### 1.1.1 Random number generation

Pseudorandom numbers have many practical applications. In cryptography specifically, they are used for purposes such as key generation, stream ciphers and asymmetric cryptosystems [7]. More broadly, they are also used for simulations, Monte-Carlo methods [8], and many others. Depending on the application, pseudorandom number sequences need to fulfill different requirements, such as the distribution of its elements, the period length of the sequence, computation speed of the next element, and unpredictability. A detailed examination of generating random sequences and using statistical tests to check their properties can be found in [9].

Tamás Herendi developed an algorithm to construct pseudorandom number generators in [1]. The algorithm can be used to create linear recurrence sequences with uniform distribution modulo powers of 2, with theoretically arbitrarily large period lengths. The elements of the sequences can likewise be arbitrarily large. Computing new elements is simple, using linear feedback shift registers. Although unpredictability does not hold for the base sequences created, the elements can be transformed in ways to make this requirement hold.

In order to ensure that the algorithm can be used in practice, it is necessary to analyse how efficiently it can be implemented, as well as test the statistical properties of the linear recurrence sequences created.

The author has researched several practical aspects of implementing the algorithm. The algorithm requires an irreducible polynomial over $GF(2)$ with high degree and order as input. A new method to generate such polynomials was developed and compared to other suitable methods. Several implementations of the computationally dominant operations were created using well-known number theoretical software libraries, as well as hardware implementations using the RIVYERA FPGA platform, and their performances tested. After creating an efficient implementation of the algorithm, several pseudorandom sequences were constructed and their properties tested using the NIST statistical test suite [10]. These results can be found in section 2.1.

### 1.1.2 Linear code generation

Since the beginning of Coding Theory, the practical question of searching for linear binary codes has been given considerable attention from researchers [11]. Nowadays, one of the most important areas where such codes are used is Code-based Cryptography. Certain cryptographic schemes using these codes are promising candidates in Post-Quantum Cryptography, that is, cryptography believed to be secure against attack using quantum computers [12].

A central problem of finding linear codes is that for a given codelength $n$, the question of whether of not codes with certain $(n, k, d)$ parameters exist cannot be answered in general. No known universal algorithm exists that can construct any possible linear code with reasonable efficiency, nor are there any theoretical results that can answer if a linear code exists with given arbitrary parameters.

Clearly a naive approach to generating linear codes would start becoming computationally infeasible even for very small codes. In practice, a number of different approaches and methods exist for constructing linear codes, but all of them are only efficient or capable of constructing certain specific families of codes.

One approach to create new linear codes is to modify already known linear codes, constraining the space of possibilities by providing a starting point to our search. One of the six basic methods typically used when modifying linear codes is *augmentation*: given an $(n, k, d)$ linear code, we construct a $(n, k+l, d')$ linear code, $l \geq 1$. The dimension of the code is increased, but

the minimum weight can at best stay equal, that is, $d' \leq d$.

The specific goal of finding linear codes can take multiple forms.

One common goal is the *classification* of linear codes. By classifying linear codes of certain $(n, k, d)$ parameters, we want to find all linear codes $C_1, C_2, \ldots, C_m$ such that for all $i, j \in \{1, 2, \ldots, m\}$, $i \neq j$, $C_i$ and $C_j$ are not permutation equivalent, and any $(n, k, d)$ linear code is permutation equivalent to exactly one code on the list.

Another possible goal can be finding linear codes with previously unknown $(n, k, d)$ parameters. Surprisingly, even for relatively small values of these parameters, there are codes whose existence or non-existence is unknown. There are several open research questions regarding the existence of certain self-dual codes that have not been found for decades, including:

- The existence of $(56, 28, 12)$ Type I codes [13].
- The existence of $(72, 36, 16)$ Type II codes [14].

The minimum distance is the most common criteria when searching for codes, along with type, orthogonality, or being self-dual, but other conditions can be important as well. Any number of other properties can be of interest, both for practical or for theoretical purposes, including weight distribution, the size and composition of the code's automorphism group, the number of codewords of a certain weight, the inclusion of exclusion of certain codewords or subcodes, and so on.

Codes fulfilling these special criteria can then be used as starting points for other methods of creating linear codes, as de-

scribed previously. As an example, [15] gives a way to construct a $(72, 36, 16)$ Type II code, given a $(56, 21, 16)$ self-orthogonal doubly even code that contains the $\mathbf{1}$ codeword, which to date has also not been found. These searches require specific knowledge related to the properties of the code in order to be efficient.

The Torch software package created by the author implements a number of search algorithms based on augmentation to generate linear codes. It is designed to be flexible enough to allow for practically arbitrary search conditions, and extensible to enable incorporating new research results. Section 2.2 lists new algorithms based on research results by the author that were incorporated into the package.

# Chapter 2

# Theses

## 2.1 Random number generation

### 2.1.1 Generating irreducible polynomials

**Main result:** A method for generating polynomials over $GF(2)$ guaranteed to not have factors with degrees under a certain threshold is developed.

Let $R$ be a randomly generated degree $n$ polynomial over $GF(2)$. The probability that $R$ is irreducible, supposing that our pick has uniform distribution, can be estimated as

$$P(R\ is\ irreducible) = \frac{G(n)}{2^n} < \frac{2}{n}.$$

where $G(n)$ is Gauss's formula.

Let $I(x)$ be an irreducible polynomial over $\mathbb{F}_2$, $deg(I) = n$. If $Q(x)$, $deg(Q) = m$, is divisible by $I(x)$, then it can be written as $Q(x) = I(x)R(x)$ where $deg(R) = m - n$. This means that the number of degree $m$ polynomials that can be divided by a given degree $n$ irreducible polynomial is equal to the number of degree $m-n$ polynomials, i. e. $2^{m-n}$. If $S(x)$ is a randomly generated degree $m$ polynomial, picked with uniform distribution, then

$$P(I(x)|S(x)) = \frac{2^{m-n}}{2^m} = \frac{1}{2^n}, \text{ and } P(I(x) \not| S(x)) = 1 - \frac{1}{2^n}.$$

Note that this probability is independent of $m$.

Let $I_1(x), ..., I_n(x)$ be irreducible polynomials. Let $S(x)$ be a randomly generated polynomial, picked with uniform distribution, and $\forall i\ deg(S) \geq deg(I_i)$. Then

$$P(\forall i I_i(x) \not| S(x)) = \prod_{i=1}^{n} 1 - \frac{1}{2^{deg(I_i)}}.$$

We use this observation when searching for a large, irreducible polynomial over $\mathbb{F}_2$. The general idea is that we first generate a candidate polynomial that is guaranteed to not have any small factors, then check for irreducibility using one of the testing methods described in subsection 3.2 of the thesis. Making sure that the candidate polynomial doesn't have small factors multiplies the chance of it being irreducible by a constant factor over randomly choosing a polynomial with uniform distribution. For very large degrees, this method is not practical, but it does provide a significant upgrade for the range of degrees we

are interested in for the purposes of this test, which is between $2^{16} - 2^{20}$.

Table 3.1 in the Appendix shows what proportion of randomly chosen polynomials $S$ do not have a factor of degree $i$ or lower.

Let $n$ be the degree of the irreducible polynomial over $GF(2)$ we wish to generate. We will create candidate polynomials $R(x)$, $deg(R(x)) = n$, which will be tested for irreducibility. The steps of creating the candidate polynomials are found in subsection 3.3.1 of the thesis.

Let $M_i$ be the set of all irreducible polynomials over $GF(2)$ with degree at most $i$. Let the polynomials $T_i(x)$ be the following:

$$T_i(x) = \prod_{P(x) \in M_i} P(x).$$

Let $j \in \mathbb{N}$ be the largest integer such that $deg(T_j) < n$.

The polynomials created by the method are guaranteed to not have any factors with degree $j$ or smaller.

In practice, this method requires the precomputed $T_i(x)$ polynomials, and a collection of irreducible polynomials of small degrees, to randomly choose from. In the current implementation, the largest $T_i(x)$ stored is $T_{23}(x)$, which has a degree of 16772836. The collection contains all irreducible polynomials of degree 23 or smaller, and 50000 polynomials each of degrees $24 \leq n \leq 47$.

Statistical tests comparing linear recurrence sequences based on polynomials generated by this method and sequences based

on polynomials generated by the Q-transform method [16] are found in section 3.4 of the thesis. Both methods produced very high quality pseudorandom sequences, passing all relevant benchmarks set by the NIST test suite. The method based on the Q-transform operation is shown to be more easily computable compared to the presented method, but only capable of producing a subset of all irreducible polynomials.

### 2.1.2 Performance tests of implementations

**Main result:** The most efficient solution to implement the computationally expensive steps of the LRS generation algorithm is determined.

Multiple implementations were created by the author of the computationally expensive steps of the algorithm, detailed in chapter 4 of the thesis. The speed of these operations is the dominant factor in determining the runtime of the algorithm. The implemented operations are the following:

- Polynomial GCD over $GF(2)$. The implementations test $gcd(P(x), Q(x)), P, Q \in \mathbb{F}_2[x]$, with $deg(P)$ up to 1000000, and $deg(Q)$ up to 2000000.
- Polynomial modulo over $GF(2)$. The implementations test $Q(x) mod\ P(x), P, Q \in \mathbb{F}_2[x], deg(Q) \geq deg(P)$, with $deg(P)$ up to 1000000, and $deg(Q)$ up to 2000000.
- Matrix multiplication over $GF(4)$. In one variant of the algorithm, exponentiation to a very high degree is required,

using square matrices. The implementations test matrices up to size $1024 \times 1024$.

The software implementations were created using the NTL and FLINT libraries.

For representing polynomials, the NTL library has an optional *gf2x* package available, which implements highly optimized algorithms specifically for polynomials over $GF(2)$. For polynomial arithmetic, FLINT offers five suitable representations for $GF(2)$, depending on the base class used.

For representing matrices over finite fields, NTL offers two suitable classes, while FLINT offers four.

Hardware implementations created by the author for the SciEngines RIVYERA S6-LX150 FPGA platform are also presented. It enables very large scale physical parallelism with the use of FPGAs (field-programmable gate arrays). The machine used in the test contains 16 Spartan-6 LX150 FPGAs.

Implementations were created utilizing all available options. Performance tests were carried out to compare the efficiency of the different software and hardware solutions for these operations. The results of these tests are available in chapter 4 of the appendix of the thesis. The tests determined the NTL software library to be the most efficient solution to be used in the implementation of the linear recurrence sequence generating algorithm.

### 2.1.3 Statistical tests of LRSs

**Main result:** The statistical properties and the practical applicability of the sequences created by the LRS generation algorithm are determined.

Tests were carried out to measure the statistical properties of the pseudorandom number sequences created using the algorithm and tools determined in the previous chapters. These tests are detailed in chapter 5 of the thesis. Several attributes can affect the statistical properties of the sequences. The ones of interest for the tests are the following:

- The degree of the irreducible polynomial used to create the LRS.
- The irreducible polynomial being *dense*, that is, having a large number of non-zero coefficients, or *sparse*.
- The initial values of the sequence, before it begins generating new pseudorandom values.
- Having a function added to the LRS that transforms each pseudorandom value generated. Seven different transformations were implemented and tested.

All tests described in this chapter use the NIST statistical test suite.

Several LRSs were created with different attributes, to test their effects on the statistical properties of the sequences.

Unless stated otherwise, the test data generated by each LRS consists of $2^{21}$ 64-bit words. The NIST test suite splits the data into 100 bitstreams. In the tables found in chapter 5 of the appendix of the thesis, the Proportion column will show the number of bitstreams that passed each test. Following the guidelines of the NIST documentation, a pass rate of 96 for a sample size of 100 is considered acceptable for a test.

The chapter also includes comparisons to other, previously known sequences that are defined in [17], [18] and [19]. These sequences have provably good pseudorandom properties, using measures defined in [20].

The test results and the practical conclusions drawn from them are included in chapter 5 of the thesis. These conclusions provide support for using the algorithm to create pseudorandom number sequences for practical purposes.

## 2.2 Linear code generation

### 2.2.1 Candidate codeword generation

**Main result:** An algorithm is developed to efficiently determine codewords to augment a given generator matrix with, in order to build a linear code with given parameters.

Let $\mu(a, b)$ be the number of coordinates i such that $a_i = b_i = 1$ for two binary vectors of equal length $a$ and $b$. When generating linear codes, the parameters of the code determine

the possible $\mu$-values allowed between codewords. This information can be precomputed in a table, and used to create sets of conditions that help determine which codewords are suitable to augment a code. This table is named the Mu table.

At each step of a search using the algorithms available in Torch, we wish to augment a generator matrix $A$ with codewords $b$ such that the augmented code $A'$ could potentionally be a subcode of the type of code we are searching for. These codewords are named candidate codewords. Subsection 8.1.2 of the thesis shows how to use the Mu table to create a set of conditions for a given matrix $A$ that a candidate codeword $b$ must satisfy. A condition can be created for each codeword in the code generated by $A$. As the number of codewords is exponential in the dimension of the code, this set can become impractically large. In practice, Torch provides an option to define an upper limit on the number of conditions created at each step. This set of conditions is used to speed up the process generating the candidate codewords at each step of a search.

Subsection 8.1.3 of the thesis describes the two default algorithms used to create candidate codewords. The first is a general algorithm that can be used for any linear code search, the second is an algorithm used only when searching for self-orthogonal codes. These algorithms are necessary since a naive approach to generating candidate codewords would be impractically slow even for relatively small linear codes.

The general algorithm takes a set of variables $v_i, 1 \leq i \leq m$ that represent different segments of a codeword. At the start, all variables are unassigned. Given a partial evaluation of the variables, an iteration repeats the following steps: the $v_i$ with

the smallest index $i$ that is currently unassigned is given a value between a predetermined upper and lower bound. The relevant $\mu$-conditions are checked. If the new partial evaluation fails a given condition, the iteration moves on to the next possible value. Otherwise, a recursive function call repeats the algorithm with the new partial evaluation. If all $v_i$ variables are assigned, passing all conditions, the evaluation is returned.

In this way, an incorrect prefix, that fails a necessary $\mu$-condition, is immediately rejected before the algorithm wastes more time trying to finish it, effectively rejecting all codewords that begin with that prefix at once.

A simplified pseudocode of the algorithm can be seen in listing 8.3 in the appendix of the thesis.

The algorithm used when searching for self-orthogonal codes, relies on selecting codewords from $A^{\perp}$, the dual code of $A$, as candidate codewords. The codewords of $A^{\perp}$ are filtered using the set of $\mu$-conditions, and other applicable search conditions specific to the current search.

The conditions used to filter the codewords when using these algorithms are necessary, but usually not sufficient to ensure that the new augmented code $A'$ can lead to an actual search result.

### 2.2.2 Clique-based linear code generation

**Main result:** An algorithm is developed that generates linear codes with given parameters, based on building a graph with nodes representing codewords, and finding cliques in the graph.

The algorithm described in section 8.2 of the thesis generates linear codes by finding cliques in a graph that represent the candidate codewords. The default search defined in Torch uses both depth-first search and the clique-based search, dynamically switching between them during execution. Under usual circumstances, searches begin as a depth-first search, and switch over to clique-based search at a given node when the necessary conditions regarding the structure of the generator matrix at that node are met.

Let $A$ be the generator matrix at a given node in the search tree that meets the conditions for the clique-based search. The basic outline of the algorithm is the following:

- All codewords that can lead to a search result when used to augment $A$ are determined.
- An undirected graph $H$ is constructed where the vertices are the above codewords. The edges are determined based on conditions described in section 8.2 of the thesis. The graph is such that a set of codewords that create a valid search result when augmenting $A$ appear as a clique in the graph.
- To optimize searching for such cliques, the graph is pruned of vertices and edges that are not part of a clique of correct size.
- A recursive algorithm is used to search the graph for cliques that satisfy the conditions of the search.

- For each unique clique found, one generator matrix is constructed as a search result.

This algorithm avoids generating some equivalent codes that a simple depth-first search would generate. Subsection 8.2.1 of the thesis contains some experimental results comparing using only the depth-first search, and the combination of depth-first search and clique based search that is used in the default configuration of Torch.

### 2.2.3 Heuristic linear code generation

**Main result:** The general hill climbing and best-first search algorithms are adapted to be suitable for linear code searches. Heuristic functions are defined that are useful in practical search scenarios.

For classification problems, when the goal is to construct an exhaustive list of linear codes that are not permutation equivalent that satisfy a given set of parameters, the entire search tree needs to be traversed by the search algorithm before the task can be considered finished. In this case, the order of traversal of the child nodes of a given node in the tree does not impact the overall runtime of traversing the entire tree.

If the goal of the search is not classification, but to find a number of linear codes that fit the search criteria, then the order in which the nodes of tree is traversed becomes important.

Informed searches are a classical topic of artificial intelligence, used in a wide variety of problems to improve the per-

formance of finding states with favorable properties in a state space. In these algorithms, special knowledge about the properties of the objects being searched are incorporated to guide the direction of the search.

The Torch framework provides the possibility of using informed search algorithms for constructing linear codes. These algorithms rely on using a heuristic function to calculate a value for each generator matrix, representing how favourable a given matrix is to the search. Practical experience has shown these algorithms to be most useful when searching for codes with very specific properties, instead of more general search scenarios.

The current implementation of the framework provides two basic informed search algorithms: a hill climbing search, and a best-first search. These are described in section 8.3 of the thesis.

The framework also provides a number of built-in heuristic functions to choose from, and the ability to define custom heuristic functions to enable experimentation. The heuristic functions take a generator matrix of any size as input. The return value of a heuristic function can be any data type that can be ordered. The provided heuristic functions and the way to combine and evaluate them are described in subsection 8.3.1.

The hill climbing search is a greedy algorithm. Given the same parameters and search criteria, the search tree constructed by the hill climbing search contains the same nodes as the tree constructed by the depth-first search. The sole difference is the order in which the nodes are constructed. When creating candidate codewords to augment a given generator matrix, the codewords yielded remain unchanged, but the order in which they are yielded is determined by the heuristic function.

Given the same set of parameters and search criteria, the search tree traversed by the best-first search is a subtree of the tree traversed by the depth-first search algorithm. Depending on the parameters of the best-first search and the size of the tree, this subtree might be a proper subtree, or it might be the same tree as built by the depth-first search. As with the hill climbing algorithm, it uses a heuristic function to determine the fitness of each created node, using it to guess which node is more likely to lead to search results.

One challenge of the best-first search algorithm being applied to linear codes is the choice of heuristic function. If the heuristic function $h$ is such that $h(G_1) > h(G_2)$ always holds if the number of rows of $G_1$ is greater then the number of rows of $G_2$, then the best-first search algorithm is effectively the same as the hill climbing algorithm. However in practice, if a generator matrix with a smaller number of rows can have a better heuristic value than a generator matrix with a greater number of rows, then the best-first search can easily get stuck at a certain depth level of the search tree, rarely moving on to greater depths to actually produce search results.

The heuristic functions defined by Torch or custom created by the user can be combined to create multi-valued heuristics. Some practical results of using these algorithms and heuristic functions are found in subsection 8.3.2 of the thesis.

### 2.2.4 Paperclip algorithm

**Main result:** An algorithm is developed that generates linear codes in a way that rarely produces permutation equivalent

codes, with very small space requirements. A uniquely determined generator matrix for any given set of permutation equivalent linear codes is defined, named the *superminimal* generator. Four efficiently computable conditions are given that are necessary but not sufficient to determine if a given generator matrix is superminimal.

The motivation of the *Paperclip* algorithm described in section 10.3 of the thesis is to create a method that fulfills multiple criteria: it should be suitable for both classification and targeted search tasks, the search results should very rarely contain permutation equivalent codes, and the algorithm should not need to record or compare matrices.

Given parameters $(n, k, d)$ that we want to search for, the ideal algorithm we wish to build has the following properties:

- It is a depth-first search algorithm. The root node of the tree is an $(n, 1, d)$ code. During each step, the generator matrix is augmented by one new line, such that at depth $k'$, all nodes are $(n, k', d)$ codes.

- Let $P(C)$ be the set of linear codes permutation equivalent to $C$. Let $R_{P(C)}$ be a uniquely determined linear code from $P(C)$, called the *representative* of $P(C)$. For each set $P(C)$ for an $(n, k, d)$ code $C$, the tree includes $R_{P(C)}$ as a leaf node, and no other codes from $P(C)$ are present in the tree.

- Let $C'$ and $C''$ be two nodes in the tree such that $C'$ is the parent of $C''$. If $C''$ is an $(n, k'', d)$ code, then $C'$ is a

$(n, k'' - 1, d)$ code and $R_{P(C')} = C'$. In other words, the parent of a representative code is the representative of its own permutation equivalence set.

Based on these properties, the presented method is classified as a Read-Faradzev algorithm [21] [22].

The algorithm requires a uniquely determined generator matrix to be defined for any linear code $C$. The *reduced row echelon* form (RRE) is suitable for the algorithm.

Let $C$ be a binary linear code. The representative element of $P(C)$, denoted $R_{P(C)}$, is the minimal element of $P(C)$ with respect to the ordering defined in subsection 10.3.2 of the thesis.

The RRE generator matrix of $R_{P(C)}$, denoted $G_{RRE}(R_{P(C)})$, is named the *superminimal generator* of $P(C)$. In the presented algorithm, the usual RRE form of a matrix is flipped upside-down to make certain conditions easier to understand and compute.

The search algorithm builds generator matrices by augmenting them one row at a time. After each augmentation, we need to decide if the new matrix is superminimal or not. Because of the typical size of $P(C)$ for any interesting code $C$, a naive approach to this decision would be extremely impractical. In subsection 10.3.4 of the thesis, four conditions are given that can be efficiently evaluated to help with this decision. These conditions are necessary, but not sufficient for deciding if an RRE generator matrix $G$ is superminimal.

- Local minimum condition: Define linear codes $C$ and $C'$ to be *transposition neighbors* if $C$ can be transformed into $C'$ with a single transposition of two coordinates of its

codewords. When calculating the superminimal generator of a set $P(C)$, a greedy algorithm that, given a code $C$ iteratively keeps moving toward the smallest transposition neighbor of $C$ is not sufficient. That is, local minima exist: codes that are smaller than all of their transposition neighbors, but whose RRE generator is not necessarily superminimal. The condition given in the thesis is sufficient and necessary to check if a given RRE generator matrix $G$ of a code $C$ is a local minimum compared to its transposition neighbors.

- Border subcode condition 1 and 2: Two conditions are given based on the existence of certain punctured subcodes, named *border subcodes*, which are defined in the thesis. If an RRE generator matrix $G$ contains a submatrix generating a border subcode that fulfills either of these two conditions, then $G$ can not be superminimal.

- Automorphism group condition: Let $Aut(C)$ denote the *automorphism group* of $C$, that is, the group of all coordinate permutations of $C$ that do not change the linear code. For a generator $G$ with rows $g_1, g_2, \ldots, g_k$ generating a linear code $C$, let $C^i$ denote the linear code generated by $G^i$ with rows $g_1, g_2, \ldots, g_i$, $1 \leq i \leq k$. If $G$ is a superminimal generator, then for all $1 \leq j < k$ and all $p \in Aut(C^j)$, $g_{j+1} \leq p(g_{j+1})$ must hold. In other words, for row $g_{j+1}$, the permutations in the automorphism group of the code generated by the previous rows $Aut(C^j)$ can not make the binary form of $g_{j+1}$ smaller if $G$ is superminimal.

Subsection 10.3.5 of the thesis presents some practical results of using the presented algorithm. Section 10.4 of the thesis compares the algorithm to another algorithm with similar goals described by Bouyukliev et al. in [23]. The general conclusion of the comparison is that the Bouyukliev's algorithm is more efficient for the very specific goal it was created for, while Paperclip is a generally slower, but more flexible and widely applicable algorithm.

# Bibliography

[1] T. Herendi, "Construction of uniformly distributed linear recurring sequences modulo power of 2," *Uniform Distribution Theory*, vol. 13, no. 1, pp. 109–129, 2018.

[2] T. Herendi and S. R. Major, "Modular exponentiation of matrices on FPGA-s," *Acta Univ. Sapientiae, Inform.*, vol. 3, no. 2, pp. 172–191, 2011.

[3] T. Herendi and S. R. Major, "Efficient random number generation on fpgas," in *ICAI 2014: Proceedings of the 9th International Conference on Applied Informatics*, pp. 313–320, 2014.

[4] T. Herendi and S. R. Major, "Using irreducible polynomials for random number generation," *Annales Mathematicae et Informaticae*, vol. 56, p. 36–46, 2022.

[5] C. Hannusch and S. R. Major, "Torch: Software package for the search of linear binary codes," in *2022 IEEE 2nd*

*Conference on Information Technology and Data Science (CITDS)*, pp. 103–106, 2022.

[6] C. Hannusch and S. R. Major, "Neighborhoods of binary self-dual codes," *Contemporary Mathematics*, vol. 4, pp. 1174–1179, 2023.

[7] A. Menezes, P. Van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.

[8] H. Niederreiter, *Random number generation and quasi-Monte Carlo methods*. Society for Industrial and Applied Mathematics, 1992.

[9] D. E. Knuth, *The art of computer programming, volume 2 (3rd ed.)*. Addison-Wesley Longman Publishing Co., Inc., 1997.

[10] NIST, "A statistical test suite for random and pseudorandom number generators for cryptographic applications." https://csrc.nist.gov/Projects/Random-Bit-Generation/Documentation-and-Software.

[11] V. Pless, R. A. Brualdi, and W. C. Huffman, *Handbook of coding theory*. Elsevier Science Inc., 1998.

[12] R. Overbeck and N. Sendrier, *Code-based cryptography. In: Post-Quantum Cryptography*, pp. 95–145. Springer Berlin Heidelberg, 2009.

[13] S. T. Dougherty, J. L. Kim, and P. Solé, "Open problems in coding theory," *Contemp. Math*, vol. 634, pp. 79–99, 2015.

[14] J. H. Conway and N. J. A. Sloane, "A new upper bound on the minimal distance of self-dual codes," *IEEE Transactions on Information Theory*, vol. 36, no. 6, pp. 1319–1333, 1990.

[15] S. Bouyuklieva, "Self-dual codes with some applications to cryptography." NATO Advanced Research Workshop, 2008.

[16] H. Meyn, "On the construction of irreducible self-reciprocal polynomials over finite fields," *Communication and Computing*, vol. 1, pp. 43–53, 1990.

[17] K. Gyarmati, "On a family of pseudorandom binary sequences," *Period. Math. Hungar*, vol. 49, pp. 45–63, 2004.

[18] L. Goubin, C. Maudit, and A. Sárközy, "Construction of large families of pseudorandom binary sequences," *J. Number Theory*, vol. 106, pp. 56–69, 2004.

[19] C. Maudit and A. Sárközy, "Construction of pseudorandom binary sequences by using the multiplicative inverse," *Acta Math. Hungar.*, vol. 109, pp. 75–107, 2005.

[20] C. Maudit and A. Sárközy, "On finite pseudorandom binary sequences I: Measure of pseudorandomness, the Legendre symbol," *Acta Arith.*, vol. 82, no. 4, pp. 365–377, 1997.

[21] R. C. Read, "Every one a winner," *Annals Discrete Math.*, vol. 2, p. 107–120, 1978.

[22] I. A. Faradzev, "Constructive enumeration of combinatorial objects," *Problemes Combinatoires et Theorie des Graphes Colloque Internat, CNRS 260 CNRS Paris*, p. 131–135, 1978.

[23] I. Bouyukliev, S. Bouyuklieva, M. Dzhumalieva-Stoeva, and V. Monev, "What is genselfdual?," *Serdica Journal of Computing*, vol. 10, no. 3-4, pp. 231–244, 2016.

UNIVERSITY AND NATIONAL LIBRARY
UNIVERSITY OF DEBRECEN
H-4002 Egyetem tér 1, Debrecen
Phone: +3652/410-443, email: publikaciok@lib.unideb.hu

Registry number: DEENK/288/2024.PL
Subject: PhD Publication List

Candidate: Sándor Roland Major
Doctoral School: Doctoral School of Informatics
MTMT ID: 10061591

## List of publications related to the dissertation

Foreign language scientific articles in Hungarian journals (1)

1. Herendi, T., **Major, S. R.**: Using irreducible polynomials for random number generation.
   *Ann. Math. Inform. 56*, 36-46, 2022. ISSN: 1787-5021.
   DOI: http://dx.doi.org/10.33039/ami.2022.12.012
   IF: 0.3

Foreign language scientific articles in international journals (2)

2. Hannusch, C., **Major, S. R.**: Neighborhoods of Binary Self-Dual Codes.
   *Contemp. Math. 4* (4), 1174-1179, 2023. ISSN: 2705-1064.
   DOI: http://dx.doi.org/10.37256/cm.4420232397

3. Herendi, T., **Major, S. R.**: Modular exponentiation of matrices on FPGA-s.
   *Acta Univ. Sap., Inf. 3* (2), 172-191, 2011. ISSN: 1844-6086.

Foreign language conference proceedings (2)

4. Hannusch, C., **Major, S. R.**: Torch: Software Package For The Search Of Linear Binary Codes.
   In: 2022 IEEE 2nd Conference on Information Technology and Data Science (CITDS): Proceedings / Fazekas István, IEEE, Piscataway, 103-106, 2022.

5. Herendi, T., **Major, S. R.**: Efficient random number generation on FPGAs.
   In: ICAI 2014: Proceedings of the 9th International Conference on Applied Informatics, vol. 1-2 / Kovács Emőd, Kusper Gábor, Kunkli Roland, Tómács Tibor, Eszterházy Károly Tanárképző Főiskola, Eger, 313-320, 2014. ISBN: 9786155297182

UNIVERSITY AND NATIONAL LIBRARY
UNIVERSITY OF DEBRECEN
H-4002 Egyetem tér 1, Debrecen
Phone: +3652/410-443, email: publikaciok@lib.unideb.hu

# List of other publications

Foreign language scientific articles in international journals (1)

6. Besenczi, R., Bátfai, N., Jeszenszky, P., **Major, S. R.**, Monori, F., Ispány, M.: Large-scale simulation of traffic flow using Markov model.
   *PLoS One. 16* (2), 1-31, 2021. EISSN: 1932-6203.
   DOI: http://dx.doi.org/10.1371/journal.pone.0246062
   IF: 3.752

Hungarian conference proceedings (2)

7. Ispány, M., Jeszenszky, P., **Major, S. R.**: A PTI BSc szak reformja a DE Informatikai Karán.
   In: Informatika a felsőoktatásban 2017 konferencia kiadványa. Szerk.: Aradi Bernadett, Bujdosó Gyöngyi, Horváth Géza, Szokol Patrícia, Debreceni Egyetem Informatikai Kar, Debrecen, 123-128, 2017. ISBN: 9789634732136

8. **Major, S. R.**: Hallgatói projektmunkák a DE-IK Programtervező Informatikus szakán.
   In: Informatika a felsőoktatásban 2017 konferencia kiadványa / Aradi Bernadett, Bujdosó Gyöngyi, Horváth Géza, Szokol Patrícia, Debreceni Egyetem Informatikai Kar, Debrecen, 281-288, 2017. ISBN: 9789634732136

Foreign language conference proceedings (1)

9. Nagy, B., **Major, S. R.**: Connection between interval-valued computing and cellular automata.
   In: 2013 IEEE 14th International Symposium on Computational Intelligence and Informatics (CINTI) / Szakál Anikó, IEEE, Piscataway, 225-230, 2013.

Foreign language abstracts (1)

10. Bátfai, N., Besenczi, R., Ispány, M., Jeszenszky, P., **Major, S. R.**, Monori, F.: Markov modeling and simulation of traffic flow.
    In: Book of Abstracts. Data Science, Statistics & Visualisation DSSV 2018 / P. Filzmoser; P.J.F. Groenen, Springer-Verlag, Vienna, 61, 2018.

**Total IF of journals (all publications): 4,052**
**Total IF of journals (publications related to the dissertation): 0,3**

DEBRECENI EGYETEM * Egyetemi és Nemzeti Könyvtár *

The Candidate's publication data submitted to the iDEa Tudóstér have been validated by DEENK on the basis of the Journal Citation Report (Impact Factor) database.

22 May, 2024

# Tartalomjegyzék

# 1. fejezet

# Bevezető

A disszertáció két kriptográfiai primitívhez kapcsolódó eredményeket mutat be. Ez a két primitív a véletlenszám generátorok és a lineáris kódok.

A disszertáció első része egy algoritmuson alapszik, amely Herendi Tamás eredménye [1]. Az algoritmussal olyan lineárisan rekurzív véletlenszám sorozatok generálhatók, amelyek egyenletes eloszlásúak, modulo 2 egy tetszőleges hatványa. A sorozat periódushossza és a sorozat elemeinek nagysága tetszőleges nagy lehet. A disszertációban bemutatott eredmények elméleti háttere Herendi Tamás és a szerző közös kutatásának eredménye. A gyakorlati alkalmazások fejlesztése, implementációja, valamint a tesztek implementációja és elemzése a szerző eredménye. A szerző ezen témakörhöz releváns publikációi [2], [3] és [4].

A disszertáció második része a lineáris kódok keresésének

problémájára épül. A cél adott, gyakorlatilag tetszőleges feltételeknek megfelelő linearis kódok generálása. A szerző által fejlesztett, Torch nevű szoftver egy kiterjeszthető, újrakonfigurálható megoldást nyújt a problémára. A Torch csomag több különböző keresési algoritmust, keresési feltételeket és egyéb opciókat támogat. Ezen keretrendszer elősegíti a kísérletezést és az új kutatási eredmények beépítését lineáris kódok keresése során. A szoftver felhasználható klasszifikációs problémákhoz, már ismert lineáris kódok generálásához, vagy jelenleg ismeretlen kódok kereséséhez. A disszertációban bemutatott eredmények elméleti háttere Carolin Hannusch és a szerző közös kutatásának eredménye. A szoftver fejlesztése, implementációja és elemzése a szerző eredménye. A szerző ezen témakörhöz releváns publikációi [5] és [6].

## 1.1. Problémafelvetés

### 1.1.1. Véletlenszám generálás

Számos gyakorlati alkalmazás használ pszeudo-véletlen számokat. A kriptográfiában használtak kulcsgeneráláshoz, folyam titkosításhoz, és asszimetrikus kriptorendszerekhez [7]. Tágabban tekintve használtak még többek között szimulációkhoz és Monte Carlo módszerekhez [8]. Az alkalmazástól függően egy pszeudo-véletlen számsorozattal szemben számos elvárás állítható, például az elemek eloszlása, a sorozat periódushossza, az elemek számításának sebessége, és a következő elem megjósolásának nehézsége. Knuth [9] részletesen elemezte a véletlen

sorozatok generálását és statisztikai tesztelésüket.

Herendi Tamás új algoritmust dolgozott ki pszeudo-véletlen számsorozatok előállításához [1]. Az algoritmussal olyan lineárisan rekurzív számsorozatokat (LRS) lehet készíteni, amelyek elemei egyenletes eloszlásúak, modulo 2 egy hatványa, és tetszőlegesen nagy periódushosszal rendelkezhetnek. Az egyes elemek mérete szintén tetszőlegesen nagy lehet. Lineáris visszacsatolt shift regiszter segítségével az új elemek számítása egyszerű. A következő elem megjósolhatatlansága nem teljesül a sorozatokra, viszont az elemek megfelelő függvénnyel történő transzformációjával elérhető.

Az algoritmus gyakorlati felhasználhatóságának elemzéséhez meg kell vizsgálnunk, mennyire hatékonyan implementálható, valamint tesztelnünk kell az előállított lineárisan rekurzív számsorozatok statisztikai tulajdonságait.

A szerző kutatása az algoritmus gyakorlati implementálásának és felhasználásának számos aspektusát vizsgálta. Az algoritmus bemenetként igényel egy $GF(2)$ feletti, nagy fokszámú és rendű irreducibilis polinomot. A szerző kidolgozott egy új módszert, amivel ilyen polinomok előállíthatók, és összehasonlította egy már ismert hasonló célú módszerrel. Számos implementáció készült, amelyek az algoritmus leginkább számításigényes lépéseit valósítják meg. Ezek az implementációk jól ismert számelméleti szoftver könyvtárakat használnak, valamint hardveres megvalósítás is készült a RIVYERA FPGA platformon. A különböző implementációk teljesítménye összehasonlításra került. Az algoritmus hatékony megvalósítását követően számos pszeudo-véletlen sorozat került előállításra, és lettek letesztelve

a NIST statisztikai tesztkészletének segítségével [10]. Ezek az eredmények a 2.1 alfejezetben találhatók.

### 1.1.2. Lineáris kód generálás

A kódelmélet kezdete óta jelentős figyelmet kapott a kutatóktól a bináris lineáris kódok előállításának kérdése [11]. Manapság a kód alapú kriptográfia az egyik legfontosabb terület, amely ilyen kódokat használ. A poszt-kvantum kriptográfia, vagyis az olyan kriptográfia, ahol a támadások kvantumszámítógépeket használnak, szintén ismer olyan kriptorendszereket, amelyek lineáris kódokat használnak [12].

A lineáris kódok keresésének egy központi problémája, hogy adott $n$ kódhosszúsághoz nem lehet általános megválaszolni, hogy valamilyen $(n, k, d)$ paraméterű kód létezik-e. Nem ismert olyan univerzális algoritmus, ami bármilyen tetszőleges lineáris kódot hatékonyan elő tudna állítani, és nem ismert olyan elméleti eredmény sem, ami tetszőleges paraméterekhez meg tudja válaszolni, hogy létezik-e azokat kielégítő kód.

Egy naiv megközelítés a lineáris kódok generálásához már nagyon kis méretű kódok esetében sem lenne praktikusan megvalósítható. A gyakorlatban több módszer is létezik, amellyel lineáris kódok állíthatók elő, de ezek mindegyike csak kódok bizonyos családjait képesek hatékonyan megkonstruálni.

Új lineáris kódok előállításának egy módja már létező lineáris kódok módosítása. Ezzel korlátozzuk a lehetőségek terét, kezdőpontot adva a kereséshez. A kódok módosításának hat alapvető módszeréből egy az *augmentálás*: adott $(n, k, d)$ lineáris kódhoz konstruálunk egy $(n, k + l, d')$ lineáris kódot, $l \geq 1$.

A kód dimenziója nő, a minimális távolság viszont csökkenhet, vagyis $d' \leq d$.

Lineáris kódok keresése több céllal is történhet.

Egy gyakori cél a kódok *klasszifikációja*. Adott $(n, k, d)$ paraméterű lineáris kódok klasszifikálásakor olyan $C_1, C_2, \ldots, C_m$ kódokat keresünk, amikre bármely $i, j \in \{1, 2, \ldots, m\}$, $i \neq j$ esetén $C_i$ és $C_j$ nem permutáció ekvivalens, és bármely $(n, k, d)$ lineáris kód permutáció ekvivalens pontosan egy kóddal a $C_i$ kódok közül.

Szintén cél lehet jelenleg ismeretlen $(n, k, d)$ paraméterű lineáris kódok keresése. Meglepő módon, viszonylag kis értékek esetén is vannak olyan kódok, amelyek létezése ismeretlen. Egyes önduális kódok létezése évtizedek óta nyitott kérdés, például:

- $(56, 28, 12)$ Egyes Típusú kód létezése [13].
- $(72, 36, 16)$ Kettes Típusú kód létezése [14].

A keresés során a leggyakoribb keresési feltételek a kód minimális távolsága, típusa, ortogonalitása vagy öndualitása, de más feltételek is fontos szerepet játszhatnak. Gyakorlati és elméleti szempontból is számtalan tulajdonság fontos lehet, úgymint a kód súlyeloszlása, a kód automorfizmus csoportjának mérete és összetétele, adott súlyú kódszavak száma, adott kódszavak vagy részkódok tartalmazása, és számos egyéb.

Speciális tulajdonságokkal rendelkező kódok használhatóak új kódokat előállító módszerek bemeneteként. Egy példát kiemelve, [15] bemutatja, hogyan konstruálhatunk $(72, 36, 16)$ Kettes Típusú kódot, ha van egy $(56, 21, 16)$ önortogonális, duplán páros kódunk ami tartalmazza az **1** kódszót. Jelenleg ennek a

kódnak a létezése is nyitott kérdés. Az ilyen keresések hatékony kivitelezéséhez a kód tulajdonságain alapuló speciális tudást használhatunk fel.

A szerző által készített Torch névre hallgató szoftver csomag számos augmentáción alapuló keresőalgoritmust implementál. A szoftver kellően rugalmas ahhoz, hogy gyakorlatilag tetszőleges keresési feltételek megadhatóak legyenek, és viselkedése kiterjeszthető új kutatási eredmények beépítésével. A 2.2 alfejezet tartalmazza azokat az új algoritmusokat, amelyek a szerző kutatási eredményein alapulnak és beépítésre kerültek a Torch csomagba.

# 2. fejezet

# Tézisek

## 2.1. Véletlenszám generálás

### 2.1.1. Irreducibilis polinomok generálása

**Fő eredmény:** Egy olyan módszer került kidolgozásra, amellyel olyan $GF(2)$ feletti polinomok generálhatók, amelyeknek garantáltan nincs egy adott értéknél kisebb fokszámú nemtriviális osztója.

Legyen $R$ egy véletlenszerűen generált $n$-ed fokú polinom $GF(2)$ felett. Annak az esélye, hogy $R$ irreducibilis, feltéve, hogy a választásunk egyenletes eloszlású volt, a következőképpen

becsülhető:

$$P(R\ irreducibilis) = \frac{G(n)}{2^n} < \frac{2}{n}.$$

ahol $G(n)$ a Gauss formula.

Legyen $I(x)$ egy irreducibilis polinom $\mathbb{F}_2$ felett, $deg(I) = n$. Ha $Q(x)$, $deg(Q) = m$ osztható $I(x)$-el, akkor felírható $Q(x) = I(x)R(x)$ alakban, ahol $deg(R) = m - n$. Ez azt jelenti, hogy azon $m$-ed fokú polinomok száma, amelyek oszthatók egy adott $n$-ed fokú irreducibilis polinommal, egyenlő az $m - n$-ed fokú polinomok számával, vagyis $2^{m-n}$. Ha $S(x)$ egy egyenletes eloszlással, véletlenszerűen generált $m$-ed fokú polinom, akkor

$$P(I(x)|S(x)) = \frac{2^{m-n}}{2^m} = \frac{1}{2^n},\ valamint\ P(I(x)\ \not|S(x)) = 1 - \frac{1}{2^n}.$$

Ez a valószínűség független $m$-től.

Legyenek $I_1(x), ..., I_n(x)$ irreducibilis polinomok. Legyen $S(x)$ egy egyenletes eloszlással, véletlenszerűen generált polinom, és $\forall i\ deg(S) \geq deg(I_i)$. Ekkor:

$$P(\forall i I_i(x)\ \not|S(x)) = \prod_{i=1}^{n} 1 - \frac{1}{2^{deg(I_i)}}.$$

Ezt az észrevételt használjuk fel nagy fokszámú, $\mathbb{F}_2$ feletti irreducibilis polinomok előállításához. Elsőnek olyan polinomokat generálunk, amelyeknek garantáltan nincs kis fokszámú tényezőjük, majd megvizsgáljuk, hogy irreducibilisek-e a disszertáció 3.2 alfejezetében bemutatott módszerek egyikével. Az így előállított polinomok egy konstans szorzóval nagyobb eséllyel lesznek irreducibilisek, mint a véletlenszerűen előállított polinomok.

Nagyon nagy fokszámok esetén ez a módszer nem praktikus, de a $2^{16} - 2^{20}$ fokszám tartományban, amely a véletlenszám generáló algoritmus számára már gyakorlatban hasznos, jelentősen javítja az esélyeket.

A disszertáció függelékében a 3.1 táblázat tartalmazza, hogy egy véletlenszerűen választott $S$ polinomnak milyen valószínűséggel nincs $i$ vagy kisebb fokú tényezője.

Tegyük fel, hogy $n$-ed fokú, $GF(2)$ feletti irreducibilis polinomot szeretnénk előállítani. Elsőnek $R(x)$, $deg(R(x)) = n$ jelölteket generálunk, amelyek irreducibilitását később teszteljük. A jelöltek előállításának lépéseit a disszertáció 3.3.1 alfejezete tartalmazza.

Legyen $M_i$ az összes $GF(2)$ feletti, legfeljebb $i$-ed fokú irreducibilis polinom halmaza. Legyen a $T_i(x)$ polinom a következő:

$$T_i(x) = \prod_{P(x) \in M_i} P(x).$$

Legyen $j \in \mathbb{N}$ a legnagyobb természetes szám, ahol $deg(T_j) < n$.

A módszer által előállított polinomoknak garantáltan nincs $j$ vagy kisebb fokszámú tényezője.

A gyakorlatban a módszerhez szükségesek az előre kiszámított $T_i(x)$ polinomok, és kis fokszámú irreducibilis polinomok egy kollekciója, amiből véletlenszerűen választani lehet. A jelenlegi implementációban a legnagyobb tárolt $T_i(x)$ a $T_{23}(x)$, aminek a fokszáma 16772836. A kollekció tartalmazza az összes 23 vagy kisebb fokszámú irreducibilis polinomot, és 50000 $n$-ed fokú polinomot, $24 \leq n \leq 47$.

A disszertáció 3.4 alfejezete olyan lineárisan rekurzív sorozatok összehasonlítását tartalmazza, amelyek az itt bemutatott módszerrel generált polinomokra, és a Q-transzformáció [16] módszerével előállított polinomokra alapulnak. Mindkét módszer rendkívül jó minőségű pszeudo-véletlen sorozatokat eredményezett, amelyek teljes mértékben megfeleltek a NIST tesztkészlet elvárásainak. A Q-transzformáció módszere egyszerűbben számítható, mint az itt bemutatott módszer, viszont csak az irreducibilis polinomok egy részhalmazát képes előállítani.

### 2.1.2. Implementációk teljesítménytesztelése

**Fő eredmény:** Meghatározásra került a leghatékonyabb megoldás az LRS generáló algoritmus leginkább számításigényes lépéseinek implementálásához.

A szerző számos implementációt készített, amelyek az algoritmus legnagyobb számításigényű lépéseit valósítják meg. Ezek az implementációk a disszertáció 4. fejezetében kerülnek bemutatásra. Ezen lépések sebessége nagy hatással van az algoritmus futási idejére. Az implementált operációk a következők:

- $GF(2)$ feletti polinomok legnagyobb közös osztója. A megvalósított művelet $gcd(P(x), Q(x)), P, Q \in \mathbb{F}_2[x]$. A tesztelés során $deg(P)$ felső határa 1000000, $deg(Q)$ felső határa 2000000.
- Polinomiális modulo $GF(2)$ felett. A megvalósított művelet $Q(x) mod\ P(x), P, Q \in \mathbb{F}_2[x], deg(Q) \geq deg(P)$. A

tesztelés során $deg(P)$ felső határa 1000000, $deg(Q)$ felső határa 2000000.

- Mátrix szorzás $GF(4)$ felett. Az algoritmus egyik változatában nagyméretű mátrixok rendkívül magas hatványra emelése szükséges. A tesztelés során a mátrixok méretének felső határa $1024 \times 1024$.

A szoftver implementációk az NTL és FLINT könyvtárak felhasználásával készültek.

Az NTL könyvtár $GF(2)$ feletti polinomok reprezenálására tartalmaz egy *gf2x* opcionális csomagot, amely magas szinten optimalizált műveleteket valósít meg. A FLINT könyvtár a $GF(2)$ feletti polinomok és műveleteik reprezentálására öt különböző osztályt biztosít.

Véges testek feletti mátrixok reprezentálásához az NTL két különböző osztályt biztosít, a FLINT négy osztályt.

A szerző hardver implementációkat is készített a SciEngines RIVYERA S6-LX150 FPGA platformon. Az FPGA (field-programmable gate array) eszközök rendkívül nagy mértékű fizikai párhuzamosítást tesznek lehetővé. A használt FPGA gép 16 Spartan-6 LX150 FPGA-t tartalmaz.

Az összes rendelkezésre álló opcióhoz készült implementáció. Az elkészült szoftver és hardver megvalósítások hatékonysága tesztelésre és összehasonlításra került. A teljesítménytesztek eredménye a dissszertáció függelékének 4. fejezetében található. A tesztek az NTL könyvtárat határozták meg, mint a leghatékonyabb megoldást a lineárisan rekurzív sorozatokat generáló algoritmus megvalósításához.

### 2.1.3. LRS-ek statisztikai tesztelése

**Fő eredmény:** Meghatározásra kerültek az LRS generáló algoritmus által előállított sorozatok statisztikai tulajdonságai és gyakorlati felhasználhatóságuk.

Az előző fejezetekben ismertetett algoritmus által generált pszeudo-véletlen számsorozatok statisztikai tulajdonságai tesztelésre kerültek. A tesztek részletei a disszertáció 5. fejezetében találhatók. A sorozatok statisztikai tulajdonságaira számos attribútum hatással lehet. A teszteléshez a következők lettek kiemelve:

- A sorozat alapjául szolgáló irreduciblis polinom fokszáma.
- Az irreducibilis polinom *sűrű* (a nem nulla együtthatók száma nagy), vagy *ritka*.
- A sorozat kezdőértékei.
- A sorozat által generált elemek transzformációja. Hét különböző transzformáló függvény lett implementálva és tesztelve.

A fejezetben ismertetett tesztek a NIST statisztikai tesztkészletet használják.

Számos, különböző attribútumokkal rendelkező sorozat került létrehozásra és összehasonlításra.

A tesztek többségénél az LRS-ek által generált tesztadatok $2^{21}$ db. 64 bites szóból álltak. A NIST tesztkészlet az adatokat

100 bitfolyamra bontja. A disszertáció függelékének 5. fejezetében található táblázatok mutatják, hogy az egyes teszteknél hány bitfolyam felelt meg a statisztikai elvárásoknak. A NIST dokumentációjában található irányelvek szerint 100 folyam esetén legalább 96 megfelelt szükséges a teszt sikeres teljesítéséhez.

A fejezet szintén tartalmaz összehasonlításokat egyéb, korábban ismert számsorozatokkal [17] [18] [19]. Ezek a sorozatok bizonyíthatóak jó pszeudo-véletlen tulajdonságokkal rendelkeznek, amelyek a [20] publikációban kerülnek bevezetésre.

A teszt eredmények és a belőlük levont gyakorlati következtetések a disszertáció 5. fejezetében találhatók. A következtetések támogatják az algoritmus által előállított pszeudo-véletlen számsorozatok gyakorlati felhasználását.

## 2.2. Lineáris kód generálás

### 2.2.1. Kódszó jelöltek generálása

**Fő eredmény:** Egy algoritmus került kidolgozásra, amely hatékonyan meghatározza azon kódszavakat, amelyekkel egy generátor mátrixot augmentálni érdemes adott paraméterű lineáris kód építésekor.

Legyen $\mu(a, b)$ azon i koordináták száma, amelyekre $a_i = b_i = 1$, ahol $a$ és $b$ két egyenlő hosszúságú bináris vektor. Egy lineáris kód generálásakor a kód paraméterei meghatározzák a kódszavak közötti lehetséges $\mu$-értékeket. Ez az információ előre

kiszámítható egy táblázatban. A keresés során a táblázat segítségével feltételhalmazok határozhatók meg, amelyek segítenek eldönteni, mely kódszavakkal érdemes augmentálni egy adott kódot. A táblázatot Mu-táblának nevezzük.

A Torch-ban elérhető keresőalgoritmusok minden lépése során a cél egy $A$ generátor mátrix augmentálása egy $b$ kódszóval úgy, hogy a kapott $A'$ potenciálisan a keresett típusú kód egy részkódja legyen. Ezeket a $b$ kódszavakat kódszó jelölteknek nevezzük. A disszertáció 8.1.2 alfejezete bemutatja, hogyan lehet a Mu-táblát felhasználni olyan feltételhalmaz előállításához, amit egy adott $A$ mátrix kódszó jelöltjeinek ki kell elégíteniük. Az $A$ által generált kód minden kódszavához meghatározható egy feltétel. Mivel a kódszavak száma exponenciálisan nő a kód dimenziójával, ez a halmaz rendkívül nagy méretű lehet. A gyakorlatban a Torch-nak egy opcióval megadható az egy lépés során létrehozott feltételek számának felső határa. Ezzel a halmazzal felgyorsítható a kódszó jelöltek generálása a keresés során.

A disszertáció 8.1.3 alfejezete bemutatja a két alapértelmezett algoritmust, amivel a Torch kódszó jelölteket állít elő. Az első egy általános algoritmus, ami bármilyen lineáris kód keresésekor használható, a második algoritmus csak önortogonális kódok esetén alkalmazható. Ezek az algoritmusok azért szükségesek, mert egy naiv megközelítés már viszonylag kis méretű kódok esetén is rendkívül lassú lenne.

Az általános algoritmus $v_i, 1 \leq i \leq m$ változókat használ, amik egy kódszó bizonyos szegmenseit reprezentálják. A kezdetben a változóknak nincs hozzárendelt értéke. Egy iterációban a változók egy parciális kiértékelése mellett a következő lépések

történnek: legyen $v_i$ a legkisebb indexű, jelenleg érték nélküli változó. Ez a változó értéket kap, ami egy meghatározott alsó és felső határ között lehet. A megfelelő $\mu$-feltételek kiértékelődnek. Ha az új parciális kiértékelés nem felel meg egy feltételnek, az iteráció továbbhalad a következő lehetséges értékre. Ha minden feltételnek megfelelt, akkor egy rekurzív hívással az algoritmus megismétlődik az új parciális kiértékelés mellett. Ha minden $v_i$ változó értéket kapott, és minden feltételnek megfelelnek, akkor a kiértékelés előállított egy új kódszó jelöltet.

Ilyen módon egy helytelen kezdőszelet, ami nem felel meg egy szükséges $\mu$-feltételnek, azonnal elutasításra kerül, mielőtt az algoritmus feleslegesen időt töltene megegyező kezdőszeletű kódszavak generálásával.

Az algoritmus leegyszerűsített pszeudokódja a disszertáció függelékének 8.3 alfejezetében található.

Az önortogonális kódok keresésekor használt algoritmus a jelenlegi kód duális kódjából válogat kódszó jelölteket. A duális kód kódszavai a $\mu$-feltételek és egyéb szükséges keresési feltételek alapján kerülnek szűrésre.

A kódszavak szűréséhez használt feltételek szükségesek, de általában nem elégségesek ahhoz, hogy biztosítsák, hogy a kódszó jelölttel augmentált kód valódi keresési eredményhez fog vezetni.

### 2.2.2. Klikk-alapú lineáris kód generálás

**Fő eredmény:** Egy algoritmus került kidolgozásra, ami adott paraméterű lineáris kódokat generál. Az algoritmus alapja egy gráf, amiben a csúcsok kódszavakat reprezentálnak. A cél klik-

kek keresése a gráfban.

A disszertáció 8.2 alfejezete egy kódgeneráló algoritmust mutat be, amely klikkeket keres egy gráfban, ami a kódszó jelöltek alapján kerül felépítésre. A Torch alapértelmezett keresője mélységi keresést és klikk-alapú keresést is használ, futás közben dinamikusan váltogatva közöttük. Tipikus esetben a keresés mélységi keresésként indul. A keresőfa egy adott csúcsánál, ha a generátor mátrix alakjára vonatkozó feltételek teljesülnek, átvált klikk-alapú keresésre.

Legyen $A$ egy generátor mátrix a keresőfa egy csúcsában, amely kielégíti a klikk-alapú keresésre váltáshoz szükséges feltételeket. Az algoritmus alapvető felépítése a következő:

- Meghatározzuk az $A$ mátrixhoz tartozó összes kódszó jelöltet.
- Létrehozzuk a $H$ irányítatlan gráfot, aminek a csúcsai a kódszó jelöltek. Az éleket a disszertáció 8.2 alfejezetében leírt feltételek határozzák meg. A gráf felépítéséből adódik, hogy azok a kódszó halmazok, amelyekkel $A$-t augmentálva egy keresett kódot kapunk, klikkeket alkotnak a gráfban.
- A klikk keresés optimalizálásához a gráfból töröljük azokat a csúcsokat, amelyeknek nincs megfelelő számú szomszédja.
- Egy rekurzív algoritmussal megkeressük a gráfban található összes olyan klikket, ami eleget tesz a keresési feltételeknek.

- Minden megtalált klikk segítségével egy generátor mátrixot konstruálunk, ami a keresés eredménye lesz.

Ezzel az algoritmussal elkerülhető bizonyos ekvivalens kódok generálása, amiket az egyszerű mélységi kereső generálna. A disszertáció 8.2.1 alfejezete összehasonlítja a mélységi kereső, és a Torch-ban alapértelmezett mélységi és klikk kereső teljesítményét.

### 2.2.3. Heurisztikus lineáris kód generálás

**Fő eredmény:** Az általános hegymászó és best-first kereső algoritmusok adaptálva lettek lineáris kódok kereséséhez. A kereséseket elősegítő heurisztikus függvények kerültek definiálásra.

Klasszifikációs problémák esetén, amikor a cél lineáris kódok olyan kimerítő listájának előállítása, amely nem tartalmaz permutáció ekvivalens kódokat és megfelelnek a keresési feltételeknek, a keresőalgoritmusnak a teljes keresőfát be kell járnia, mielőtt a feladatot késznek tekinthetjük. Ebben az esetben egy adott csúcsból a gyerek csúcsok bejárási sorrendje nem befolyásolja a teljes fa bejárásának futási idejét.

Ha a keresés célja nem klasszifikáció, hanem bizonyos számú, adott feltételeknek megfelelő lineáris kód előállítása, akkor a keresőfa csúcsainak bejárási sorrendje fontos a futási idő szempontjából.

Az informált keresések a mesterséges intelligencia klasszikus témaköre. Széles körben használtak olyan helyzetekben, amikor a cél bizonyos kedvező tulajdonságú állapotok megtalálása

egy állapottérben. Ezeknél az algoritmusoknál felhasználjuk a keresett objektumokkal kapcsolatos ismereteinket a keresés megfelelő irányba tereléséhez.

A Torch keretrendszer lehetőséget ad informált keresőalgoritmusok használára lineáris kódok konstruálásához. Ezek az algoritmusok egy heurisztikus függvényt használnak, amely egy adott generátor mátrixhoz egy értéket rendel, ami reprezentálja a keresés szempontjából a mátrix jóságát. A gyakorlati tapasztalat szerint ezek az algoritmusok hasznosabbak specifikus keresési feltételek mellett mint általánáosabb kereséseknél.

A keretrendszer jelenlegi implementációja két alapvető informált keresőalgoritmust biztosít: a hegymászó algoritmust és a best-first keresést. Ezeket a disszertáció 8.3 alfejezete ismerteti.

Szintén megtalálható a keretrendszerben számos beépített heurisztikus függvény, valamint lehetőséget ad saját heurisztika definiálására, ezzel elősegítve a kísérletezést. A heurisztikus függvények egy tetszőleges méretű generátor mátrixot kapnak bemenetként. A függvény értéke tetszőleges adattípus lehet, amin értelmezhető rendezés. A disszertáció 8.3.1 alfejezete mutatja be a beépített heurisztikus függvényeket és a függvények kombinálásának módját.

A hegymászó keresés egy mohó algoritmus. Megegyező keresési feltételek és paraméterek mellett a hegymászó algoritmus és a mélységi keresés által készített keresőfák csúcsai megegyeznek. A különbség csak a csúcsok létrehozásának sorrendjében van. Amikor kódszó jelölteket keresünk egy adott generátor mátrix augmentálásához, mindkét algoritmus ugyanazokat a jelölteket fogja generálni, viszont a hegymászó algoritmus esetében a sorrendet a heurisztikus függvény fogja meghatározni.

Megegyező keresési feltételek és paraméterek mellett a best-first kereső által bejárt keresőfa a mélységi kereső fájának részfája. A pontos keresési körülményektől és a fa méretétől függően ez a részfa lehet valódi részfa, vagy megegyezhet a mélységi kereső által épített fával. A hegymászó algoritmushoz hasonlóan itt is egy heurisztikus függvény segít eldönteni, hogy a keresés közben melyik csúcs valószínűbb, hogy keresési eredményhez fog vezetni.

Lineáris kódok best-first algoritmussal való keresésekor a fő kihívás a megfelelő heurisztikus függvény kiválasztása. Ha a $h$ heurisztikus függvényre $h(G_1) > h(G_2)$ mindig teljesül, ha $G_1$ sorainak száma nagyobb, mint $G_2$ sorainak száma, akkor a best-first kereső gyakorlatilag megegyezik a hegymászó algoritmussal. A gyakorlatban viszont ha egy kevesebb sorral rendelkező generátor mátrix jobb heurisztikus értéket kaphat, mint egy több sorral rendelkező mátrix, akkor a best-first kereső könnyen megragadhat a keresőfa egy bizonyos mélységénél. Ilyen esetben a keresés nehezen lép tovább nagyobb mélységbe és ad vissza keresési eredményt.

A Torch-ban definiált vagy a felhasználó által létrehozott heurisztikus függvények kombinálhatók többértékű függvényekké. A disszertáció 8.3.2 alfejezete bemutatja az algoritmusok és heurisztikus függvények használatának néhány gyakorlati eredményét.

### 2.2.4. Paperclip algoritmus

**Fő eredmény:** Egy algoritmus került kidolgozásra, amely ritkán generál permutáció ekivalens lineáris kódokat, alacsony tár-

használattal. Bevezetésre kerül a *szuperminimális* generátor mátrix fogalma, ami permutáció ekvivalens kódok tetszőleges halmazához egy egyértelműen meghatározott generátort definiál. Négy hatékonyan számítható, szükséges de nem elégséges feltételt adunk, amely segít eldönteni, hogy egy adott generátor mátrix szuperminimális-e.

A disszertáció 10.3 alfejezetében bemutatott *Paperclip* algoritmus létrehozását több szempont is motiválta: olyan módszert szeretnénk alkotni, amely alkalmas klasszifikációs és célzott keresésekhez egyaránt, a keresési eredmények ritkán tartalmazzanak permutáció ekvivalens kódokat, és az algoritmusnak ehhez ne kelljen mátrixokat tárolnia vagy összehasonlítania.

Tegyük fel, hogy $(n, k, d)$ paraméterű kódokat keresünk. Az ideális algoritmus, amit építeni szeretnénk, a következő tulajdonságokkal rendelkezik:

- Az algoritmus mélységi keresést végez. A fa gyökér csúcsa egy $(n, 1, d)$ kódot tartalmaz. Minden lépésben a generátor mátrixot egy új sorral augmentáljuk, vagyik $k'$ mélységben a csúcsok $(n, k', d)$ kódokat tartalmaznak.

- Legyen $P(C)$ azon lineáris kódok halmaza, amelyek permutáció ekvivalensek $C$-vel. Legyen $R_{P(C)}$ egy egyértelműen meghatározott kód a $P(C)$ halmazból, amit a $P(C)$ *képviselőjének* nevezünk. Bármely $(n, k, d)$ paraméterű $C$ kódhoz a keresőfa levélelemként tartalmazza $R_{P(C)}$-t, és más kód $P(C)$-ből nincs jelen a fában.

- Legyen $C'$ és $C''$ két csúcs a fában, és legyen $C'$ a $C''$

csúcs szülője. Ha $C''$ egy $(n, k'', d)$ kód, akkor $C'$ egy $(n, k'' - 1, d)$ kód, és $R_{P(C')} = C'$. Más szóval, egy képviselő szülője szintén legyen a saját permutáció ekvivalencia halmazának képviselője.

Ezen tulajdonságok alapján a bemutatott módszer a Read-Faradzev algoritmusok csoportjába tartozik [21] [22].

Az algoritmushoz szükséges egy generátor mátrix alak, ami bármely $C$ lineáris kódhoz egyértelműen meghatározott. A *reduced row echelon* (RRE) alak megfelelő az algoritmus számára.

Legyen $C$ egy bináris lineáris kód. A $P(C)$ halmaz képviselője, $R_{P(C)}$, legyen $P(C)$ minimális eleme a disszertáció 10.3.2 alfejezetében definiált rendezés szerint.

Jelölje $G_{RRE}(R_{P(C)})$ az $R_{P(C)}$ kód RRE alakú generátor mátrixát. Ez a generátor a $P(C)$ halmaz *szuperminimális* generátora. Az algoritmus implementációjában a tipikus RRE alak függőlegesen meg van fordítva, hogy bizonyos feltételek megértése és számítása egyszerűbb legyen.

A keresőalgoritmus soronként augmentálva építi a generátor mátrixokat. Minden augmentáció után meg kell vizsgálnunk, hogy a kapott mátrix szuperminimális-e vagy sem. A gyakorlatban érdekes esetekben a $P(C)$ halmazok hatalmas mérete miatt egy naiv megközelítés nem célravezető. A disszertáció 10.3.4 alfejezetében négy hatékonyan számítható feltételt adunk, amelyek szükségesek, de nem elégségesek azt eldönteni, hogy egy adott RRE alakú generátor mátrix szuperminimális-e.

- Lokális minimum feltétel: Legyen két lineáris kód $C$ és $C'$ *transzpozíció szomszédok*, ha $C$-ből megkaphatjuk $C'$-t a kódszavak két koordinátájának felcserélésével. Ha a

$P(C)$ halmaz szuperminimális generátorát akarjuk kiszámítani, akkor egy mohó algoritmus, amely egy kódnak minden lépésben a legkisebb transzpozíció szomszédja felé mozdul el, nem elégséges. Léteznek lokális minimumok: olyan kódok, amelyek az összes transzpozíció szomszédjuknál kisebbek, de nem feltétlenül szuperminimálisak. A disszertációban megtalálható feltétel szükséges és elégséges ahhoz, hogy egy adott generátor mátrixról eldöntsük, az általa generált kód lokális minimum-e a transzpozíció szomszédaihoz képest.

- Határ részkód feltétel 1 és 2: A disszertáció két feltételt ad, ami bizonyos *punctured* részkódok létezését vizsgálja. Ezeket a részkódokat *határ részkódoknak* nevezzük, és a disszertációban kerülnek definiálásra. Ha egy RRE alakú generátor tartalmaz olyan részmátrixot, ami olyan határ részkódot generál, ami megfelel a két feltétel bármelyikének, akkor a generátor nem lehet szuperminimális.

- Automorfizmus csoport feltétel: Legyen $Aut(C)$ a $C$ kód *automorfizmus csoportja*, vagyis azon koordináta permutációk halmaza, amelyek nem változtatják meg $C$-t. Jelölje $g_1, g_2, \ldots, g_k$ a $G$ generátor sorait, és legyen $C$ a $G$ által generált kód. Legyen $C^i$ a $G^i$ mátrix által generált kód, ahol $G^i$ sorai $g_1, g_2, \ldots, g_i$, $1 \leq i \leq k$. Ha $G$ egy szuperminimális generátor, akkor minden $1 \leq j < k$ és minden $p \in Aut(C^j)$ esetén $g_{j+1} \leq p(g_{j+1})$ teljesül. Ez azt jelenti, hogy a $g_{j+1}$ sor bináris alakját az $Aut(C^j)$ automorfizmus csoportban levő permutációk nem csökkenthetik, ha $G$ szuperminimális.

A disszertáció 10.3.5 alfejezete bemutatja az algoritmus alkalmazásának néhány gyakorlati eredményét. A disszertáció 10.4 alfejezete összehasonlítja az algoritmust egy másik hasonló célú algoritmussal, amely Bouyukliev-től származik [23]. Az általános következés szerint Bouyukliev algoritmusa hatékonyabb arra a specifikus célra, amire készült, amíg a Paperclip algoritmus gyakran lassabb, viszont flexibilisebb és szélesebb körben felhasználható algoritmus.

# Irodalomjegyzék


[1] T. Herendi, „Construction of uniformly distributed linear recurring sequences modulo power of 2," *Uniform Distribution Theory*, vol. 13, no. 1, pp. 109–129, 2018.

[2] T. Herendi and S. R. Major, „Modular exponentiation of matrices on FPGA-s," *Acta Univ. Sapientiae, Inform.*, vol. 3, no. 2, pp. 172–191, 2011.

[3] T. Herendi and S. R. Major, „Efficient random number generation on fpgas," in *ICAI 2014: Proceedings of the 9th International Conference on Applied Informatics*, pp. 313–320, 2014.

[4] T. Herendi and S. R. Major, „Using irreducible polynomials for random number generation," *Annales Mathematicae et Informaticae*, vol. 56, p. 36–46, 2022.

[5] C. Hannusch and S. R. Major, „Torch: Software package for the search of linear binary codes," in *2022 IEEE 2nd*

*Conference on Information Technology and Data Science (CITDS)*, pp. 103–106, 2022.

[6] C. Hannusch and S. R. Major, „Neighborhoods of binary self-dual codes," *Contemporary Mathematics*, vol. 4, pp. 1174–1179, 2023.

[7] A. Menezes, P. Van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.

[8] H. Niederreiter, *Random number generation and quasi-Monte Carlo methods*. Society for Industrial and Applied Mathematics, 1992.

[9] D. E. Knuth, *The art of computer programming, volume 2 (3rd ed.)*. Addison-Wesley Longman Publishing Co., Inc., 1997.

[10] NIST, „A statistical test suite for random and pseudorandom number generators for cryptographic applications." https://csrc.nist.gov/Projects/Random-Bit-Generation/Documentation-and-Software.

[11] V. Pless, R. A. Brualdi, and W. C. Huffman, *Handbook of coding theory*. Elsevier Science Inc., 1998.

[12] R. Overbeck and N. Sendrier, *Code-based cryptography. In: Post-Quantum Cryptography*, pp. 95–145. Springer Berlin Heidelberg, 2009.

[13] S. T. Dougherty, J. L. Kim, and P. Solé, „Open problems in coding theory," *Contemp. Math*, vol. 634, pp. 79–99, 2015.

[14] J. H. Conway and N. J. A. Sloane, „A new upper bound on the minimal distance of self-dual codes," *IEEE Transactions on Information Theory*, vol. 36, no. 6, pp. 1319–1333, 1990.

[15] S. Bouyuklieva, „Self-dual codes with some applications to cryptography." NATO Advanced Research Workshop, 2008.

[16] H. Meyn, „On the construction of irreducible self-reciprocal polynomials over finite fields," *Communication and Computing*, vol. 1, pp. 43–53, 1990.

[17] K. Gyarmati, „On a family of pseudorandom binary sequences," *Period. Math. Hungar*, vol. 49, pp. 45–63, 2004.

[18] L. Goubin, C. Maudit, and A. Sárközy, „Construction of large families of pseudorandom binary sequences," *J. Number Theory*, vol. 106, pp. 56–69, 2004.

[19] C. Maudit and A. Sárközy, „Construction of pseudorandom binary sequences by using the multiplicative inverse," *Acta Math. Hungar.*, vol. 109, pp. 75–107, 2005.

[20] C. Maudit and A. Sárközy, „On finite pseudorandom binary sequences I: Measure of pseudorandomness, the Legendre symbol," *Acta Arith.*, vol. 82, no. 4, pp. 365–377, 1997.

[21] R. C. Read, „Every one a winner," *Annals Discrete Math.*, vol. 2, p. 107–120, 1978.

[22] I. A. Faradzev, „Constructive enumeration of combinatorial objects," *Problemes Combinatoires et Theorie des Graphes Colloque Internat, CNRS 260 CNRS Paris*, p. 131–135, 1978.

[23] I. Bouyukliev, S. Bouyuklieva, M. Dzhumalieva-Stoeva, and V. Monev, „What is genselfdual?," *Serdica Journal of Computing*, vol. 10, no. 3-4, pp. 231–244, 2016.

DEBRECENI EGYETEM
EGYETEMI ÉS NEMZETI KÖNYVTÁR
H-4002 Debrecen, Egyetem tér 1, Pf.: 400
Tel.: 52/410-443, e-mail: publikaciok@lib.unideb.hu

Nyilvántartási szám: DEENK/288/2024.PL
Tárgy: PhD Publikációs Lista

Jelölt: Major Sándor Roland
Doktori Iskola: Informatikai Tudományok Doktori Iskola
MTMT azonosító: 10061591

## A PhD értekezés alapjául szolgáló közlemények

Idegen nyelvű tudományos közlemények hazai folyóiratban (1)

1. Herendi, T., **Major, S. R.**: Using irreducible polynomials for random number generation.
   *Ann. Math. Inform. 56*, 36-46, 2022. ISSN: 1787-5021.
   DOI: http://dx.doi.org/10.33039/ami.2022.12.012
   IF: 0.3

Idegen nyelvű tudományos közlemények külföldi folyóiratban (2)

2. Hannusch, C., **Major, S. R.**: Neighborhoods of Binary Self-Dual Codes.
   *Contemp. Math. 4* (4), 1174-1179, 2023. ISSN: 2705-1064.
   DOI: http://dx.doi.org/10.37256/cm.4420232397

3. Herendi, T., **Major, S. R.**: Modular exponentiation of matrices on FPGA-s.
   *Acta Univ. Sap., Inf. 3* (2), 172-191, 2011. ISSN: 1844-6086.

Idegen nyelvű konferencia közlemények (2)

4. Hannusch, C., **Major, S. R.**: Torch: Software Package For The Search Of Linear Binary Codes.
   In: 2022 IEEE 2nd Conference on Information Technology and Data Science (CITDS): Proceedings / Fazekas István, IEEE, Piscataway, 103-106, 2022.

5. Herendi, T., **Major, S. R.**: Efficient random number generation on FPGAs.
   In: ICAI 2014: Proceedings of the 9th International Conference on Applied Informatics, vol. 1-2 / Kovács Emőd, Kusper Gábor, Kunkli Roland, Tómács Tibor, Eszterházy Károly Tanárképző Főiskola, Eger, 313-320, 2014. ISBN: 9786155297182

DEBRECENI EGYETEM * Egyetemi és Nemzeti Könyvtár *

DEBRECENI EGYETEM

DEBRECENI EGYETEM
EGYETEMI ÉS NEMZETI KÖNYVTÁR
H-4002 Debrecen, Egyetem tér 1, Pf.: 400
Tel.: 52/410-443, e-mail: publikaciok@lib.unideb.hu

# További közlemények

Idegen nyelvű tudományos közlemények külföldi folyóiratban (1)

6. Besenczi, R., Bátfai, N., Jeszenszky, P., **Major, S. R.**, Monori, F., Ispány, M.: Large-scale simulation of traffic flow using Markov model.
*PLoS One. 16* (2), 1-31, 2021. EISSN: 1932-6203.
DOI: http://dx.doi.org/10.1371/journal.pone.0246062
IF: 3.752

Magyar nyelvű konferencia közlemények (2)

7. Ispány, M., Jeszenszky, P., **Major, S. R.**: A PTI BSc szak reformja a DE Informatikai Karán.
In: Informatika a felsőoktatásban 2017 konferencia kiadványa. Szerk.: Aradi Bernadett, Bujdosó Gyöngyi, Horváth Géza, Szokol Patrícia, Debreceni Egyetem Informatikai Kar, Debrecen, 123-128, 2017. ISBN: 9789634732136

8. **Major, S. R.**: Hallgatói projektmunkák a DE-IK Programtervező Informatikus szakán.
In: Informatika a felsőoktatásban 2017 konferencia kiadványa / Aradi Bernadett, Bujdosó Gyöngyi, Horváth Géza, Szokol Patrícia, Debreceni Egyetem Informatikai Kar, Debrecen, 281-288, 2017. ISBN: 9789634732136

Idegen nyelvű konferencia közlemények (1)

9. Nagy, B., **Major, S. R.**: Connection between interval-valued computing and cellular automata.
In: 2013 IEEE 14th International Symposium on Computational Intelligence and Informatics (CINTI) / Szakál Anikó, IEEE, Piscataway, 225-230, 2013.

Idegen nyelvű absztrakt kiadványok (1)

10. Bátfai, N., Besenczi, R., Ispány, M., Jeszenszky, P., **Major, S. R.**, Monori, F.: Markov modeling and simulation of traffic flow.
In: Book of Abstracts. Data Science, Statistics & Visualisation DSSV 2018 / P. Filzmoser; P.J.F. Groenen, Springer-Verlag, Vienna, 61, 2018.

**A közlő folyóiratok összesített impakt faktora: 4,052**
**A közlő folyóiratok összesített impakt faktora (az értekezés alapjául szolgáló közleményekre): 0,3**

DEBRECENI EGYETEM * Egyetemi és Nemzeti Könyvtár *

A DEENK a Jelölt által az iDEa Tudóstérbe feltöltött adatok bibliográfiai és tudománymetriai ellenőrzését a tudományos adatbázisok és a Journal Citation Reports Impact Factor lista alapján elvégezte.

Debrecen, 2024.05.22.