Comparison of Convolutional Codes and Random Codes Based on Quasigroups for transmission in BSC

Lina Lumburovska[1], Aleksandra Popovska-Mitrovikj[2], Verica Bakeva[3]
Faculty of Computer Science and Engineering,
Ss. Cyril and Methodius, University of Skopje, R.N. Macedonia
lina.lumburovska@students.finki.ukim.mk[1]
aleksandra.popovska.mitrovikj@finki.ukim.mk[2]
verica.bakeva@finki.ukim.mk[3]

Abstract: *Error-correcting codes are widely used in modern coding theory, and their applications in networks and communication cannot be omitted. Together with the error detecting codes, they are the core of every possible transmission and communication. In coding theory, information theory and telecommunications, error-correcting codes are used to control errors in data which are transmitted over different communication channels. Convolutional block codes are one of the most popular error-correcting codes which are applied in many networks. On the other side, Random Codes Based on Quasigroups (RCBQ) are cryptcodes defined elsewhere. These codes provide a correction of a certain number of errors in the transmitted data and an information security in one algorithm. There are a few modifications of RCBQ, but here we will consider performances of Cut-Decoding algorithm. In this paper, we investigate and compare the bit error probability (BER) of these two codes for rate 1/4 and different values of bit-error probability in the binary symmetric channel. From the obtained experimental results, we conclude that for lower bit-error probability in the binary symmetric channel, the RCBQs are slightly better than convolutional codes. The advantage of RCBQs is that they have some cryptographic properties, but convolutional codes are faster than RCBQs.*

Keywords*: CONVOLUTIONAL CODES, RCBQ, BIT ERROR PROBABILITY, BINARY SYMMETRIC CHANNEL*

1. Introduction

In the last several decades, coding theory together with its error-correcting and error-detecting codes has evolved significantly. The formal definition to declare this branch as a separate discipline was created in 1948 when Claude Shannon announced his paper with a title ”A Mathematical Theory of Communications”. The primary focus of his paper is solving the problem of how to best send an encoded information (message) from a sender to a receiver. In his continuous work, he proved that it is possible to encode a message where the number of extra added bits is minimal. A few years later in 1968, Richard Hamming, who was in the same organization as Shannon, won the Turing Award where he produced a 3- bit code for four data bits. This code by Hamming was invented after various failed attempts to break out a message on a paper using the parity code. In his frustration, he said the statement: “If it can detect the error, why can’t I correct it!”, which became his main motivation for finding error-correcting codes. Since then, the 1950s, coding theory has changed and now consists of different models and codes. Today, error-correcting codes are widely used across various computing systems and telecommunication channels [1].

In this paper, we compare performances of convolutional codes and Random Codes Based on Quasigoups (RCBQs). Convolutional codes are one of the most used and researched codes, and RCBQs are interesting cryptcodes, proposed in [7], and they use cryptographic properties of algebraic structure called quasigroups. In the experiment for the purpose of this paper, binary symmetric channels (BSC) with different bit-error probabilities are used. [2].

The rest of the paper is organized in the following way. In Section 2 we explain convolutional codes and their different implementations of encoding and decoding process. For RCBQs there are different algorithms for encoding/decoding, but in this paper we will consider the performances using the Cut-Decoding algorithm, proposed in [8]. Therefore, in Section 3, we describe the encoding and decoding process of Cut-Decoding algorithm of RCBQs. The experimental results for both codes and their comparison are given in Section 4. At the end, we will draw some conclusions from the conducted analysis.

2. Convolutional Codes

The two main categories of error-correcting codes are block codes and convolutional codes. Block codes work with fixed-size blocks of bits or symbols of predetermined size. On the other hand, convolutional codes work on bits or symbols of arbitrary length. The convolutional codes are defined with three parameters: n, k and K and each convolutional code can be written as (n, k, K). The similarity with block codes is that they process input data of k bits at a time and give an output of n bits for each incoming k bits. In case of convolutional codes, the parameters n and k are quite small, but this is not the case for block codes. Apart from their similarity, the main difference is that the convolutional codes have memory which is noted with the parameter K. Due to their memory, the current output of n bits depends not only on the value of the current block of k input bits, but also on the previous $K - 1$ blocks of k input bits. Basically, this is the main principle of using memory, executing the current values with the knowledge of the previous ones and the states depend on each other. In other words, convolutional coding is a widely used coding that is not based on blocks of bits, but the output bits are determined by logic operations that connect two parts: the present bit in a stream and previous bits. As each bit enters at the left of the register, the previous bits are shifted to the right while the oldest bit in the register is removed and that is how the memory is constantly updated with the newest [3, 4].

There are many ways to implement encoding and decoding of convolutional codes and some of the most used are: convolutional codes with sliding parity bit calculation, convolutional codes as state machines and convolutional codes with trellis structures [5].

The simplest implementation of convolutional codes is that with a sliding window. The sliding window is used as a memory with length K, and this parameter is also known as constraint length. The memory window is used to select which message bits may participate in the parity calculations, which is done using the operation XOR. Before the calculation of the next parity bits, the window is moved to the right for one place. In such a case, larger values of K provide better error-correcting capabilities and greater redundancy [5].

For more complex error-correcting codes, the sliding window is not recommended, and therefore the convolutional codes are usually represented with state machines or trellis structures. As its name says, the state machine is a finite automata which contains states and has a starting state. In this case, the constraint length K is used to calculate the number of states: 2^{K-1}. For each state (including the starting one) it is defined which is the next state if the value is 0 or if the value is 1. Also, for each input bit, there is an output of X (depending on the code rate) bits for each of the next states. For this paper, we made experiments for code rate 1/4, so that when the

state machine moves from one state to another, the output is four bits [6].

The most sophisticated way to explain the convolutional codes is with trellis structures. Every convolutional code that can be presented with a state machine, it can be presented with a trellis structure as well. In our experiments, we use implementation with trellis structure due to their compatibility with Matlab. Transmitted bits track a unique, single path of branches through the trellis. So, the output of the whole trellis is just one path. Moving within the trellis structure is more or less the same as in state machines. The encoding starts in the starting state, checks the current bit on the input, chooses the right path (0 or 1) and gives the suitable four output bits for that current input bit. The procedure ends when the last bit from the input is being encoded [5].

During the years, many algorithms were developed for decoding convolutional codes. The most frequently used decoding algorithm for convolutional codes is the Viterbi algorithm. This algorithm was developed by Andrew Viterbi in 1967 and today is still the most suitable algorithm for decoding convolutional codes. Viterbi algorithm performs the decoding by finding the most likely path through the trellis. The trellis is the same as for encoding, but the procedure is different. There are two decoding techniques of Viterbi algorithm: Hard Decision and Soft Decision. In our experiments, we are using the Hard Decision Algorithm. Viterbi uses the Hamming distance in order to calculate the most likely path. The Hamming distance gives the difference between the value in the trellis and the value that needs to be decoded. Each branch is labeled with a metric and in this case the metric is the Hamming distance which is calculated separately for every branch. Except for the starting state, all other states have a unique predecessor state. The path in the trellis that has the least number of different bits (different bits from the input bits in the trellis structure), is used for decoding of the current two output bits. Once a valid path is selected as the correct path, the decoder can recover the input data bits from the most likely output code bits [1].

4. Random Codes Based on Quasigroups

RCBQs are designed using algorithms for encryption and decryption from the implementation of TASC (Totally Asynchronous Stream Ciphers) by quasigroup string transformation [9]. These cryptographic algorithms use the alphabet Q and a quasigroup operation * on Q together with its parastrophe /.

- ***Encoding process***

Standard coding algorithm is first algorithm for RCBQs proposed in [7]. In this algorithm, first the message $M = m_1m_2 \dots m_l$ (of $N_{block} = 4l$ bits where $m_i \in Q$ and Q is an alphabet of 4-bit symbols (nibbles)) is extended to a message $L=L^{(1)}\, L^{(2)} \dots\, L^{(s)} = L_1L_2 \dots L_m$ by adding redundant zero symbols. The produced message L has $N = 4m$ bits ($m = rs$), where $L^{(i)}$ are sub-blocks of r symbols from Q and $L_i \in Q$. In this way we obtain (N_{block}, N) code with rate $R= N_{block} /N$. The codeword is produced by applying the encryption algorithm of TASC (given in Fig. 1) on the message L. For this aim, a key $k=k_1k_2 \dots k_n \in Q^n$ should be chosen. The obtained codeword of M is $C=C_1C_2 \dots C_m$, where $C_i \in Q$.

In Cut-Decoding algorithm [8], instead of using a (N_{block}, N) code with rate R, we use together two (N_{block}, $N/2$) codes with rate $2R$, that encode/decode the same message of N_{block} bits. So, for coding we apply the encryption algorithm, given in Fig. 1, on the same redundant message L twice using different parameters (different keys or quasigroups). The codeword of the message is obtained with concatenation of the two codewords of $N/2$ bits.

- ***Decoding process***

In all decoding algorithms of RCBQs, after transmission through a noisy channel, the received message D is divide in s blocks of r nibbles. Then we choose an integer B_{max} (assumed maximum number of bit errors that occur in a block during transmission) and in each iteration we generate the sets $H_i = \{\alpha \mid \alpha \in Q^r\ ,\ H(D^{(i)}, \alpha) \le B_{max}\}$, for $i = 1, 2, \dots, s$ and the decoding candidate sets $S_0, S_1, S_2, \dots, S_s$. These sets are defined iteratively and $S_0 = (k_1k_2 \dots k_n; \lambda)$, where λ is the empty sequence. In the i-th interaction, we form set S_i of all pairs (δ, $w_1w_2 \dots w_{4 \cdot r \cdot i}$) obtained by using the sets S_{i-1} and H_i as follows (w_j are bits). For each (β, $w_1w_2 \dots w_{4 \cdot r \cdot (i-1)}$) $\in S_{i-1}$ and each element $\alpha \in H_i$, we apply the decryption algorithm given in Fig. 1 with input (α, β). If the output is the pair (γ, δ) and if the sequences γ has the redundant zeros in the right positions (according to the chosen pattern), then the pair (δ, $w_1w_2 \dots w_{4 \cdot r \cdot (i-1)}c_1c_2 \dots c_{4r}$) $\equiv$ (δ, $w_1w_2 \dots w_{4 \cdot r \cdot i}$) is an element of S_i.

Encryption	**Decryption**
Input: Key $k = k_1k_2 \dots k_n$ and message $L = L_1L_2 \dots L_m$ **Output**: message (codeword) $C = C_1C_2 \dots C_m$	**Input**: The pair ($a_1\, a_2 \dots\, a_s, k_1k_2 \dots k_n$) **Output**: The pair ($c_1\, c_2 \dots\, c_s, K_1K_2 \dots K_n$)
For $j = 1$ to m $X \leftarrow L_j$; $T \leftarrow 0$; For $i = 1$ to n $X \leftarrow k_i * X$; $T \leftarrow T \oplus X$; $k_i \leftarrow X$; $k_n \leftarrow T$; **Output**: $C_j \leftarrow X$	For $i = 1$ to n $K_i \leftarrow k_i$; For $j = 0$ to $s - 1$ $X, T \leftarrow a_{j+1}$; $temp \leftarrow K_n$; For $i = n$ down to 2 $X \leftarrow temp \setminus X$; $T \leftarrow T \oplus X$; $temp \leftarrow K_{i-1}$; $K_{i-1} \leftarrow X$; $X \leftarrow temp \setminus X$; $K_n \leftarrow T$; $c_{j+1} \leftarrow X$; **Output**:($c_1c_2 \dots\, c_s, K_1K_2 \dots K_n$)

Fig. 1 TASC algorithm for encryption and decryption.

In Cut-Decoding algorithm, after transmitting through a noisy channel, we divide the outgoing message $D = D^{(1)}D^{(2)} \dots D^{(s)}$ in two messages D^1 and D^2 with equal lengths, and we decode them parallel with the corresponding parameters. In this decoding algorithm, in each iteration, we reduce the number of elements in the decoding candidate sets in the following way. Let $S_i^{(1)}$ and $S_i^{(2)}$ be the decoding candidate sets obtained in the i^{th} iteration of two parallel decoding processes, $i = 1, \dots, s/2$. Before the next iteration, we eliminate from $S_i^{(1)}$ all elements whose second part does not match with the second part of an element in $S_i^{(2)}$, and vice versa. In the $(i+1)^{th}$ iteration, the both processes use the corresponding reduced sets. In this way, the size of the lists (decoding candidate sets) becomes smaller.

After the last iteration, if the reduced sets have only one element with the same second component, then this component is the decoded message. In this case, we say that we have a successful decoding. If the decoded message is not the correct one, then we have an undetected-error. If the reduced sets have more than one element after the last iteration, we have more-candidate-error. In this case we randomly select a message from the reduced sets in the last iteration, and we take this message as the decoded message. If in some iteration all decoding candidate sets are empty, then the process will finish (we say that a null-error appears). But, if we obtain one nonempty decoding candidate set in an iteration, then the decoding continues with the nonempty set.

In [10] and [11] authors have proposed methods for decreasing the number of null and more-candidate errors by backtracking. In the experiments for this paper, we use the following combination of these two methods with backtracking. If the decoding ends with null-error, then the last two iterations are cancelled and the first of them is reprocessed with B_{max}+ 2 (the next iterations use the previous value of B_{max}). If the decoding ends with

more-candidate-error, then the last two iterations of the decoding process are cancelled, and the penultimate iteration is reprocessed with $B_{max} - 1$. In the decoding of a message, we use at most one backtracking for null-error and at most one backtracking for more-candidate-error.

3. Experimental Results and Analysis

In this section we will compare the experimental results for bit-error probabilities obtained using a convolutional code and RCBQ when the messages are transmitted through a binary symmetric channel. For both codes, experiments are made for code rate ¼.

Experiments for RCBQ with Cut-Decoding algorithm are made for code (72, 288) using the following code parameters:

- redundancy pattern: 1100 1110 1100 1100 1110 1100 1100 1100 0000
- two different keys: $k_1 = 01234$ and $k_2 = 56789$
- quasigroup of order 16 on the set Q of nibbles given in [12]

Experiments for convolutional codes are made using trellis structure and following parameters:

- data bits $k = 1$, for each incoming bit, there are n bits
- n-bit codewords, $n = 4$, each bit is coded with 4 bits
- constraint length $K = 3$, number of states is 4 (2^{K-1})

In all experiments the input is a randomly generated list with 100000 bits (the source messages) and each experiment is performed with the following steps:

1. The input is read from a text file (input.txt). The input is encoded with a suitable code (convolutional or RCBQ) and the encoded messages are written in another file (coded value.txt).
2. The encoded bits are read from the text file coded value.txt and sent through a BSC. The output of the channel is written in another file (value from channel.txt).
3. Then the output from the channel, file channel.txt, is decoded with a corresponding code (convolutional or RCBQ) and the decoded messages are written in another file called decoded value.txt.
4. After the decoding process, decoded bits from the file decoded value.txt are compared with the bits in the input file input.txt, and we calculate the bit-error-probability for the experiment.

The binary symmetric channel can have different probability for incorrectly transmitted bits. Here, we will present and analyze results for BSC with the following six different probabilities: 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08 and 0.09. Experimental results for bit-error probabilities (BER) for both considered codes are given in Table 2. In this table, *BER_rcbq* denotes the bit-error probability for RCBQ and *BER_c* the bit-error probability obtained by using convolutional code. Also, the results for BER are graphically presented on Fig. 2.

Table 1: *Bit-error probabilities for convolutional codes and RCBQ*

p	*BER_rcbq*	*BER_c*
0.02	0.00001	0.0033
0.03	0.00024	0.0054
0.04	0.00142	0.0081
0.05	0.00507	0.0119
0.06	0.00869	0.0165
0.07	0.02017	0.0215
0.08	0.03459	0.0270
0.09	0.05378	0.0332

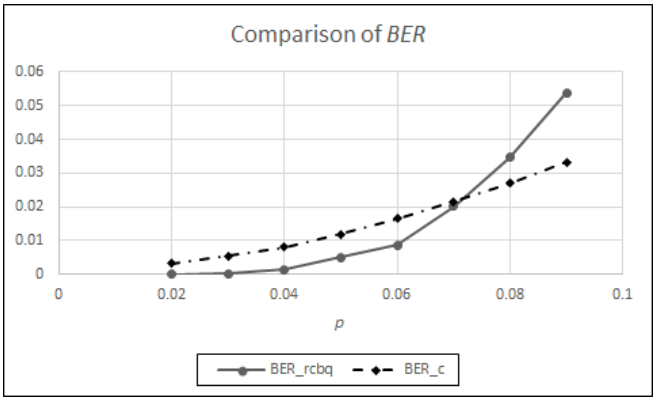

Fig. 2 *Comparison of BER*

From the results in Table 1 and Fig. 1, we can conclude that both codes have good performances in correction of errors in the binary-symmetric channel with the considered values of p. Namely, in each row, experimental results for *BER* (*BER_rcbq* and *BER_c*) are smaller than the bit-error probabilities in the channel.

Comparing the results obtained with RCBQ and the convolutional code, we can see that for smaller values of bit-error probabilities ($p < 0.08$) in the channel with RCBQs we obtain better results for *BER*. Nevertheless, in all experiments, the time efficiency of the convolutional codes is better than that of RCBQ. On the other hand, the advantage of the RCBQ is that they have cryptographic properties. Namely, if the data are encoded with RCBQs, then the recipient can decode the message only if s/he knows exactly which parameters (redundancy pattern, keys and quasigroup) are used in the encoding/decoding process, even if the channel is noiseless.

4. Conclusion

Error-correcting codes have an important role in communications and transmissions of data. Different types of codes have been developed over the years, but their usage remains still the same.

The different implementations for convolutional codes such as sliding window, state machines and trellis structures, show one more time that they are widely extended. Also, their compatibility and flexibility in Matlab are another proof that they can be designed in many ways. Convolutional codes are the basics for understanding turbo codes, which are parallel implementation for convolutional codes and are widely used everywhere.

On the other hand, Random Codes based on Quasigroups are relatively new codes, which have cryptographic properties which are very important in data transmission nowadays. These codes enable the correction of erroneously transmitted bits in a noisy channel and security of transmitted messages, using a single algorithm. Also, from the experimental result presented in this paper we can conclude that for lower bit-error probabilities in BSC, RCBQs have better performances in correction of errors than convolutional codes.

At the end we must note that there is no right or wrong code since each code is better in different situations. In order to determine which code is most appropriate for a particular purpose, all circumstances must be taken into consideration. RCBQs have some cryptographic properties, but the convolutional codes are faster.

5. References

[1] S. Mukherjee. Morgan Kaufmann. *Architecture design for soft errors*. (2011)
[2] L. Wang, G.W. Wornell, L. Zheng. IEEE Transactions on Information Theory. *Fundamental limits of communication with low probability of detection*. ***62*(6)** 3493-3503. (2016)
[3] C. Beard, S. William. Pearson. *Wireless communication networks and systems*. (2015).
[4] A. Bensky. Newnes. *Short-range wireless communication*. (2019)
[5] https://www.cs.princeton.edu/courses/archive/spring18/cos463/lectures/L09-viterbi.pdf (27.05.2021)
[6] T. Richardson, U. Ruediger. Cambridge university press. *Modern coding theory*. (2008)
[7] D. Gligoroski, S. Markovski, Lj. Kocarev, Error-correcting codes based on quasigroups, in Procedings of 16th Intern. Confer. Computer Communications and Networks, ICCCN 2007, Honolulu, pp. 165-172. (2007)
[8] A. Popovska-Mitrovikj, S. Markovski, V. Bakeva, Increasing the decoding speed of random codes based on quasigroups'. in: S. Markovski, M. Gusev (Eds.), ICT Innovations 2012, Web proceedings, ISSN 1857-7288, 2012, pp. 93-102. (2012)
[9] D. Gligoroski, S. Markovski, Lj. Kocarev, Totally asynchronous stream ciphers + Redundancy = Cryptcoding. in: Aissi, S., Arabnia, H.R. (eds.), Proceedings of the International Conference on Security and management, SAM 2007, CSREA Press, Las Vegas, pp. 446-451. (2007)
[10] A. Popovska-Mitrovikj, S. Markovski, V. Bakeva, Performances of error-correcting codes based on quasigroups', in Gómez, D., Davcev, J.M (Ed.) ICT innovations 2009, Springer, Berlin, Heidelberg, pp. 377–389 (2010)
[11] A. Popovska-Mitrovikj, S. Markovski,V. Bakeva, Increasing the decoding speed of random codes based on quasigroups, in Proceedings of ICT innovations 2012, pp. 93–102. (2012)
[12] A. Popovska-Mitrovikj, S. Markovski,V. Bakeva, Performances of error-correcting codes based on quasigroups. in: Davcev, D., Gomez, J.M. (eds.) ICT-Innovations 2009, pp. 377-389. Springer (2009)