NONLINEAR TURBO CODES BASED ON QUASIGROUP STRING PROCESSING

Dejan Spasov

Faculty of Computer Science and Engineering

Skopje, Macedonia

ABSTRACT

Use of quasigroup transformations to build error-correcting codes was first proposed by Gligoroski, Markovski, and Kocarev [1]. However, their decoding algorithm was based on the exponential-time minimum-distance decoding algorithm, in which the brute-force search for the error pattern is confined in blocks of 16 bits and upper-bounded to two or three errors per block. We point out that any quasigroup-based error-correcting code, designed so far, can be modeled as finite state machine, thus it can be decoded in polynomial time with the well-known Viterbi algorithm or the MAP decoding algorithm. In order to improve the error-correcting capability of quasigroup codes, in this paper we build an error-correcting system based on the Turbo-code design principles. We present the error-correcting capabilities of our Turbo-code system over the Gaussian channel.

I. INTRODUCTION

Let Q be a finite alphabet of $|Q|$ letters and let Q^n be the set of all strings of length n over Q. A *code* C is a subset of Q^n of M elements. Elements of the code $c_i \in C$ are called *codewords.*

Let $d(x,y)$ denote the *Hamminng distance* between two strings $x, y \in Q^n$, i.e. $d(x,y) \in \mathrm{N}$ is the number of positions in which x and y differ. We say that C is an (n,M,d) code,

$$d = \min\{d(c_i, c_j) \mid c_i, c_j \in C, i \neq j\} \quad (1)$$

where d is the so-called *(minimum) distance* of the code C.

Some codes are gifted with efficient (polynomial) procedure to find the nearest codeword $\hat{c} \in C$ (in terms of Hamming distance) for any string $x \in Q^n$, namely

$$\hat{c} = \arg\min\{d(x, c_i) \mid c_i \in C\}. \quad (2)$$

The process of finding the nearest codeword is known as *decoding*. Codes with polynomial decoding procedures can be used for transmission of digital information over a noisy channel. A *k*-letter message $m \in Q^k, k < n,$ is one-to-one encoded into an *n*-letter codeword $c_m \in C$ and sent over a noisy channel. The channel can arbitrarily change few letters, but the receiver may still be able to deduce which codeword has been sent, thus recovering the original message m.

II. CONVOLUTIONAL CODES

Convolutional codes are one of the oldest known classes of error-correcting codes that were introduced by P. Elias in 1955 [2]. Given a binary operation $*$ and a *k*-letter message $M \in Q^k$, the idea is to produce an *n*-letter codeword $c_M \in C$ by pseudo-randomly applying the binary operation $*$ over the letters of the message M Thus a convolutional encoder is a circuit made of two parts: a shift register that stores the message m and a combinatorial logic which is commonly implemented with XOR gates (Fig. 1).

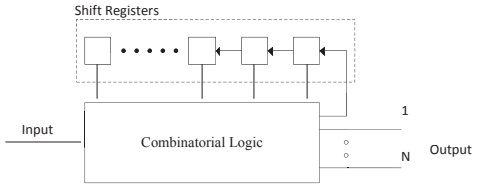

Figure 1: Diagram of convolutional encoder.

For information sequence M of k letters, a convolutional encoder with $l, l \leq k,$ shift registers produces N output sequences of at most $k+l$ letters each, thus producing a code with rate approximately equal to $1/N$. Further increase of the code rate is possible by puncturing out some of the output letters from the encoder's output. The i-th letter in each of the N output sequences is obtained by combining the i-th letter from the information sequence M and the content of the shift register. Fig. 2 is an example of a rate ½ convolutional encoder with four memory registers. The combinatorial logic of the encoder is described with the generators of the convolutional code, expressed as octal numbers. The generators of the convolutional code on fig. 2 are $g_1 = 31$ for the upper output and $g_2 = 21$ for the lower out.

There are two main categories of convolutional encoders: *recursive* and *non-recursive* convolutional encoders. The convolutional encoder on fig. 2 is an example of non-recursive convolutional encoder. A convolutional encoder is

non-recursive if for the i-th letter in each of the N output sequences the content of the memory registers depends only on the i-th letter and the previous $i-l$ letters of the input sequence m. A convolutional encoder is recursive if for the i-th letter in each of the N output sequences the content of the memory registers depends on all i letters of the input sequence m. Thus recursive encoders are circuits with infinite impulse response. We can make recursive encoder from a non-recursive on (like the (37,21)) by making a feedback with one of the outputs (Fig. 3).

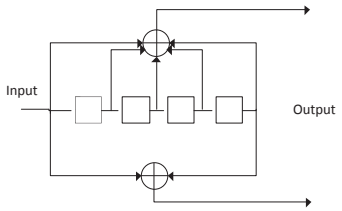

Figure 2: The (37,21) convolutional encoder

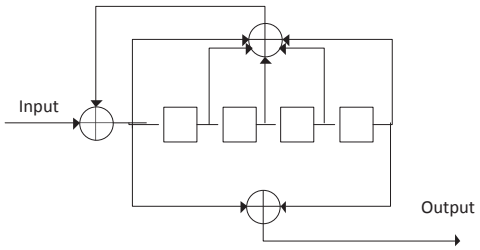

Figure 3: Design of a recursive encoder from the (37,21) non-recursive encoder

We say that the encoder is *systematic* if one of the N output sequences is identical to the input information sequence.

A. Trellis Diagram and decoding of convolutional codes

A convolutional encoder with l registers is finite state machine with 2^l states. *Trellis diagram* is labelled k-partite graph, in which every path represents a valid codeword. The trellis diagram on fig. 4 corresponds to the recursive encoder on fig. 3. Vertices of the k disjoint sets in the trellis represent all possible 2^l states of the encoder. On fig. 4 vertices are labelled as decimal numbers, such that the content of the leftmost register corresponds to the most significant bit in the decimal number. Edge labels represent the input letters to the encoder and the appropriate output letters produced by the encoder separated by the slash symbol.

Trellis diagram of convolutional codes gives a hint about the decoding process; if the received sequence does not represent a valid path through the trellis diagram, then we can conclude that errors have occurred. The decoding objective is to find the most probable valid path though the trellis. Several decoding algorithms exist for decoding convolutional codes. The most famous ones are the Viterbi algorithm [3],[4] and the BCJR algorithm [5]. The Viterbi algorithm is universally used and is highly parallelizable.

$$\mathrm{LLR}(x_k) = \log \frac{\Pr\{x_k = 1 / observation\}}{\Pr\{x_k = 0 / observation\}} \qquad (3)$$

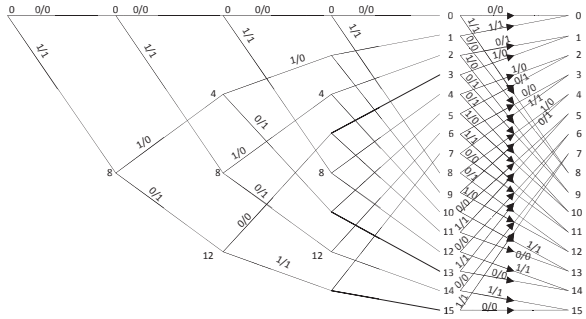

Figure 4: Trellis diagram of a convolutional code

B. Turbo codes

Invented by C. Berrou, A. Glavieux, and P. Thitimajshima [6], turbo codes were the first practical system that achieved signal-to-noise ratio of 0.7 dB above the Sahnnon's limit to provide bit error probability of 10^{-5}. The turbo encoder is a special type of convolutional encoder that uses two or more recursive encoders connected via parallel concatenation scheme (fig. 5). It is obvious from figure 5 that turbo codes are systematic, i.e. the input sequence appears as output sequence x. Turbo encoder on figure 5 has a code rate of 1/3, but higher coding rates can be achieved by puncturing out parts of the outputs from the recursive encoders y_1 and y_2. The purpose of the interleaver is to provide random permutation of the input sequence, thus allowing identical recursive encoders to be used in the design.

Since a turbo code is made of two parallel systematic recursive convolutional codes, the decoding process involves separate decoding of each of the systematic recursive convolutional codes (Fig. 6).

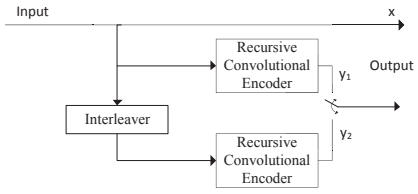

Figure 5: Turbo encoder

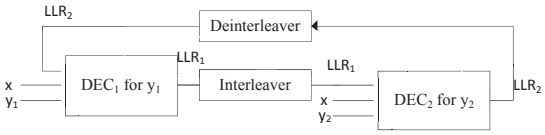

Figure 6: Turbo decoder

DEC_1 and DEC_2 are modified BCJR decoders [5] that output the logarithm of likelihood ratio (LLR) for each bit x_k

where $\Pr\{x_k = 1/observation\}$ is the a posteriori probability of the bit x_k. DEC_1 outputs LLR quantities for each bit x_k. These quantities are fed to DEC_2 to produce its own estimates LLR_2. The turbo decoding process is iterative, and in the next iteration LLR_2 quantities are fed into DEC_1.

III. Codes Derived from Quasigroup Transformations

A *quasigroup* is an alphabet Q endowed with binary operation $*: Q \times Q \to Q$, such that for all $u, v \in Q$ there exist unique $x, y \in Q$ such that

$$\begin{cases} u * x = v \\ y * u = v \end{cases}.$$

From engineering perspective, quasigroup operations can be seen as circuit with two inputs and one output (Fig. 7 a)). The circuit works the following way: any random string ($\in Q^n$) that enters on the input 1 will change its representation according to the quasigroup table (Fig. 7 b)) and the values on the input 2.

Given a string $S = (s_1, \ldots, s_n)$ over the alphabet Q and random letter $a \in Q$; E-transformation is the string $E = (e_1, \ldots, e_n)$ such that [1]

$$\begin{cases} e_1 \overset{def}{=} a * s_1 \\ e_i \overset{def}{=} s_i * e_{i-1} \end{cases}.$$

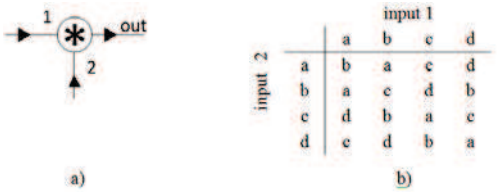

Figure 7: Circuit representation of quasigroup operations

The size of the alphabet Q is called *order of the quasigroup*. In this paper we will work with quasigroups of order 4. We will use these quasigroups over binary strings in such a way that each pair of adjacent binary symbols from the string enter in parallel in the quasigroup circuit as one letter from the alphabet of size 4 (Fig. 8 a)). Each two-bit combination is mapped to a letter from the alphabet of size 4 (Fig. 8 b)).

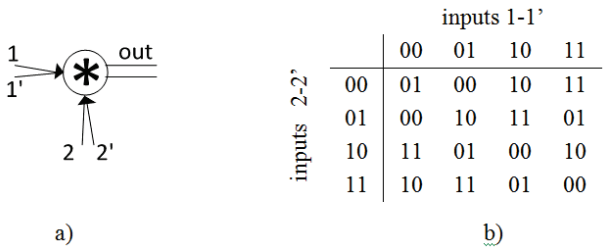

Figure 8: Quasigroup operations of order 4 over binary strings

Assuming existence of a circuit that performs quasigroup operations (Fig. 7), the E-transformation circuit will need additional memory register to store the previous letter (Fig. 9 a)).

Given a string $S = (s_1, \ldots, s_n)$ over the alphabet Q and random letter $a \in Q$; D-transformation is the string $D = (d_1, \ldots, d_n)$ such that [1]

$$\begin{cases} d_1 \overset{def}{=} a * s_1 \\ d_i \overset{def}{=} s_i * s_{i-1} \end{cases}.$$

Circuit representation of the D-transformation is given on Fig. 9 b).

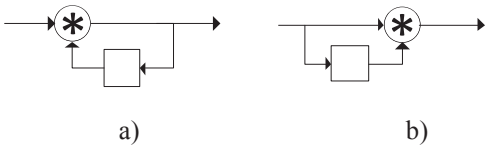

Figure 9: Circuit representation of E-transformation and D- transformation

We can combine E and D transformations to build more complex transformations. For example, four E-transformation circuits can be connected in cascade line (Fig. 10).

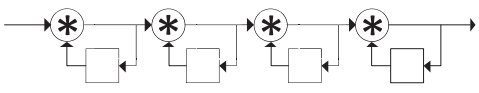

Figure 10: Complex quasigroup-based transformation

Let $M = (m_1, \ldots, m_k)$ be a message that we wish to encode, i.e. a string over Q. Let v, $v \notin Q$, be a special letter, i.e. a blank space, and let $V = (v, \ldots, v)$ is a string of $n-k$ blank spaces. Let the string I consists of all letters from M and V arranged in pseudo-random fashion. For example,

$$I = (m_1, m_2, v, v, m_3, m_4, v, v \ldots, m_{k-1}, m_k, v, v). \quad (4)$$

Then the codeword $C=(c_1,\ldots,c_n)$ that corresponds to the message M is obtained from a complex quasigroup-based transformation (as in Fig. 10) over the string I.

Let $M'=(m_1,\ldots,m_k')$ and $M''=(m_1,\ldots,m_k'')$ are two messages that differ in the last letter. Then it is obvious that regardless of the complex quasigroup transformation, the two codewords C' and C'' will differ in at most 3 positions. Hence, the construction (4) cannot be better than Hamming codes. Intuitively, we expect that the string $I=(m_1,m_2,m_3,m_4,v,v,v,v\ldots,m_{k-3},m_{k-2},m_{k-1},m_k,v,v,v,v)$ may produce a better result than (4) and the best results we expect to happen with

$$I=(m_1,m_2,m_3,m_4,\ldots,m_{k-1},m_k,v,v,\ldots,v) \qquad (5)$$

Thus in this paper we seek to develop codes that are systematic in nature.

IV. Turbo codes based on quasigroup transformations

We have developed an error-correcting system based on turbo codes in which XOR operations are replaced with quasigroup operations. The new turbo encoder looks like the old turbo encoder (Fig. 5), but employs modified recursive encoders based on quasigroup operations (Fig. 11).

Each memory cell can store one letter from the alphabet Q; thus an encoder with l memory cells is a finite state machine with $l^{|Q|}$ states. We use an encoder with $l=2$ memory cells that operates with quasigroups of order 4. The resulting encoder will have 16 states thus making it comparable with the original turbo encoder designed by Berrou, Glavieux, and Thitimajshima [6]. S ince we wanted to work with binary strings, we had to adapt the encoder from figure 11 work with binary sequences (Fig. 12).

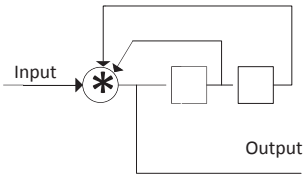

Figure 11: Recursive convolutional encoder based on quasigroup transformation

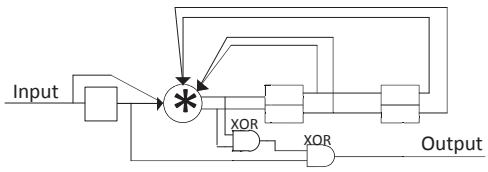

Figure 12: Recursive convolutional encoder based on quasigroup transformation

	a	b	c	d
a	b	a	c	d
b	a	c	d	b
c	d	b	a	c
d	c	d	b	a

non-fractal

	a	b	c	d
a	a	b	c	d
b	b	a	d	c
c	c	d	a	b
d	d	c	b	a

fractal

Figure 13: Non-fractal and fractal quasigroups of order 4

Quasigroups of order 4 are studied in detail in [7] and divided into two classes: fractal and non-fractal. It has been hypothesized that non-fractal quasigroups are suited for practical use in coding theory. Our encoder (Fig.12) uses non-fractal quasigroup (Fig.13). If we use the fractal quasigroup, the trellis diagram of the encoder will be irregular and the error rate (BER) at the decoder will increase. However, this fact does not exclude the use of fractal quasigroups in designing turbo encoders.

The turbo decoder in this paper is based on the turbo decoder published in [8] with appropriate modifications to accommodate the new trellis diagram of a quasigroup-based turbo encoder.

Fig. 14 shows simulation results of our turbo-code-like system for messages of length $N=1024$ bits over AWGN channel, while figures 15 and 16 compare our system with the original turbo-code system [6].

IV. Conclusion

In [9] we have asked about feasible error-correcting systems based on quasigroup transformations. Here we presented one possible solution to this problem. Moreover, we have shown that any quasigroup-based error-correcting system developed so far can be decoded in polynomial time. In order to achieve this goal, we have used systematic recursive codes that are connected in turbo-code scheme. We have tested our system over AWGN channel and compared the results with the performance of the original turbo-code system [6]. From figures 15 and 16 we see that our system outperforms the original turbo-code system for small block lengths.

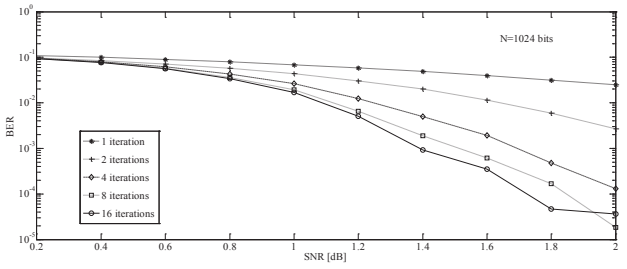

Figure 14: Bit Error Rate performance of our turbo system over AWGN channel

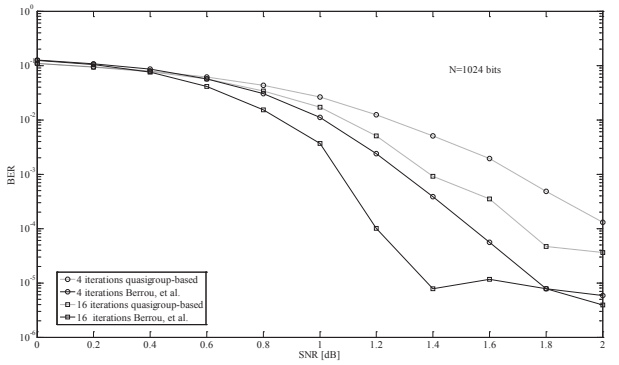

Figure 15: Comparison of our turbo system with the original turbo system for long block lengths

Several questions remain open. We have not provided a design of a quasigroup circuit (Fig. 7 and 8). In this context, we ask if fractal quasigroups can be used to reduce the design complexity and if turbo-code systems can be built using fractal quasigroups. However, the most important open problem is to investigate the performance of quasigroup-based turbo-codes over higher alphabets. We believe that larger alphabets are more natural to this class of error-correcting codes and that quasigroups may particularly be useful over alphabets that cannot have field structure.

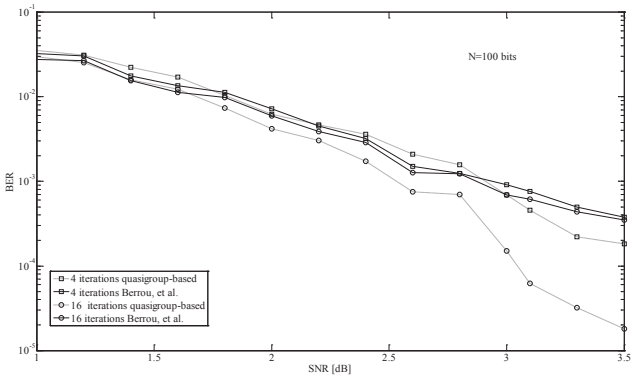

Figure 16: Comparison of our turbo system with the original turbo system for short block lengths

REFERENCES

[1] D. Gligoroski, D., Markovski, S., Kocarev, L. "Error-Correcting Codes Based on Quasigroups." *Proceedings of 16th International Conference on Computer Communication and Networks*, pp. 16-172, 2007.

[2] Elias, P. "Coding for noisy Channels." *IRE Convention Record*, Part IV, pp. 37-46 (1955).

[3] Viterbi, A. J. "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm." *IEEE Transactions on Information Theory* vol. 13, no. 2, pp. 260–269, 1967.

[4] Forney, G. D. "The Viterbi Algorithm." *IEEE Transactions on Information Theory* vol. 61, no. 3, pp. 268–278, 1973.

[5] Bahl, L., Cocke, J., Jelinek, F., Raviv, J. "Optimal Decoding of Linear Codes for minimizing symbol error rate." *IEEE Transactions on Information Theory*, vol. 20, no. 2, pp.284-287, 1974.

[6] Berrou, L., Cocke, J., Jelinek, F., Raviv, J. "Optimal Decoding of Linear Codes for minimizing symbol error rate." *IEEE Transactions on Information Theory*, vol. 20, no. 2, pp.284-287, 1974.

[7] Dimitrova, V. *Quasigroup processed arrays, their Boolean presentation and application in cryptography and coding*. PhD Thesis, University Ss. Cyril and Methodius, Skopje, 2010.

[8] Spasov, D., Maggio, G. M., Kocarev, L. " A Practical Algorithm for turbo decoding Enhancement." *ISCAS 2004* vol. 4, pp.621-624, 2004.

[9] Spasov, D. "Open Problems in Designing Convolutional Codes Based on Quasigroups." *CIIT 2011* vol. 8, pp.22-24, Bitola, 2011.