Decoding of Block Codes by using Genetic Algorithms and Permutations Set

Saïd Nouh[1], Idriss Chana[1] and Mostafa Belkasmi[1]

[1] Department of Communication Networks, National School of Computer Science and Systems Analysis (ENSIAS), Mohammed V-Souissi University, Rabat, Morocco
nouh_ensias@yahoo.fr, idrisschana@gmail.com, m.belkasmi@um5s.net.ma

***Abstract*:** Recently Genetic algorithms are successfully used for decoding some classes of error correcting codes. For decoding a linear block code C, these genetic algorithms computes a permutation π of the code generator matrix depending of the received word. Our main contribution in this paper is to choose the permutation π from the automorphism group of C. This choice allows reducing the complexity of re-encoding in the decoding steps when C is cyclic and also to generalize the proposed genetic decoding algorithm for binary nonlinear block codes like the Kerdock codes. In this paper, an efficient stop criterion is proposed and it reduces considerably the decoding complexity of our algorithm. The simulation results of the proposed decoder, over the AWGN channel, show that it reaches the error correcting performances of its competitors. The study of the complexity shows that the proposed decoder is less complex than its competitors that are based also on genetic algorithms.

***Keywords*:** Genetic algorithms, Error correcting block codes, Cyclic codes, Automorphism group, Soft decision decoding algorithms.

1. Introduction

The telecommunication and storage systems are used, more and more, to guarantee transmission and memorization of several types of data such as texts, pictures, videos and voice. The reliability of these data, notably on channels that are subject to noise, represents a big worry for users. An efficient solution of this problem is based on the detection and the correction of errors, by the channel coding technique. In effect this technique consists in adding redundancy in data to protect, by using error correcting codes which enable reconstruction of the original data.

The decoding of error correcting codes is in general a NP-Hard problem; the quality of a decoder depends on the industrial requests. In certain cases, the main criterion is the realization of good performances in terms of the BER (bit error rate), independently on the temporal complexity. In other cases, some errors are tolerated, but the reduction of the temporal complexity is imposed.

There are two classes of error correcting codes: convolutional codes and block codes. The class of block codes contains two subclasses: nonlinear codes and linear codes. The principle of encoding by using a block code C(n,k) is as follows: the initial message is cut out into blocks of length k. The length of the redundancy is n-k, thus the length of transmitted blocks is n.

There are two categories of decoding algorithms: Hard decision and Soft decision algorithms. Hard decision algorithms work on the binary form of the received information. In contrast, soft decision algorithms work directly on the received symbols [1]. The decoding category depends on the industrial requests and the communication channel. When the channel allows measuring the reliabilities (float symbols) of the sequence r to decode, the soft decision decoders working on these reliabilities allow to win generally about 2 dB more than the hard decision decoders working on the binary form of r can do.

Many decoding algorithms benefit from the proportionality between the symbols reliabilities and the probability that these later are correct [2-7]. For the binary channels, the new majority voting procedure introduced by Nouh et al [8] allows to compute these measures and to enhance the decoder performane. The decoder of Chase [4] tries to find some errors in the least reliable symbols in a first step. Then it uses an auxiliary hard decision decoder like the BMA algorithm of Berlekamp-Massey [9] or the permutation decoding algorithm of MacWilliams [10] for finding the complementary error.

Recently Genetic algorithms are successfully used for decoding linear block codes [2,5,11,12]. For decoding a linear code C^1 of generator matrix G^1, for non binary channels, the decoder of Maini et al [2] starts by finding a generator matrix G^2 of another linear code C^2 equivalent to C^1 and the permutation π which binds these two codes. G^2 is obtained by applying the Gauss operations on G^1 after sorting the received sequence r by reliabilities. This treatment allows reducing the number of errors in the information part of the permuted sequence $\pi(r)$. In a second step, these decoding algorithms try to find the closest codeword to $\pi(r)$ by re-encoding a certain number of information vectors. The second step requires encoding by a generator matrix, which is more complex when comparing with encoding by the polynomial generator, when the code C^1 is cyclic, and isn't available when this code is nonlinear. The SDGA algorithm of Azouaoui et al [5] applies the Chasing technique over information set decoder based on genetic algorithms. Unlike Maini decoder [2], which works on the generator matrix, the decoder DDGA of Azouaoui et al [11] works on the parity check matrix; this allows reducing the complexity of their genetic algorithm for codes of high rate. However, the complexities of both decoders are the same for the codes of rate about 0.5, like the Quadratic Residue codes (QR). The PGAD algorithm [12], uses many genetic algorithms working in parallel for decoding linear block codes.

Unlike the genetic decoding algorithms of Maini et al and the DDGA algorithm, we propose in this paper, a new genetic algorithm which uses the available encoding procedure and without necessity to pass to any equivalent code. The proposed decoder uses a small part of the automorphism

group, called a permutation set, for moving, at most, the least reliable symbols in the redundancy part. This choice allows to reduce the complexity of re-encoding when the code is cyclic, and to also generalize the proposed genetic decoding algorithm for binary nonlinear block codes like the Kerdock codes [13]. In this algorithm, a new metric is used and it reduces considerably the decoding complexity.

The automorphism group of the quadratic residue codes contains the PSL2 group given in [14]. The one of the BCH codes is discussed in the work of Berger and Charpin [15]. One of the most important works that use the part of the automorphism group to decode cyclic codes is the algorithm of Chana et al [6-7]. These algorithms use some cyclic permutations and the Chasing technique on the least reliable symbols remained in the information part of some sequences. Another work that uses this group is the famous permutation decoding algorithm of MacWilliams [10], for binary channels.

The remainder of this paper is structured as follows. In the section 2, we present the proposed genetic algorithm for decoding systematic block codes (AutDAG). In the section 3 we give some simulation results of AutDAG, and we make a comparison with other decoders. In the section 4, we present a simplification of the Maini algorithm. In the section 5, we study the complexities of some decoders. Finally, a conclusion and a possible future direction of this research are outlined in Section 6.

2. The Proposed Decoder

A genetic algorithm (GA) is a heuristic search algorithm premised on the natural selection and genetic [16-18].These algorithms are successfully used for decoding linear codes [2, 5, 11, 12, 19]. They are also used for finding the weight enumerator of linear block codes [20]. The genetic algorithms have the possibility of working in parallel; thus, they are implemented in parallel for decoding linear block codes in the work of Ahmadi et al [12]. Genetic algorithms are used in Search of good Tailbiting Codes [21], they are from the family of evolutionary algorithms which are recently applied to the synthesis of unequally spaced antenna arrays [22].

Before giving the AutDAG steps we give the following definitions:

Definition 1: We call the information reliability length of a sequence $r \in IR^n$ relating to a threshold S and we note IRL(r,S), the number of the symbols r_i having reliabilities more than S and which are in the information part of r : IRL(r,S) = cardinal ($i \in \{1,2,\ldots,k: |r_i| \geq S\}$).

In practice IRL(r,S) is computed as follows:

$$\mathrm{IRL(r,S)} = \sum_{i=1}^{k} V_i \quad , \quad \begin{cases} V_i = 1 & if \quad |r_i| \geq S \\ V_i = 0 & if \quad |r_i| < S \end{cases}$$

Definition 2: Let be $L = \{\sigma^{(1)}, \sigma^{(2)}, \sigma^{(3)}, \ldots, \sigma^{(z)}\}$ a list of z automorphisms of a block code C (a permutation set). We call a maximum reliable automorphism related to a sequence $r \in IR^n$ and a threshold S, over L and we note $\sigma_L^{[r,S]}$, the permutation $\sigma^{(i)}$ such that $\forall j \leq z : IRL(\sigma^{(i)}(r,S)) \geq IRL(\sigma^{(j)}(r,S))$

The AutDAG algorithm works as follows:

Inputs:

- A permutation set L of z automorphisms
- The sequence r to decode.
- The systematic encoding procedure Encod()
- The population size Ni
- The maximum number of generations Ng
- The crossover probability p_c
- The mutation probability p_m
- The threshold of reliability S
- The threshold of decoding T

Outputs: *the decoded word c.*

Begin

Step 1. Find a maximum reliable automorphism σ related to r and S over L
Step 2. Compute h, the hard version of the sequence r
Step 3. h←σ(h) ; r←σ(r);
Step 4. Generate an initial population, of Ni individuals; the first individual is the information part of h and the others are a binary vectors of length k uniformly generated.
Step 5. N← 1; continue ← true; c←the codeword corresponding to the information part of h.
Step 6. While(N≤ Ng and continue=true) do:
Sub Step 1. Compute the fitness of each individual which is equal to the Euclidean distance between the encoded individual, obtained by applying the function Encod(), and the sequence r.
Sub Step 2. Copy the best individual (of small fitness) in the intermediate population.
Sub Step 3. For i from 2 to Ni do:
-Randomly select two individuals: p1 and p2.
-Cross p1 and p2 to obtain children ch according to the crossover probability p_c.
-If (fitness(c)≤fitness(ch)) then mute the individual ch according to the mutation probability p_m
-Insert the individual ch in the intermediate population.
Sub Step 4. c←the best individual in the current population
Sub Step 5. if the Hamming distance Hd(c,r) between c and h is less than or equal to T then continue←false.
Step 7. c←σ^{-1}(c)

End.

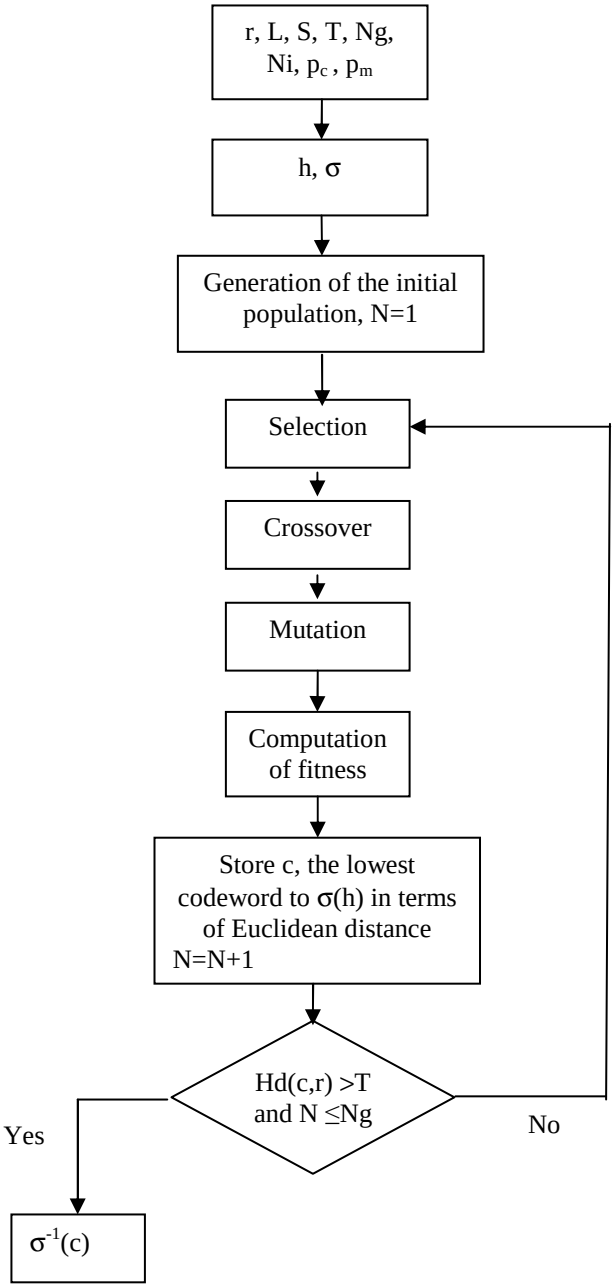

Figure1. AutDAG diagram

The figure 1 presents a diagram of AutDAG. In this genetic algorithm we propose to use the same crossover given by Maini et al [2].
All error correcting codes have an automorphism group, therefore the permutation set L exists for linear and nonlinear codes. Contrary to the Maini and DDGA algorithms, the AutDAG algorithm doesn't contains any instruction requiring the property of code linearity. Thus it can be generalized for non linear block codes like the Kerdock codes [13].

3. Simulation results

In order to show the error correcting performances of AutDAG, we do intensive simulations. With the exception of the cases where the simulation parameters are explicitly mentioned, the simulations where made with default parameters outlined in Table 1. The performances are given in terms of BER (bit error rate) as a function of the signal to noise ratio SNR $=E_b/N_0$(the energy per bit to noise power spectral density ratio). The simulation will be presented for Quadratic-Residue codes (QR), Bose, Ray-Chaudhuri and Hocquenghem codes (BCH) and the Nordstrom-Robinson code (NR).

Table 1. Default simulation parameters of AutDAG

Ni	300	T	0
Ng	50	Default code	QR(47,24)
p_c	0.97	Channel	AWGN
p_m	0.08	Modulation	BPSK
z	500	Minimum residual errors	100
S	0.7	Minimum transmitted blocks	1000

Table 2. Simulation parameters for NR and some QR codes with T=0

Code	Ni	Ng	z	T
QR(31,16)	500	100	200	0
QR(41,21)	600	100	400	0
QR(47,24)	500	100	500	0
QR(71,36)	1300	100	1000	0
NORDSTROM-ROBINSON	40	3	100	0

The figure 2 shows the AutDAG performances, without threshold of decoding, for the QR codes of length 31, 41, 47 and 71 obtained by using the simulation parameters illustrated in the table 2.

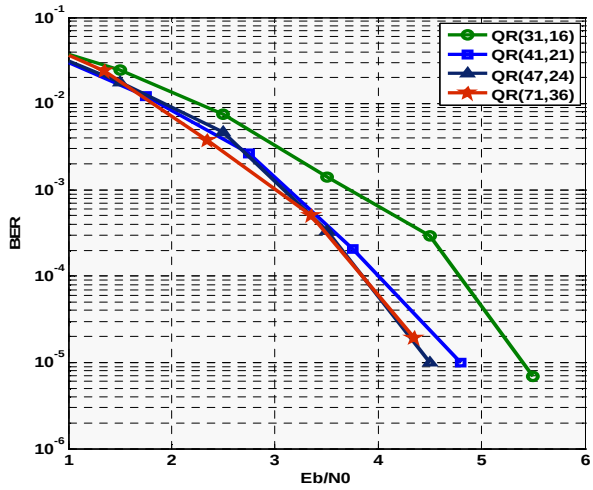

Figure 2. Impact of the code length on the AutDAG performances for T=0

The figure 3 shows the AutDAG performances, with a threshold of decoding, for the QR codes of length 31, 41, 47 and 71 obtained by using the simulation parameters illustrated in the table 3.
The figures 2 and 3 show an improvement of performances, when the code length is increased from 31 to 71.

Table 3. Simulation parameters for some QR codes with T≠0

Code	Ni	Ng	z	T
QR(41,21)	500	100	600	4
QR(47,24)	300	80	400	5
QR(71,36)	1000	100	1000	5

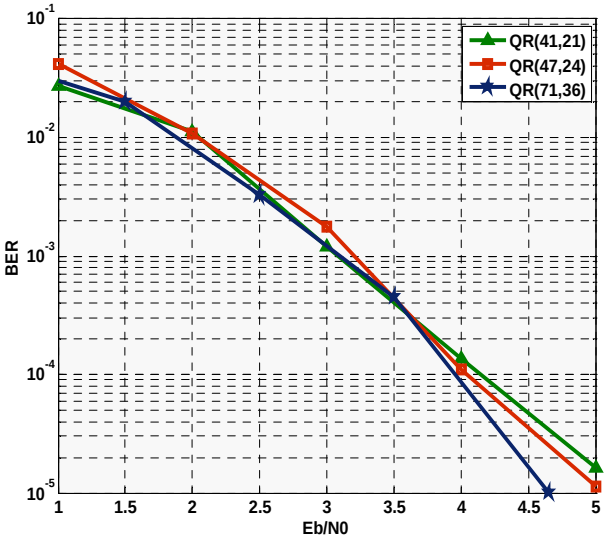

Figure 3. Impact of the code length on the AutDAG performances for T=error correcting capability

The main difference between the simulation parameters given in tables 2 and 3 is the value of the threshold of decoding T which is equal to 0 for the simulation results plotted in the figure 2 and it is equal to the error correcting capability for those given in the figure 3. As we will show in the next section, the use of a big value of T reduces considerably the time complexity of AutDAG. The comparison between the simulation results plotted in the figures 2 and 3 proves that the stop criterion given by T is efficient in the terms of performances, thus the complexity of AutDAG is reduced without touching a lot the performances.

To show the effect of the permutations number z on the error correcting performances of AutDAG applied to the QR(47,24) code, we give in the figure 4 the results obtained by varying z from 1 to 5000 . The use of 100 permutations allows to win about 1 dB at BER=3.10^{-5}. The gain becomes negligible when z passes 100.

To show the impact of the population size Ni on the error correcting performances of AutDAG applied to the QR(47,24) code, we elucidate in the figure 5 the results obtained by varying Ni from 1 to 500 . The use of 200 individuals allows to win about 4 dB at BER=3.10^{-5} comparing to populations with only ten individuals. The gain becomes negligible when Ni passes 200.

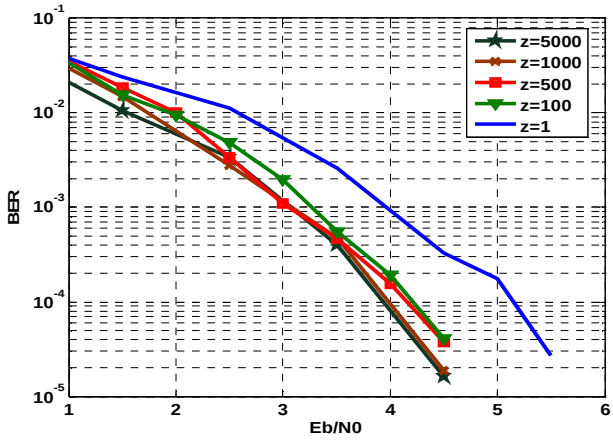

Figure 4. Impact of the permutations number on the AutDAG performances

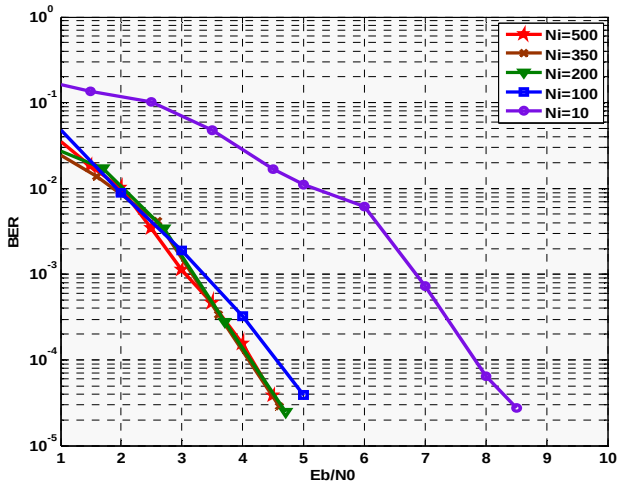

Figure 5. Impact of the population size on the AutDAG performances

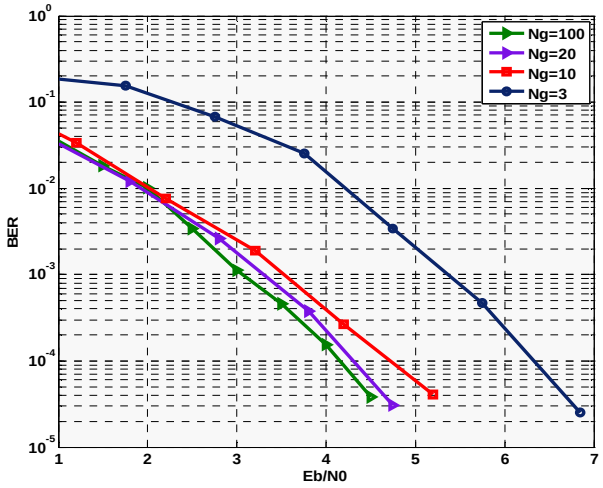

Figure 6. Impact of the number of generations on the AutDAG performances

To show the impact of the generations number Ng on the error correcting performances of AutDAG, applied to the QR(47,24) code, we illustrate in the figure 6 the results obtained by varying Ng from 1 to 500 . The use of 20 generation allows to win about 2 dB at BER=3.10^{-5} comparing to AutDAG with only 3 generations. The gain becomes negligible when Ng passes 20.

To show the impact of the crossover probability p_c on the error correcting performances of AutDAG, we illustrate in the figure 7 the simulation results obtained by varying pc from 0.05 to 0.97. This figure demonstrates that small values of pc damage the performances and their big values improve them. At BER=2.10^{-5} the good choice of pc allows to win about 1dB.

To show the impact of the crossover type on the error correcting performances of AutDAG we illustrate in the figure 8 the simulation results obtained by the classical crossover in one and two point, the uniform crossover and the proposed crossover. The proposed crossover based on the channel reliabilities improves considerably the performances of the proposed decoder.

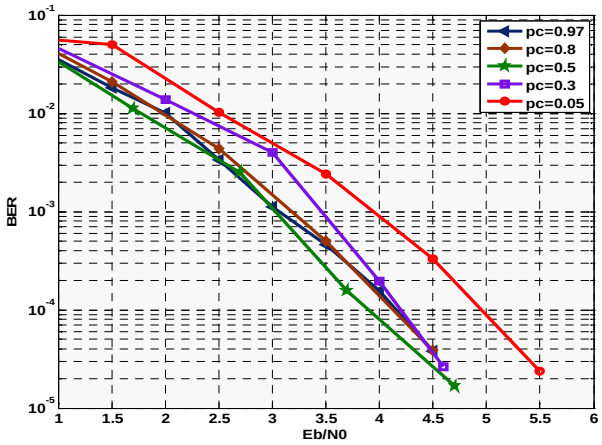

Figure 7. Impact of the crossover probability on the AutDAG performances

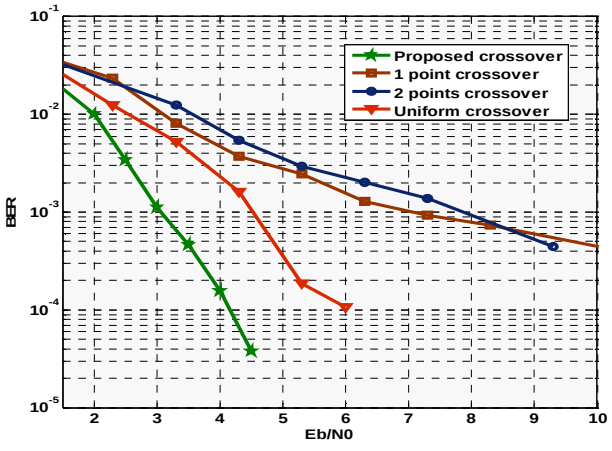

Figure 8. Impact of the crossover type on the AutDAG performances

To illustrate the impact of the mutation probability p_m on the error correcting performances of AutDAG we give in the figure 9 the simulation results corresponding to some values of pm. By analyzing this figure, the null and the big values of pm damage the performances; however the values 0.08 and 0.2 improve them. At BER=2.10^{-5} the good choice of pc allows to win about 2 dB.

To illustrate the impact of the threshold of reliability S on the error correcting performances of AutDAG we give in the figure 10 the simulation results corresponding to some values of S. By analyzing this figure, the increase of s from 0 to 0.7 improves the performances; however the big value 1.1 damages them. At BER=3.10^{-5} the good choice of S allows to win about 1.3 dB.

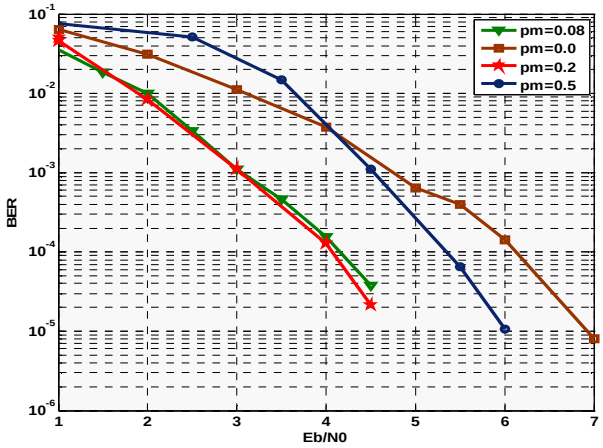

Figure 9. Impact of the crossover probability on the AutDAG performances

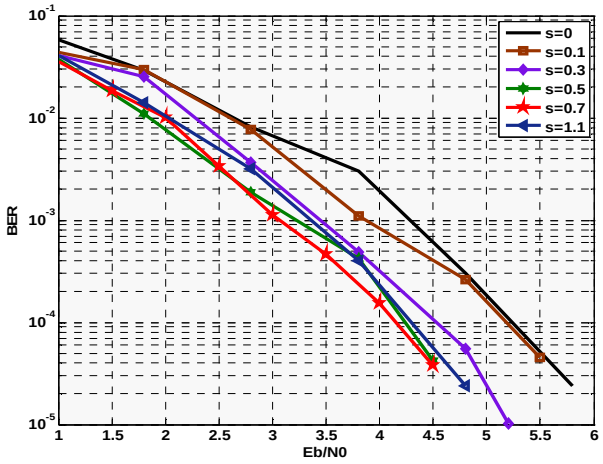

Figure 10. Impact of the threshold of reliability on the AutDAG performances

The threshold of decoding T allows to stops AutDAG when this later converges to a codeword c at Hamming distance from the received word h less than or equal to T. The minimum distance of the QR(47,24) code is equal to 11 and its error correcting capability is then equal to 5. The figure 11 shows the impact of T on the error correcting performances of AutDAG for this code. It shows that the values of T which are less than or equal to the error correcting capability of this code give the same performances. However, the values 6 and 7 damage them. At BER=10^{-5} the good choice of T between 0 and 7 allows to win about 2.5 dB.

The figure 12 presents a comparison between the error correcting performances of AutDAG and those of the Chase-2 algorithm [4] applied to the BCH (63, 39) and QR(47,24) codes. For the first code we have applied the Chasing technique on the hard decision permutation decoding algorithm [10] with 1000 permutations. For the second code

we used the hard decision Berlekamp-Massey decoder in the chasing technique and only the 63 cyclic permutation in AutDAG with 100 generations (z=63, Ng=100). The figure 12 shows that AutDAG passes the Chase algorithm in terms of error correcting performances. At BER=10^{-5} the difference between these decoders is about 0.8 dB.

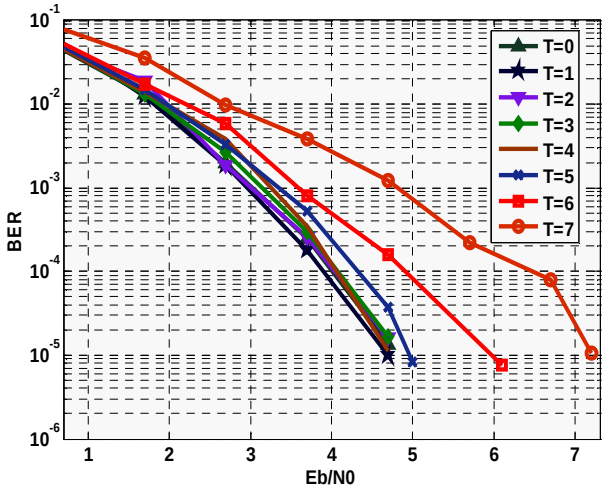

Figure 11. Impact of the threshold of decoding on the AutDAG performances

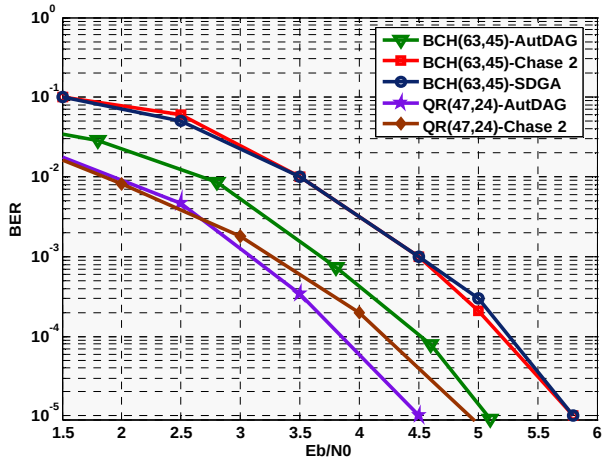

Figure 12. Comparison between the error correcting performances of AutDAG and Chase-2 algorithms

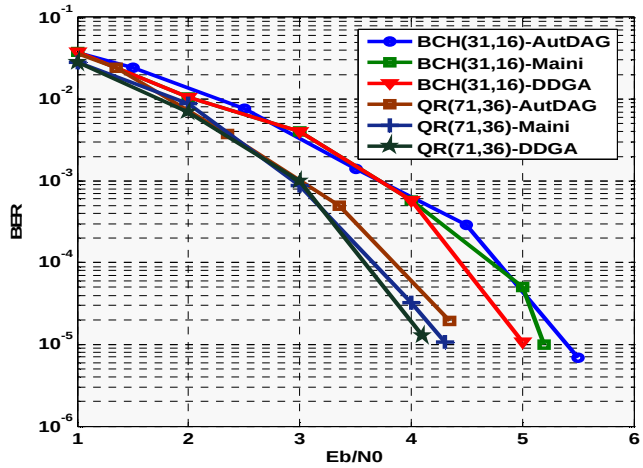

Figure 13. Comparison between the error correcting performances of AutDAG and those of the Maini and DDGA Algorithms.

The figure 13 presents a comparison between the error correcting performances of AutDAG and those of the Maini and DDGA Algorithms applied to the QR (71, 36) and QR(31,16) codes by using the simulation parameters given in the table 2. This figure shows that the difference between these three decoders in terms of the error correcting performances is negligible. Conversely the AutDAG algorithm is less complex.

The figure 14 shows that AutDAG performs the same as the OSD[3] algorithm [3] for code QR (71,36,11) by using the simulation parameters given in the table 2. The complexity of AutDAG is less than that of the OSD[3]. At BER=10^{-5} the difference between AutDAG and OSD[1] decoders is more than 1 dB.

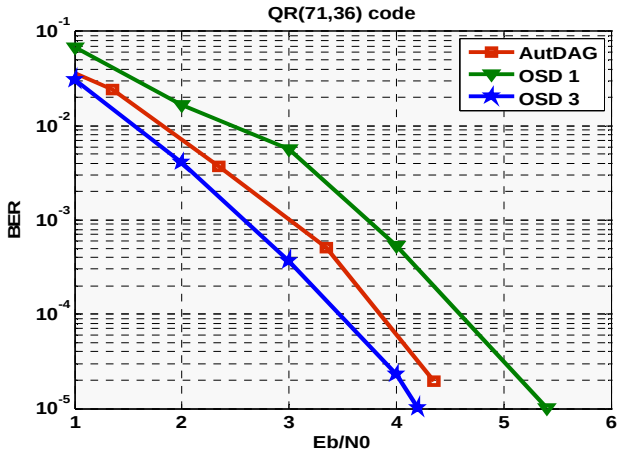

Figure 14. Comparison between the error correcting performances of AutDAG and the OSD algorithms for the QR(71,36) code.

The figure 15 shows that AutDAG reaches the error correcting performances of the Maximum Likelihood Decoding algorithm (MLD) for the Nordstrom-Robinson code (NR) which is a binary nonlinear code [13] by using the simulation parameters given in the table 2.

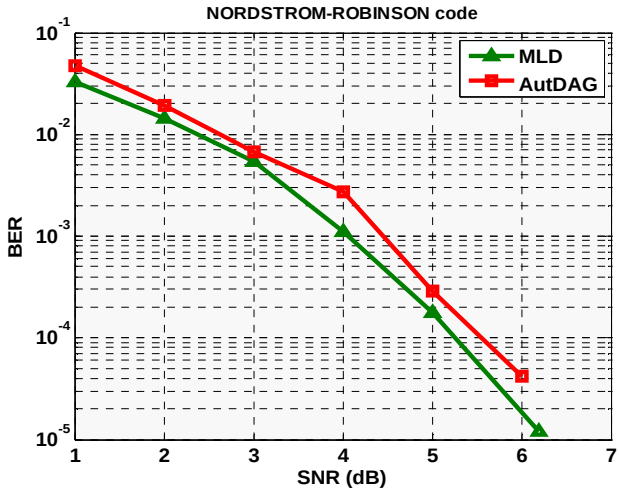

Figure 15. Comparison between the error correcting performances of AutDAG and the MLD algorithms for the Nordstrom-Robinson code.

For the simulation plotted in the figure 15, AutDAG uses very little number of generations and a small population size, so its complexity is less than that of the MLD. At BER=4.10^{-5} the difference between AutDAG and MLD decoders is about 0.25 dB.

The figure 16 shows that AutDAG performs better than the algorithm of Chana et al [7-8] for the QR (71,36,11) code by using the simulation parameters given in the table 3.
The figure 17 presents a comparison between the error correcting performances of AutDAG and those of the PGAD algorithm[12] for the BCH (63, 30) code with 500 generations (z=376, Ng=500, Ni=600). The figure 17 shows that AutDAG and PGAD have the same performances for this code. However, AutDAG is less complex than PGAD as we are going to see in next section.

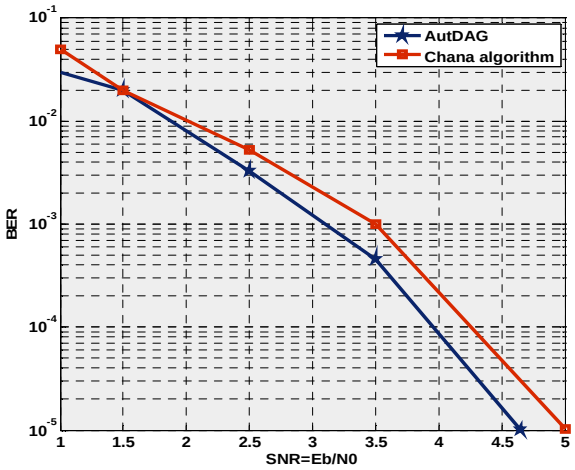

Figure 16. Comparison between the error correcting performances of AutDAG and the Chana algorithms for the QR(71,36) code.

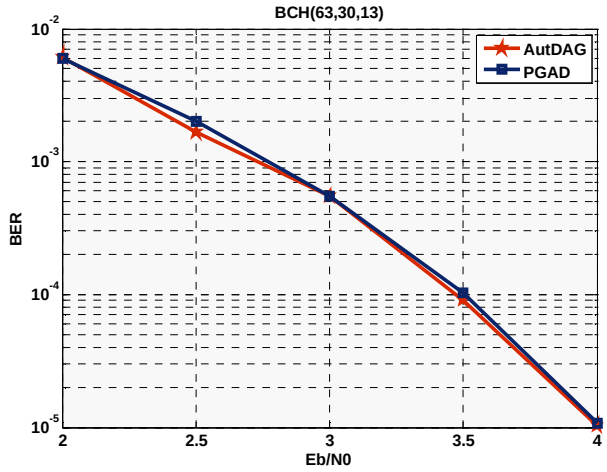

Figure 17. Comparison between the error correcting performances of AutDAG and PGAD for the BCH(63,30) code.

4. A Simplified Maini Algorithm

The stop criterion of AutDAG, introduced in the sixth step by finding a codeword at Hamming distance less than or equal to the threshold of decoding T, from the received word h, allows reducing considerably the run time of this algorithm.

In order to simplify the Maini decoder, the same stop criterion is used, the simulation results are plotted in the figure 18 and they are obtained by using the default parameters at exception of those given in the table 4.

Table 4. Simulation parameters

Decoder	Ni	Ng	p_c	p_m	T	z	S
AutDAG	100	100	0.97	0.08	5	300	0.6
Maini Simplified Version	100	300	0.97	0.03	5	-	-
Maini Origial Version	100	300	0.97	0.03	-	-	-

The figure 18 shows that the difference between the error correcting performances of the simplified version of the Maini algorithm (T=5) and those of the AutDAG algorithm is negligible. However, the original version of the Maini algorithm performs better.
The efficiency of a decoder is measured in general by the rate: performances to the time complexity. Thus, we have computed the run time of these three decoding algorithms (AutDAG, Maini simplified and original versions) for 1000 sequences. The simulation results are plotted in the figure 19 and they are obtained by using the default parameters at exception of those given in the table 4. All these simulations are obtained by using a simple configuration machine: Intel (R) Core ™ 2 Duo CPU T7300 @ 2.00 GHz, RAM: 2.00 Go.

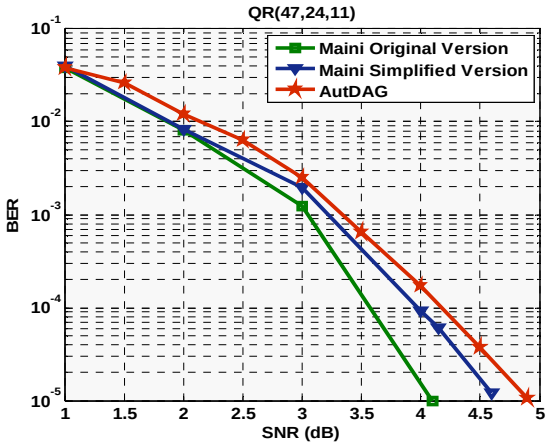

Figure 18. Comparison between the error correcting performances of AutDAG, Maini original version and simplified version for the QR(47,24) code.

The figure 19 proves that the run time of AutDAG is less than the one of the simplified version of the Maini algorithm which is less than the one of the original version. When the SNR value increases, the run time deceases at exception of the original version of the Maini algorithm which it remains constant.

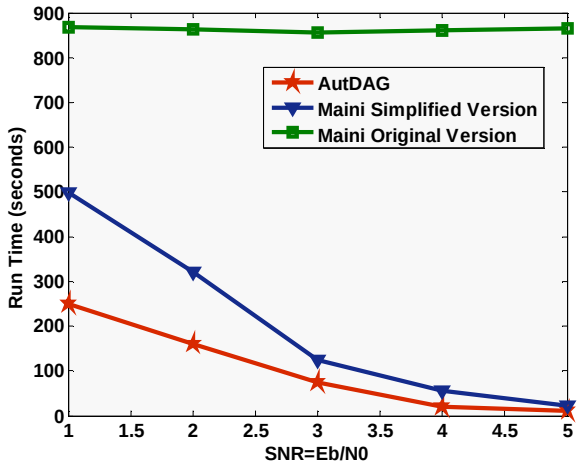

Figure 19. Comparison between the run time of AutDAG, Maini original version and simplified version for the QR(47,24) code.

5. Study of the complexities

At any given stage, we maintain a few sets of *Ni* * *k* arrays, therefore the memory complexity of this algorithm is *O*(*Ni.k*). The AutDAG algorithm has the same memory complexity comparing to the Maini and DDGA algorithms.

In order to get the temporal complexity of our algorithm, we will compute the complexity of each step:

Step 1 has temporal complexity of $O(z)$. This complexity is independent on the length and the dimension of the code and it is negligible comparing to the other steps.

Step 4 has temporal complexity of $O(k^2 n)$ [2]. This complexity depends on the random number generator in use, but the cost is negligible compared to that of other steps. In addition, the AutDAG algorithm can always use the same initial population which can be generated uniformly and stored in memory before the starting of AutDAG.

Step 6 has temporal complexity equal to the one of encoding multiplied by the product *NiNg* [2,11] which is the worst case complexity.

The polynomial encoding has complexity of order *O(log(n).log(n-k))* [6-7] which is very reduced comparing to the matrix encoding which is of complexity of *O(k.n)*. The AutDAG algorithm benefit from the simplicity of polynomial encoding, contrary to the DDGA, Maini and OSD decoders which require using the generator matrix for encoding even if the code is cyclic.

The other steps contain some assignments and one operation of coding as consequence their complexity is negligible.

The table 5 gives an upper bound of the AutDAG complexity for cyclic, linear and random systematic block codes. It gives also the complexities of Maini algorithm, OSD, Chase-2, SDGA, Chana algorithm, DDGA, PGAD. These complexities measurement are more detailed respectively in [2-4, 5-6, 11-12]. In the table 5, C(HD), C(Encoding), Nc and Ns denote respectively the complexity of the hard decision decoder HD, the one of encoding, the offspring number and the processor number. The table 5 shows that AutDAG is less complex than their presented competitors.

Table 5. Complexity of some decoding algorithms

Decoding algorithm	Type of binary block code	Temporal complexity
PGAD	Linear	$O(n.\ln(n)+k^2.n+k.n.(Ni+Ng.Nc)+Ng.Ni.\ln(Ni)+Ns.\ln(Ns))$
AutDAG	Cyclic	Less than or equal to $O(Ni.Ng(\log(n).\log(n-k)))$
	Linear and not cyclic	Less than or equal to $O(Ni.Ng.k.n))$
	Systematic	Less than or equal to $O(Ni.Ng.C(Encoding)))$
Maini algorithm	Cyclic or linear	$O(Ni.Ng(k.n+\log(Ni)))$
DDGA	Cyclic or linear	$O(Ni.Ng(k.(n-k)+\log(Ni)))$
SDGA	Cyclic or linear	$O(2^t.(Ni.Ng(k.n^2+k.n+\log(Ni))))$
Chase2-HD	Binary block code	$O(2^t.C(HD))$
Chana algorithm	Cyclic	$O(2^{p+1}.k.\log(n).(n+\log(n-k)))$
OSD of order m	Cyclic or linear	$O(n^{m+1})$

The stop of AutDAG when the best codeword in the current population has Hamming distance, from the received word, less than or equal to the threshold of decoding T, allows reducing considerably the temporal complexity of AutDAG by minimizing the number of the effectively generated populations. In order to test the impact of this stop criterion on the time complexity of AutDAG we have computed the average number of the populations generated by this decoder as function of SNR (E_b/N_0) and the threshold of decoding T. This statistical study is made on 10000 received sequences at each SNR for the QR(47,24) code with the default parameters and only 50 generations (Ng=50). The figure 20 gives the obtained results; it shows that when T and/or the SNR increase, the complexity of AutDAG decreases. For the code under study, the convenient value of T is equal to 5 because this value reduces considerably the complexity and its impacts on the error correcting performances are negligible as it was viewed previously in the figure 11. It is important to note that the value 5 is equal to the error correcting capability of this code.

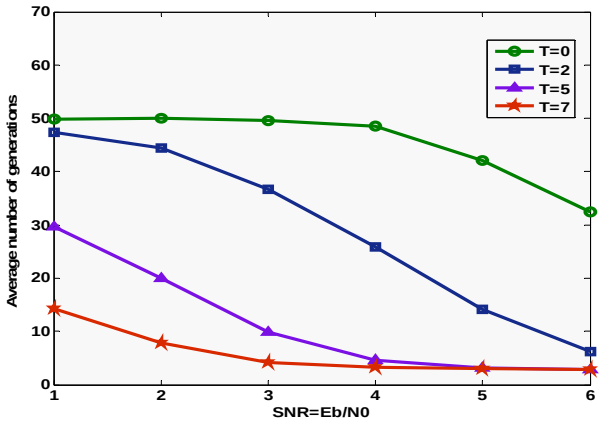

Figure 20. Average number of generations used in AutDAG as function of SNR and the threshold of decoding T for the QR(47,24) code.

Comparing to the Maini decoder, the DDGA algorithm reduces the complexity of decoding the codes of high rates. For the codes of rate about 0.5 like the QR codes the DDGA and Maini decoders have the same complexity, however the AutDAG algorithm is less complex than these two competitors in all cases.

6. Conclusion and perspectives

In this paper we have used genetic algorithms for decoding systematic block codes. The simulations applied on some BCH and QR codes show that the proposed algorithm is an efficient soft-decision decoding algorithm. In summary the proposed decoder has two main advantages comparing to their competitors OSD, Maini and DDGA algorithms. The fist characteristic of AutDAG is its possibility to decode some binary non linear code like the Nordstrom-Robinson code. The second characteristic is its low complexity obtained by the threshold of decoding and also by the polynomial encoding in the case of cyclic codes. The obtained results will open new horizons for the artificial intelligence algorithms in the coding theory field.

References

[1] G. C. Clarck, J.B Cain, “Error-Correction Coding for Digital Communication”, New York Plenum. 1981.

[2] H. Maini, K. Mehrotra, C. Mohan and S. Ranka, “Soft decision decoding of linear block codes using genetic algorithms,” IEEE International Symposium on Information Theory, p. 397, Trondheim , Norway. 1994.

[3] M.P.C Fossorier and S. Lin, "Soft decision decoding of linear block codes based on ordered statistics", IEEE Trans. information theory Vol. 41, pp. 1379-1396. 1995.

[4] D. Chase, “A class of algorithms for decoding block codes with channel measurement information,” IEEE Transaction on Information Theory, vol IT-18, pp 170-182. 1972.

[5] A. Azouaoui, I. Chana and M. Belkasmi “Efficient Information Set Decoding Based on Genetic Algorithms,” International Journal of Communications, Network and System Sciences, Vol. 5, No. 7, pp. 423-429, 2012.

[6] I. Chana, H. Allouch and M. Belkasmi, “An efficient new soft-decision decoding algorithm for binary cyclic codes”, IEEE International Conference On Multimedia Computing and Systems (ICMCS’11) proceedings , pp 823-828, Ouarzazate, Morocco, April 2011.

[7] I. Chana, H. Allouch and M. Belkasmi, “New Turbo Decoding of Product Codes Based on Cyclic Codes”, Journal of Telecommunications, volume 11, issue 2, pp 39-48, December 2011.

[8] S. Nouh, A. El khatabi and M. Belkasmi, “Majority voting procedure allowing soft decision decoding of linear block codes on binary channels”. International Journal of Communications, Network and System Sciences, N° 9, Vol 5. 2012.

[9] J. L. Massey “Shift-register synthesis and BCH decoding” IEEE Transaction on Information Theory, IT-15 Vol.1, pp. 122–127. 1969.

[10] F. J. MacWilliams “Permutation decoding of systematic codes”. Bell System Tech. J., 43:485–505. 1964.

[11] A. Azouaoui, M. Belkasmi and A. Farchane, “Efficient Dual Domain Decoding of Linear Block Codes Using Genetic Algorithms,” Journal of Electrical and Computer Engineering, Vol. 2012, 2012, Article ID: 503834. doi:10.1155/2012/503834

[12] A. Ahmadi, F. El Bouanani, H. Ben-Azza, Y. Benghabrit, “A Novel Decoder Based on Parallel Genetic Algorithms for Linear Block Codes,” International Journal of communications, Network and System Sciences, N° 1, Vol 6, 2012. doi:10.4236/ijcns.2013.61008

[13] R. Hammons, P.V. Kumar, A.R. Calderbank, N.J.A Sloane, and P. Solé, “Kerdock, Preparata, Goethals and Other Codes are linear over Z4”. IEEE Transactions on information theory, 40/301-319. 1994.

[14] F.J. MacWilliams and N.J.A Sloane, “The theory of Error-Correcting Codes,” Publishing Company, North-Holland. 1977.

[15] T. P. Berger and P. Charpin, “The automorphism groups of BCH codes and of some affine-invariant codes over extension fields”, Design, Codes and Cryptography. Vol. 18, Issue 1-3, pp 29-53, 1999.

[16] J. McCall, “Genetic Algorithms for Modelling and Optimization”, J of Computational and Applied Math, Vol. 184, No 1, pp. 205 – 222. 2005.

[17] D. E. Goldberg, “Genetic Algorithms in Search, Optimization and Machine Learning Reading”, MA Addison Wesley. 1989.

[18] A. M. S Zalzala, and P. J. Fleming, "Genetic Algorithms in Engineering Systems," Control engineering Series 55, Institution of Electrical Engineers. London. 1999.

[19] A. G. Scandura, A.L. Daipra, L. Arnone, L. Passoni, J.C. Moreira, "A Genetic Algorithm Based Decoder for Low Density Parity Check Codes" Latin American Applied Research, N° 3, Vol. 36, pp. 169-172. 2006.

[20] S. Nouh and M. Belkasmi, “Genetic algorithms for finding the weight enumerator of binary linear block codes”, International Journal of Applied Research on Information Technology and Computing IJARITAC N°3, Vol 2, 2011.

[21] P. Remlein, D. Szłapka, “Genetic Algorithm used in Search of good Tailbiting Codes”, International Journal of Communication Networks and Information Security IJCNIS, Vol. 1, No. 3, December 2009.

[22] C. Lin, A. Qing and Q. Feng1, “Synthesis of unequally Spaced Antenna Arrays by a new Differential Evolutionary Algorithm”, International Journal of Communication Networks and Information Security (IJCNIS). Vol. 1, No. 1, 2009.