TNC Transactions on Networks and Communications
SOCIETY FOR SCIENCE AND EDUCATION UNITED KINGDOM
VOLUME 2, ISSUE 3
ISSN: 2054 - 7420

Four Parallel Decoding Schemas of Product BlockCodes

Abdeslam Ahmadi[(1)], Faissal El Bouanani[(2)], Hussain Ben-Azza[(1)]
[(1)]Ecole Nationale Supérieure d'Arts et Métiers-Meknes, Morocco;
[(2)]Ecole Nationale Supérieure d'Informatique et d'Analyse des Systàmes-Rabat, Morocco;
ab_ahmadi@hotmail.com,elbouanani@ensias.ma,hbenazza@yahoo.com

ABSTRACT

This paper presents four new iterative decoders of two dimensional product block codes (2D-PBC) based on Genetic Algorithms. Each one runs in parallel on a number of processors connected by a network. As for the conventional iterative decoder, each elementary decoder of these new schemas uses as input, the received word and the extrinsic information computed by the previous elementary decoder. They have polynomial complexities in parameters of the code and those of the genetic algorithm. These are almost the same of the conventional iterative decoder complexity, but the performances are improved. Indeed, at each iteration, the new parallel decoders preserve the better of extrinsic information computed by elementary decoders running simultaneously on all processors.

Keywords: Error Correcting Codes, Product Block Codes, Genetic Algorithms, Parallel Decoding, Iterative Decoding, Time Complexity.

1. INTRODUCTION

In digital transmission, the information encoded in a binary sequence may be translated (modulated) into an analog signal to cross the communication channel, which is always noisy. The Noise due to channel parasites and modulation /demodulation processes can alter the useful signal. Likewise, the data stored in the storage media can be corrupted because of several factors (scratches, wear, etc). The encoder receives the information symbols provided by the source and adds redundancy symbols, carefully chosen, so that the maximum of infiltrated errors can be corrected. Upon arrival, the decoder attempts to restore the original sequence, using the redundancy symbols.

The analytical decoding techniques prove to be limited. Either they do not give satisfactory performance, or they are efficient, but require a high execution time and/or very large memory space like Maximum Likelihood decoding. This is why research in coding theory is oriented towards probabilistic, iterative, or meta-heuristic decoding techniques, where

DOI: 10.14738/tnc.23.229
Publication Date: 12th June 2014
URL: http://dx.doi.org/10.14738/tnc. 23.229

performances of some decoders approaching the Shannon limit [1]. This is the case for example for Turbo codes [2] and LDPC codes [3-4].

The challenge is to find methods which ensure a compromise between their correction capabilities and their complexities(acceptable execution time and memory space). Thus, in 1975, Holland introduced Genetic Algorithms(GAs) inspired by biological laws and natural selection [5]. They were then developed and popularized by Goldberg [6].

In 1994, Maini proposes a decoder based on GAs (GAD) giving good performances [8]. The works we have proposed in [9-12], have as objectives to improve performances and/or complexities of decoders based on GAs. In this paperwe propose four new parallelization schemas of two dimensional product block codes iterative decoding, where an elementarydecoder based on GAs is used [8].

This paper is organized as follows. Section 2 reminds some fundamental theoretical concepts. Section 3 presents elementary,iterative and parallel decoders that we use in the proposed schemas. Section 4 describes our parallelization schemas of aniterative decoding. In section 5 we discuss and study their time complexities. Finally, we give in section 6 our conclusions andperspectives of this work.

2. BACKGROUND

In In this section, we first make a quick reminder of product block codes. We then define the main classes of complexity, and we finish by presentation of genetic algorithms.

2.1 Product blockcodes

2.1.1 Linear block codes

Let $F_2 = \{0, 1\}$ be the binary alphabet and $(F_2)^n$be the set of vectors of length *n*. i.e. :

$$(\mathbb{F}_2)^n = \{x_1|...x_n / x_1, ..., x_n \in \mathbb{F}_2\} \tag{1}$$

A linear code C of length *n* on F_2 is a vector subspace of $(F_2)^n$. Such that the hamming distance between two different code-words is greater than the minimum distance of the code. i.e. :

$$\begin{cases} \forall\, x, y \in C \;/\; x \neq y,\; d_H(x,y) \geq d_{\min} \\ \forall\, x, y \in C,\; x + y \in C\;; \\ \forall x \in C, \forall \lambda \in \mathbb{F}_2,\; \lambda x \in C\;. \end{cases} \tag{2}$$

Note that in F_2, the second condition is equivalent to $0 \in C$. Let$k = dim(C)$,be the dimension of *C*, and $B = (g_i)_{1 \leq i \leq k}$a base of *C*.Since $g_i \in C$, then $length(g_i) = n$. The matrix *G* whose rows are the vectors of the base, is called the generator matrix of the code *C*. Note that the matrix *G*is not unique, since the base *B* is not. Thus:

$$G = \begin{pmatrix} g_1 \\ \dots \\ g_k \end{pmatrix} = \begin{pmatrix} g_{11} & g_{12} & \dots & g_{1n} \\ \dots & \dots & \dots & \dots \\ g_{k1} & g_{k2} & \dots & g_{kn} \end{pmatrix}$$

So we have $C=\{\alpha G/\alpha \in (F_2)^k\}$.i.e :

$$\forall x = (x_1, ..., x_n) \in C, \exists! \alpha = (\alpha_1, ..., \alpha_k) \in (\mathbb{F}_2)^k / \; x = \alpha G$$

$$\iff \quad (x_1, ..., x_n) = (\alpha_1, ..., \alpha_k) \begin{pmatrix} g_{11} & g_{12} & \dots & g_{1n} \\ \dots & \dots & \dots & \dots \\ g_{k1} & g_{k2} & \dots & g_{kn} \end{pmatrix}$$

$$\iff \begin{cases} x_1 = \alpha_1 g_{11} + ... + \alpha_k g_{k1} \\ x_2 = \alpha_1 g_{12} + ... + \alpha_k g_{k2} \\ \dots \\ x_n = \alpha_1 g_{1n} + ... + \alpha_k g_{kn} \end{cases}$$

The parity check matrix H is a marix such that $GH^T = HG^T = 0$, where H^T is the transpose of matrix H. We have then $v = uG \Leftrightarrow vH^T = Hv^T = 0$. H has a crucial role in decoding. Indeed, the received word $v$is a codeword (without errors) if and only if $vH^T = 0$. It is said, in this case, that the word $v$satisfies the parity constraints of the code C.

The dual code of C, denoted by $C^T(n,n-k)$, is a linear block code whose generator matrix is H(parity matrix of C). Then, we can write:

$$C^T = \{y = (\mathbb{F}_2)^n \; / \; \forall x \in C, \quad < x, y >= 0\} \tag{3}$$

where$<x,y>$is the scalar product of $x$and y.

2.1.2 **Product block codes**

A product code is built from two or more elementary block codes, generally linears. Let $C_1(n_1, k_1, d_1)$and $C_2(n_2,k_2, d_2)$be two linear block codes. The product code $C= C_1 \otimes C_2$ is constructed as follows:

• The information symbols are arranged in a matrix of k_1 rows and k_2 columns ($k_1 \times k_2$ symbols) ;

• Eachone of the k_1 rows is coded by the code C_2;

• Eachone of the n_2 columns is coded by the code C_1.

Thus, a codeword of the product code $C$is a block of n_2 rows and n_1 columns ($n_2 \times n_1$ symbols). We show [13] that allrows are codewords of C_1 and all columns are codewords of C_2. Furthermore, the parameters of the product code $C(n, k, d)$are :

• $k= k_1 \times k_2$;

• $n= n_1 \times n_2$;

• $d= d_1 \times d_2$;

• $R = R_1 \times R_2$;

Where R_1,R_2, and $R$are respectively the rates of C_1,C_2, and C.

Product block codes represent a particular case of serial concatenated codes. Their highlight is that they allow the construction of codes of large lengths and large minimum distances, by concatenating two or more codes of small lengths and small minimum distances. Product code built with a large minimum distance will have then a large capacity for detection and correction of errors.

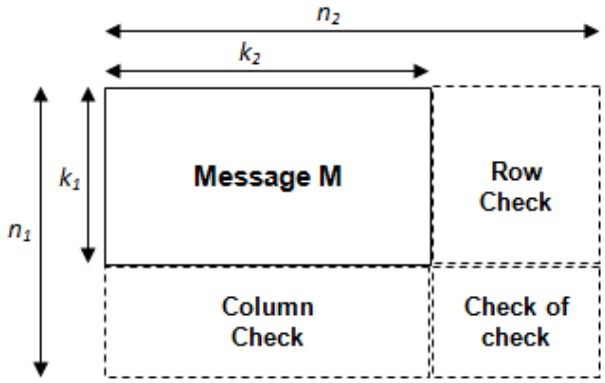

Figure 1: Product block code

2.2 Decoding Complexity

2.2.1 Complexity Classes

The complexity theory is a recent discipline that aims to classifying problems, according to their degree of difficulty of resolution. Several classes of complexity as well temporal as spatial have been defined. The best known are the classes P, NP and NP-Complete. Class P includes problems for which there exists an algorithm with a polynomial running time in the size of data, allowing solving them. The problems of this class are called "easy". The class NP contains all optimization problems where the number of potential solutions, is at worst exponential such that we can verify in a polynomial time whether a potential solution satisfies the question. A problem X of class NP is said NP-complete (or NP-hard), if each problem Y of class NP is polynomially reducible to X. i.e., there is a polynomial algorithm used to brought back the search of solution of Y to the search of solution of X. NP-Complete class contains all the problems of class NP such that if one of them is proven to be easy (solvable in polynomial time) then all NP problems are easy. If one of them is hard, then P≠ NP.

2.2.2 Complexity of linear codes decoding

In 1978, Berlkamp, McEliece and Van Tilborg [14] have stated two conjectures:

i) the problem of decoding linear codes is NP-complete;

ii) computing the minimum weight of a code is an NP-complete problem.

For a linear code, the second conjecture is equivalent to the conjecture of calculating the minimum distance. If this is calculated, the error correction capacity is then determined. This conjecture was open until 1997, when it has been proved by Vardy [15].

The fact that a problem is NP-complete causes its computational difficulty, i.e., the inefficiency of deterministic algorithms to solve it. Thus, mathematicians and computer scientist's recourse to meta-heuristic methods to find good solutions (not necessarily optimal) to NP-complete problems in polynomial time.

2.3 Genetic Algorithms

Genetic Algorithms (GAs) have been inspired in genetics and the theory of evolution of species, presented by Darwin in1860. It refutes the idea that the natural system is fixed forever. For him, the species are gradually adapting to their natural environment that could change depending on external parameters and constraints to which it is exposed. The AGs are designed to simulate processes of the natural systems required for evolution, especially those who respect the principle of "survival of stronger". The Strongest individuals will reproduced and their offspring are improved over generations. The lowest ones will disappear.

Researchers have tried to program and simulate natural phenomena since the 50s. However, these attempts were not very fruitful, because of the limitation of computer performance at this period. The use of GAs, made a big boom in the last three decades, when Holland has posed their theoretical foundations in 1975 in his book "Adaption in Natural and Artificial Systems"[5], and when Goldberg wrote in 1989 his famous book "Genetic algorithms in search, optimization, and machine learning"[6]. Thus we moved from natural Darwinism to artificial evolution, which began to be used increasingly in the solving of problems with very high complexity, in several fields.

2.3.1 Principle of GAs

GA generates randomly n individuals to form the initial population. For Every individual, which is a potential solution to the problem to be optimized, we associate a fitness (or cost) that measures its quality as solution. Then we select, with a probability that depends on the fitness, the best individuals that may be crossed to give birth of new individuals (children). These undergo, with a certain probability, to mutations in their genes. This forms the new population of the next generation. We repeat the same treatment until the stop condition is satisfied.

Here is the basic genetic algorithm, as shown in figure 2:

Algorithm: Basic GA

1. [Initialization]: generate a random population of n individuals :$(I_i)_{1\leq i\leq n}$;

2. [Fitness]: compute the fitness $f(I_i)$ for each individual I_i of the current population ;

3. [Reproduction]: create a new population by repeating the following steps:

   • [Selection]: select two parents, taking into account their fitness;

   •[Crossover]: cross, with a probability p_c, the parents to create new individuals ;

   • [Mutation]: mutate, with probability p_m, the new individuals ;

   • [Insert]: add new individuals to the new population;

4. [Replace]: use new generated population for the next iteration;

5. [Test]: if the stop condition is satisfied, then return the best individual of the current population;

6. [Loop]: else go to the step 2.

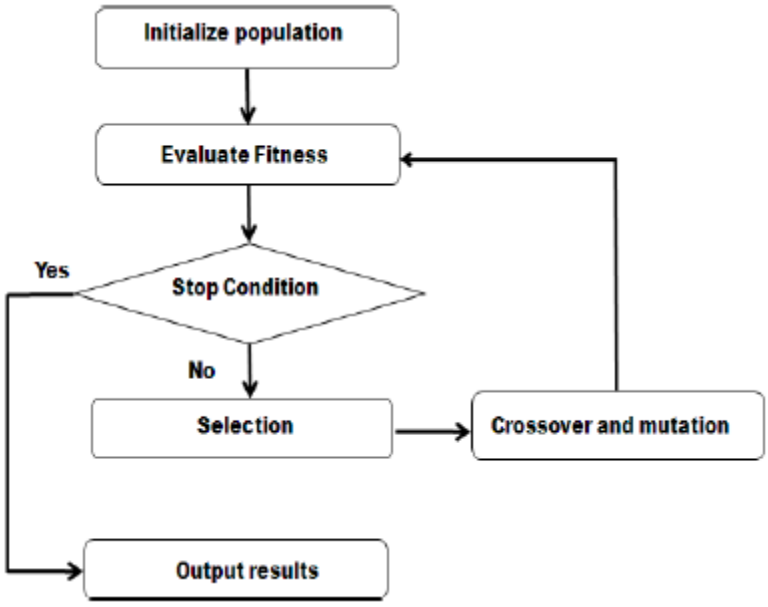

Figure 2: Genetic algorithm flowchart

2.3.2 **Convergence of GAs**

The researches consisting of laying the theoretical foundations of GAs are very rare. The reference works in this sense are those of Eiben et al. [16], Fogel [17] and Rudolph [18].They have, first, given a mathematical formulation of the basic genetic algorithm, modeled its evolution as a Markov chain, and finally defined sufficient conditions for its convergence toward an optimum. They have shown that the convergence to the global optimum is not an inherent property of the basic genetic algorithm, but a consequence of the idea of keeping track of the best solution found over time (from one generation to another). In other words, the basic GA can be considered an optimization algorithm for static optimization problems,

because it is provable that it does not converge toward any subset of the set of states containing at least one global solution, even in infinite time.

2.4 Parallel Systems

The choice of distributed systems or parallel machines is strongly imposed for applications requiring very important treatment and/or memory space. There are many types of parallel machines which are classified by Flynn (1972) according to two independent concepts: the instruction stream and data stream used by these instructions. Thus, there are four possible combinations [7] :

- Single Instruction Single Data: the machine executes one instruction on a data each clock cycle. It's not really a parallel machine but rather a classical Von Newman computer;
- Single Instruction Multiple Data: a processor, with a single control unit (CU) and multiple arithmetic logic units (ALUs),executes the same instruction on different data each clock cycle ;
- Multiple Instruction Single Data: systems executing multiple instructions on the same data at each clock cycle
- Multiple Instructions Multiple Data (MIMD): these systems have multiple independent processors. i.e., each processor has its own CU and its own ALU. In the same clock cycle, each processor executes a different instruction on a different data.

These instructions can be synchronous or asynchronous. Most parallel systems are MIMD. A MIMD system can be either a multiprocessor, or a multi-computer or a hybrid of both.

A multiprocessor contains several autonomous processors and a single shared memory, figure 3. This last can also provide communication between different processors. A multi-computer contains a given number of autonomous computers, having a control unit, an ALU and a private memory each one, figure 4. The communication between the computers is provided by a network.

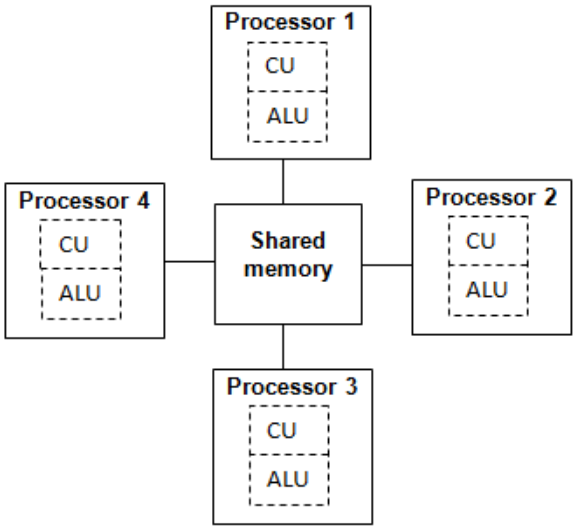

Figure 1:A Parallel system with 4 processors

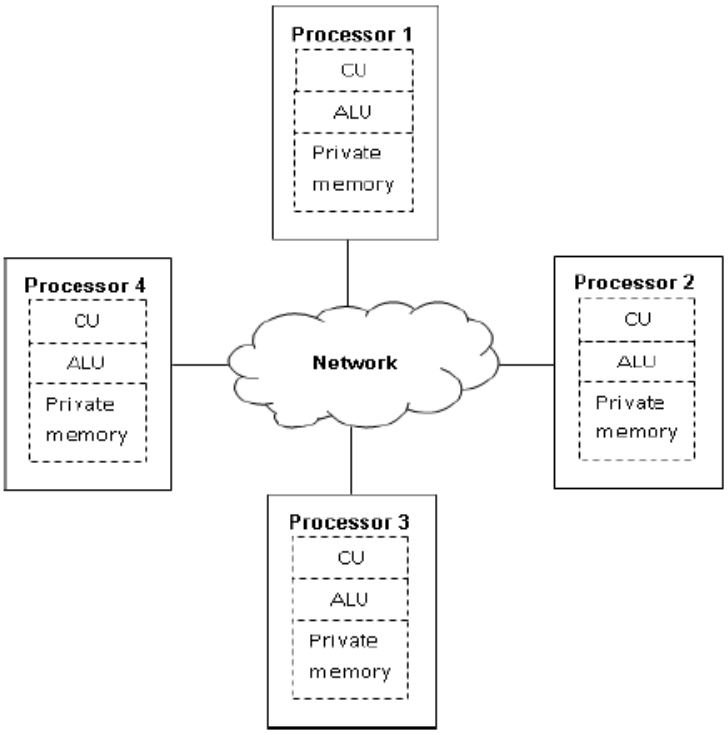

Figure 4:A multi-computer with 4 computers

A multiprocessor contains several autonomous processors and a single shared memory, figure 3. This last can also provide communication between different processors. A multi-computer contains a given number of autonomous computers, having a control unit, an ALU and a private memory each one, as shown in figure 4. The communication between the computers is provided by a network.

3. DECODING ALGORITHMS BASED ON GENETIC ALGORITHMS

Let $C(n,k,d)$ be a linear block code with generator matrix G, and $F = (F_1,...,F_n)$, $r = (r_1,...,r_n)$, respectively the vector of fading and the received sequence (associated with the transmitted sequence). The parameters N_p, N_g, N_e, p_c, $p_m$are respectively the size of the population, the number of generations, the number of elites, the crossover and the mutation rates.

3.1 Elementary decoder

We take GAD the decoder of block codes based on GAs that we have already used in [10-12]. It's a HISO (Hard-In Soft-Out) and uses GAs to decide the code word D, knowing r and F.

Algorithm : GAD

$D = GAD\ (r, F,k,n,N_p, N_g, N_e, p_c, p_m)$

- **Step1 :**sort the elements of received vector r in descending order of magnitude to have a new vector $r^{(1)}$. i.e., find a permutation π_1 such that $r^{(1)} = \pi_1(r)$ and $|r_1^{(1)}| \geq |r_2^{(1)}| \geq ... \geq |r_n^{(1)}|$.This will put, for BPSK modulation, reliable elements in the first ranks. Indeed, when the absolute value is large (far from 0), there is no risk to decode the bit 1 to 0 or 0 to 1. Let $F^{(1)}$ be the permutation of F by π_1. i.e., $F^{(1)} = \pi_1(F)$. Likewise, $G^{(1)} = \pi_1(G)$. Then, permute $G^{(1)}$ by π_2 to have G', such that its first k columns are linearly independent, permute the vectors $r^{(1)}$ and $F^{(1)}$ by the same permutation. Let $r' = \pi(r)$ and $F' = \pi(F)$, where $\pi = \pi_2 \circ \pi_1$.
- **Step 2** : quantize the first k bits of r' ($r'_i \in \mathbb{R}$) to obtain a binary vector I_1 and randomly generate $(N_i - 1)$ information vectors of k bits each one. This vectors form with vector I_1 the initial population of N_p individuals $(I_1, ..., I_{Np})$.
- **Step 3** : encode individuals of the current population, using G' to obtain code words : $C_i = I_i G'$ $(1 \leq i \leq N_p)$. Then, compute individuals fitness, defined as Euclidian distance between C_i and r', and sort individuals in ascending order of fitness.
- **Step 4** : place the first N_e individuals (N_e: elite number $\leq N_p$) to the next population, which will be completed by offsprings generated using reproduction operators : selection of two best individuals as parents (a,b) using the following linear ranking :

$$W_i = \frac{W_{\max} - 2(i-1)(W_{\max} - 1)}{N_i - 1},\ \forall i \in \{1, ..., N_p\} \tag{4}$$

  whereW_i is the ith individual weight and W_{max} weight assigned to the fittest (nearest) individual.

- **Step 5** :Reproduce the $(N_p - N_e)$ remaining individuals of the next population using crossover and mutation operations. Let *Rand* be a uniformly random value between 0 and 1, generated at each time.

  if$Rand < p_c$ then $\forall i \in \{N_e+1, ..., N_p\}$, $\forall j \in \{1, ..., k\}$,

$$I_{ij} = \begin{cases} a_j \ if \ Rand < (1 - a_j + a_j b_j \ ) + \frac{a_j - b_j}{1+e^{\frac{-4r'_j F'_j}{N_0}}} \\ b_j \quad else \end{cases} \tag{5}$$

and then,

$$I_{ij} = 1 - I_{ij} \quad if \ Rand < p_m \tag{6}$$

else,

$$I_i = \begin{cases} a \ \text{if} \ Rand < 0.5 \\ b \ \text{else} \end{cases} \tag{7}$$

end if

Repeat steps 3 to 5 for N_g − 1 next generations.

- **Step 6** :The first (fittest) individual *D'* of the last generation is the nearest to *r'*. So, the decided code word is $D = \pi^{(-1)}(D')$

3.2 Iterative decoder

An iterative algorithm receives at its input soft information and produces another one called extrinsic information which depends on the decided code word. This extrinsic information will be combined with the received word and fed back to its input. The processing is repeated Nit times and the decided code word will be the one decided at the last iteration.

For product block codes, an iterative decoder consists of placing in series two or three decoders. The extrinsic information computed by the ith decoder is combined with the received word and the result is fed to the input of the (*i* + 1)th decoder.

The result of the combination of the extrinsic information of the last decoder and the received word is injected at the input of the first decoder. The process is repeated Nit times. The iterative decoder for product block codes with two dimensions is depicted in figure 5.

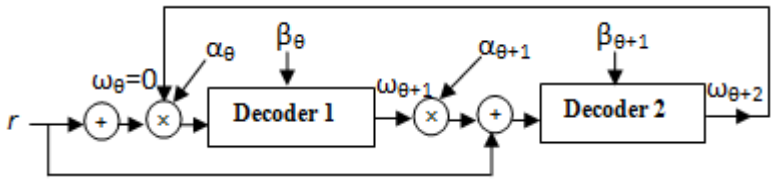

Figure 5:The iterative decoder based on AGs (IGAD)

Our iterative decoder here, is IGAD (Iterative decoder based on Genetic Algorithms), where the elementary decoders are GADs.

Let *D* denote the GAD decision of the input sequence *r*, and ω be the extrinsic information, and $H^{(j)}$ be the competitor codeword of *D* corresponding to the *j*th bit defined by :

$$\left\| H^{(j)} - r \right\| = \min_{2 \le p \le N_t} \left\{ \left\| Q^{(p)} - r \right\|, Q_j^{(p)} \neq D_j \right\},$$

where :

- $Q^{(p)}$ is the *p*th codeword of the last generation

- $Q_j^{(p)}, D_j$ are the *j*th bits of $Q^{(p)}$, *D*

- $\| . \|$ is the Euclidean distance.

The algorithm executed by each of the elementary decoders of the iterative decoder, accepts as input *r, k, n*, N_p, N_g, N_e, N_{it}, p_c, p_m, and coefficients $(\alpha_j)_{1 \le j \le 2Nit}$ and $(\beta_j)_{1 \le j \le 2Nit}$. These coefficients are optimized for each code and SNR to enhance the algorithm performance.

Algorithm : IGAD

(ω,*D*) = IGAD(*r, F,k,n,*N_p, N_g, N_e, N_{it}, p_c, p_m, α,β)

•**Step1** :$\theta = 0$, $\omega_0 = 0$;

•**Step2** :*Iterative decoding*

While ($\theta < 2(N_{it} - 1)$) **do**

- Run D(θ) = GAD (r+ α(θ)ω(θ), F, k,n,Np, Ng, Ne, pc, pm) on the first decoder to decide the codeword D(θ) ;
- Compute the extrinsic information ω(θ+1) in terms of D(θ)

For *j* = 1 **to***n***do**

If $H^{(j)}$ *exists* **then**

$$\omega_j^{(\theta+1)} = \widetilde{D}_j^{(\theta)} \left[\frac{\left\| H^{(j)} - r \right\| - \left\| D^{(\theta)} - r \right\|}{4} \right] \quad (8)$$

$$= \widetilde{D}_j^{(\theta)} \sum_{\substack{p=1, p \neq j \\ H_p^{(j)} \neq D_p^{(\theta)}}}^{n} r_p \widetilde{D}_p^{(\theta)}$$

else

$$\omega_j = \beta \widetilde{D}_j^{(\theta)} \quad (9)$$

$$\widetilde{D}_j^{(\theta)} = 2D_j^{(\theta)} - 1$$

End if

End for

- Run $D^{(\theta+1)}$ = GAD ($r+ \alpha^{(\theta+1)}\omega^{(\theta+1)}$, F, k,n,N_p, N_g, N_e, p_c, p_m) on the second decoder to decide the codeword $D^{(\theta+1)}$;
- Compute the extrinsic information $\omega^{(\theta+2)}$ in terms of $D^{(\theta+1)}$
- $\theta = \theta + 2$;

End While

•**Step3** :*Decision*

✓ Select the codeword decided by the second decoder at the last iteration $D^{(2(Nit-1))}$

3.3 Parallel decoder

We give here the sub-algorithm of our parallel decoder PGAD, based on GAs that we proposed in [12]. It runs in parallel on each one of the N_s processors:

Algorithm : PGAD

D = PGAD (r, F, k,n,N_p, N_g, N_e, p_c, p_m,N_s, N_c)

•**Step1**:*Permutations and Initialization*

✓ Run the two first steps of the algorithm GAD

•**Step 2**:*Reproduction*

✓ Encode individuals of the current population to obtain code words $C_i = I_i G'$ ($1 \le i \le N_p$)

✓ Compute individual fitness, defined as Euclidian distance between C_i and r' :

$$f(C_i) = \sum_{j=1}^{n} (C_{ij} - r'_j)^2 , \forall i \in \{1, \dots, N_p\}$$

✓ Sort the current population individuals in descending order of their fitness ;

✓ $gen \leftarrow 0$ (gen is the current generation number).

While ($gen<N_g$) **do**

✓ Copy the $N_e$best individuals (elites) from the current population to the new one ;

✓ Select parents from the N_p–N_e individuals of the current population ;

✓ Quantize the first k bits of each selected parent ;

✓ Cross with probability p_c the selected parents to generate N_c new individuals of k bits

✓ Mutate the new individuals with a probability p_m if their parents are crossed ;

✓ Encode and the N_c new individuals, compute their fitness, and insert them into the new population ;

✓ Receive N_m migrant elites (with their fitness) from the previous population of each N_s– 1 other processors, to complete the new population ;

✓ Send the best N_m= (N_I–N_e–N_c)/(N_s– 1) individuals (with their fitness) to the new populations of N_s–1 other processors ;

✓ Sort the N_p individuals (codewords) in descending order of their fitness ;

✓ Replace the current population with the new one ;*gen*←*gen* + 1 ;

End While

•**Step 3**:*Decision*

✓ Get the best individuals $(D'^{(i)})_{1\le i\le Ns}$ of the last populations of all processors. The best one *D'* of them is the closest to *r'*.i.e. *D'* = arg min { $\| D'^{(i)} - r' \|$, $1 \le i \le N_s$}. The decided codeword is then $D = \pi^{(-1)}(D')$.

The flowcharts of the previous algorithm are illustrated in both figure 6 and figure 7.

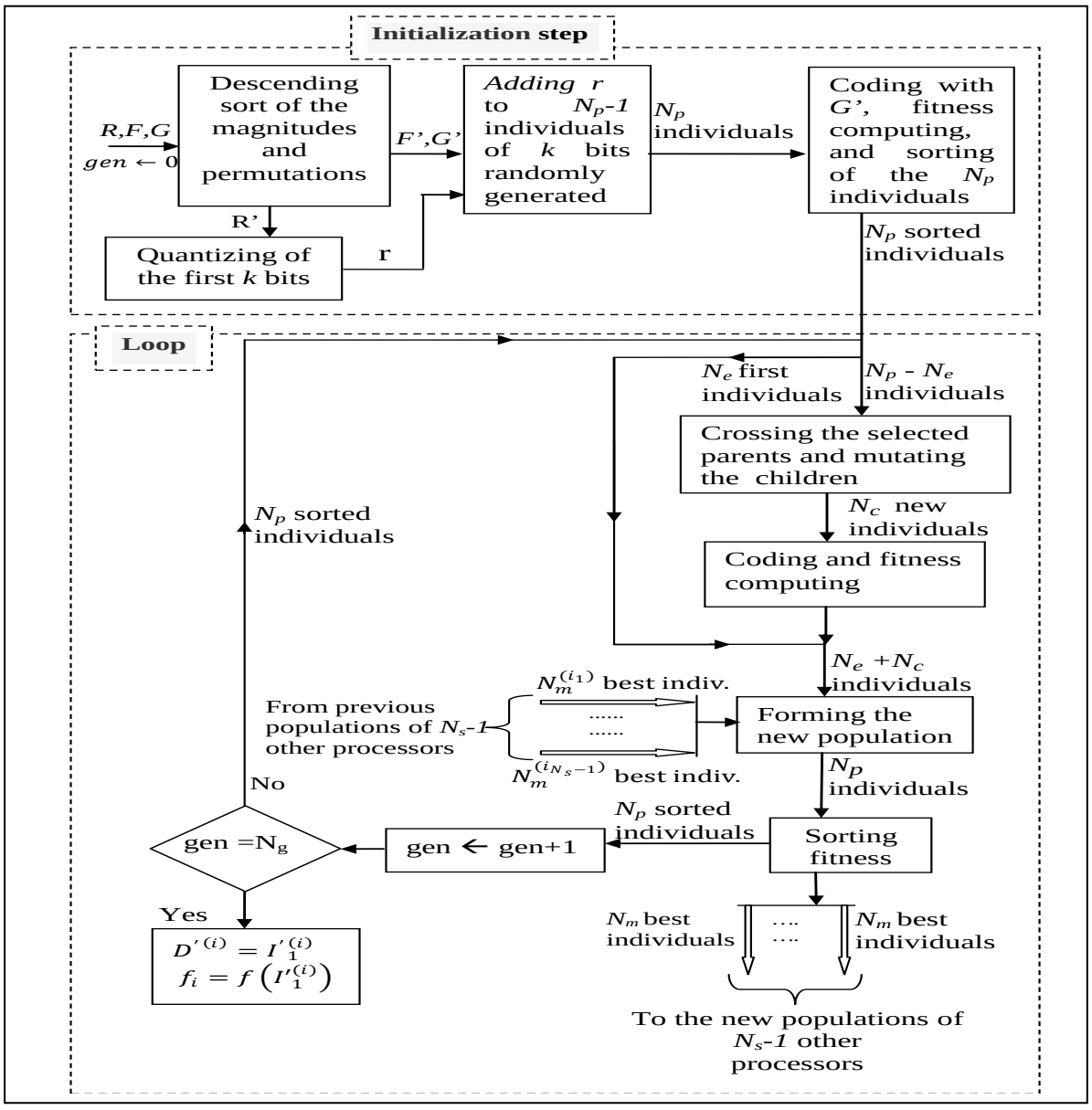

Fig. 6.The flowchart of the proposed PGAD sub-algorithm running on the *i*th processor.

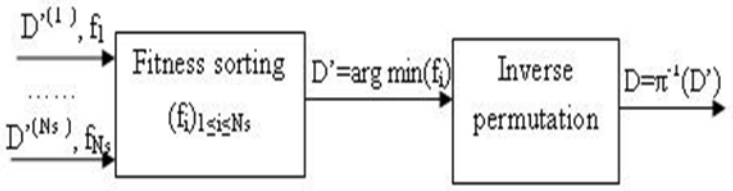

Fig. 7.The codeword decision flowchart of the proposed PGAD algorithm.

4. PARALLELIZATION SCHEMAS OF ITERATIVE 2D-PBC DECODING

The purpose is not only to reduce the execution time and occupied memory space, but also improve the quality of error correction. So, the parallelization schemas can affect the entire iterative decoder or just its elementary decoders.

Their corresponding algorithms can be run on parallel machines containing enough processors (multi-processors, multicomputer)or a network of computers. Communications between processors are performed using global variables stored in a common memory, or via send/receive primitives. To simplify the graphs, we have considered just four processors.

4.1 First schema

The figure 8 depicts the first parallel iterative decoder that we propose. It is simply runs in parallel, a conventional 2Diterative decoder on each one of the N_s processors of a parallel machine or a distributed system, a given number of iterations N_{it}. At the last iteration, the ith processor decides the codeword $D^{(i)}$. The final codeword to decide is the one with the best fitness of all $D^{(i)}$, where $1 \leq i \leq N_s$.

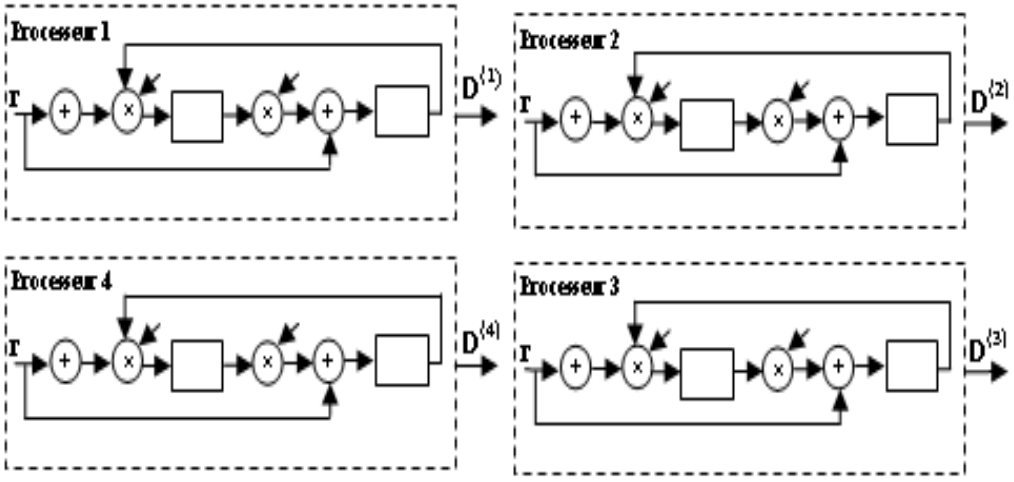

Figure 8.The first iterative parallel decoding schema on 4 processors.

The corresponding algorithm is given below:

Algorithm: PARAL_ITER1

For i=1**to**N_s**do**

Run *IGAD on* the $i$thprocessor *;*

Get the decided codeword$D^{(i)}$*;*

End For

$D=arg\ max\ \{fitness(D^{(i)}),\ 1 \le i \le N_s\}$

4.2 Second schema

As shown in figure 9, the second schema consists in executing, Nit times, a parallel decoder which is composed of N_sIGADs decoders. Each one is running on a dedicated processor. The processors are connected by a network. At each iteration, all decoders run in parallel according to their inputs. At the first iteration, the extrinsic information at the input of all decoders is zero. At tth iteration, the decoder of ith processor provides as output, extrinsic information which will be injected, in combination with the received word r, to the decoder input of processor s+1, at the iteration t+1. The extrinsic information given by the decoder of the last processor is fed to the decoder input of the first one.

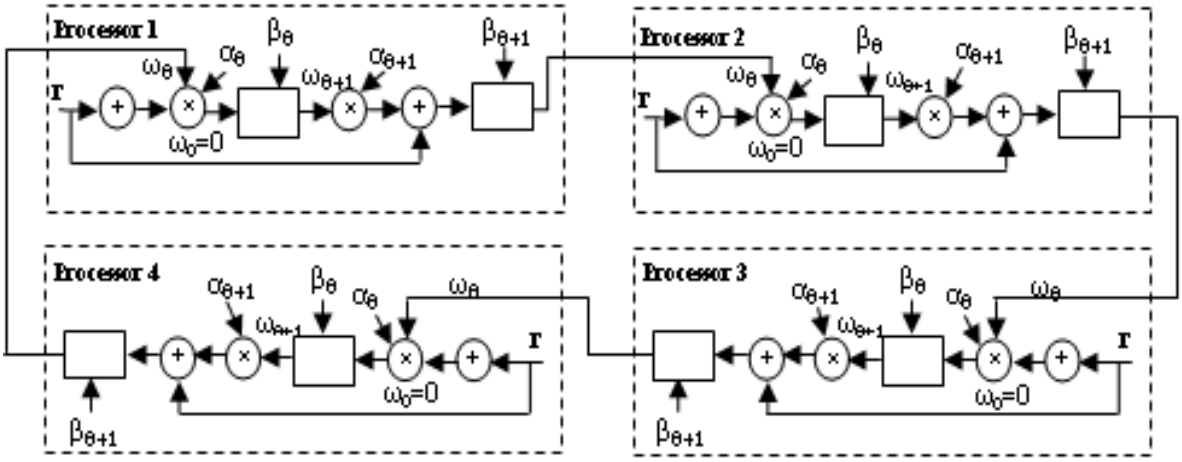

Figure 9. Second Parallel iterative decoding schema on 4 processors

The algorithm of this schema is:

Algorithm: PARAL_ITER2

θ=0 *;* ω_0=0 *;*

While $\theta \le 2(N_{it}-1)$**do**

For$i$=1**to**N_s**do**

Run$IGAD_1^{(\theta/2)}$ *;*

Get the decided codeword on the$i$th processor$D_i^{(\theta/2)}$*;*

Compute$\omega_i^{(\theta+2)}$based on$D_i^{(\theta/2)}$*;*

End For

Temp=$\omega_{Ns}^{(\theta+2)}$ *;*

For $i = N_s$ **to** 2 **do**

$\omega_i^{(\theta+2)} = \omega_{i-1}^{(\theta+2)}$;

End For

$\omega_1^{(\theta+2)} = temp$;

$\theta = \theta + 2$;

End while

$D = D_{Ns}^{(Nit-1)}$;

4.3 Third schema

The elementary decoder in the conventional iterative decoder which running on a single processor is replaced by the parallel decoder which will be run on N_s interconnected processors. At each iteration, these processors run their basis decoders independently, and each one decides its own code word. The extrinsic information provided by the parallel decoder at this iteration, is computed based on the best decided codeword.

The first decoder accepts an input that depends on the received word r and the extrinsic information $\omega^{(\theta)}$, and provides another information $\omega^{(\theta+1)}$ which will form with r , the input of the second parallel decoder. Similarly, the second decoder makes a processing based on its input, to decide the codeword. It also provides a new extrinsic information $\omega^{(\theta+2)}$ to be injected in combination with r to the input of the first decoder. This processing is performed N_{it} times, before getting the code word decided by the second parallel decoder (figure 10).

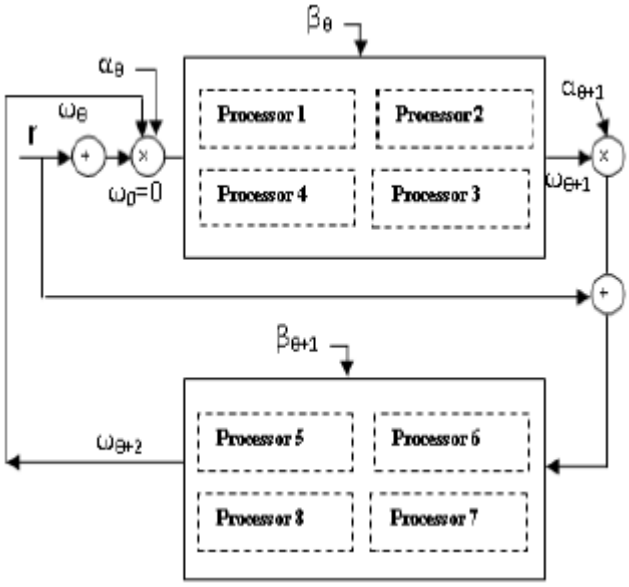

Fig. 10. Third Parallel iterative decoding schema on 4 processors

The following algorithm describes well the conduct of this parallelization scheme of iterative decoding:

Algorithm : PARAL_ ITER3

θ=0 ; ω_0=0 ;
While $\theta \leq 2(N_{it}-1)$ **do**
  For i=1 **to** N_s **do**
    Run $IGAD_1^{(\theta/2+1)}$;
    Get the decided codeword on the ith processor $D_i^{(\theta/2+1)}$;
  End For
  Get $\omega^{(\theta+1)}$ using $D_1^{(\theta/2+1)} = arg\ max\{fitness(D_i^{(\theta/2+1)}, 1 \leq i \leq N_s\}$;
  For $i = N_s+1$ **to** $2N_s$ **do**
    Run $GAD_i^{(\theta/2+1)}$;
    Get $D_i^{(\theta/2+1)}$;
  End For
  Get $\omega^{(\theta+2)}$ using $D_2^{(\theta/2+1)} = arg\ max\ \{fitness\ (D_i^{(\theta/2+1)}),\}$;
  $\theta = \theta+2$; $N_s+1 \leq i \leq 2N_s$
End while
$D = arg\ max\ \{fitness\ (D_i^{(Nit)}),\ N_s+1 \leq i \leq 2N_s\}$

4.4 Fourth schema

The principle of the fourth scheme is the same as the previous one, by replacing the two elementary parallel decoders based on GAD by two elementary parallel decoders PGAD1 and PGAD2. As shown in figure 5, the processors do not runindependent decoders but decoders working together.

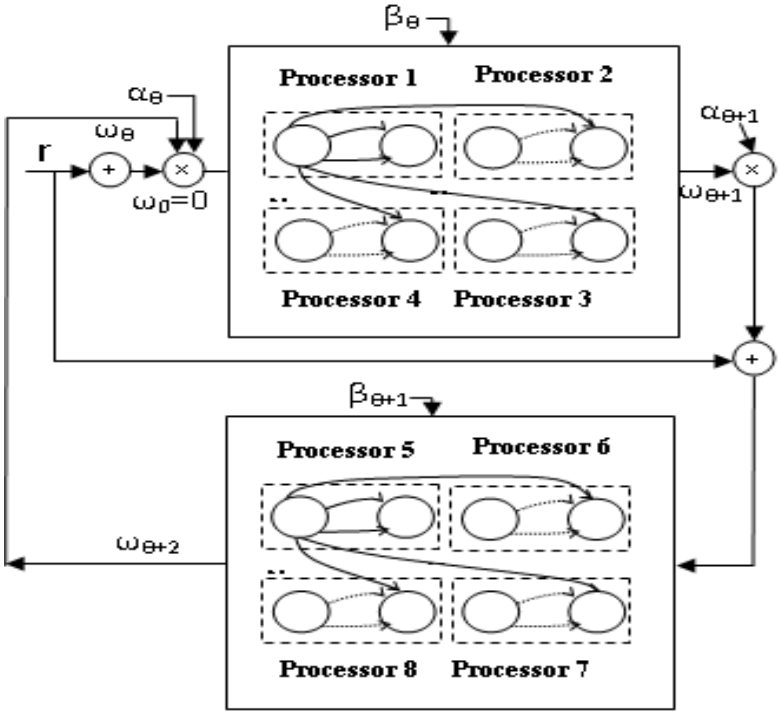

Fig. 11. Fourth Parallel iterative decoding schema on 4 processors

Its algorithm is presented below:

Algorithm: PARAL_ITER4

θ=0 ; ω_0=0 ;
  While $\theta \leq 2(N_{it}-1)$ **do**
    Run$PGAD_1$;
    Get$D^{(\theta+1)}$;
    Compute $\omega^{(\theta+1)}$ using $D^{(\theta+1)}$;
    Run$PGAD_2$;
    Get$D^{(\theta+2)}$;
    Compute $\omega^{(\theta+2)}$ using $D^{(\theta+2)}$;
    $\theta=\theta+2$;
  End while
$D=D^{(Nit)}$;

5. TIME COMPLEXITY OF THE PROPOSED SCHEMES

In this section, we interest to the expression of the time complexity of each proposed algorithm.

The complexity of GAD is [9] :

$$O[k^2n + NpNg(kn + \ln(Np))] \tag{10}$$

For the IGAD decoder, its complexity is [11] :

$$O\left(N_{it}\left[k^2 g\left(k^1, n^1, N_p, N_g\right) + n^1 g\left(k^2, n^2, N_p, N_g\right)\right]\right), \tag{11}$$

where

$$g(k, n, N_p, N_g) = k^2n + N_pN_g(kn + \ln(N_p)) + N_pn + n^2 \tag{12}$$

The complexity of PGAD is derived in [12] as :

$$O[nln(n) + k^2n + kn\left(N_p + N_gN_c\right) + N_pN_g \ln\left(N_p\right) + N_p \ln\left(N_p\right)] \tag{13}$$

5.1 First iterative Parallel decoding complexity

The algorithm PARAL_ITER1 contains a loop of instructions that run in parallel on Ns processors. The last instruction which sorts, in descending order, code words decided by all processors, has a complexity of $N_s\ln(N_s)$. Thus the complexity of PARAL _ITER1 is Complexity(IGAD) + $N_s\ln(N_s)$. i.e. :

$$O(N_{it}[k_2 g(k_1, n_1, N_p, N_g) + n_1 g(k_2, n_2, N_p, N_g)] + N_s \ln(N_s)) \quad (14)$$

Note that the complexity of this new iterative decoder is increased by $N_s\ln(N_s)$ relative to that of the conventional one (IGAD). However, it increases the probability of having decided the correct code word, $N_s$times.

5.2 Second Iterative Parallel decoding complexity

The block instructions of the first loop of the algorithm PARAL_ITER2 are run in parallel on each one of N_s processors. Therefore their complexity is the same as that of IGAD. The complexity of the loop in the second algorithm is $N_{it}N_s$. So, the total complexity of this algorithm is :

$$O(N_{it}[N_s + k_2 g(k_1, n_1, N_p, N_g) + n_1 g(k_2, n_2, N_p, N_g)]) \quad (15)$$

However, the turbo effect with N_{it} iterations in IGAD is equivalent to that of PARAL_ITER2 with N_{it}/N_s iterations. Furthermore, this scheme has the advantage of being able to cascade different types of decoders, where some of them can compensate inefficient other at each iteration.

5.3 Third Iterative Parallel decoding complexity

For this scheme, the extrinsic information computed by the two elementary decoders, depends on the best codeword decided by their N_s processors. The complexity of descending sort of these decided words for N_{it} iterations is $N_{it}N_s\ln(N_s)$. The last instruction has a complexity of $N_s\ln(N_s)$. Thus its complexity is increased by $N_{it}N_s\ln(N_s)$ compared to IGAD. i.e. :

$$O(N_{it}[k_2 g(k_1, n_1, N_p, N_g) + n_1 g(k_2, n_2, N_p, N_g) + N_s ln(N_s)]) \quad (16)$$

5.4 Fourth Iterative Parallel decoding complexity

It is clear that the complexity of PARAL_ITER4 is $N_{it}\times$ complexity(PGAD). i.e. :

$$O[nln(n) + k^2 n + kn\left(N_p + N_g N_c\right) + N_p N_g \ln(N_i) + N_p \ln\left(N_p\right)] \quad (17)$$

The complexity is multiplied by the iteration number, but the turbo effect is significantly augmented. So the correction capacity is improved.

6. CONCLUSION

We have presented four iterative parallel decoders for two dimensional product block codes. They can run on a parallel machine with enough processors or on a network of computers. One of their advantages is that they allow combining elementary decoders with the

same or different genetic parameters. This allows benefiting from the highlights of each code, at each iteration. We have shown that their complexities were slightly increased, but improve the decoding performances.

We intend to implement these schemes and test them on certain codes like BCH codes to validate the theoretical results with simulations.

The proposed schemes can also be applied to three dimensional product block codes. Also, for the first three schemes, their elementary decoders can be any ones, not necessarily based on GAs.

REFERENCES

[1]. C. E. Shannon, ”A mathematical theory of communication,” Bell System Technical Journal, vol. 27, pp. 379-423 and 623-656, July and October, 1948.

[2]. Berrou, G., A. Glavieux, and P. Thitimajshima. ”Near Shannon limit error-correcting coding : Turbo Codes,” in proc. 1993 Int. Conf. Commun. Geneva,Switzerland, May 1993, pp. 1064-1070.

[3]. Gallager, R.G. Low Density parity Check Codes. Th`ese. Cambridge, Mass.1963. IRE Transactions on Information Theory.1962.

[4]. D.J.C Mackay, and R.M. Neal, ”Near Shannon Limit Performance of Low Density Parity Check Codes”, Electronics Letters, Vol. 32, no. 18, pp.1645-1646, August 1996.

[5]. J. H. Holland, ”Adaptation In Natural And Artificial Systems, University of Michigan Press”, 1975.

[6]. D.E. Goldberg, ”Genetic Algorithms in Search, Optimization, and Machine Learning”, Addison-Wesley, 1989.

[7]. A.TANENBAUM, ”Computer architecture”. Dunod 2001.

[8]. H. S. Maini, K. G. Mehrotra,C. Mohan, S. Ranka, ”Genetic Algorithms for Soft Decision Decoding of Linear Block Codes”, Journal of EvolutionaryComputation, Vol.2, No.2, pp.145-164, Nov.1994.

[9]. F. El Bouanani, H. Berbia, M. Belkasmi, H. Ben-azza, ”Comparaison des d´ecodeurs de Chase, l’OSD et ceux basés sur les algorithmes génétiques”,GRETSI 2007, Troyes, France 11-14 Septembre 2007 in french.

[10]. M. Belkasmi, H. Berbia, F. El Bouanani, ”Iterative decoding of product block codes based on the genetic algorithms”, Source and Channel Coding(SCC), 2008 7th International ITG Conference on Source and Channel Coding (SCC’08), 2008.

[11]. A. Ahmadi, F. El Bouanani, H. Ben-Azza, and Y. Benghabrit, ”Reduced Complexity Iterative Decoding of 3D-Product Block Codes Based on GeneticAlgorithms”, Journal of Electrical and Computer Engineering, Volume 2012.

[12]. A. Ahmadi, F. El Bouanani, H. Ben-Azza, and Y. Benghabrit, ”A Novel Decoder Based on Parallel Genetic Algorithms for Linear Block Codes”,International Journal of Communications, Network and System Sciences, 2013, 6, 66-76.

[13]. F.J. Macwilliams and N.J.A. Slone, ”The Theory of error correcting rrror”, North-Holland publishing comapny, 1978, pp. 567-580.

[14]. E.R. Berlkamp, R. J. McEliece, and H. C. A. Van Tilborg, ”On the inherent intractability of certain coding problems”, IEEE Transactions on InformationTheory, IT-24 :384-386, 1978.

[15]. A. Vardy, ”The intractability of computing the minimum distance of a code”, IEEE Transactions on Information Theory, IT-43 :1757-1766, 1997.

[16]. A. E. Eiben, E. H. L. Aarts, and K. M. Van Hee, ”Global convergence of genetic algorithms: A markov chain analysis”, in Parallel Problem Solvingfrom Nature H. -P. Schwefel and R. M¨anner (Eds), Berlin and Heidelberg : Springer, 1991, pp. 4-12.

[17]. D.B. Fogel, ”Evolving Arti cial Intelligence”, PhD dissert., San Diego University of California, 1992.

[18]. G. Rudolph, ”Convergence analysis of canonical genetic algorithms”, IEEE Trans. on Neural Networks, 5(1): 96-101, 1994.