Int. J. Communications, Network and System Sciences, **2013, 6, 66-76**
http://dx.doi.org/10.4236/ijcns.2013.61008 Published Online January 2013 (http://www.scirp.org/journal/ijcns)

A Novel Decoder Based on Parallel Genetic Algorithms for Linear Block Codes

Abdeslam Ahmadi[1], Faissal El Bouanani[2], Hussain Ben-Azza[1], Youssef Benghabrit[1]
[1]Department of Industrial and Production Engineering, Moulay Ismail University,
National High School of Arts and Trades, Meknès, Morocco
[2]Department of Communication Networks, National High School of Comptuer Science and
System Analysis, Rabat, Morocco
Email: ab_ahmadi@hotmail.com, elbouanani@ensias.ma, hbenazza@yahoo.com, you_benghabrit@yahoo.fr

Received November 8, 2012; revised December 5, 2012; accepted December 15, 2012

ABSTRACT

Genetic algorithms offer very good performances for solving large optimization problems, especially in the domain of error-correcting codes. However, they have a major drawback related to the time complexity and memory occupation when running on a uniprocessor computer. This paper proposes a parallel decoder for linear block codes, using parallel genetic algorithms (PGA). The good performance and time complexity are confirmed by theoretical study and by simulations on *BCH*(63,30,14) codes over both AWGN and flat Rayleigh fading channels. The simulation results show that the coding gain between parallel and single genetic algorithm is about 0.7 dB at BER = 10^{-5} with only 4 processors.

Keywords: Channel Coding; Linear Block Codes; Meta-Heuristics; Parallel Genetic Algorithms; Parallel Decoding Algorithms; Time Complexity; Flat Fading Channel; AWGN

1. Introduction

The error correcting codes began with the introduction of Hamming codes [1] in the same period that the remarkable work of Shannon [2]. They consist in correcting data corruption when saved in storage media (erasure CD/ DVD, etc.) or transmitted over noisy communication channel. **Figure 1** shows the canonical system diagram of numerical communication.

The encoder takes the information symbols and adds to them redundancy symbols, carefully chosen such that a maximum of errors, which are infiltrated throughout the process of signal modulation, noisy transmission channel, and demodulation, can be corrected. The resulting binary code word is then transmitted over the noisy and memoryless channel under assumption that the sending messages are independent from the added noise. At the reception, the decoder attempts, given channel observations and using the redundancy symbols, to find the most probable message. The techniques used by the decoder are diverse. The most optimal criterion is Maximum Likelihood [3]. Given its high complexity (computation time, memory occupation), other suboptimal techniques with acceptable performance are used in practice like Turbo codes [4] and LDPC [5]. Several other decoders developed were inspired from the artificial intelligence field as Han decoding which using algorithm A^* [6], Genetic Algorithms (GAs) [7] and Neural Networks [8]. In 2007, it was shown that decoders based on GAs have good performance and time complexity lower than those of classical algebraic decoders [9]. In 2008, the performance of these decoders has been further improved by using iterative decoding for product block codes in two dimensions [10]. In 2012, we made an extension of concatenated codes by passing to three dimensions [11]. In this last work, we have proposed two iterative decoding algorithms based on GAs, which can be applied to any arbitrary 3D binary product block codes, without the need of a Hard-In Hard-Out decoder.

The first decoder outperforms the Chase-Pyndiah [12] one. The second algorithm, which uses the *List-Based* SISO Decoding Algorithm (LBDA) based on order-*i* reprocessing [13], is more efficient than the first one. We have also showed that the two proposed decoders are less complex than both Chase-Pyndiah algorithm for codes with large correction capacity, and LBDA for large *i* parameter.

All these proposed decoders have been designed to run on a single processor machines. Their instructions are executed sequentially and then so slowly. In addition, we do not take advantage of the parallel nature of GAs, which is a major advantage related to their exploration and convergence. So, the aim of this new work is to overcome these drawbacks by applying Parallel Genetic Algorithms (PGAs) to decoding linear block codes. Thus,

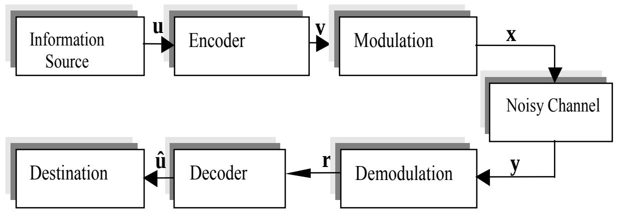

Figure 1. The canonical diagram of numerical communication.

we have developed a new decoder using parallel genetic algorithms, which runs on a parallel computer with multiple processors and a shared memory, or on a distributed system. So, it reduces the time complexity and increases little performance.

This paper is organized as follows. Section 2 presents some background on Linear Block Codes, Parallel System, and Genetic Algorithms. Section 3 describes the proposed Parallel Genetic Algorithms Decoder (PGAD), studies its time complexity and compares it with the one of SGAD. In Section 4 we discuss the simulation results obtained for PGAD. Finally, we give in Section 5 our conclusions and perspectives of this work.

2. Background

2.1. Linear Block Codes

Let $\mathbb{Z}_2 = \{0,1\}$ be the binary alphabet and $(\mathbb{Z}_2)^n$

The set of all words of length *n i.e.*:

$$(\mathbb{Z}_2)^n = \{x_1 \cdots x_n / x_1, \cdots, x_n \in \mathbb{Z}_2\}.$$

A linear code *C* of length *n* on $\mathbb{Z}_2$ is a subspace of the vector space $(\mathbb{Z}_2)^n$ [14]. *i.e.*:

$$\forall x, y \in C, \forall \lambda \in \mathbb{Z}_2, x + y \in C, \text{ and } \lambda x \in C.$$

Let $x, y \in C$ such that $x = \{x_1, \cdots, x_n\}$ and $y = \{y_1, \cdots, y_n\}$. The Hamming distance between *x* and *y*, noted $d_H(x, y)$, is defined by:

$$d_H(x, y) = \text{card}\{i / x_i \neq y_i, 1 \leq i \leq n\}.$$

The minimum Hamming distance *d* of a code *C* is the Smallest non zero distance between all its vectors, taken two by two. *i.e.*:

$$d = \min_{x,y \in C}\{d_H(x, y) / x \neq y\} \qquad (1)$$

Let $k = \dim(C)$ be the dimension of code *C*. *C* is then said $d(n,k,d)$-linear code. The code rate is the ratio k/n

Let $B = (b_1, \cdots, b_k)$ be a basis of *C*. Since $b_i \in C$, Then length (b_i) = *n*. The matrix *G* for which rows are the basis vectors is called generating matrix of the code *C*. It can be written as:

$$G = \begin{pmatrix} b_1 \\ \vdots \\ b_k \end{pmatrix} = \begin{pmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{k1} & \cdots & b_{kn} \end{pmatrix}$$

Thus, we have:

$$C = \{\alpha G / \alpha \in (\mathbb{Z}_2)^k\} \quad i.e. \quad \forall x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \in C,$$

$\exists! \alpha = (\alpha_1, \cdots, \alpha_k) \in (\mathbb{Z}_2)^k$ such that $x = \alpha G$ *i.e*:

$$\begin{cases} x_1 = \alpha_1 b_{11} + \cdots + \alpha_k b_{k1} \\ \qquad \vdots \\ x_n = \alpha +_1 b_{1n} + \cdots + \alpha_k b_{kn} \end{cases}$$

2.2. Parallel Systems

The choice of parallel machines or distributed systems is more imposed for applications requiring a very high processing power and/or a big memory space. There are plenty of parallel systems which are classified as below [15].

2.2.1. Taxonomy of Parallel Systems

These systems have been classified by Flynn (1972) according to two independent concepts: the *stream of instructions* and the *stream of data* used by these instructions. There are four possible combinations:

- SISD (Single Instruction, Single Data): The machine executes one instruction on one data at each clock cycle. It is not really a parallel machine but a classical computer (Von Newman).
- SIMD (Single Instruction Multiple Data): A processor having a single Control Unit (CU) and multiple Arithmetical and Logical Units (ALUs) executes the same instruction on different data at each clock cycle.
- MISD (Multiple Instruction, Single Data): Systems running multiple instructions on the same data at each clock cycle.
- MIMD (Multiple Instruction Multiple Data): These systems are multiple independent processors *i.e.* each processor has its CU and its ALU. At the same clock cycle, each processor executes a different instruction on a different data. These instructions can be synchronous or asynchronous. The majority of parallel systems are MIMD. The MIMD system can be either a Multiprocessor or a Multi-computer or a hybrid of them. We explain in the next subsections the architecture of each one.

2.2.2. Multiprocessor

It is a MIMD parallel system with certain number of autonomous processors (CU+ALU) and a single shared memory. *i.e.* All processors use a common physical memory which composed of several memory blocks **Figure 2**. These blocks are interconnected with the processor and between them [15].

If two processors want to communicate, it suffices that the first one writes data (or message) in the memory and the other one recuperates it. So their programming is easy since the programmer does not have to focus on the explicit communication (message exchange) between the different processors. These allowed multiprocessors making a great success. However, it would be difficult to build a multiprocessor when the number of its communicating elements (processors and memory blocks) is very important. Indeed, the interconnection of all these elements is not always an easy task.

2.2.3. Multi-Computer

A multi-computer is also a MIMD system but relatively simple to build. It contains a number of computers (CU + ALU + private memory) linked together by an interconnection network. Each computer has its own memory **Figure 3**. This reduces significantly the number of elements to be interconnected. The communication between computers is achieved by exchanging messages as primitives Send/Receive, programmed and integrated by the programmer in its application [15]. This may complicate more his task. The following points must be taken into account for each parallel programming:

- For an efficient communication between different elements (processors and memory blocks) of a parallel or distributed system, the suitable choice of the topology, routing and switching mode of its interconnection network affects its performance remarkably.
- To benefit from the advantages of a multiprocessor and those of a multi-computer and minimize the disadvantages of each one of them, hybrid systems have been designed.
- To take advantage of parallel systems, we must execute on them parallel programs or containing a significant part of parallel instructions.

2.3. Genetic Algorithms

The Genetic Algorithm (GA), initiated in 1970 by Holland [16] is an Evolutionary Algorithm (EA) inspired from the natural biological evolution. It use search stochastic techniques to solve problems not having an analytical resolution or when the time to resolve them by classical algorithms is not reasonable [17]. Their applications cover, in addition to error correcting codes in telecommunication, several other fields as optimization, artificial intelligence, economic markets, etc. As shown in algorithm below [18], an EA especially a GA has an iterative character:

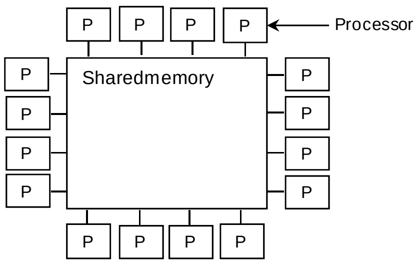

Figure 2. A multiprocessor with 16 processors sharing a commonmemory.

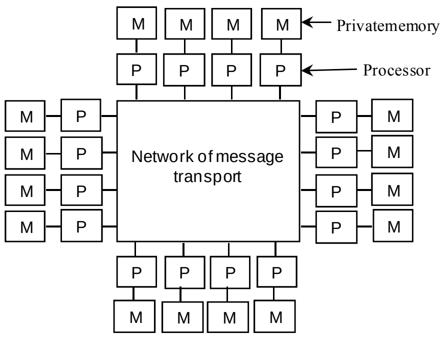

Figure 3. A multi-computer with 16 interconnected processors where each one has its own memory.

```
t=0;
initialize and evaluate [P(t)] ;
while not stop-condition do
P'(t)=variation [P(t)] ;
Evaluate [P'(t)] ;
P(t+1) =select [P'(t), P(t)];
t=t+1;
end while.
```

It generates an intermediate population $P'(t)$ by applying variation operators (often stochastic) on individuals (which may be here, information sequences or codewords) of the current population $P(t)$. Then, it evaluates the quality (of being solution) of individuals from $P'(t)$ using a criterion known as *fitness*. It finally creates the new population $P(t+1)$ in which individuals are selected from those of $P'(t)$ and eventually from $P(t)$. The process is repeated until the stopping condition is satisfied. In general, the process is stopped when the optimal solution is found or when the maximum number of iterations is reached.

The Genetic Algorithms can be singles (SGA) or Parallels (PGA).

2.3.1. Single Genetic Algorithms

The previous general algorithm can be adapted in SGA as follows:

```
Generate an initial population of n individuals randomly;
while not stop-condition do
Calculate the fitness f (x) for any individual x;
```

```
while not Filled-New-Population do
    Select two parent individuals;
    Cross them to have a new(s) individual(s);
    The new individuals undergo eventual mutations;
    Insert the new individual in the new population;
end while
Replace the old population by the new one;
end while.
```

The three classical operators inspired from natural evolution to generate a new population from the current one in both SGA and PGA are selection, crossover and mutation:

- The selection operator describes how to select the parents in the current population to cross them and generate new individuals (offspring) which will be inserted in the new population. The individuals are sorted in ascending order of their fitness to give more chances to the best ones (fittest) to be selected (assuming that better parents reproduce better children). However, there are cases where the crossing of bad parents can generate good offspring. The most popular selection methods are: Proportional, Linear Ranking, Whitley's Linear Ranking and Uniform Ranking [19].
- The crossover operator generates new individuals inheriting from their parents. *i.e.* Their genes are a mixture of those of their parents. There are several crossing techniques that can be either generic (robust) *i.e.* applied in a wide variety of problems, or specific to particular problems, or hybrid techniques combining generic and specific ones.
- As in natural evolution, the genes of some individuals may undergo changes (mutations). This occurs in very rare cases. The mutation plays a very important role in the convergence of SGA and PGA algorithms. Indeed, it will prevent their premature convergence by guiding them to explore other more promising areas and avoid a local optimum.

2.3.2. Parallel Genetic Algorithms

The PGAs are GAs running on parallel systems discussed in the second subsection.Their purpose is not only accelerating the convergence and/or use a large memory space but also improve the performance. There are two models of PGAs: 1) Island model (or network model) which runs an independent GA with a sub-population on each processor, and the best individuals are communicated either to all other sub-populations or to neighboring population [20]; 2) Cellular model (or neighborhood model) which runs an individual on each processor, and cross with the best individual among its neighbors [21]. In fact, the second model is a particular case of the first one. Indeed, its population is reduced to a single individual on each processor.

The general algorithm of PGAs is given below:

```
Generate an initial population of n individuals randomly;
while not stop-condition do
    Calculate the fitness f(x) for any individual x;
    if Interval-Exchange then
        Exchange of individuals;
    end if
    while not Filled-New-Population do
        Select individual parents;
        Cross them to make new people;
        New individuals undergo eventual mutations;
        Insert new individuals in the new population;
    end while.
    Replace the old population by the new one;
end while.
```

3. The Proposed Parallel Decoder Based on Genetic Algorithms

This work is a parallelization of the decoder that we have already used in [9,10], and [11] by exploiting some techniques of parallel genetic algorithms used in other domains like Optimization and Artificial Intelligence. It can be run on a multiprocessor where the data exchanged are saved in global or shared variables, or on a multi-computer which exchange data between its processors via a network.

Let $F=(F_1,\cdots,F_n)$ and $R=(R_1,\cdots,R_n)$ be respectively the fading vector and the received sequence (associated to the transmitted sequence) at the decoder input of a binary linear block code $C(n,k,d)$ with a generator matrix G. The parameters N_p, N_e, N_c, N_m, N_s, N_g are respectively the population size, elite number, offspring number, migrant number, processor number and maximum number of generations such that $N_m=(N_p-N_e-N_c)/(N_s-1)$ is a positive integer.

3.1. The PGAD Algorithm

The sub-algorithm which will run in parallel on each processor is given below:

Step 0: Initialization

- *Sort the elements of the received vector R in descending order of their magnitude to produce another vector* $R^{(1)}$ *i.e. find a permutation* π_1 *such that* $R^{(1)}=\pi_1(R)$ *and* $|R_1^{(1)}|\ge|R_2^{(1)}|\ge\cdots\ge|R_n^{(1)}|$. *This will put reliable elements in the first ranks.* $F^{(1)}$ *is the permutation of F by* π_1 *i.e.* $F^{(1)}=\pi_1(F)$. *Then, permute G by* π_2 *to produce G' such that the first k columns of G' are linearly independent i.e.* $G'=\pi_2(G)$. *The two vectors* $R^{(1)}$ *and* $F^{(1)}$ *are permuted by* π_2 *to R' and F'. So,* $R'=\pi_2(R^{(1)})=\pi_2(\pi_1(R))=\pi(R)$ *and*

$F' = \pi_2\left(F^{(1)}\right) = \pi_2\left(\pi_1\left(F\right)\right) = \pi\left(F\right)$, *with* $\pi = \pi_2 \circ \pi_1$.

- *Quantize the first k bits of* R' *to obtain binary vector r and randomly generate* $N_p - 1$ *information vectors of k bits each one. These vectors form with vector r the initial population of* N_p *individuals* $\left(I_1, \cdots, I_{Np}\right)$;

Step* 1: *Reproduction

gen←0 (gen is the current generation number).

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

- *Encode individuals of the current population, using* G' *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' :

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

- *Sort the current population individuals in descending order of their fitness*;
- *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*;
- *Cross with probability* p_c *the selected parents to generate* N_c *new individuals*;
- *Mutate the new individuals with a probability* p_m *if their parents are crossed*;
- *Insert these* N_c *new individuals with their fitness in the new population* ***Figure* 4**;
- *Send the best* $N_m = \left(N_p - N_e - N_c\right) / \left(N_s - 1\right)$ *individuals (with their fitness) to the new populations of* $N_s - 1$ *other processors as shown in* ***Figure* 5**;
- *Receive* N_m *migrant elites (with their fitness) from the previous population of each* $N_s - 1$ *other processors, to complete the new population as shown in* ***Figure* 6**;
- *Replace the current population with the new one*;
- *gen←gen+*1;

end while

Step* 2: *Decision

Get the best individuals $\left(D'^{(i)}\right)_{1 \le i \le N_s}$ *of last populations of all processors. The best one* D' *of them is the closest to* R'. *i.e.* $D' = \operatorname{argmin}\left\{\left\|D'^{(i)} - R'\right\|, 1 \le i \le N_s\right\}$. *So, the decided codeword is* $D = \pi^{-1}\left(D'\right)$.

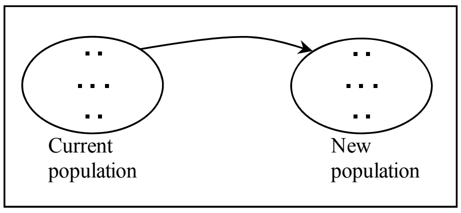

Figure 4. The *i*th processor which works on the current population to give the next one.

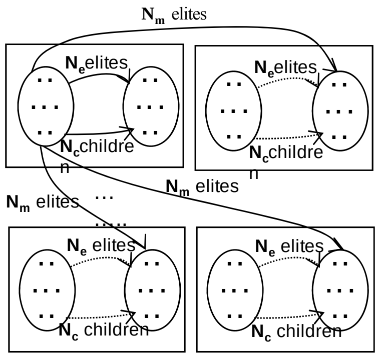

Figure 5. In a given generation, the *i*th processor sends its best N_m individuals to other processors.

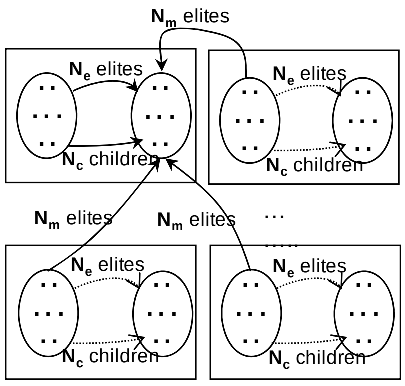

Figure 6. In a given generation, the *i*th processor receives the best N_m individuals from each other processor.

The flowchart of the previous algorithm is illustrated in **Figures 7** and **8**.

3.2. The PGAD Characteristics

The main elements characterizing a basic PGA are [19]: evolution mode, reproduction operators (crossover and mutation), selection and replacement policy of individuals (local and foreign) and migration.

3.2.1. Evolution Mode

It defines the granularity of the evolution step in a sub-algorithm of the PGA. *i.e.* how the new population is created from the current one. There are two techniques that are often used. The first one is the generational GA (GGA) where the new population replaces all the old ones. The second technique inserts only a few new (generated) individuals in the current population. In our implementation, we have used the GGA evolution mode with elitism.

3.2.2. Reproducing Operators

To generate N_c individuals $\left(I_i\right)_{N_e+1 \le i \le N_e+N_c}$ in the new population, we have used, like in our works [10], and [11], these operators:

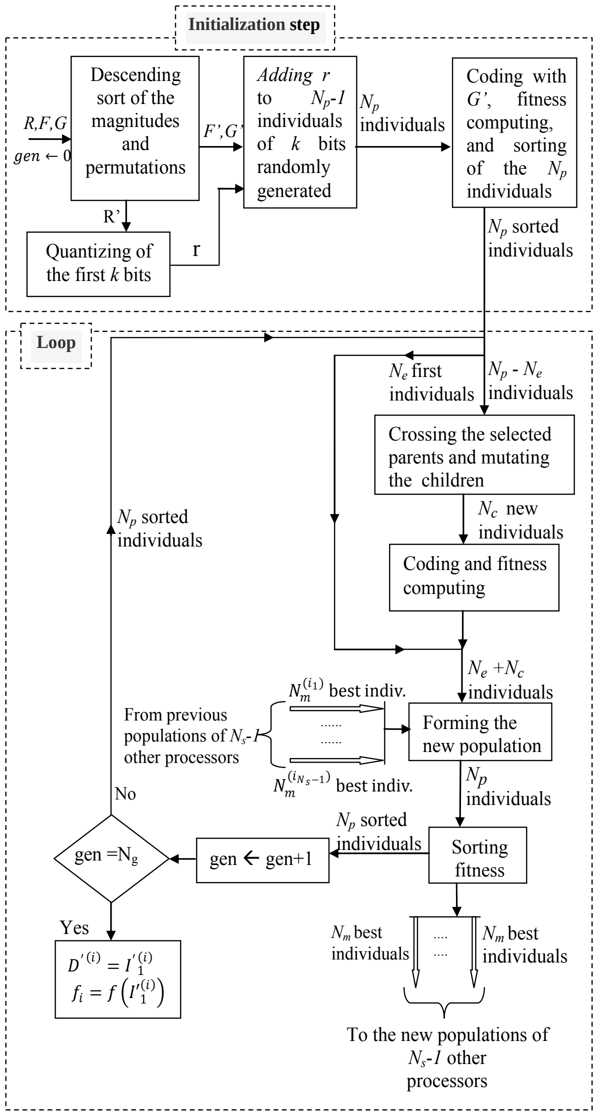
Initialization step
R,F,G
gen ← 0
Descending sort of the magnitudes and permutations
F',G'
Adding r to Np-1 individuals of k bits randomly generated
Np individuals
Coding with G', fitness computing, and sorting of the Np individuals
R'
Quantizing of the first k bits
r
Np sorted individuals
Loop
Ne first individuals
Np - Ne individuals
Crossing the selected parents and mutating the children
Nc new individuals
Coding and fitness computing
Np sorted individuals
Ne +Nc individuals
best indiv.
From previous populations of Ns-1 other processors
Forming the new population
best indiv.
Np individuals
No
Np sorted individuals
gen =Ng
gen ← gen+1
Sorting fitness
Yes
Nm best individuals
Nm best individuals
To the new populations of Ns-1 other processors

Figure 7. The flowchart of the proposed PGAD sub-algorithm running on the *i*th processor.

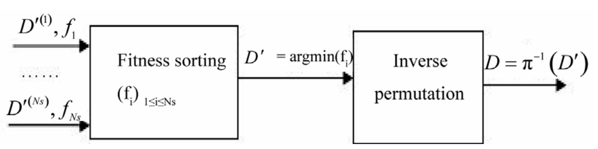

Figure 8. The codeword decision flowchart of the proposed PGAD algorithm.

- Selection of two individuals as parents (P,Q) using the following linear ranking:

$$\text{weight}_i = \frac{\text{weight}_{\max} - 2(i-1)(\text{weight}_{\max} - 1)}{N_p - 1}, \quad (2)$$
$$\forall i \in \{1, \cdots, N_p\}.$$

where weight$_i$ is the ith individual weight and weight$_{\max}$ is the weight assigned to the fittest (closest) individual.

- Let p_c, p_m be respectively, the probabilities of crossover and mutation, and let Rand be a uniformly random value between 0 and 1, generated at each time.

if Rand $< p_c$ *then*

$$\forall i \in \{N_e + 1, \cdots, N_e + N_c\}, j \in \{1, \cdots, k\}:$$

$$I_{ij} = \begin{cases} P_j & \text{if Rand} < \left(1 - P_j + P_j Q_j\right) + \dfrac{P_j - Q_j}{1 + e^{\frac{-4R'_j F'_j}{N_0}}} \\ Q_j & \text{else} \end{cases} \quad (3)$$

and then

$I_{ij} \leftarrow 1 - I_{ij}$ if Rand $< p_m$

else

$$I_i = \begin{cases} P & \text{if Rand} < 0.5 \\ Q & \text{else} \end{cases}$$

end if.

We note that on an AWGN (Additive White Gaussian Noise Channel) channel, we have

$$F'_j = 1, \forall j \in \{1, \cdots, k\}.$$

3.2.3. Selection and Replacement Strategies

They define the selection of individuals to replace in the current population of a sub-algorithm PGA and those to migrate from other processors. The individuals can be replaced by new local ones or by the migrant ones:

- The main replacement policies are [19]: 1) *Inverse Proportional*; where the worst individual is more likely to be replaced; 2) *Uniform Random*; where all individuals have the same chance of being replaced; 3) *Worst*; where the worst individual is always replaced. In some problems, it is replaced only if it is worse than the new one; 4) *Generational*; where the entire current population is replaced by a new one. In our work, we keep the best local individuals (elitism) and the bad ones are replaced by new local offspring and by the migrants from other processors.
- There are two cases to choose migrant individuals: choose the best ones, or choose them randomly. In our algorithm, we chose the best migrants.

3.2.4. Migration

This is an operation that helps to diversify and enrich the new population of a sub-algorithm which runs in a processor by migrant individuals from other cooperator processors of the PGA. There are two parameters characterizing the migration operation: Migration Gap and Migration Rate. The first one specifies the frequency of exchange of individuals. *i.e.* The number of steps that a sub-algorithm must run before sending or receiving migrant individuals. It can also specify the probability of migration at each stage. The second migration parameter defines the number of exchanged individuals (migrants).

The exchange (send/receive) of individuals is done in two modes: synchronous or asynchronous.

The synchronous mode suspends, periodically, the execution of a sub-algorithm of the PGA and waits for the receiving migrant individuals from other nodes before continuing its execution. In asynchronous mode, the sub-algorithm does not wait. Once the migrant individuals arrive, it deals with them.

In our algorithm, we send/receive at each new population the $N_m = \left(N_p - N_e - N_c\right) / \left(N_s - 1\right)$ first best individualsto/from every other processor according to the asynchronous mode. To reduce the number of messages that flow through the network connecting processors, the number of migrant individuals should be less than the number of individuals locally generated for each new population. *i.e.* $N_m \le N_e + N_c$.

3.2.5. Heterogeneity

A PGA is called homogeneous if its nodes run the same type of algorithm. Otherwise, it is called heterogeneous. Our PGAD is homogeneous. Indeed, all nodes are symmetric and perform the same genetic algorithm having same parameters and operators.

3.3. PGAD Time Complexity

We give in **Table 1** the complexity of the different stages of the proposed decoding algorithm PGAD in the order oftheir appearance in the flowcharts of **Figures 7** and **8** before deducting its global complexity.

We note that:

- The crossover operation will be made almost always, since its probability is close to 1. So its time complexity is $O(kN_c)$.
- The complexity of mutation operation is neglected. Indeed, this operation will occur rarely since its probability is close to 0.

The total time complexity of PGAD is then:

Table 1. Time complexities of different steps of the PGAD.

Initialization step	**Time complexity**
Descending sort of the magnitude of R	$O(n\ln n)$
Quantizing the first k bits	$O(k^2n)$
Coding with G', fitness computing, and sorting of the N_p individuals	$O(knN_p) + O(nN_p) + O(N_p\ln N_p)$
Loop steps	**Time complexity**
Crossover and mutation of information vectors	$O(kN_c)$
Coding and fitness computing	$O(knN_c) + O(nN_c)$
Sorting of N_p individuals	$O(N_p\ln N_p)$
Decision step	**Time complexity**
Sorting of $D'^{(i)}$, $1 \le i \le N_s$	$O(N_s\ln N_s)$

$$\begin{aligned}
&O\Big(n\ln n + k^2n + knN_p + nN_p + N_p\ln N_p \\
&+\left(N_g - 1\right)\left(kN_c + knN_c + nN_c + N_p\ln N_p\right) + N_s\ln N_s\Big) \\
&= O\Big(n\ln n + k^2n + kn\left(N_p + N_gN_c\right) \\
&\qquad + N_gN_p\ln N_p + N_s\ln N_s\Big)
\end{aligned} \tag{4}$$

From the formula (4), it is clear that the complexity of PGAD is lower than that of SGAD. Indeed, there are terms that are common to both decoders (initialization and sorting at the end of the loop). The term $N_s\ln N_s$ can be neglected when the processor number is small. The difference between SGAD and PGAD consists in the crossing, coding, and fitness computing in the loop. For each generation, we must cross parents to have $N_p - N_e$ new individuals in the case of SGAD against only N_c in PGAD ($N_c < N_p - N_e$). Likewise we encode in the loop only N_c individuals in our decoder against $N_p - N_e$ in SGAD. For the fitness in the loop, it is also computed for N_c individuals instead of $N_p - N_e$.

From the foregoing, we can summarize the advantages of our decoder as follows:

- Improvement of performance. Indeed, It corrects errors better than simple algorithms studied in [9,10] as shown in simulation section.
- Reducing the time complexity of the decoding process: 1) It is run in parallel on multiple processors for almost the same number or lower of total individuals (N_gN_p) used in a single decoder; 2) It reduces the time of encoding and fitness computing, since the algorithm receives ($N_p - N_e - N_c$) migrant individuals (encoded) with their fitness already computed; 3) It reduces the reproduction time since the number of parents to cross is reduced to $N_c(N_c < N_p - N_e)$.

4. Simulation Results

Our PGAD is run on 4 processors ($N_s = 4$). To study the performance of the presented PGAD decoder, we have simulated a binary communication system with BPSK modulation and both AWGN and Rayleigh fading channels. We give in this section, the impact of each parameter N_g, N_p, p_c, p_m, N_e, N_c and code rate on the performance of our decoder and we finish with a comparison with the SGAD decoder. The chosen code is the linear block *BCH*(63,30,14) code. The minimum number of sent erroneous frames is 30. The performance will be given as figures showing the Bit Error Rate (BER) versus the energy per bit to noise power spectral density ratio E_b/N_0.

The figures corresponding to each channel are given in **Table 2**.

4.1. Effect of Generation Number and Population Size

Generally, increasing the number of evaluated code-

Table 2. Simulation figures corresponding to each channel.

AWGN channel	Fading channel
Figures 1 to **14**	**Figure 15**

words N_pN_g, the probability to find the codeword closest to the input sequence becomes high. This makes it possible to improve the BER performances. The effect of increasing the number of evaluated code words on the *BER* improvement for code *BCH*(63,30,14) at the 12th iteration is presented in **Figures 9** and **10**. The values N_g = 100 and N_p = 100 can be the optimal values in a large range E_b/N_0. The other genetic parameters for the first optimization are: $N_p = 100$, $N_c = 80$, $N_e = 5$, $p_c = 0.99$, $p_m = 0.03$ and $N_g = 100$, $N_c = 80$, $p_c = 0.99$, $p_m = 0.03$ for the second one.

4.2. Crossover Rate Effect

The crossover is a very important operation insofar as it allows a large exploration, and an effective exploitation. In Indeed, it creates new individuals may be good solutions to the problem. For most problems, the probability of crossover is high. This is the case also for the error correcting. The **Figure 11** shows that among the studied probabilities p_c= 0.99 offers the best performance. For this simulation, we have fixed the other parameters as follows: N_g = 10, N_p = 100, N_c = 80, N_e = 5, and p_m = 0.03.

4.3. Mutation Rate Effect

The effect of mutation rate for *BCH*(63,30,14) is depicted in **Figure 12**. It is shown that $p_m = 0.03$ is the optimal BER value for all SNRs. One reason of this value close to 0 may be the stability of members in vicinity of optima for low mutation rates. The fixed values are: N_g = 100, N_p = 100, N_c = 80, N_e = 5, and p_c = 0.99.

4.4. Elite Number Effect

Among the best individuals of the current population, some may survive and move to the new population. This can be justified by the fact that their elitism may give birth to other best descendants. As shown in the **Figure 13**, the greater the number of elites survived, more performances improve. However, when we exceed five elites the performances begin to decline. We deduce that worse individuals can also create, by crossover and mutation, better individuals. So we choose N_e = 5. The fixed values for this simulation are: N_g = 100, N_p = 100, N_c = 80, p_c = 0.99 and p_m = 0.03.

4.5. Offspring Number and Migration Rate Effects

The number of individuals migrating from each nodeis

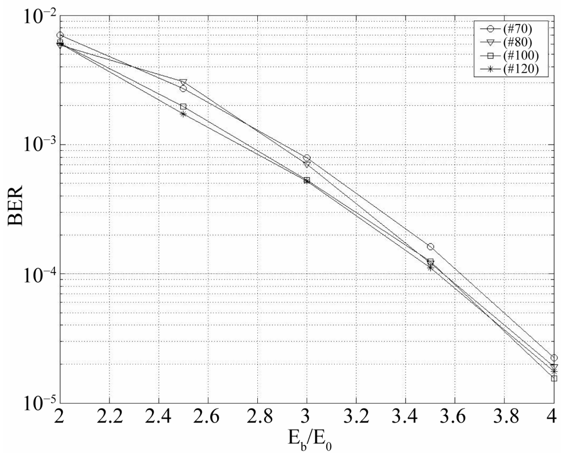

Figure 9. Effect of the generation number for *BCH*(63,30,14).

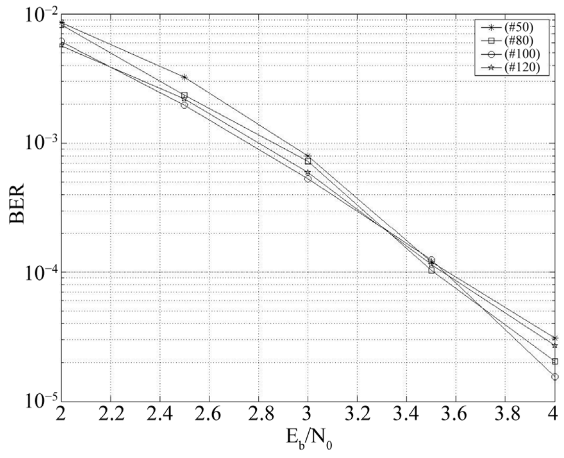

Figure 10. Effect of the population size for *BCH*(63,30,14).

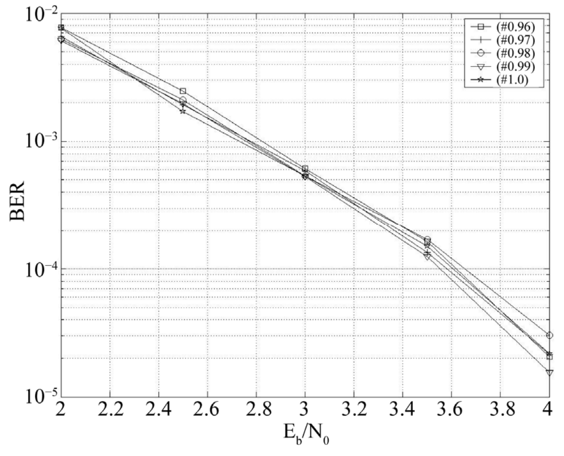

Figure 11. Effect of the crossover probability for *BCH*(63,30,14).

$(N_p - N_e - N_c)/(N_s - 1) = (9 - N_c)/3$. So when this number decreases, the number of individuals created locally N_c increases. From the curves plotted in **Figure 14**, performances increase when N_c increases up to a threshold

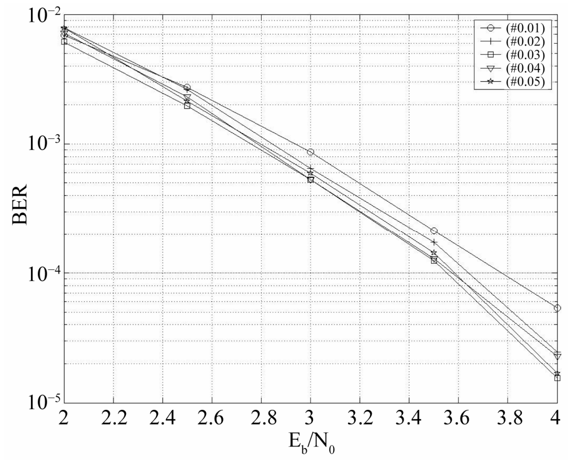

Figure 12. Effect of the mutation rate for *BCH*(63,30,14).

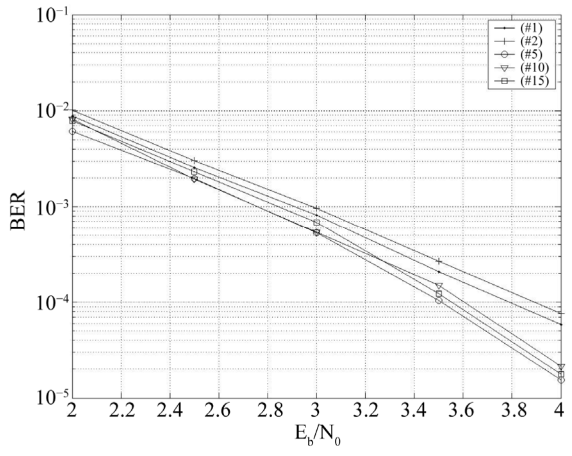

Figure 13. Effect of the elite number for *BCH*(63,30,14).

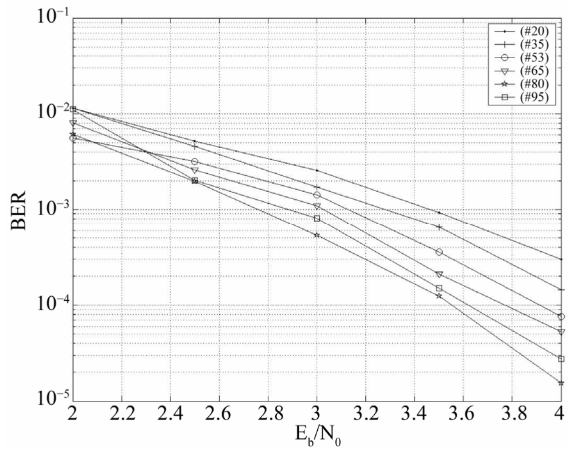

Figure 14. Effect of the offspring number for *BCH*(63,30,14).

$N_c = 80$. We also note that when there is no ex- change of individuals ($N_c = 95$ *i.e.* the number of migrant individuals is equal to 0) the performance is worse than with the exchange. The fixed values of parameters are: $N_g = 100$, $N_p = 100$, $N_e = 5$, $p_c = 0.99$ and $p_m = 0.03$.

4.6. Code Rate Effect

We have studied the performance of the following BCH codes: (63,30), (63,39), (63,45), and (63,51). From the **Figure 15**, we note that when the rate decreases, performance increases. This is explained by the fact that when the code length *n* increases for the same dimension *k*; the number of redundancy bits (check) increases. *i.e.* it corrects better with more redundancy; which makes sense. In this simulation, we adopted the optimal values in **Table 3** previously found.

4.7. Performance Comparison between PGAD and SGAD

We have studied the performance of our PGAD on a Gaussian channel AWGN and flat Rayleigh fading channel with flat fading using the same parameters in **Table 3**. We remark from **Figures 16** and **17** that the parallel decoder not only reduces the time complexity, but also has good performance compared to simple decoder studied in [9] and just for 4 processors.

5. Conclusion

We have presented a new decoder based on parallel genetic algorithms which can be applied to any linear block code. The simulations show that it provides better performance than a decoder based on SGA and the gain is

Table 3. Time complexities of different steps of the PGAD.

N_g	100
N_p	100
N_e	5
N_c	80
p_c	0.99
p_m	0.03

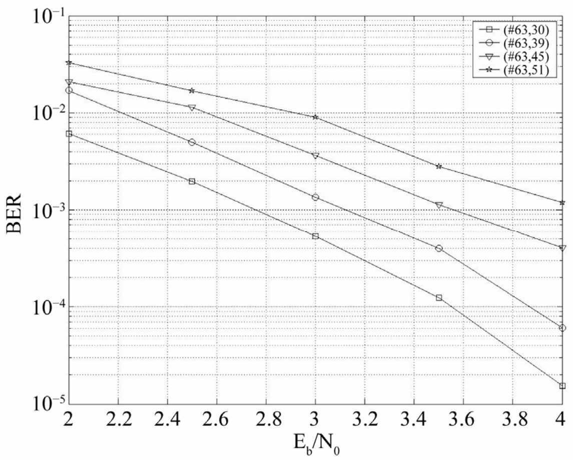

Figure 15. Effect of code rate.

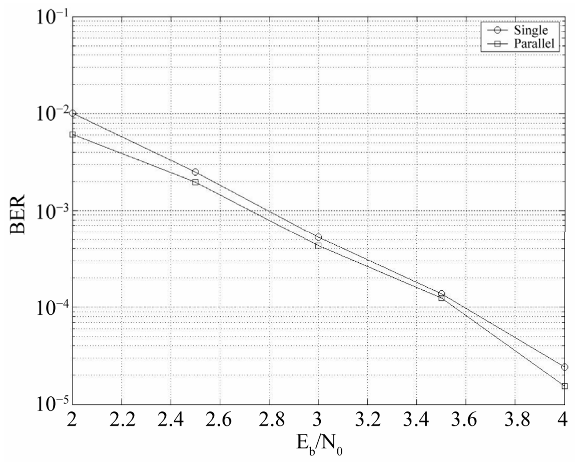

Figure 16. Performance comparison between SGAD and PGAD on AWGN channel for *BCH*(63,30,14).

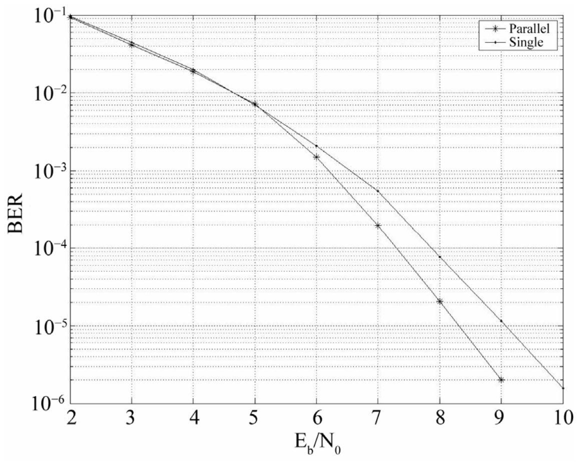

Figure 17. Performance comparison between SGAD and PGAD on Rayleigh channel for *BCH*(63,30,14).

about 0.7 dB with 4 processors only. This is due to its parallel architecture which can exploit and explore more individuals and avoid the premature convergence to local optimum. In addition, this decoder has a lower complexity because it runs on multiple processors simultaneously, and because it reduces the time of encoding and fitness computing, and the time of generation of new individuals. Its performance can be further improved by adjusting algorithm parameters and characteristics (processor number, topology, migration, selection and replacement, ...) and by parallelizing tasks running on the same processor, or by putting it in hybrid with other decoders. We could also envisage an iterative decoding based on PGAs.

REFERENCES

[1] R. W. Hamming, "Error Detecting and Error Correcting Codes," *Bell System Technical Journal*, Vol. 29, No. 2, 1950, pp. 47-160.

[2] C. E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, Vol. 27, 1948, pp. 379-423, 623-656.

[3] S. Roman, "Introduction to Coding and Information Theory," Spring Verlag, New York, 1996.

[4] C. Berrou and A. Glavieux, "Near Optimum Error Correcting Coding and Decoding: Turbo Codes," *IEEE Transactions on Communications*, Vol. 44, No. 10, 1996, pp. 1261-1271. doi:10.1109/26.539767

[5] D. J. C. Mackay and R. M. Neal, "Good Codes Based on Very Sparse Matrices," 5*th IMA Conference on Cryptography and Coding*, Lecture Notes in Computer Science number 1025, October 1995, Springer, Berlin, pp. 100-111. doi:10.1007/3-540-60693-9_13

[6] Y. S. Han, C. R. P. Hartmann and C.-C. Chen, "Efficient Maximum Likelihood Soft-Decision Decoding of Linear Block Codes Using Algorithm A^*," Technical Report SU-CIS-91-42, Syracuse University, Syracuse, 1991.

[7] H. S. Maini, K. G. Mehrotra, C. Mohan and S. Ranka, "Genetic Algorithms for Soft Decision Decoding of Linear Block Codes," *Journal of Evolutionary Computation*, Vol. 2, No. 2, 1994, pp. 145-164. doi:10.1162/evco.1994.2.2.145

[8] J. L. Wu, Y. H. Tseng and Y. M. Huang, "Neural Networks Decoders for Linear Block Codes," *International Journal of Computational Engineering Science*, Vol. 3, No. 3, 2002, pp. 235-255. doi:10.1142/S1465876302000629

[9] F. El Bouanani, H. Berbia, M. Belkasmi and H. Ben-Azza, "Comparison between the Decoders of Chase, OSD and Those Based on Genetic Algorithms," 21 Colloquium of GRETSI (Group of Study and Signal and Pictures Processing), Troyes, 11-14 September 2007, pp. 1153-1156.

[10] M. Belkasmi, H. Berbia and F. El Bouanani, "Iterative Decoding of Product Block Codes Based on the Genetic Algorithms," 7*th International ITG Conference on Source and Channel Coding* (*SCC*'08), Ulm, 14-16 January 2008, pp. 1-6.

[11] A. Ahmadi, F. El Bouanani, H. Ben-Azza and Y. Benghabrit, "Reduced Complexity Iterative Decoding of 3D-Product Block Codes Based on Genetic Algorithms," *Journal of Electrical and Computer Engineering*, Vol. 2012, 2012, Article ID: 609650.

[12] R. Pyndiah, A. Glavieux, A. Picart and S. Jacq, "Near Optimum Decoding of Product Codes," *IEEE Proceedings of Global Telecommunications Conference*, San Francisco, 28 November-2 December 1994, Vol. 1, pp. 339-343.

[13] P. A. Martin, D. P. Taylor and M. P. C. Fossorier, "Soft-Input Soft-Output List-Based Decoding Algorithm," *IEEE Transactions on Communications*, Vol. 52, No. 2, 2004, pp. 252-264.

[14] S. Lin and D. Costello, "Error Control Coding: Fundamentals and Applications," Prentice-Hall, Upper Saddle River, 1983.

[15] A. Tanenbaum, "Computer Architecture," Dunod, Paris, 2001.

[16] J. H. Holland, "Adaptation in Natural and Artificial Sys-

tems," University of Michigan Press, Ann Arbor, 1975.

[17] D. E. Goldberg, "Genetic Algorithms in Search, Optimization, and Machine Learning," Addison-Wesley, Boston, 1989.

[18] E. Alba and M. Tomassini, "Parallelism and Evolutionary Algorithms," *IEEE Transactions on Evolutionary Computation*, Vol. 6, No. 5, 2002, pp. 443-462.

[19] E. Alba and J. M. Troya. "A Survey of Parallel Distributed Genetic Algorithms," *Complexity*, Vol. 4, No. 4, 1999, pp. 31-52. doi:10.1002/(SICI)1099-0526(199903/04)4:4<31::AID-CPLX5>3.0.CO;2-4

[20] T. Starkweather, D. Whitley and K. Mathias, "Optimization Using Distributed Genetic Algorithm," Springer Verlag, New York, 1991, pp. 176-185. doi:10.1007/BFb0029750

[21] V. S. Gordon and D. Whitley, "A Machine-Independent Analysis of Parallel Genetic Algorithms," *Complex Systems*, Vol. 8, 1994, pp. 181-214.