Low-Complexity Decoding for Non-Binary LDPC Codes in High Order Fields

Adrian Voicila, David Declercq, Francois Verdier, Marc Fossorier, Pascal Urard

Low-complexity decoding for non-binary LDPC codes in high order fields

Adrian Voicila[*‡], David Declercq[‡], François Verdier[‡], Marc Fossorier[†], Pascal Urard[*]

[‡]ETIS ENSEA/UCP/CNRS UMR-8051 95014 Cergy-Pontoise, (France)

[†]Dept. Electrical Engineering Univ. Hawaii at Manoa Honolulu, HI 96822, (USA)

[*]STMicroelectronics Crolles, (France)

Abstract

In this paper, we propose a new implementation of the Extended Min-Sum (EMS) decoder for non-binary LDPC codes. A particularity of the new algorithm is that it takes into accounts the memory problem of the non-binary LDPC decoders, together with a significant complexity reduction per decoding iteration. The key feature of our decoder is to truncate the vector messages of the decoder to a limited number n_m of values in order to reduce the memory requirements. Using the truncated messages, we propose an efficient implementation of the EMS decoder which reduces the order of complexity to $\mathcal{O}(n_m \log_2 n_m)$. This complexity starts to be reasonable enough to compete with binary decoders. The performance of the low complexity algorithm with proper compensation is quite good with respect to the important complexity reduction, which is shown both with a simulated density evolution approach and actual simulations.

Index Terms

Iterative decoding, non-binary LDPC codes, low complexity algorithm

I. Introduction

It is now well known that binary low density parity check (LDPC) codes achieve rates close to the channel capacity for very long codeword lengths [1], and more and more LDPC solutions have been proposed in standards (DVB, WIMAX, etc). In terms of performance, binary LDPC codes start to show their weaknesses when the code word length is small or moderate, or when

higher order modulation is used for transmission. For these cases, non-binary LDPC (NB-LDPC) codes designed in high order Galois fields have shown great potential [2], [3], [4], [5].

However, the performance gain provided by LDPC codes over GF(q) comes together with a significant increase of the decoding complexity. NB-LDPC codes can be decoded efficiently with message passing algorithms as the belief propagation (BP) decoder, but the size of the messages varies in the order q of the field. Therefore, a straightforward implementation of the BP decoder has complexity in $\mathcal{O}(q^2)$. A Fourier domain implementation of the BP is possible like in the binary case, reducing the complexity to $\mathcal{O}(q \log q)$ [2], [6], but this implementation is only convenient for messages expressed in the probability domain. This is a problem since several authors have identified that the use of log-density-ratios (LDR) representation is mandatory to avoid complicated operations like multiplications and divisions. Any LDR-based implementation of the BP requires also $q-1$ values per message in the graph.

In this paper, we propose a new decoding algorithm for NB-LDPC codes. Our algorithm has both low computing complexity and reduced storage requirements, and therefore becomes a good solution for hardware implementation.

In one of the algorithms presented in [7] the authors introduced the idea of using only a limited number n_m of reliabilities in the messages at the input of the check node in order to reduce the computational burden of the check node update. The complexity at each check node was reduced to the order of $\mathcal{O}(n_m\ q)$, and the same memory storage complexity as BP was needed. In this paper, we keep the basic idea of using only $n_m \ll q$ values for the computation of messages, but we extend the principle to all the messages in the Tanner graph, that is, both at the check nodes and the variable nodes input. Moreover, we propose to store only n_m reliabilities instead of $q-1$ for each message. The truncation of messages from $q-1$ to n_m values has to be done in an efficient way in order to reduce its impact on the performance of the decoder. The truncation technique that we propose is described in details in Section III, together with an efficient offset correction to compensate the performance loss. Using the truncated messages representation, and a recursive implementation of the check node update, we propose a new implementation of the Extended Min-Sum (EMS) decoder whose complexity is dominated by $\mathcal{O}(n_m \log n_m)$, with $n_m \ll q$. This is an important complexity reduction compared to all existing methods [7], [8], [9]. Our new algorithm is developed in Section IV and a study of its complexity/performance trade-off is presented in Section V. Section VI is dedicated to non-binary adaptation of the shuffled scheduling for the special class of cycle codes. In Section VII the robustness of the

algorithm to the effects of a finite precision representation of messages is studied. In Section VIII-A, the simulation results verify that the proposed low complexity decoder still performs very close to the BP decoder that we use as benchmark. We conclude the paper in section VIII-A by a fair comparison between the proposed non-binary decoding algorithm and the binary corrected Min-Sum (MS) algorithm [10] applied to binary irregular LDPC codes, in terms of computational complexity and error performance.

II. Preliminaries

An NB-LDPC code is defined by a very sparse random parity check matrix H, whose components belong to a finite field GF(q). The matrix H consists of M rows and N columns; the code rate is defined by $R \leq \frac{N-M}{N}$. Decoding algorithms of LDPC codes are iterative message passing decoders based on a factor (or Tanner) graph representation of the matrix H [11]. In general, an LDPC code has a factor graph consisting of N variable nodes and M parity check nodes with various degrees. To simplify the notations, we will only present the decoder equations for isolated nodes with given degrees. We denote d_v the degree of a symbol node and d_c the degree of a check node. In order to apply the decoder to irregular LDPC codes, simply let d_v (resp. d_c) vary with the symbol (resp. check) index. A single parity check equation involving d_c variable nodes (codeword symbols) c_n is of the form:

$$\sum_{n=0}^{d_c-1} h_n c_n = 0 \quad \text{in GF}(q) \tag{1}$$

where each h_n is a nonzero value of the parity matrix H.

As for binary decoders, there are two possible representations for messages : probability weights vectors or LDR vectors. The use of the LDR form for messages has been advised by many authors who proposed practical LDPC decoders. The LDR values, which represent real reliability measures on the bits or the symbols are less sensitive to quantization errors due to the finite precision coding of the messages [12]. Also, LDR measures operate in the logarithm domain, which avoids complicated operations (in terms of hardware implementation) like multiplications or divisions. The following notation will be used for an LDR vector of a random variable $z \in GF(q)$:

$$\mathbf{L}(z) = [L[0] \dots L[q-1]]^T$$

where

$$L[i] = \log \frac{P(z=\alpha_i)}{P(z=\alpha_0)} \tag{2}$$

with $P(z=\alpha_i)$ being the probability that the random variable z takes on the values $\alpha_i \in GF(q)$. With this definition $L[0]=0, \quad L[i] \in \mathbb{R}$.

The log-likelihood-ratio (LLR) messages at the channel output are $q-1$ dimensional vectors in general denoted by $\mathbf{L}_{ch}=[L_{ch}[k]_{k\in\{0,\ldots,q-1\}}]^T$ and are defined by $q-1$ terms of the type (2). The values of the probability weights $P(z=\alpha_i)$ depend on the transmission channel statistics. The decoding algorithm that we propose is independent of the channel, and we just assume that a demodulator provides the LLR vector $\mathbf{L}_{ch}$ to initialize the decoder. We have applied the NB-LDPC codes to communicate over two types of channels: BI-AWGN and QAM-AWGN.

For the BI-AWGN case, each symbol of the codeword $c_n,\ n \in \{0,\ldots,N-1\}$ can be converted into a sequence of $\log_2(q)$ bits $c_{n_i} \in GF(2),\ i \in \{0,\ldots,\log_2(q)-1\}$. The binary representation of the codeword is then mapped into a BPSK constellation and sent on the AWGN channel:

$$y_{n_i} = BPSK\left(c_{n_i}\right) + w_{n_i}$$

with y_{n_i} being the received noisy BPSK symbol, and w_{n_i} being a real white Gaussian noise random variable with variance $\frac{N_0}{2E_bR}$, where $\frac{E_b}{N_0}$ is the SNR per information bit.

The NB-LDPC iterative decoding algorithms are characterized by three main steps corresponding to the different nodes depicted in Fig. 1: *(i)* the variable node update, *(ii)* the permutation of the messages due to non zeros values in the matrix H and *(iii)* the check node update which is the bottleneck of the decoder complexity, since the BP operation at the check node is a convolution of the input messages, which makes the computational complexity grow in $O(q^2)$ with a straightforward implementation.

We use the following notations for the messages in the graph (see Fig. 1). Let $\{\mathbf{V}_{p_iv}\}_{i\in\{0,\ldots,d_v-1\}}$ be the set of messages entering into a variable node *v* of degree d_v, and $\{\mathbf{U}_{vp_i}\}_{i\in\{0,\ldots,d_v-1\}}$ be the output messages for this variable node. The index 'pv' indicates that the message comes from a *permutation* node to a *variable node*, and 'vp' is for the other direction. We define similarly the messages $\{\mathbf{U}_{p_ic}\}_{i\in\{0,\ldots,d_c-1\}}$ (resp. $\{\mathbf{V}_{cp_i}\}_{i\in\{0,\ldots,d_c-1\}}$) at the input (resp. output) of a degree d_c check node.

In [7], the EMS algorithm reduces the complexity of the check node update by considering only the n_m largest values of the messages at the input of the check node. However, the output messages of the check node are still composed of q values. As a consequence, the EMS complexity of a single parity check node varies in $O(n_m.q)$ and all messages in the graph are stored with their full representation of q real values, which implies a high memory requirements.

In this paper, we present a new implementation of the EMS algorithm, whose main originality is to store exactly $n_m \ll q$ values in all vector messages $\mathbf{U}_{vp}$, $\mathbf{V}_{cp}$. As a result not only the memory requirements are reduced but also the computational complexity. In the following section we present our procedure to truncate the messages from q to n_m values and discuss the impact on the error correction performance of the decoder.

III. Structure and compensation of the truncated messages

The vector messages $\mathbf{V}_{cp}$ and $\mathbf{U}_{vp}$ are now limited to only n_m entries which are assumed to be the largest reliability values of the corresponding random variable. Moreover, the values in a message are sorted in decreasing order. That way, $V_{cp}[0]$ is the maximum value and $V_{cp}[n_m - 1]$ is the minimum value in $\mathbf{V}_{cp}$. We need to associate to the vectors $\mathbf{V}_{cp}$, $\mathbf{U}_{vp}$ of size n_m the additional vectors $\beta_{\mathbf{V_{cp}}}$ and $\beta_{\mathbf{U_{vp}}}$ (of size n_m) which store the field elements $\alpha_k \in GF(q)$, associated to the largest LDR values of vectors $\mathbf{V}_{cp}$ and $\mathbf{U}_{vp}$. For example, $U_{vp}[k]$ is the LDR value that corresponds to the symbol value $\beta_{U_{vp}}[k] \in GF(q)$.

Although interesting in terms of memory and computation reduction, the truncation of messages obviously looses potentially valuable information which leads to performance degradation on the error rate curves. This loss of performance could be mitigated by using a proper compensation of the information that has been truncated. Because our main concern is the development of low complexity decoders, we have chosen to compensate the $q - n_m$ truncated values with a single scalar value γ, which is the simplest model one can use. The following definition is used for a compensated message:

Definition

Let **A** *be any message in the graph which represents an LDR vector of size* q*. A truncated version* **B** *of* **A** *is composed of the* n_m *largest values of* **A** *sorted in decreasing order, plus an additional* $(n_m + 1)$*-th value* $\gamma_A \in \mathbb{R}$*, whose goal is to compensate for the information loss due to the truncation of* $q - n_m$ *values.* ∎

The compensated-truncated message **B** has then $(n_m + 1)$ components, and the value γ_A is seen as a constant real value that replaces the $q - n_m$ missing reliabilities. A full representation of the truncated message **B** would then be:

$$\mathbf{B} = [B[0] \dots B[n_m - 1]\ \gamma_A \dots \gamma_A]^T$$

This means in particular that $\gamma_A \leq B[n_m - 1]$.

Let us first analyze a possible solution to compute the value of γ_A using normalization of probability messages. We consider $\mathbf{P}_A$ the probability domain representation of the LDR vector $\mathbf{A}$

$$P_A[k] = P(z = \alpha_k) = P_A[0]e^{A[k]} \qquad k \in \{0, \ldots, q-1\}$$

and let $\mathbf{P}_B$ be the vector of size n_m with the values

$$P_B[k] = P(z = \beta_B[k]) = P_A[0]e^{B[k]} \qquad k \in \{0, \ldots, n_m - 1\}$$

Remember that $\mathbf{A}$ is unsorted while $\mathbf{B}$ is sorted, which explains the difference in these two definitions.

Because $\mathbf{P}_A$ is a probability weight vector, we have:

$$\sum_{k=0}^{q-1} P_A[k] = 1 \qquad \sum_{k=0}^{n_m-1} P_B[k] < 1 \tag{3}$$

A clever way to fix a good value on the scalar compensation γ_A is to assume that the truncated message should represent a probability weight vector with a sum equal to one, so that $\sum_{k=0}^{n_m-1} P_B[k] + (q - n_m)P_{\gamma_A} = 1$ is satisfied. The probability weight associated with LDR value γ_A is $P_{\gamma_A} = P_A[0]e^{\gamma_A}$. The normalization of vector $\mathbf{P}_B$ is then

$$(q - n_m)P_{\gamma_A} = 1 - P_A[0] \sum_{k=0}^{n_m-1} e^{B[k]}$$

$$\frac{P_{\gamma_A}}{P_A[0]} = \frac{\frac{1}{P_A[0]} - \sum_{k=0}^{n_m-1} e^{B[k]}}{q - n_m}$$

$$\log \frac{P_{\gamma_A}}{P_A[0]} = \log \left(\sum_{k=0}^{q-1} e^{A[k]} - \sum_{k=0}^{n_m-1} e^{B[k]} \right) - \log(q - n_m)$$

and finally

$$\gamma_A = \log \left(\sum_{k=0, A[k] \notin B}^{q-1} e^{A[k]} \right) - \log(q - n_m) \tag{4}$$

As a first remark, we note that the computation of the additional term requires the $q - n_m$ ignored values of vector $\mathbf{A}$, and the computation of a non linear function. The non linear function can be expressed in terms of the $\max^*(x_1, x_2)$ operator, used in many papers (e.g. [9]), and in order to simplify (4), we approximate this operator by:

$$\max^*(x_1, x_2) = \log\left(e^{x_1} + e^{x_2}\right) \approx \max(x_1, x_2) \tag{5}$$

Equation (4) becomes:

$$\begin{aligned}\gamma_A &= \max_{k=0,A[k]\notin B}{}^{*}\{A[k]\} - \log(q-n_m) \\ &\approx \max_{k=0,A[k]\notin B}\{A[k]\} - \log(q-n_m) \\ &\approx B[n_m] - \log(q-n_m) \end{aligned} \quad (6)$$

where $B[n_m]$ is the largest value among the $(q-n_m)$ ignored values of vector **A**.

By using the approximation (6) we obtain a simple computational formula for the supplementary term γ_A, since we just need to truncate the LDR vector **A** with its (n_m+1) largest values instead of its n_m largest values. On the other hand, this approximation introduces a degradation of the error performance of the decoder. The approximation (5) is well known to over-estimate the values of the LDR messages [13], and needs compensation.

In principle, the compensation of the over-estimation should be different for each message since the accuracy of approximation (5) depends on the values it is applied to. An adaptive compensation would be obviously too complicated with regards to our goal of proposing a low complexity algorithm. We have then chosen to compensate *globally* the over-estimation of the additional term γ_A with a single scalar offset, constant for all messages in the graph and also constant for all decoding iterations:

$$\gamma_A = B[n_m] - \log(q-n_m) - offset = B[n_m] - Offset \quad (7)$$

There are several ways of optimizing the value of a global offset correction in message passing decoders. We have chosen to follow the technique proposed in [7], which consists of minimizing the decoding threshold of the LDPC code, computed with simulated density evolution. Because of the lack of space, we do not discuss in this paper the optimization of the global offset, and we recall that estimated density evolution is just used as a criterion to choose the correction factor and not to compute accurate thresholds.

IV. Description of the algorithm

A. Decoding steps with messages of size $n_m \leq q$

We now present the steps of the EMS decoder that uses compensated-truncated messages of size n_m. We assume that the LLR vectors of the received symbols are known at the variable nodes, either stored in an external memory or computed on the fly from the channel measurements.

Using the notations of Fig. 1, the basic steps of the algorithm are:

1) Initialization: the n_m largest values of the LLR vectors are copied in the graph on the $\{\mathbf{U}_{vp_i}\}_{i\in\{0,\ldots,d_v-1\}}$ messages.
2) Variable-node update: the output vector messages $\{\mathbf{U}_{vp_i}\}_{i\in\{0,\ldots,d_v-1\}}$ (of size n_m) associated to a variable node v passed to a check node c are computed given all the information propagated from all adjacent check nodes and the channel, except this check node itself.
3) Permutation step: this step permutes the messages according to the nonzero values of H (see (1)). In our algorithm, it just modifies the indices vectors and not the message values:

$$\beta_{U_{p_ic}}[k] = h_i.\beta_{U_{vp_i}}[k] \quad k \in \{0,\ldots,n_m-1\} \tag{8}$$

where the multiplication is performed in GF(q).
4) Check-node update: for each check node, the values $\{V_{cp_i}[k]\}_{i\in\{0,\ldots,d_c-1\},k\in\{0,\ldots,n_m-1\}}$ sent from check a node to a permutation node are defined as the probabilities (expressed in LDR format) that the parity-check equation is satisfied if the variable node v is assumed to be equal to $\beta_{V_{p_iv}}[k]$.
5) Inverse permutation step: this is the permutation step from check nodes to symbol nodes, so it is identical to step 3), but in the reverse order.

For steps 2) and 4), a recursive implementation combined with a forward/backward strategy is a well known efficient implementation of node update when the associated degree is larger than four. This implementation technique has been widely presented in the literature for binary LDPC codes, and also for non-binary LDPC codes in [9]. It is based on a decomposition of the node neighborhood using dummy variables and adding corresponding edges that carry intermediate messages, that are named **I** in this paper. This decomposition allows to express the check or variable node equations using several *elementary steps*. One elementary step is defined by a node update that assumes only two input messages and one output message. The decomposition of a degree $d_c = 5$ check node and the associated forward/backward scheduling is depicted on figure 2. In this figure, the intermediate messages **I** are assumed to be stored also with n_m values, like the other messages. Using this strategy, the d_c incoming messages are used to compute $2 * (d_c - 3)$ intermediate messages by a forward/backward recursion, then the d_c outgoing messages are computed using either a combination of one input and one intermediate message, or two intermediate messages. Note that the intermediate messages are stored only until the outputs have been updated.

Remark

In order to ensure the numerical stability of the EMS algorithm, a post-processing step is necessary. We simply substrate to all values the smallest one. Without this step, the values of the LDR messages would converge to the highest achievable numerical value in a few iterations. The LDR values equation (9) are real numbers in domain $[0, +\infty)$.

$$\begin{aligned} U_{vp_i}[k] &= U_{vp_i}[k] - U_{vp_i}[n_m - 1] \;\; i \in \{0, \ldots, d_v - 1\} \;\; k \in \{0, \ldots, n_m - 1\} \\ V_{cp_i}[k] &= V_{cp_i}[k] - V_{cp_i}[n_m - 1] \;\; i \in \{0, \ldots, d_c - 1\} \;\; k \in \{0, \ldots, n_m - 1\} \end{aligned} \tag{9}$$

Since the EMS algorithm only involves linear operations, the terms $U_{vp_i}[k]$, $V_{cp_i}[k]$ *have the same LDR structure as defined in (2).* ∎

B. Variable node elementary step

Let assume that an elementary step describing the variable node update has **V** and **I** as input messages and **U** as output message. The vectors **V**, **I** and **U** of size n_m are sorted in decreasing order. We note also by $\beta_{\mathbf{V}}$, $\beta_{\mathbf{I}}$ and $\beta_{\mathbf{U}}$ their associated index vectors. Using the BP equations in the log-domain for the variable node update [9], the goal of an elementary step is to compute the output vector containing the n_m largest values among the $2n_m$ candidates (10) (stored in an internal vector message **T**). The processing of the elementary step in the case of a variable node update is described by:

$$T[k] = V[k] + Y \qquad T[n_m + k] = \gamma_V + I[k] \qquad k \in \{0, \ldots, n_m - 1\} \tag{10}$$

with

$$Y = \begin{cases} I[l] & \text{if } \beta_I[l] = \beta_V[k] \qquad k, l \in \{0, \ldots, n_m - 1\} \\ \gamma_I & \text{if } \beta_I[l] \notin \beta_{\mathbf{V}} \end{cases}$$

The compensation value γ is used when the required symbol index is not present in an input message.

Whenever the **V** input corresponds to the LLR channel vector of the received symbol, the equation (10) becomes:

$$T[k] = V[k] + Y \qquad T[n_m + k] = \mathrm{L}_{ch}[\beta_I[k]] + I[k] \qquad k \in \{0, \ldots, n_m - 1\}$$

since we do not assume that LLR vectors are truncated/compensated messages.

C. Low complexity implementation of a check node elementary step

This section describes in details the algorithm that we propose for an elementary component of the check node. This step is the bottleneck of the algorithm complexity and we discuss its implementation in details in the rest of the paper. The check node elementary step has **U** and **I** as input messages and **V** as output message. All these vectors are of size n_m are sorted in decreasing order. Similar to the variable node update, we note also by $\beta_{\mathbf{U}}$, $\beta_{\mathbf{I}}$ and $\beta_{\mathbf{V}}$ their associated index vectors. Following the EMS algorithm presented in [7], we define $S(\beta_V[i])$ as the set of all the possible symbol combinations which satisfy the parity equation $\beta_V[i] \oplus \beta_U[j] \oplus \beta_I[p] = 0$. With these notations, the output message values are obtained with:

$$V[i] = \max_{S(\beta_V[i])} (U[j] + I[p]) \qquad i \in \{0, \ldots, n_m - 1\} \tag{11}$$

Just as in the variable node update, when a required index is not present in the truncated vector **U** or **I**, its compensated value γ is used in equation (11). Without a particular strategy, the computation complexity of an elementary step is dominated by $O(n_m^2)$.

We propose a low computational strategy to skim the two sorted vectors **U** and **I**, that provide a minimum number of operations to process the n_m sorted values of the output vector **V**. The main component of our algorithm is a sorter of size n_m, which is used to fill the output message. For the clarity of presentation, we use a virtual matrix M built from the vectors **U** and **I** (cf. Fig.3), each element of M being of the form $M[i,p] = U[j] + I[p]$. This matrix contains the n_m^2 candidates to update the output vector **V**. The goal of our algorithm is to explore in a efficient way M in order to compute iteratively its n_m largest values, using the fact that M is build from sorted messages. For instance, we remark that the n_m largest values of M are located in the upper part of the anti diagonal of the matrix. The basic operations of the elementary step are:

1) Initialization: the values of the first column of M are introduced in the sorter.
2) Output: the largest value is computed.
3) Test: does the associated GF(q) index of the output value already exist in the output vector.
   - Yes: no action
   - No: the value is moved in the vector **V**
4) Evolution: The right neighbor - with regard to the M matrix - of the filled value is introduced in the sorter.
5) Go to (2)

In order to ensure that all values of the output vector **V** correspond to different symbols $\alpha_V \in GF(q)$, we can not stop the algorithm after only n_m steps, because it is possible that among the computed values after n_m steps, two or more values correspond to the same index α_V. Let us define n_c as the number of necessary steps so that all the n_m values of the output vector are computed. The parameter n_c is used to indicate the computational complexity of our new EMS implementation. We note that $n_c \in [n_m, \frac{n_m^2}{2}]$. Of course, the value of n_c depends on the LDR vectors **U** and **I**, and a strictly valid implementation of the elementary step should take into account the possibility of the worst case. However, we have found that n_c is most of the time quite small. As a matter of fact, the distribution of n_c has an exponential shape and decreases very rapidly, e.g. $prob(n_c \leq n_m + 4) = 0.9816$, for a regular GF$(256)$-LDPC code, $n_m = 32$ and a signal to noise ratio in the waterfall region of the code. Based on this observation, it seems natural to consider that the bad situations with large n_c are sufficiently rare so that they do not really impact on the decoder performance. We have verified this claim by simulations of density evolution and found that using $n_{c_{max}} = 2n_m$ does not change the value of the decoding threshold for various LDPC code parameters. Note that with $n_{c_{max}} = 2n_m$, sometimes the output vector **V** could be filled with less than n_m values and in those cases, we fill the rest of the vector with a constant value equal to the additional term γ_V. The worst case for the complexity of an elementary step is then $\mathcal{O}(n_{c_{max}} \log_2 n_m) = \mathcal{O}(2n_m \log_2 n_m)$, which corresponds to the number of $\max$ operations needed to insert $n_{c_{max}}$ elements into a sorted list of size n_m. In the next section, we study in details the complexity of our new implementation of the EMS algorithm.

V. Complexity and memory evaluation of the algorithm

The computational complexity per bit of a single parity node and a single variable node are indicated in table I in terms of their connexion degree d_c (resp. d_v). This complexity applies both for regular and irregular non binary LDPC codes, the local value of the connexion degree following the connectivity profile of the code. This complexity assumes the use of truncated messages of size n_m, and the implementation of the check node update presented in this paper. Note that we indicated the worst case complexity for the check node with $n_c = n_{c_{max}}$ and that the average complexity is often less than that. The complexity associated with the update of vectors **U** at the variable node output is obtained with a recursive implementation of the variable node, which is used only for connexion degrees $d_v \geq 3$. As a result, the complexity of our decoding algorithm is dominated by $\mathcal{O}(n_m \log_2(n_m))$ for both parity and

variable nodes computation. Interestingly, the complexity $Comp_{CN}$ of a check node and $Comp_{VN}$ of a variable node are somewhat balanced, which is a nice property that should help an efficient hardware implementation based on a generic processor model. Moreover, one can remark that the complexity of the decoder does not depend on q, the order of the field in which the code is considered. Let us again stress the fact that the complexity of our decoder varies in the order of $\mathcal{O}(n_m \log_2(n_m))$ and with $n_m \ll q$, which is a great computational reduction compared to existing solutions [7], [8], [9].

Finally, for a complete characterization of the computational complexity of our non-binary LDPC decoding algorithm, we also reported in table I the associated complexity of the permutation step ($Comp_{Perm}$) and the complexity of the post-processing ($Comp_{Post}$).

The memory space requirement of the decoder is composed of two independent memory components, the memory corresponding to the channel messages $\mathbf{L}_{ch}$ and the edge memory corresponding to the extrinsic messages $\mathbf{U}$, $\mathbf{V}$ with their associated index vectors β. Storing each LDR value on $Nbits$ bits in finite precision would therefore require a total number of $n_m * N * d_v * (Nbits + \log_2 q)$ bits for the edge memory. Thus, the memory storage depends linearly on n_m, which was the initial constraint that we put on the messages.

Since n_m is the key parameter of our algorithm that tunes the complexity and the memory of the decoder, we now need to study for which values of n_m the performance loss is small or negligible. In order to give a first answer to this question, we have made an asymptotic threshold analysis of the impact of n_m on the threshold value. For a rate $R = 0.5$ LDPC code with parameters $(d_v = 2, d_c = 4)$, Fig.4 plots the estimated threshold in $(E_b/N_0)_{dB}$ of our algorithm for different values of n_m and two different field orders GF(64) and GF(256). In this paper, we do not claim that the EMS algorithm verifies the necessary symmetry conditions that ensures the convergence of density evolution. Therefore, the validity of the threshold values is not proved. However, the estimated thresholds are a good indicator of the decoder behavior when the codeword length is large and the nonzeros values in the matrix are chosen uniformly.

The BP thresholds are equal to $\delta = 0.58dB$ for the GF(64) code and $\delta = 0.5dB$ for the GF(256) code [7]. As expected, the thresholds become better as n_m increases, and can approach the threshold of BP with much less complexity. We can use the plots on Fig.4 as first indication for choosing the field order of the LDPC code that corresponds to a given complexity/performance trade-off. Note, however, that this asymptotic study has to be balanced with the girth properties of finite length codes, since it has been identified in [3], [4] that ultra-sparse LDPC codes in

high order fields and with high girth have excellent performance.

VI. Special case: further memory reduction for cycle codes

It has been shown that for high order fields $q \geq 64$, the best GF(q)-LDPC codes decoded with BP should be *ultra sparse* (cycle codes, $d_v = 2$) [2], [3]. In the EMS implementation, an improved trade-off memory space/performance can be achieved for the decoding of cycle codes, by considering a modified scheduling of the decoding steps described in Section IV. We have adapted the shuffled scheduling proposed in [14] to the non-binary case, with the objective of greater storage memory reduction. Note that the adaptation of the shuffled scheduling for NB-LDPC codes has been proposed independently in [15], but the authors did not study the memory reduction that this scheduling implies.

Using a shuffled scheduling allows to store only the messages **U** in the edge memory, and the intermediate messages **I** and the messages **V** can be stored *locally* in a processing unit. It is therefore possible to consider more than n_m values for the **I** and **V** without increasing the storage capacity of the decoder. Let us denote by n_{m_I} (respectively n_{m_V}) the number of LDR values that form the truncated versions of messages **I** (respectively **V**) inside the processing unit. By construction, the different sizes verify $n_{m_U} \leq n_{m_I} \leq n_{m_V}$.

The shuffled scheduling is defined as follows. For each and every check node, let $\{v_1, \ldots, v_{d_c}\}$ be the set of variable nodes connected to this check node. The shuffled processing unit takes all incoming messages $\mathbf{U}_{vp}$ that are on the edges of the check node, computes locally the $\mathbf{V}_{pv}$ messages on the same edges with the EMS algorithm, and then updates the $\mathbf{U}_{vp}$ messages that are on the edges of $\{v_1, \ldots, v_{d_c}\}$ which are not connected to the current check node. In the case of $d_v = 2$ LDPC codes, this last step is performed only with the knowledge of the channel LLRs $\{\mathbf{L}_{v_k}\}_{k=1,\ldots,d_c}$. We can consider that the shuffled processing unit works with two types of messages: the external **U** vectors which determine the dimension of the edge memory and the internal **V** and **I** vectors which determine the computational complexity of decoder. Using different values for $(n_{m_U}, n_{m_I}, n_{m_V})$ has then an impact on the trade-off between the overall complexity of the decoder and its performance. We now discuss this advantage of the shuffled scheduling with a comparison with the classical flooding scheduling.

Let us consider a code $(d_v = 2, d_c = 4)$ code in GF(256) of size $N_b = 848$ (see section VIII-A for more details), and let us use truncated messages of size $n_m = 18$ in a flooding implementation of the EMS decoder. We consider the two following cases for a shuffled scheduling, and the

corresponding frame error rate simulations are plotted on figure Fig.5:

- (*a*) The same computational complexity for the two schedules.
  In this case, the size of the vectors **V** and **I** is set to $n_{m_V} = n_{m_I} = 18$. The size of the vectors **U** is set to $n_{m_U} = 9$. This choice corresponds to a memory space reduction of roughly $\frac{n_{m_U}}{n_{m_V}}$, with a small error performance degradation compared to the flooding implementation (Fig.5, B and C curves).
- (*b*) The same edge memory space for the two schedules.
  In this case, the size of the vectors **U** is kept at $n_{m_U} = 18$, but the size of vectors **V** and **I** is increased to $n_{m_V} = n_{m_I} = 36$. The shuffled scheduling provides an improvement of the error performance (Fig.5, A and B curves), without increasing the memory requirement of the decoder. Of course, this also induces an increase of the algorithm complexity .

As a conclusion, implementing the shuffled scheduling for non binary LDPC codes has the same advantage of reducing the average number of decoding iterations, as for the binary shuffled scheduling (see [15] for more details), but also provides additional degrees of freedom for the storage/complexity/performance trade-off of an EMS decoder.

VII. Quantization of the EMS algorithm

Toward practical hardware implementation, quantization is an indispensable issue that needs to be resolved. The goal of this section is to find the best trade-off between the hardware complexity, messages storage space and the error performance of the EMS algorithm. We investigate only the impact of uniform quantization schemes. The choice of the uniform quantization scheme is motivated by the fact that the hardware implementation of the EMS algorithm does not require nonlinear operations and the uniform quantifier has the advantage that it is simple and fast.

Let (b_i, b_f) represent a fixed-point number with b_i bits for the integer part (dynamic range) and b_f bits for the fractional part. So by fixed-point representation, a real number x is mapped to a binary sequence $\mathbf{x} = \left[\mathrm{x}_0 \ldots \mathrm{x}_{b_i+b_f-1}\right]$. A direct consequence of the post-processing defined by equation (9), is that we can use an unsigned fixed-point representation (12) to quantify the LDR messages of the EMS algorithm.

$$x \rightarrow \sum_{j=0}^{b_i+b_f-1} \mathrm{x}_j 2^{b_i-1-j} \tag{12}$$

This representation corresponds to a limit range of the LDR values of $\left[0, 2^{b_i+1} - 2^{-b_f}\right]$ with a precision of 2^{-b_f}. Various schemes (b_i, b_f) are examined, in order to find the best trade-off

between the number of quantization bits ($b_i + b_f$) and the error performance degradation of the decoder. The most representative results are summarized in Fig.6, which presents the simulation results of the EMS algorithm for an LDPC code over GF(64) of rate $R = 1/2$, for two sets of parameters $(n_{m_U}, n_{m_V}) = (8, 16)$ and $(n_{m_U}, n_{m_V}) = (16, 32)$.

We remark that a fixed point quantization scheme with $b_i = 5$ bits provides error performance close to the floating implementation of the EMS algorithm, while all the quantizations having $b_i = 4$ bits caused an error floor region. It turns out that the apparition of this phenomenon is due to the insufficient dynamic range of the LDR messages [16].

With the goal of speed and low storage in mind, we advice a quantization of all messages with 5 bits, with $(b_i = 5, b_f = 0)$. This representation of messages provides a balanced trade-off between low storage and good performance. We have conducted the same finite precision study for various rates and code lengths and have observed that $(b_i = 5, b_f = 0)$ is good in all cases. The EMS algorithm requires then only a few quantization bits, close to the fixed-point representation of the extrinsic messages in binary LDPC decoders [18].

VIII. Experimental Results of the EMS decoder

A. Performance loss compared to the non-binary BP algorithm

In this section, we present the simulation results of our low complexity EMS algorithm, compared with the BP algorithm considered as reference. We have made the comparison with regular GF(q)-LDPC codes over high order fields, of rate $R = 1/2$ ($d_v = 2, d_c = 4$), applied on a BPSK-AWGN channel. The BP has been implemented in floating point precision, and a quantization of $(b = 5, q = 0)$ is used for the EMS algorithm, as pointed out in the preceding section. In figure Fig.7, we have reported the frame error rate (FER) of a short code with length $N_b = 848$ equivalent bits, corresponding to a length $N = N_b/\log_2(q)$ non-binary LDPC code. The maximum number of iteration has been fixed to 1000, and a stopping criterion based on the syndrome check is used. Note that the average number of decoding iterations is rather low for all the simulation points below $FER = 10^{-3}$ (as an example, the average number of iterations for the $(2, 4)$ GF(64) code at $FER = 6 * 10^{-4}$ is equal to 3).

We denote by $\text{EMS}^{\text{GF(q)}}_{n_{m_U}, n_{m_V}}$ the EMS decoder over the field GF(q) with parameters n_{m_U}, n_{m_V} and $n_{m_I} = n_{m_V}$. Let us first discuss the performance of the EMS decoder with respect to the BP decoder. For the code over GF(64), the $\text{EMS}^{\text{GF(64)}}_{8,16}$ is the less complex algorithm presented. It performs within $0.25dB$ of the BP decoder in the waterfall region. The $\text{EMS}^{\text{GF(64)}}_{16,32}$ algorithm has

$0.06dB$ performance loss in the waterfall region and performs even better than the BP decoder in the error floor region. The fact that the EMS can beat the BP decoder in the error floor is not surprising and is now well known in the literature. This behavior comes from the fact that for small code lengths, an EMS algorithm corrected by an offset could be less sensitive to pseudo-codewords than the BP.

Note that with this example, the only advantage of using a GF(256) code in terms of performance/complexity trade off is that it provides an error floor region lower than the GF(64) code. Finally, it is interesting to compare the error performance of $\text{EMS}_{16,32}^{\text{GF(64)}}$ and $\text{EMS}_{16,32}^{\text{GF(256)}}$ because they offer the same decoding complexity. In the waterfall region of the codes $\text{EMS}_{16,32}^{\text{GF(64)}}$ performs better that $\text{EMS}_{16,32}^{\text{GF(256)}}$ with a gain of $0.19dB$. The good performance of the GF(64) code in the waterfall region is determined by the value of $n_{m_V} = 32$ parameter, which is sufficiently close to the field order to provide a good threshold. At low FER, the performance gap between the two codes becomes smaller, which seems to indicate that the $GF(256)$ LDPC code will perform better that the $GF(64)$ LDPC code at very low FER (FER<10^{-7}), without increasing the decoder complexity. Note that this observation balances the conclusions of Section V, and stresses another advantage of considering very high order field non-binary LDPC codes. Moreover, the EMS is quite robust since the complexity reduction from $q = 256$ to $n_{m_V} = 32$ is a lot higher than from $q = 64$ to $n_{m_V} = 32$, and the performance loss stays acceptable. Note that the other approaches proposed in the literature [8], [9] were not illustrated on high order fields and that - to our knowledge - the EMS decoder is the first decoder that proposes a good performance complexity trade-off for field orders $q \geq 64$.

In order to quantify the influence of the offset parameter (γ) on the decoder's performances, we have also reported in Fig.7 the simulations results of the EMS decoder in the particular case when the offset is zero (EMS without offset). We remark that the error performances of the $\text{EMS}_{16,32}^{\text{GF(256)}}$ algorithm are greatly improved by using a proper offset, and its influence is less significant in the case $\text{EMS}_{8,16}^{\text{GF(64)}}$. Generally, the influence of the offset parameter on the error performances of the EMS decoder depends on the loss of information induced by the truncation procedure ($q - n_m$). If the difference $q - n_m$ is non-negligible the use of a proper offset is recommended.

For lack of space reasons, we present only the results for the code/decoder parameters of figure Fig.7, but we have conducted extensive simulations for various other code/decoder parameters and the same kind of behavior has been observed. As seen on the results presented in this

section, the error performance of a hardware implementable version of the EMS is quite close to the performance of floating BP algorithm. Its good performance and its reduced complexity and memory space requirement make the EMS algorithm a good candidate for the hardware implementation of non binary LDPC decoders.

In order to improve the performance of the decoder without sacrificing much the complexity, it would be interesting to study more precisely if the performance degradation compared to BP comes from the truncation of the messages or from the use of a max operator at the check node update. A correction strategy more elaborate than a single offset correction (dynamical offset along the iterations, nonlinear correction, etc) could be more effective on either approximations.

B. Comparison with binary decoders

The main idea of this section is to compare in terms of computational complexity and error performance the proposed EMS algorithm to its binary equivalent, the corrected Min-Sum (MS) algorithm [10]. The complexity of the corrected MS algorithm for a single check node of degree d_c is equal to: $3(d_c-2)/d_c$ min operations per bit, $(2d_c-1)/d_c$ XOR operations per bit to compute the sign of the output and 2 real additions that correspond to the correction operation. Also, for a bit node of degree d_v the complexity is equal to $(2d_v-1)/d_v$ real additions per bit. For a fair computational complexity comparison of algorithms, we have decided to compare only the operations that are common to both algorithms. We thus compare the number of max operations of the EMS algorithm (see table I) with the min operations of the MS algorithm and the number of real additions necessary to two algorithms (per iteration). The specific operations of the algorithms are not taken into account in the complexity comparison (the additions over GF(q) for EMS algorithm and the sign computation for the MS).

The comparison has been made for short and moderate code lengths over BI-AWGN and QAM-AWGN channels. The choice of the code length is motivated by the fact that the non-binary LDPC codes can achieve performance very close to the Shannon limit for these lengths. The binary codes that we used are from [17], irregular codes of size $N_b = 504$ (short length) and $N_b = 1008$ bits (moderate length) and of coderate $R = 0.5$. The corresponding non-binary codes are of equivalent length $N = 84$ symbols over GF(64) (short length) and $N = 126$ symbols over GF(256) (moderate length). The non-binary codes are regular $(d_v = 2, d_c = 4)$ and of coderate $R = 0.5$.

In Fig.8, we have reported the frame error rate (FER) of binary and non-binary short length

codes. We denote by $\text{EMS}_{n_m}^{\text{GF(q)}}$ the EMS decoder over the field GF(q) with parameters $n_m = n_{m_U} = n_{m_I} = n_{m_V}$. Let us first discuss the performance of the EMS algorithm with respect to the corrected MS algorithm. The $\text{EMS}_{18}^{\text{GF(64)}}$ algorithm performs better than the corrected MS with a gain of $0.375dB$ in the waterfall region. Furthermore for a smaller value of n_m ($n_m = 12$ approximately 20% of q) the EMS algorithm still outperforms the MS. Concerning the complexity of these two version of the EMS, the $\text{EMS}_{18}^{\text{GF(64)}}$ is 9 times more complex than the MS, and the $\text{EMS}_{12}^{\text{GF(64)}}$ is 5 times more complex than the MS. We have also plotted the error performance of the $\text{EMS}_{6}^{\text{GF(64)}}$ algorithm, which has a complexity equivalent to the binary decoder. The loss of performance in the waterfall region is explained by the small value of $n_m = 6$ (approximately 10% of q), which is not sufficiently close to the field order to provide a good threshold.

For short code lengths, the $\text{EMS}_{18}^{\text{GF(64)}}$ and $\text{EMS}_{12}^{\text{GF(64)}}$ have better error performance than the MS decoder on a very good binary LDPC code (for this rate and length) and in the same time the complexity of our non binary decoder remains reasonably close to the complexity of the binary decoder.

Over QAM-AWGN channels, the non-binary LDPC codes with a field order greater or equal to the size of constellation has the advantage that the encoder/decoder works directly with symbols. All mapping choices of the codeword symbols to the constellation points are equivalent and lead to the same performance. This means that there is no loss of performance due to the demapping process at the receiver. This is a clear advantage comparing to the binary codes. In Fig.9, we have plotted the simulation results of the EMS algorithm and the binary MS algorithm for the moderate length codes, over a 256-QAM-AWGN channel. We have used a Bit-Interleaved Coded Modulation scheme to transmit the binary code over the 256-QAM-AWGN channel and a field order equal to $q = 256$ for the non-binary LDPC codes. Note that the non-binary LDPC codes have been optimized with the technique described in [4].

Over the QAM256-AWGN channel the $\text{EMS}_{36}^{\text{GF(256)}}$ algorithm performs $0.5dB$ better than the corrected MS algorithm which is a quite important improvement. Concerning the complexity comparison, the EMS algorithm has approximately 25 times the complexity of the binary algorithm. The $\text{EMS}_{6}^{\text{GF(256)}}$ and $\text{EMS}_{12}^{\text{GF(256)}}$ algorithms have a performance loss in the waterfall region due to the small value of n_m. The $\text{EMS}_{6}^{\text{GF(256)}}$ has roughly the same complexity than the MS decoder. As in the BI-AWGN channel case, the EMS decoder on non-binary LDPC codes performs better than the MS algorithm on binary LDPC codes, with a reasonable increase in complexity. Our efficient decoder shows that non-binary LDPC codes could be a reliable

alternative for coding schemes with short to moderate codeword lengths.

Note that the EMS decoder has a quite fast convergence since the average number of decoding iterations when a syndrome stopping criterion is used is typically half the one of the binary case. For example, with $(q = 64, n_m = 18)$ at $FER = 1e-5$, the average number of iterations for the EMS algorithm is equal to 3.3 and for its binary equivalent (Min-Sum) the average number of iterations is 6.8. This remark remains valid in the case of an 256-QAM-AWGN transmission, where for the $\text{EMS}_{36}^{\text{GF(256)}}$ algorithm (Fig.9) the average number of iterations is equal to 5 at $FER = 1e-5$ and for the Min-Sum algoritm the average number of iterations is approximatively 9.5.

IX. Conclusion

We have presented in this paper a general low complexity decoding algorithm for non binary LDPC codes, using log-density-ratio as messages. The main originality of the proposed algorithm is to truncate the vector messages to a fixed number of values $n_m \ll q$, in order to solve the complexity problem and to reduce the memory requirements of the non binary LDPC decoders. We have also shown that by using a correction method for the messages, our EMS decoding algorithm can approach the performance of the BP decoder and even in some cases beat the BP decoder. The complexity of the proposed algorithm is dominated by $\mathcal{O}(n_m \log_2(n_m))$. For values of n_m providing near-BP error performance, this complexity is smaller than the complexity of the BP-FFT decoder, and by far lower than the solutions proposed in the literature. Note that the single parameter n_m tunes both the computational complexity and memory space requirements. It also defines efficiently the trade-off performance/complexity. We have also proposed a non-binary adaptation of the shuffled scheduling in order to induce a new degree of freedom in the algorithm, which allows a reduction of the memory space requirements for the cycle codes.

We have compared the error performance of our algorithm with non-binary BP and binary corrected MS algorithms, in order to demonstrate that the proposed low complexity, low memory EMS decoding algorithm becomes a good candidate for a hardware implementation. Since its complexity and its memory space requirements has been greatly reduced and the performance degradation is small or negligible, the EMS algorithm applied on non-binary LDPC codes build in very high order fields could be an alternative to existing solutions.

Although the EMS algorithm could be applied to irregular LDPC codes as described in this paper, an interesting issue would be to study if the number n_m of values kept in messages

needs to be optimized with respect to the degree of the variable nodes. This issue is of particular importance since good irregular LDPC codes are usually more dense than regular ones, increasing thereby the memory requirements for message storage.

The authors are grateful to the reviewers for insightful comments and suggestions, which have improved this paper.

REFERENCES

[1] T.J. Richardson, M.A. Shokrollahi and R.L. Urbanke, "Design of Capacity-Approaching Low-Density Parity Check Codes" *IEEE Trans. Inform. Theory*, vol.47, pp.619-637, Feb. 2001

[2] M. Davey and D.J.C. MacKay, "Low Density Parity Check Codes over GF(q)," *IEEE Commun. Lett.*, vol. 2, pp. 165-167, June 1998.

[3] X.-Y. Hu and E. Eleftheriou, "Binary Representation of Cycle Tanner-Graph GF(2^q) Codes," *The Proc. IEEE Intern. Conf. on Commun.*, Paris, France, pp. 528-532, June 2004.

[4] C. Poulliat,M. Fossorier and D. Declercq, "Design of non binary LDPC codes using their binary image: algebraic properties," *ISIT'06*, Seattle, USA, July 2006.

[5] A. Bennatan and David Burshtein, "Design and Analysis of Nonbinary LDPC Codes for Arbitrary Discrete-Memoryless Channels," *IEEE Trans. on Inform. Theory*, vol. 52, no. 2, pp. 549-583, Feb. 2006.

[6] L. Barnault and D. Declercq, "Fast Decoding algorithm for LDPC codes over GF(2^q)," *The Proc. 2003 Inform. Theory Workshop*, Paris, France, pp. 70-73, March 2003

[7] D. Declercq and M. Fossorier, "Decoding Algorithms for Nonbinary LDPC Codes over GF(q)", *IEEE Trans. on Commun.*, vol. 55(4), pp. 633-643, April 2007.

[8] H. Song and J.R. Cruz, "Reduced-Complexity Decoding of Q-ary LDPC Codes for Magnetic Recording," *IEEE Trans. Magn.*, vol. 39, pp. 1081-1087, Mar. 2003.

[9] H. Wymeersch, H. Steendam and M. Moeneclaey, "Log-Domain Decoding of LDPC Codes over GF(q)," *The Proc. IEEE Intern. Conf. on Commun.*, Paris, France, June 2004, pp. 772-776.

[10] J. Chen, A. Dholakia, E. Eleftheriou, M. Fossorier and X.-Y. Hu "Reduced Complexity Decoding of LDPC codes" *IEEE Trans. Commun.*, vol. 53, pp. 1288-1299, July 2005

[11] R.M. Tanner, "A Recursive Approach to Low Complexity Codes", *IEEE Trans. Inform. Theory*, vol. 27, pp. 533-547, 1981.

[12] L. Ping and W.K. Leung, "Decoding low density parity check codes with finite quantization bits", *IEEE Commun. Lett.*, 4(2):pp.62-64, February 2000.

[13] J. Chen and M. Fossorier, "Density Evolution for Two Improved BP-Based Decoding Algorithms of LDPC Codes," *IEEE Commun. Lett.*, vol. 6, pp. 208-210, May 2002.

[14] J. Zhang and M. Fossorier "Shuffled Iterative Decoding" *IEEE Trans. Lett.*, vol. 53, pp. 209-213, February 2005

[15] N. Yacov, H. Efraim, H. Kfir, I. Kanter and O. Shental, "Parallel vs. Sequential Belief Propagation Decoding of LDPC Codes over GF(q) and Markov Sources," *ArXiv Computer Science e-prints*, cs/0605069, May 2006

[16] H. Wymeersch, H. Steendam and M. Moeneclaey, "Computational complexity and quantization effects of decoding algorithms of LDPC codes over GF(q)," *In Proc. ICASSP*, Montreal, Canada, May 2004

[17] D.J.C. MacKay, "Online database of low-density parity check codes", http://www.inference.phy.cam.ac.uk/mackay/codes/data.html.

[18] T. Zhang, Z. Wang and K.K. Pahri "On finite precision implementation of low parity check codes" *In Proc. ISCAS*, Sydney, Australia, May 2001

Per bit per iteration	No. max	No. real add	No. add over GF(q)
$Comp_{CN}$	$(3(d_c-2)n_{c_{max}}\log_2 n_m)/(d_c\log_2 q)$	$3(d_c-2)(n_{c_{max}}+n_m)/(d_c\log_2 q)$	$3(d_c-2)(n_{c_{max}}+n_m)/(d_c\log_2 q)$
$Comp_{VN}$	$(3d_v-4)n_m\log_2(2n_m)/(d_v\log_2 q)$	$(3d_v-4)2n_m/(d_v\log_2 q)$	0
$Comp_{Post}$	0	$n_m/(\log_2 q)$	0
$Comp_{Permp}$	0	0	$n_m/(\log_2 q)$

TABLE I

COMPUTATIONAL COMPLEXITY OF THE MESSAGE UPDATES WITH THE EMS ALGORITHM AND MESSAGES OF SIZE n_m

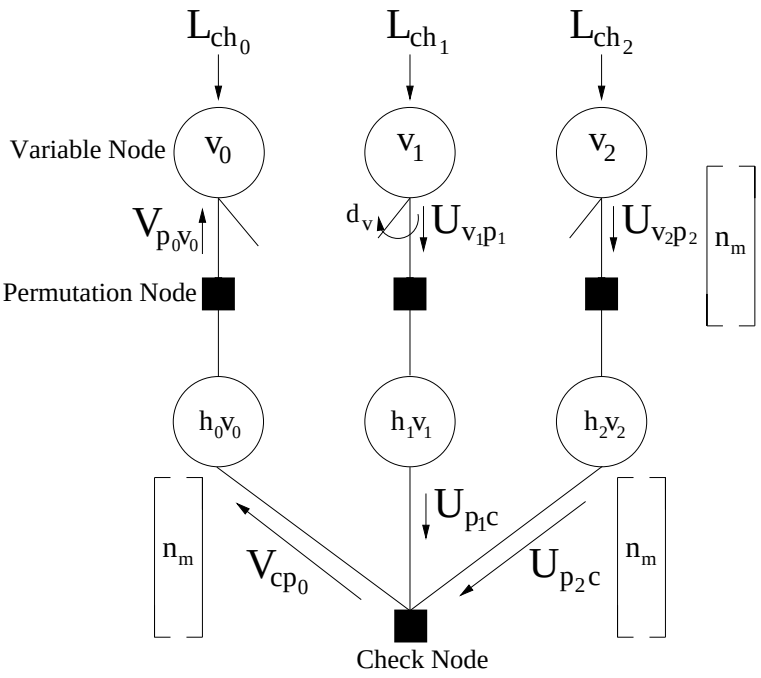

Fig. 1. Factor graph structure of a parity check node of degree $d_c = 3$ for a non-binary LDPC code

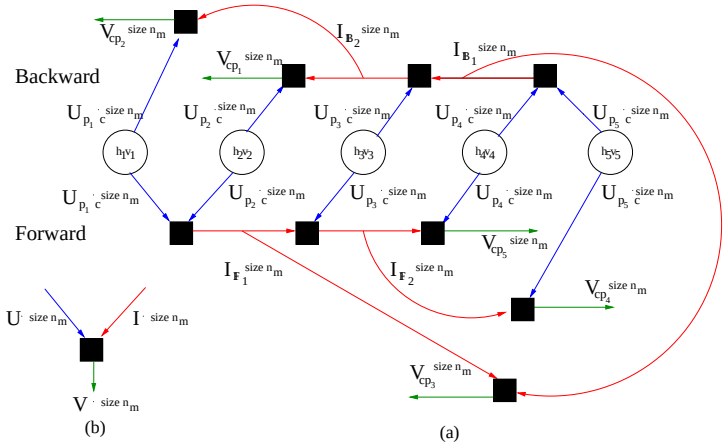

Fig. 2. The recursive structure of a degree $d_c = 5$ check-node (a); The elementary step (b)

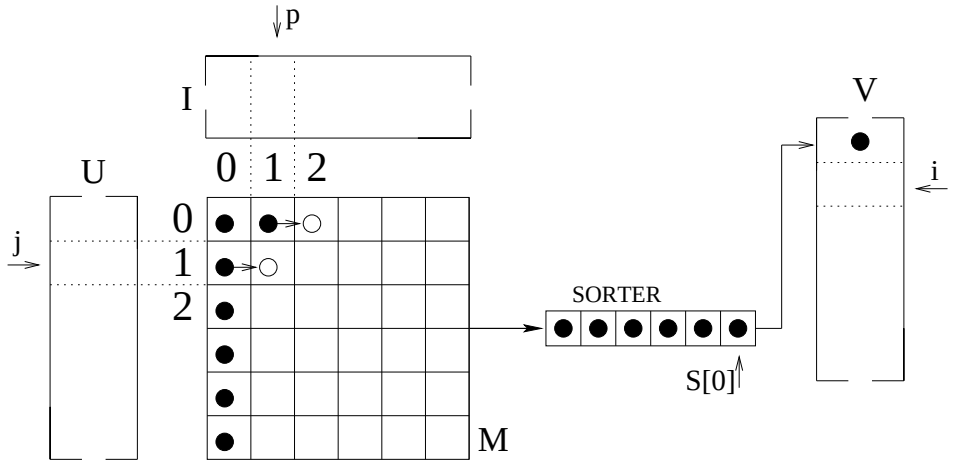

Fig. 3. Diagram of the low complexity algorithm of the elementary step

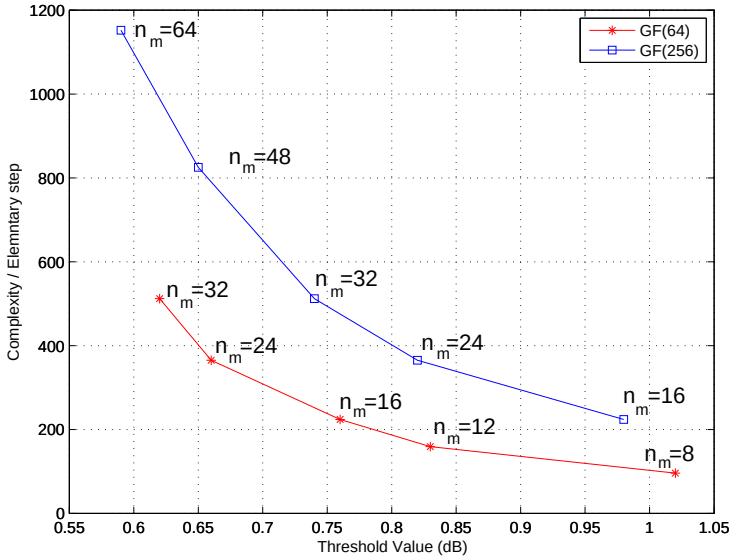

Fig. 4. Estimated decoding threshold vs. Complexity

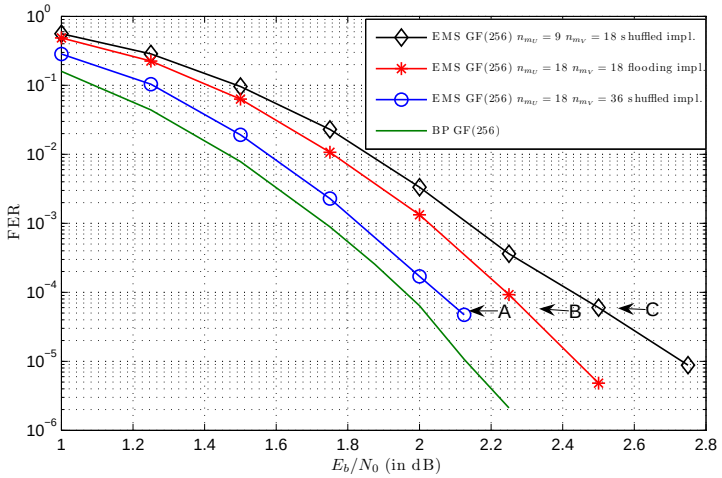

Fig. 5. EMS decoding algorithm, Shuffled vs. Flooding implementation, for an GF(256)-LDPC code (R=0.5, N_b=848 bits) over BI-AWGN channel

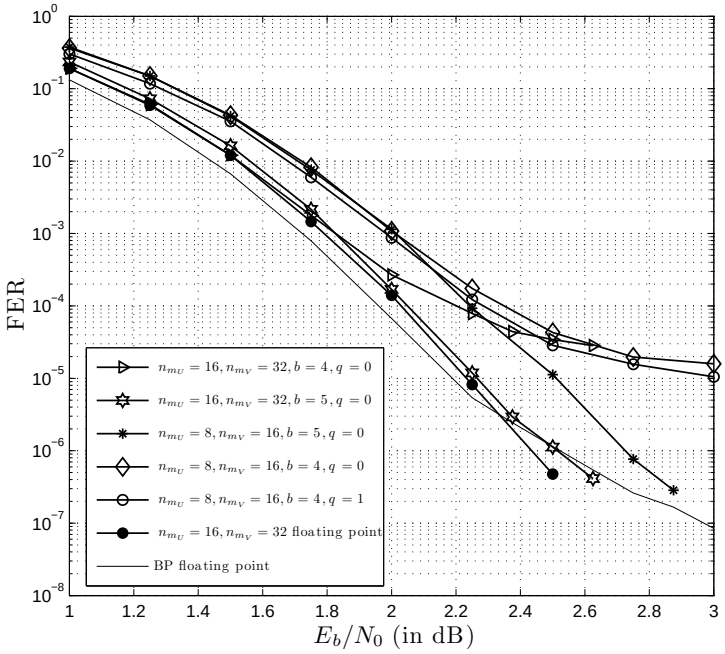

Fig. 6. EMS decoding algorithms, different fixed-point implementations,for an GF(64)-LDPC code (R=0.5, N_b=852 bits) over BI-AWGN channel

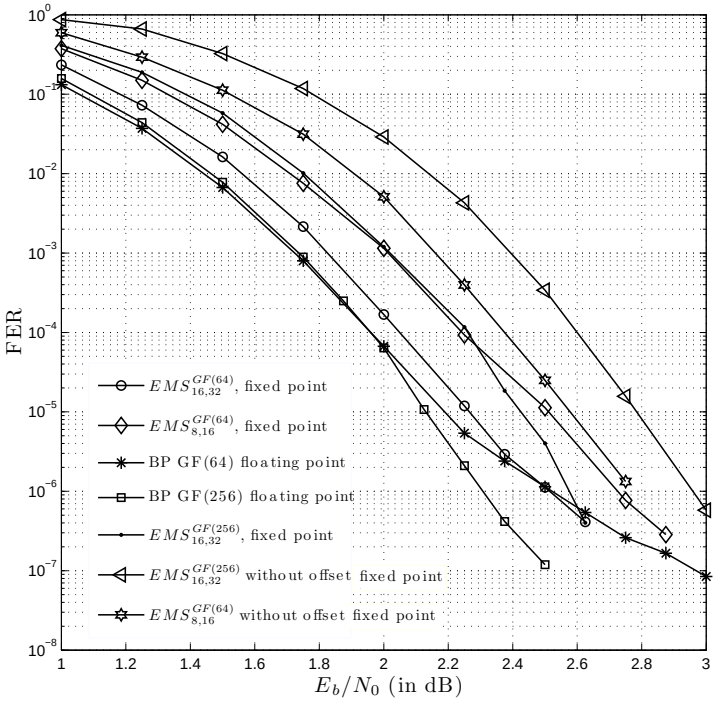

Fig. 7. Comparison between BP and EMS decoding algorithms,for an LDPC code (R=0.5, N_b=848 bits) over BI-AWGN channel

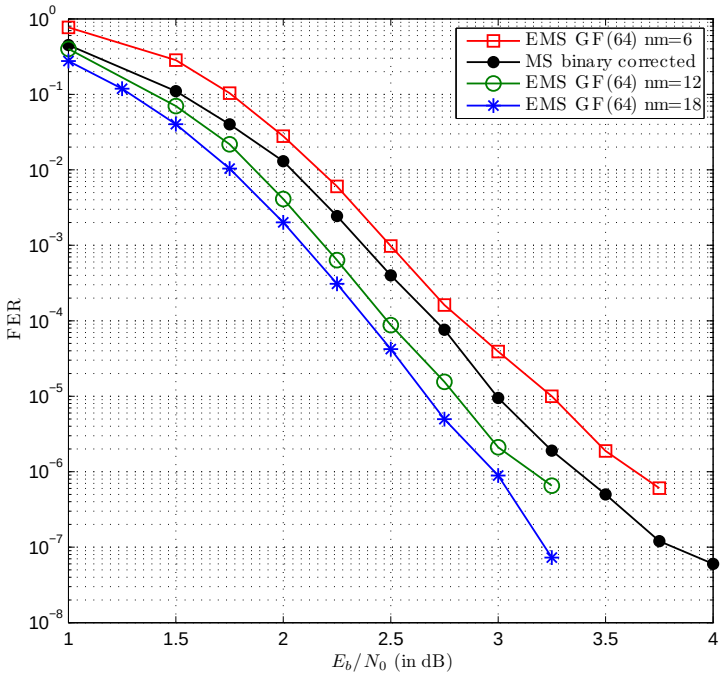

Fig. 8. Comparison between EMS decoder and binary MS decoder, for an LDPC code (R=0.5, N_b=504 bits) over BI-AWGN channel

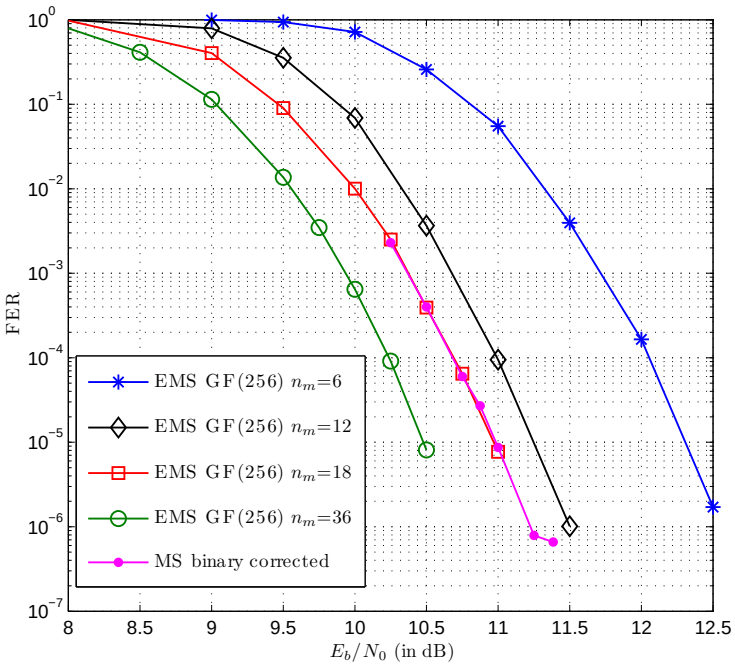

Fig. 9. Comparison between EMS decoding algorithm and binary MS algorithm, for an LDPC code (R=0.5, N_b=1008 bits) over 256-QAM-AWGN channel