Low-complexity, Low-memory EMS algorithm for non-binary LDPC codes

Adrian Voicila*‡,David Declercq‡, François Verdier‡
‡ETIS
ENSEA/UCP/CNRS UMR-8051
95014 Cergy-Pontoise, (France)

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

Pascal Urard*
*STMicroelectronics
Crolles, (France)

***Abstract*— In this paper, we propose a new implementation of the EMS decoder for non binary LDPC codes presented in [7]. 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 $\mathcal{O}(n_m \log_2 n_m)$, which starts to be reasonable enough to compete with binary decoders. The performance of the low complexity algorithm with proper compensation are quite good with respect to the important complexity reduction, which is shown both with a simulated density evolution approach and actual FER simulations.**

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, and more and more LDPC solutions are proposed in standards (DVB, WIMAX, etc). In terms of performance, binary LDPC codes start to show their weaknesses when the code size is small or moderate, and higher order modulation is used for transmission. For these cases, non binary LDPC (NB-LDPC) codes designed in high order Galois fields GF(q) have shown great interest [1], [2], [3], [4]. However, the performance gain brought by using LDPC codes over GF(q) comes together with a significant increase of the decoding complexity. NB-LDPC codes are decoded 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 $\mathcal{O}(q \log q)$, but this implementation is only convenient for messages expressed in the probability domain.

In [7], the authors have proposed a decoding algorithm, called extended min-sum (EMS) algorithm, which uses 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. With $n_m \ll q$, the complexity of a check node varies in $\mathcal{O}(n_m.q)$, using LDR representations of the messages, without sacrificing too much in performance, even for high order field codes, up to GF(256). This complexity is still very high and one should try to reduce it even more in order for NB-LDPC codes to compete with binary codes in terms of performance/complexity trade off. Moreover, the larger amount of memory required to store the messages of size q is an issue that has not been addressed in [7].

In this paper, we propose several improvements of the EMS algorithm that allows us to reduce both the memory and the complexity of the decoder. 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 data nodes input. Moreover, we propose to store only n_m reliabilities instead of q in each message. The truncation of messages from q to n_m values has to be done in an efficient way in order to reduce its impact on the performance of the code. The technique that we propose is described in details in section III, together with an efficient offset correction to compensate for the performance loss. Using the truncated messages representation, and a recursive implementation of the check node update, we propose a new implementation of the EMS decoder whose complexity is dominated by $\mathcal{O}(n_m \log n_m)$, with $n_m \ll q$, which is an important complexity reduction compared to all existing methods [5], [6], [7]. Our new algorithm is depicted in section IV and a study of its complexity/performance trade off is made in section V. We conclude the paper by simulation results that demonstrate that our low complexity decoder still performs very close to the BP decoder that we use as benchmark.

II. Preliminaries

A non binary LDPC code is defined by a very sparse random parity check matrix H whose components belong to a finite field GF(q). Decoding algorithms of LDPC codes are iterative message passing decoders based on a factor (or Tanner) graph representation of the matrix H. In general, an LDPC code has a factor graph consisting of N variable nodes with various connexion degrees and M parity check nodes with also various degrees. To simplify our notations, we will only present the decoder equations for isolated nodes with given degrees, and we denote d_v the degree of the symbol node and d_c the degree of the 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_t is of the form:

$$\sum_{t=1}^{d_c} h_t c_t = 0 \quad \text{in GF(q)} \tag{1}$$

where h_t is a nonzero value of the parity matrix H.
An important difference between non binary and binary LDPC decoders is that the former's messages that circulate on the factor graph are multidimensional vectors, rather than scalar values. Like the binary decoders, however, there are two possible representations for the messages : probability weights vectors or log-density-ratio (LDR) vectors. The use of the LDR form for messages has been advised by many authors who proposed practical LDPC decoders. Indeed, LDR values which represent real reliability measures on the bits or the symbols are less sensitive to quantization errors due to finite precision coding of the messages [8]. Moreover, 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 a LDR vector of a random variable $z \in GF(q)$:

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

where

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

with $P(z = \alpha_i)$ the probability that the random variable z takes on the values $\alpha_i \in GF(q)$ and $L[0] = 0, \quad L[i] \in \mathbb{R}$.
For example, the log-likelihood-ratio (LLR) vector message at the channel output is denoted $\mathbf{L}_{ch} = \{L_{ch}[k]_{k=\{0,\ldots,q-1\}}\}$ defined by q-1 terms of the type (2), the values of the probability weights $P(z = \alpha_i)$ depending on the transmission channel statistics. The decoding algorithm that we propose is independent of the channel, and we just assume that a demodulator provides a LLR vector $\mathbf{L}_{ch}$ that initializes the decoder (the BI-AWGN channel is used in our simulations, see section VI).
The GF(q)-LDPC iterative decoding algorithms are characterized by three main steps corresponding to the different nodes depicted in figure Fig. 1: *(i)* the variable node update, *(ii)* the permutation of the messages due to nonzeros 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 $O(q^2)$.

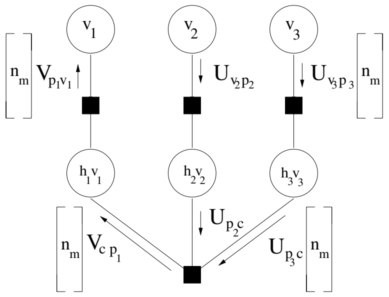

Fig. 1. Factor graph structure of a parity check node for a non-binary LDPC code

We use the following notations for the messages in the graph (see Fig. 1). Let $\{\mathbf{V}_{p_iv}\}_{i=\{0,\ldots,d_v-1\}}$ be the set of messages entering a variable node of degree d_v, and $\{\mathbf{U}_{vp_i}\}_{i=\{0,\ldots,d_v-1\}}$ 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 $\{\mathbf{U}_{p_ic}\}_{i=\{0,\ldots,d_c-1\}}$ ($\{\mathbf{V}_{cp_i}\}_{i=\{0,\ldots,d_c-1\}}$) at the input (output) of a 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 storage complexity.
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 $\mathbf{U}_{vp}$, $\mathbf{V}_{cp}$. This has the advantage of reducing the memory requirements, but also to further reduce the computational complexity with a proper algorithm which is described in details in section IV. Let us first present the way we truncate the messages from q to n_m values and discuss its 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 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 $\mathbf{A}$ *be any message in the graph which represents a LDR vector of size q. A truncated version* $\mathbf{B}$ *of* $\mathbf{A}$ *is composed of the* n_m *largest values of* $\mathbf{A}$ *sorted in decreasing order, plus an additional* (n_m+1)*-th value* $\gamma_A \in$ *, whose goal is to compensate for the information loss due to the truncation of* $q - n_m$ *values.* ■

The compensated-truncated message $\mathbf{B}$ 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 $\mathbf{B}$ would then be:

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

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 $\mathbf{A}$

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

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

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

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 $\sum_{j=0}^{n_m-1} P_B[j] + (q - n_m)P_{\gamma_A} = 1$ is satisfied. With P_{γ_A} the probability weight associated, with LDR value γ_A, that is $P_{\gamma_A} = P_A[0]e^{\gamma_A}$. The normalization of vector $\mathbf{P}_B$ is then:

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

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

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

and finally

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

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 [6], and in order to simplify the equation (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:

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

where $B[n_m]$ is the largest value among the $(q - n_m)$ ignored values of vector $\mathbf{A}$.
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 $\mathbf{A}$ 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 [9].
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:

$$\gamma_A = B[n_m] - \ln(q - n_m) - offset = B[n_m] - Offset \tag{7}$$

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 message $\{\mathbf{U}_{vp_i}\}_{i \in \{0,\ldots,d_v-1\}}$ associated to a variable node v passed to a check node c is 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:

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

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 $\beta_{V_{p_i v}}[k]$.
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 (Fig.2(a)) minimizes the number of elementary updates needed for both check node and variable node updates. Lets put first some notations on the recursive implementation, whose principle is to decompose the parity check and variable node into several *elementary steps* (Fig.2(b)). One elementary step assumes only two input messages and one output message. The decomposition of the check nodes (variable nodes) with degree $d_c \geq 4$ ($d_v \geq 3$) implies therefore the computation of *intermediate messages* denoted $\mathbf{I}$ (Fig.2(a)), which are assumed to be stored also with n_m values. The output $\mathbf{V}_{cp_i}$ ($\mathbf{U}_{vp_i}$) of a check node (variable node) is then - in most cases - computed from the combination of an input message $\mathbf{U}$ (respectively $\mathbf{V}$) and an intermediate message $\mathbf{I}$.

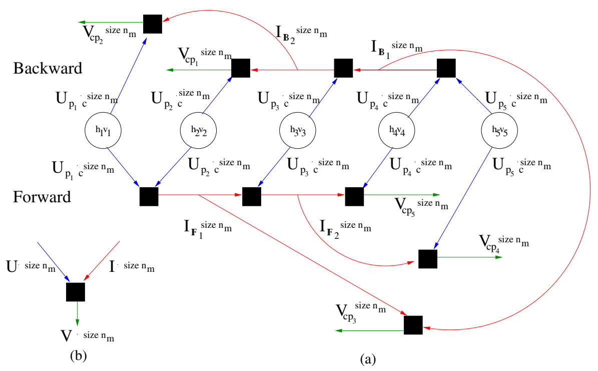

Fig. 2. The recursive structure of a degree $d_c = 5$ check-node (a); The elementary step (b)

B. Variable node: elementary step

Let assume that an elementary step describing the variable node update has $\mathbf{V}$ and $\mathbf{I}$ as input messages and $\mathbf{U}$ as output message. The vectors $\mathbf{V}$, $\mathbf{I}$ and $\mathbf{U}$ 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 [6], the goal of an elementary step is to compute the output vector containing the n_m largest values among the $2n_m$ candidates (9) (stored in an internal vector message $\mathbf{T}$). The processing of the elementary step in the case of a variable node update is described by:

$$T[k] = V[k]+Y \quad T[n_m+k] = \gamma_V+I[k] \quad k \in \{0,\dots,n_m-1\} \tag{9}$$

with

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

The compensation value γ is used when the required symbol index is not present in an input message. Whenever the $\mathbf{V}$ input corresponds to the LLR channel vector, the γ_V term (9) is substituted by $\mathrm{L}_{ch}[\beta_I[k]]$.

C. Low complexity implementation of a check node elementary step

The this section describes in details the algorithm that we propose for an elementary component of the check node (Fig.2(b)). 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 $\mathbf{U}$ and $\mathbf{I}$ as input messages and $\mathbf{V}$ 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:

$$V[i] = \max_{S(\beta_V[i])} (U[j] + I[p]) \qquad i \in \{0,\dots,n_m-1\} \tag{10}$$

Just as in the variable node update (step (2) in the previous section), when a required index is not present in the truncated vector $\mathbf{U}$ or $\mathbf{I}$, its compensated value γ is used in equation (10). 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 $\mathbf{U}$ and $\mathbf{I}$, that provide a minimum number of operations to process the n_m sorted values of the output vector $\mathbf{V}$. 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 $\mathbf{U}$ and $\mathbf{I}$ (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 $\mathbf{V}$. 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 steps of the algorithm are:

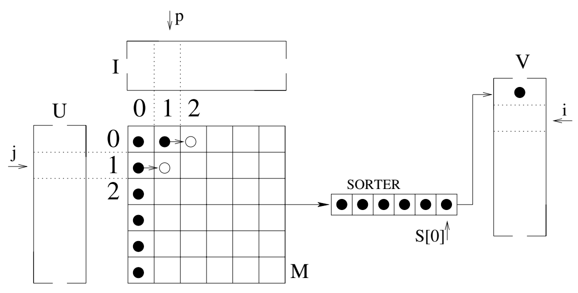

Fig. 3. Diagram of the low complexity algorithm

1) Initialization: the values of the first column 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 exists in the output vector.
   - Yes: no action
   - No: the value is moved in the vector $\mathbf{V}$
4) Evolution: The right neighbor - with regard to the M matrix - of the filled value is introduced in the sorter.
5) Go to step 2

In order to insure that all values of the output vector $\mathbf{V}$ 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 can correspond to the same α_V. Let us define K as the number of necessary steps so that all the n_m values of the output vector are computed. The parameter K is used to indicate the computational complexity of our new EMS implementation. We note that K$\in [n_m, \frac{n_m^2}{2}]$. Of course, the value of K depends on the LDR that compose the input vectors $\mathbf{U}$ and $\mathbf{I}$, and a strictly valid implementation of the elementary step should take into account the possibility of the worst case. However, we have found that K is most of the time quite small. As an example, Fig. 4 shows the discrete empirical probability density function of K for a regular GF(256)-LDPC code, $n_m = 32$ and a signal to noise ratio in the waterfall region of the code.

As a matter of fact, the distribution of K has an exponential

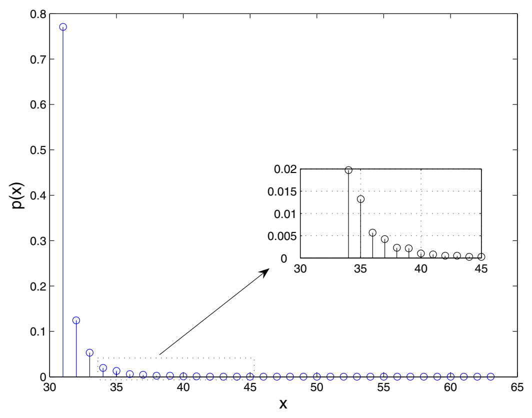

Fig. 4. Discrete probability density function of K

shape and decreases very rapidly, e.g. $prob(K \leq n_m + 4) = 0.9816$. Based on this observation, it seems natural to consider that the bad situations with large K 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 $K_{max} = 2n_m$ does not change the value of the decoding threshold for various LDPC code parameters. Note that with $K_{max} = 2n_m$, sometimes the output vector $\mathbf{V}$ 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 scenario for the complexity of an elementary step is then to $\mathcal{O}(K_{max} \log_2 n_m) = \mathcal{O}(2n_m \log_2 n_m)$, which corresponds to the number of $\max$ operations needed to insert K_{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.

V. Complexity and memory evaluation of the algorithm

The computational complexity 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

TABLE I
Computational complexity of the message updates with the EMS algorithm and messages of size n_m

	Complexity check node
No. max	$3(d_c-2)K_{max}\log_2 n_m$
No. real add	$3(d_c-2)(K_{max}+n_m)$
No. add over GF(q)	$3(d_c-2)(K_{max}+n_m)$

	Complexity variable node
No. max	$(d_v+3(d_v-2))n_m\log_2(2n_m)$
No. real add	$(d_v+3(d_v-2))2n_m$

size n_m, even for the channel LLR vector $\mathbf{L}_{ch}$ 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 $K = K_{max}$ and that the average complexity is often less than that.

The complexity associated with the update of vectors $\mathbf{U}$ 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 results, the complexity of our decoding algorithm is dominated by $\mathcal{O}(n_m \log_2(n_m))$ in the case $K_{max} = 2n_m$ for both parity and variable nodes computation. Interestingly, the complexity of a check node respectively 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 [5], [6], [7].

The memory space requirements of the decoder is composed by two independent memory components, that correspond to the channel messages $\mathbf{L}_{ch}$ and 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. So, 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 tune 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.5 plot 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).

As expected, the thresholds become better as n_m increases, and can approach closely the threshold of BP with much less complexity. The BP thresholds are equal to $\delta = 0.58dB$ for the GF(64) code and $\delta = 0.5dB$ for the GF(256) code [7]. We can use the plots on Fig.5 as first indication for choosing the field order of the LDPC code that corresponds to a given complexity/performance trade off. Note however

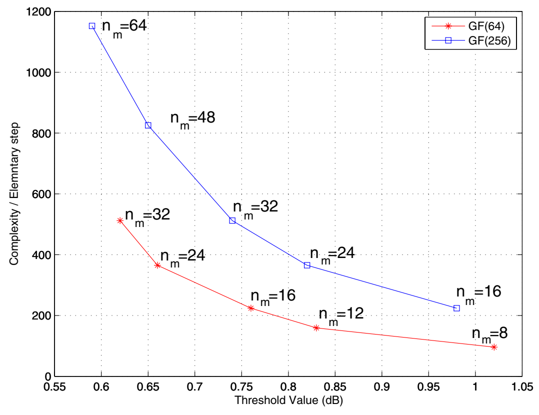

Fig. 5. Estimated decoding threshold vs. Complexity

that this asymptotic study has to be balanced with the girth properties of finite length codes, since it has been identified in [2], [3] that ultra-sparse LDPC codes in high order fields and with high girth have excellent performance.

VI. Experimental Results

In this section, we present the simulation results of our low complexity EMS algorithm, compared with the BP algorithm. In order to make a fair comparison, floating point implementation have been used for both algorithms, and we will discuss in a future paper the effect of quantization on the EMS decoder performance. We focus on a regular GF(q)-LDPC code over high order fields, of rate $R = 1/2$ ($d_v = 2, d_c = 4$). In Fig.6, we have reported the frame error rate (FER) of short code of length $N_b = 848$ equivalent bits, at convergence.

We denote by $\text{EMS}_{\text{GF(q)}}^{n_m}$ the EMS decoder over the field GF(q)

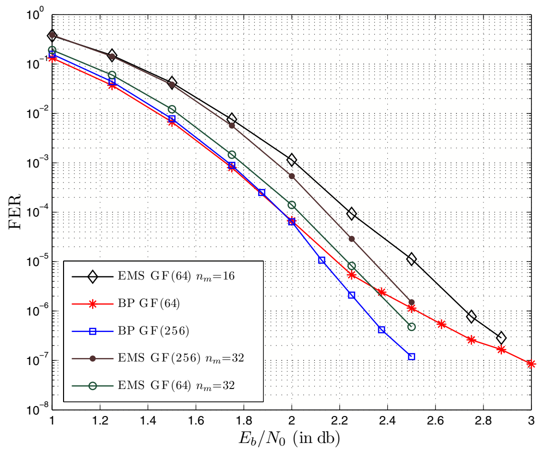

Fig. 6. Comparison between BP and EMS decoding algorithms, floating point implementation

with parameter n_m. Let us first compare our low complexity decoder to the BP decoder. For the code over GF(64), the $\text{EMS}_{\text{GF(64)}}^{16}$ is the less complex algorithm presented. It performs within $0.25dB$ of the BP decoder in the waterfall region. The $\text{EMS}_{\text{GF(64)}}^{32}$ algorithm has $0.05dB$ performance loss in the waterfall region and performs even better than the BP decoder in the error floor region. This behavior is now well known in the literature and comes from the fact that for small code lengths, an EMS algorithm corrected by an offset is less sensitive to pseudo-codewords than the BP.

Note that with this example, there is no advantage of using a GF(256) code in terms of performance/complexity trade off since $\text{EMS}_{\text{GF(64)}}^{32}$ has better performance than $\text{EMS}_{\text{GF(256)}}^{32}$ for all considered (E_b/N_0), although they share the same complexity. Our low complexity algorithm is however quite robust since the complexity reduction from $q = 256$ to $n_m = 32$ is a lot higher than from $q = 64$ to $n_m = 32$, which was a drawback of the solutions proposed in [5], [6]. The same kind of behavior have been observed for other code and decoder parameters.

VII. 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 same 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.

The proposed low complexity, low memory EMS decoding algorithm then becomes a good candidate for the hardware implementation of non binary LDPC decoders. Since its complexity and its memory space has been greatly reduced compared to the other suboptimal decoding algorithms for the non binary LDPC codes and the performance degradation is small or negligible.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] D. Declercq and M. Fossorier, “Decoding Algorithms for Nonbinary LDPC Codes over GF(q)”, to appear in *IEEE Trans. on Commun.*, 2007.

[8] L. Ping and W.K. Leung, “Decoding low density parity check codes with finite quantization bits”, *IEEE Commun. Lett.*, pp.62-64, February 2000.

[9] 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.