Non-Binary LDPC Codes defined over General Linear Group: finite length design and practical implementation issues .

W. Chen*, C. Poulliat*, D. Declercq*, L. Conde-Canencia†, A. Al-Ghouwayel and E. Boutillon†
* ETIS, ENSEA, University of Cergy-Pontoise, CNRS
F-95000, Cergy-Pontoise, France
Email: {weigang.chen, poulliat, declercq}@ensea.fr
† Lab-STICC (UMR 3192 UBS/CNRS)
Centre de Recherche, BP 92116 Lorient 56321 Cedex, France
Email: {laura.conde-canencia, ali.al-ghouwayel, emmanuel.boutillon}@univ-ubs.fr

[1]

***Abstract*—Non-binary LDPC codes are now recognized as a potential competitor to binary coded solutions, especially when the codeword length is small or moderate. More and more works are reported with good performance/complexity tradeoffs, which make non-binary solutions interesting for practical applications, such as 4G-wireless systems or DVB-like systems.**

In this paper, we show that proposing non-binary LDPC codes build on finite fields is actually a limitation, both from performance and implementation aspects. By considering non-binary codes on the general linear group, we show in particular that one can obtain a slight performance improvement compared to Galois field codes, with reasonable additional cost in the hardware implementation. The performance gain is quite small, but comes at a slight extra decoding cost, and is obtained by proper generalization of the code optimization techniques that are standard for non-binary LDPC codes on fields.

I. Introduction

It has been shown in several recent papers that non-binary codes can have very good performance/complexity tradeoffs, when the order of the Galois field GF(q) in which the codes are considered is high $q \geq 64$ and when the minimum symbol node connexion $d_v = 2$ is used for the Tanner graph of the code [1], [2]. Those codes are then nowadays considered as real competitors to binary LDPC and Turbo-codes in the future standarts of digital communication (4G, DVB, etc).

In this paper, we deal with a much more general family of non-binary LDPC code, that is generalized low-density parity-check (GLDPC) codes over the general linear group (GLG). By considering codes in a wider ensemble, it is therefore possible to find better codes without changing the Tanner graph density or the order of the symbols finite set, therefore without increasing significantly the decoding complexity. We have therefore generalized the approaches proposed in [1] from field codes to codes over the general linear group, and proposed an efficient hardware implementation of the generalized decoder, which improves the performance complexity tradeoff of the coding system.

[1]this work is supported by the European FP7 ICT-STREP DAVINCI project

In Section II, we give a brief introduction to non-binary LDPC codes defined on the general linear group. Then, in Section III, we present the optimization method that we use as efficient code design. This approach is indeed a generalization of the work presented in [1]. We show in particular that it is possible to choose better code components (corresponding to a single non-binary parity check) in the GLG set than in the corresponding field GF(q), then we adapt the global minimum distance maximization using a local full rank criterion (FRC) to the GLG case. Then, in Section IV, we present the impact of using GLG codes instead of field codes in the decoder architecture, and show that the extra cost induced can negligible. Finally, a performance comparison between optimized field codes and optimized GLG codes is made in Section V. For code rate R$= 1/2$ and various lengths, we show a slight improvement in the waterfall region with expected neglegible decoding complexity, while comparison at rate R$= 3/4$ shows an improvement both in the waterfall and the error floor.

II. Non-binary LDPC codes defined on the general linear group.

A. Non-binary LDPC codes ensembles

An LDPC code is a linear block code defined on a very sparse parity-check matrix $\mathbf{H}$ with the dimensions of $M \times N$, which can be defined over the binary Galois field or high order Galois fields. Let $\mathbf{x} = [x_0 \dots x_{N-1}]$ be a codeword. If the code is defined over a finite field GF(q) with $q = 2^p$, the $i-$th parity check equation can also be written as

$$\sum_{j:h_{ij} \neq 0} h_{ij} x_j \equiv 0$$

where h_{ij} are non-zero elements from GF(q). This definition can be generalized to the case of non-binary code ensembles defined over the general linear group [3] or more generally to the case of codes defined over the finite Abelian group $G(2^p) = \mathbb{F}_2^p$ [4]. In these cases, the $i-$th parity check equation

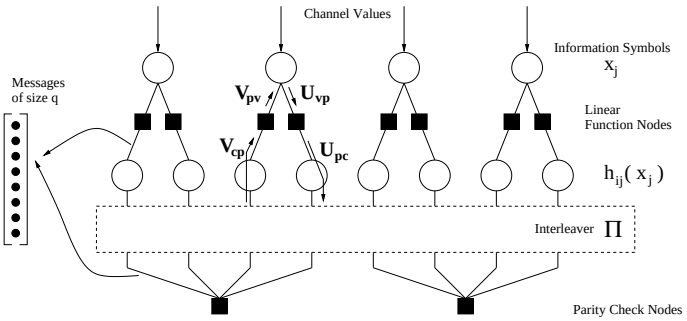

Fig. 1. Tanner graph of an LDPC code defined over a finite group $G(q)$.

can be written as

$$\sum_j h_{ij}(x_j) \equiv 0 \text{ in } G(2^p) \tag{1}$$

where $h_{ij} : G(2^p) \rightarrow G(2^p)$ is a linear function, also called mapping, associated with each edge of the non-binary Tanner graph representing the parity-check matrix $\mathbf{H}$. The corresponding Tanner graph of a non-binary code defined over the GLG is given in Figure 1.

Using this representation, for codes defined over a field GF(q), the $2^p - 1$ mapping functions reduce to the multiplicative group of GF(q) and they are equivalent to cyclic permutations. For the general linear group, the set $\mathcal{H}$ of the mappings $h_{ij}(.)$ is the set of the bijective linear mappings whose cardinality is $|\mathcal{H}| = \prod_{n=0}^{p-1}(2^p - 2^n)$ [3], whereas for the last case of codes defined over an Abelian group $h_{ij}(.)$ can be either linear or non linear mappings [4].

In the following, we will only consider the design of non binary LDPC codes defined over the general linear group. As is the case for non-binary LDPC codes over GF(q), we can easily derive an equivalent binary representation of the non-binary LDPC codes defined on the GLD. To this end, let us consider the binary mapping of the non-binary symbols over $G(2^p)$. The symbols x_j can be represented using p-tuples defined over GF(2). Using a binary vector notation, we can write

$$\sum_{j:\mathbf{H}_{ij}\neq\mathbf{0}} \mathbf{H}_{ij}\mathbf{x}_j^T = \mathbf{0}^T \tag{2}$$

where $\mathbf{H}_{ij}$ is the $p \times p$ invertible matrix over the binary field associated with the bijective linear mapping $h_{ij}(.)$, $\mathbf{x}_j$ is the p-tuple associated symbol element x_j and T holds for transpose. The vector $\mathbf{0}$ is the all zero component vector. Note that the mappings $h_{ij}(.)$ define now some permutations of the p-tuples. Considering the $i-$th generalized parity-check equation of $\mathbf{H}$ involving exactly $d_{c,i}$ codeword symbols, we can define $\mathbf{H}_i$ as $\mathbf{H}_i = [\mathbf{H}_{ij_0} \cdots \mathbf{H}_{ij_m} \cdots, \mathbf{H}_{ij_{d_{c,i}-1}}]$ as the equivalent binary parity-check matrix of the $i-$th row constraint, with $\{j_m : m = 0, 1, \cdots, d_{c,i} - 1\}$ is the set of the indexes of the codeword symbols involved in the $i-$th constraint. Let $\mathbf{X}_i = [\mathbf{x}_{j_0} \cdots \mathbf{x}_{j_{d_{c,i}-1}}]$ be the binary representation of the symbols of the codeword $\mathbf{x}$ involved in the $i-$th generalized parity-check equation. When using the binary representation, the $i-$th parity-check equation of $\mathbf{H}$, can be finally written as $\mathbf{H}_i\mathbf{X}_i^T = \mathbf{0}^T$. We define $d_{\min}^{(i)}$ as the minimum distance of the binary code associated with $\mathbf{H}_i$. This representation will be used in the following to select good code components. More generally, when using the binary matrix representation of mapping functions, we can associate with the overall code an equivalent binary code with associated parity-check matrix $\mathbf{H}_b$.

B. Decoders for non-binary LDPC codes

Aside from the Belief propagation decoder on GF(q) [5], [6], other types of decoders have been introduced in the litterature to decode efficiently non-binary codes in high order fields [2]. The different proposed decoders are essentially different for the symbol and/or the checknode updates, but are exactly the same with respect to the edge computation and especially in their treatment of the non-zeros labels supported by the edges of the Tanner graph (depicted as "linear function nodes" on figure 1). For example, a BP-like decoder with proper modifications to encompass the GLG case is shortly depicted below in this section. A thourough discussion on the efficient implementation of the *edge update* or *linear function nodes update* is conducted in section IV.

We will refer to the belief propagation decoder on group as *groupBP* decoder. The Tanner graph of an LDPC code over a finite group is depicted on Figure 1, in which we indicated the notations we use for the vector messages. Additionally to the classical variable and check nodes, we add function nodes to represent the effect of the bijective linear mappings $h_{ij}(.)$. The groupBP decoder has four main steps which use $q = 2^p$ dimensional probability messages:

- *Data node update:* the output extrinsic message is obtained from the term by term product of all input messages, including the channel likelihood message, except the one carried on the same branch of the Tanner graph.
- *Function node update:* the messages are updated through the function nodes $h_{ij}(.)$. In the case of general linear functions from $G(2^p)$ to $G(2^p)$ denoted $\beta = f_{ij}(\alpha)$, the update operation is:

$$U_{pc}[\beta_j] = \sum_i U_{vp}[\alpha_i] \qquad j = 0 \ldots q-1, \;\; \beta_j = h_{ij}(\alpha_i)$$

- *Check node update:* this step is identical to BP decoder over finite fields and can be efficiently implemented using a Fast Fourier Transform. See e.g. [4] for more details.
- *Inverse function node update:* the update equation is

$$V_{pv}[\alpha_i] = V_{cp}[\beta_j] \qquad \forall \alpha_i \, : \, \beta_j = h_{ij}(\alpha_i)$$

We do not have enough space in this paper to present in details the BP equations, or its reduced compexity versions, but the reader can refer to [4], [3], [2] for complete details.

III. Finite length design of regular $(2, d_c)$ non-binary code over GLG.

For codes defined over GF(q), when addressing finite length design, it has been shown in [5] and [1] that selecting

carefully the non binary entries of the parity-check matrix can improve the overall performance of the code when compared to randomly chosen coefficients. The selection of the non zero values can impact both on the waterfall and the on error floor. The observed performance gains are dependent of both the field order and the code rate.

In the waterfall region, selecting the edges label row-wise is critical. It is shown in [1] that "best" rows are selected according to their equivalent binary minimum distance and multiplicity of the minimum distance. In addition to that, for ultra-sparse non-binary codes (i.e. strictly regular $(2, d_c)$ codes, also called cyclecodes), it has been also shown in [1] that it is possible to lower the error floor by avoiding low weight codewords induced by some algebraic topological structures of the underlying Tanner graph, such as cycles or stopping sets. Choosing properly the edge labels of the stopping sets has a direct influence on the local minimum distance of the code, and therefore on the global minimum distance as well. Since the error floor performance of ultra-sparse non-binary LDPC codes are limited by the global minimum distance and not the pseudo-distance as for the binary LDPC codes, it is therefore very important to maximize the minimum distance of the code by proper optimization. In this paper, we aim at generalizing the method proposed in [1] to the case of non-binary codes over the general linear group. We will restrict our contribution to the case of ultra-sparse non binary codes for two main reasons:

- (a) It has been shown in [1][7] that ultra-sparse codes can perform very well under iterative decoding, being competitive in both the waterfall and the error-floor region in comparison with the state-of-the-art iteratively decodable codes.
- (b) it has been pointed out by [3] that codes defined over GF(q) and G(q) of the same order seem to have approximately the same thresholds. As a consequence, the same behavior under iterative decoding is expected, indicating that the use of LDPC edge distributions well suited for iterative decoding over GF(q) is a reasonable choice for G(q) codes.

Before describing in detail our design method, let us first introduced the main features. Basically, the proposed finite length design is based on two main steps: (i) building the graph, $i.e.$ optimizing the edge connections and (ii) selecting the non zeros entries of H, $i.e.$ choosing carefully the application $h_{ij}(.)$. The first part can be efficiently addressed using some instances of the PEG algorithm [8] [9], aiming at maximizing the local girth. It can be shown as a first requirement to ensure good achievable minimum distance when considering direct extensions of the results in [1] to our case. Then, non zeros entries are selected carefully to ensure both good waterfall behavior and low error floors.

To this end, we will first consider the search for good code components, $i.e.$ having good minimum distance properties. Based on these sets of potentially good codes, we then describe how we perform the optimization using random bitwise permutations of these component codes.

A. Component Code Selection

In [1], the authors selected the best row entries according to the maximum of $d_{\min}$ using the equivalent binary parity-check matrix of each row. This can be naturally extended to the case of non-binary code ensemble defined over the GLG by selecting component codes having good minimum distance and minimum multiplicity for each row. Using this selection criterion, we can then consider better codes in the GLG than in GF(q). For example, the best d_c-tuples of coefficients for GF(64) with $d_c = 4$ have minimum distance of 3 in GF(q) while it is possible to consider component codes with $d_{\min} = 4$ in the GLG. This will have a direct impact on the waterfall of the LDPC code.

This motivates the search for components codes achieving the best bound in terms of minimum distance and multiplicity [10]. For our example, this can be done for example by finding good codes using carefully chosen shortened versions of a $(63, 57)$ Hamming code, or by shortening a $(32, 26)$ extended-Hamming code. This is not the optimal choice, but we will see in the performance results section that this example is sufficient to ensure some gain compared to GF(q) LDPC codes. Future work will aim at considering the optimum choice for the component codes. Once a code component or a collection of code components has been selected, we can easily generate other good codes using bitwise permutations. We make use of the bitwise permutation technique in order to maximize the global minimum distance with a generalization of the algorithm presented in [1]. This optimization method is depicted in the next section.

B. Code Optimization with Random Permutations of One Component Code

In this paper, we further propose rank-guaranteed random bitwise permutations to expand the possible entries which can construct a component code with good distance properties. With bitwise permutations, more component codes with good minimum distance can be generated ensuring the diversity of the non zero entries to fully benefit from the optimization procedure of [1]. We can denote the bitwise permutation using a $pd_c \times pd_c$ permutation identity matrix $\mathbf{\Pi}$, and then d_c different non-zero entries over $G(2^p)$ are achieved as follows.

$$\begin{aligned} \mathbf{H}'_i &= [\mathbf{H}_{ij_0} \dots \mathbf{H}_{ij_m} \dots \mathbf{H}_{ij_{d_{c,i}-1}}] \cdot \mathbf{\Pi} \\ &= [\mathbf{H}'_{ij_0} \dots \mathbf{H}'_{ij_m} \dots \mathbf{H}'_{ij_{d_{c,i}-1}}] \end{aligned} \tag{3}$$

On one hand, we limit $\mathbf{H}'_{ij_m}$, $0 \leq m \leq d_{c,i}-1$, to be invertible or to be full rank. This guarantees that $h_{ij_m}(\cdot)$ is a bijective mapping or $\mathbf{H}'_{ij_m} \in G(2^p)$. In this way, all the function node update step can use the similar component and the decoder has a uniform architecture. Additionnaly, it is obvious that the good minimum distance property is maintained with the bitwise permutation and thus the new row entries are also the best entries in terms of minimum distance and its multiplicity. Then, when considering these permutations, we apply a direct generalization of the optimization method proposed in [1].

The optimization consists in iteratively selecting the randomly permuted rows to ensure the FRC condition for the cycles of the underlaying graph while maximizing the binary minimum distance over the set of the topological stopping sets.

IV. Efficient Implementation Architectures for Non-Binary LDPC Codes over GLG

Proposing efficient codes, like GLG LPDC, would only remain a good theoretical work if no consideration is given to its hardware implementation. In the case of GLG LDPC codes, the only difference between the non-binary LDPC decoder and the non-binary GLG LDPC decoder is the updating of the edges (defined by the mapping functions). In this section we recall some complexity issues in non-binary LDPC decoders and then we focus on the implementation of the edge updates.

A. GLD LDPC decoder architectures

So far, there are very few, if any, reported implementation of non-binary LPDC decoders. The only works deal with the simplification of the GF(q) check node (see [12] and [2]). Those papers reduce the complexity of the check node processing from $q \cdot log_2(q)$ and $n \cdot log_2(n)$ arithmetic operations, respectively ($n << q$). The last solution, the Extended Min-Sum (EMS), allows the implementation of an LDPC decoder at a hardware cost competitive with binary LDPC codes or Turbo-codes. The FP7 European project DaVinci aims to build such decoder [11], but in this paper, we only focus on the difference between non-binary LDPC decoder and GLG LDPC decoder, i.e., in the edge computation.

B. Edge computation

For a given edge, its associated permutation h can be represented in binary by a $P \times P$ binary matrix H. Using the binary representation $X = [x_0, x_1, ..., x_{p-1}]$ of the GF(q) symbol $\mathbf{x}$, the permutation $\mathbf{y} = h(\mathbf{x})$ is then given by $Y = H \cdot X^T$, where Y is the binary representation of GF(q) symbol $\mathbf{y}$. The direct storage of the H matrix requires p^2 bits and the computation of Y requires p^2 AND functions and p p-input XOR functions. With a classical non-binary LDPC decoder, a single GF(q) value is enough to characterise the transformation (i.e. p binary elements). At first glance, moving towards GLG LDPC increases by a factor p the size of the memory to store the edge transformation. However, there are simple tricks that allow to limit, or even avoid, this increase of memory.

First, if the architecture is fully parallel, i.e., a hardware unit is dedicated for each branch computation, then both Galois Fields and group permutation will have a same hardware complexity, since both implement a wired permutation. In this case, GLPDC has no hardware penalties compared to non-binary LDPC.

Second, there are exactly $2N$ edges on the bipartite graph of the code, i.e., $2N$ different permutations (and inverse permutation). It is possible to limit the search of permutation in a subgroup of permutation. In this case, instead of storing the whole permutation matrix, we can store only a value that defines the permutation. For example, all rotations on a vector of p bits can be characterised by an offset of p bits. More generally, if we define e elementary permutations $\{m_r\}_{r=0..e-1}$, then 2^e different permutations can be defined by an e-bit vectors $p = (p_{e-1}, ..., p_1, p_0)$ as $h_p = m_{e-1}^{p_{e-1}} \circ ... m_1^{p_1} \circ m_0^{p_0}$ where $\circ$ stands for the composition operator and $m_r^{p_r}$ is the identity permutation if $p_r = 0$, the m_r permutation if $p_r = 1$.

Finally, although not yet formally verified, the same permutation can be shared by several different edges. In this case, permutation matrices (or, as seen above, methods to generate a permutation) can be stored in a shared memory. An index is then associated to each edge to refer to the permutation associated to this edge. In that case, the increase of memory size is very limited.

To conclude, compared to a classical non-binary LDPC, a direct implementation of a GLG LDPC requires an increase of a factor p of the memory size. In this section, we proposed several ideas that could limit this increase.

V. Simulation results.

In this section, we present some simulation results for different code lengths and rates for some codes defined over G(64). The results will be compared to optimized codes defined over GF(64) with identical parameters. For our simulations, we are using non binary belief Propagation decoding over fields or groups using 100 decoding iterations.

First, we consider three rate one-half codes with code lengh $N = 48$ symbols defined over GF(64) and G(64) respectively. All codes have the same graph built using the RPEG algorithm [9]. For the code defined over GF(64), we perform the global minimum distance optimization using the method described in [1]. Then, for the last two codes, we consider the optimization using, in one case, component codes with minimum distance $d_{\min} = 3$ and, in the other case, $d_{\min} = 4$. The resulting codes are noted 'GLG-d3' and 'GLG-d4' respectively. For the 'GLG-d3' code, the component codes are obtained using bitwise interleaved versions of a shortened version of the $(63, 57)$ Hamming code. The bitwise permutations are carefully selected in order to mimimize the global minimum distance using a generalization of the method described in [1]. For the 'GLG-d4'code, the component codes are obtained using bitwise interleaved versions of a shortened version of the extented $(32, 26)$ Hamming code. Then, the optimization is performed as previously described. Simulation results are given in figure 2. The results show that all codes behave almost the same in the error floor region, but the optimized GLG code with $d_{\min} = 4$ exhibits a slight performance improvement in the error floor region. In this case, the 'GLG-d4' code seems to outperfom the 'GLG-d3' code based on a "weaker" component code. The same behavior has been also observed for rate one-half codes with code lengh $N = 192$ symbols defined over GF(64) and G(64) respectively. As shown in figure 3, the error floors are the same and the waterfall improvement is almost 0.1 dB. Here, the 'GDG-d4' code outperfoms clearly the 'GDG-d3' code. Note that the codes have been optimized with the same parameters as before in all three cases but targeting a different codelength. Finaly, we have designed a code for

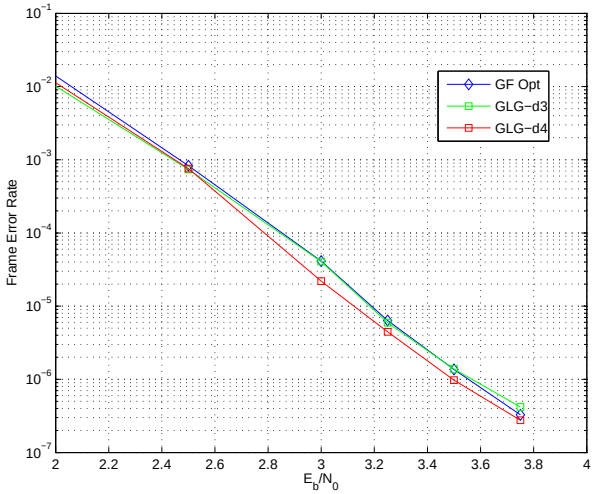

Fig. 2. Performance comparison of rate $R = 1/2$ codes defined over GF(64) and G(64) for $N = 48$ symbols.

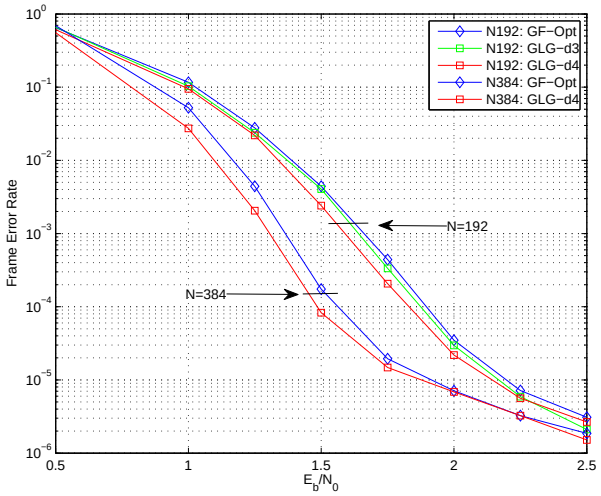

Fig. 3. Performance comparison of rate $R = 1/2$ codes defined over GF(64) and G(64) for $N = 192$ and $N = 384$ symbols

the code length $N = 384$. As seen in figure 3, the error floors are exactly the same, but the designed 'GLG-d4' code outperfoms the GF(q) code by 0.1 dB in the waterfall. Finally, following the same optimization method, we have performed the optimization of a rate $R = 3/4$ code of length $N = 192$. For that rate et codeword length, it can be seen in figure 4 that the GLG code outperforms slightly the code over GF(q) both in the waterfall and in the error floor region.

VI. Conclusion

In this paper, we have shown that by considering non-binary codes on the general linear group, one can obtain a slight performance improvement compared to Galois field codes, with reasonable additional cost in the hardware implementation.

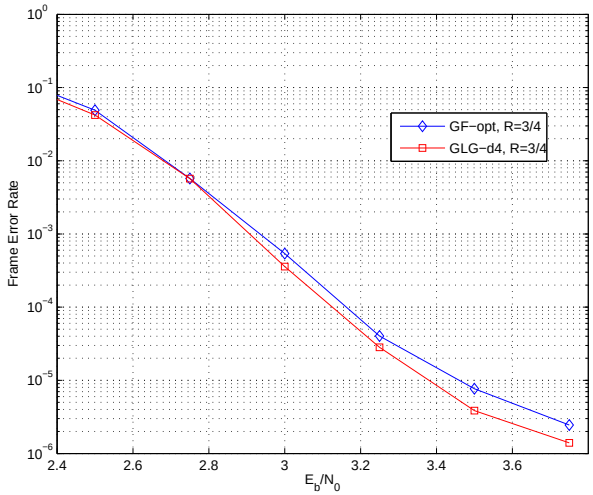

Fig. 4. Performance comparison of rate $R = 3/4$ codes defined over GF(64) and G(64) for $N = 192$ symbols.

The performance gain is quite small, but comes at a slight extra decoding cost, and is obtained by proper generalization of the code optimization techniques that are standard for non-binary LDPC codes on fields.

References

[1] C. Poulliat, M. Fossorier and D. Declercq, "Design of regular $(2, d_c)$-LDPC codes over GF(q) using their binary images", *IEEE Trans. on Commun.*, vol. 56, no. 10, pp. 1626–1635, October 2008.

[2] A. Voicila, D. Declercq, F. Verdier, M. Fossorier and P. Urard, "Low-Complexity Decoding for non-binary LDPC Codes in High Order Fields", to appear in IEEE Trans. Commun., 2009.

[3] V. Rathi and R. Urbanke, "Density evolution, thresholds and the stability condition for non-binary LDPC codes," *IEE Proceedings-Communications*, Vol. 152, No. 6, pp. 1069 – 1074, Dec. 2005

[4] A. Goupil, M. Colas, G. Gelle and D. Declercq, "FFT-based BP Decoding of General LDPC Codes over Abelian Groups," *IEEE Trans. on Commun.*, vol. 55, no. 4, pp. 644-649, April 2007.

[5] M.C. Davey and D. MacKay, "Low-Density Parity-Check Codes over GF(q)", *IEEE Commun. letters*, vol. 2, pp. 165C167, June 1998.

[6] L. Barnault and D. Declercq, "Fast Decoding Algorithm for LDPC over GF(2^q)", IEEE ITW, Paris, France, 2003.

[7] X.Y. Hu and E. Eleftheriou, "Binary Representation of Cycle Tanner-Graph GF(2^q) codes",*EEE ICC*, Paris, France, June 2004.

[8] X.-Y. Hu, E. Eleftheriou and D.M. Arnold, "Regular and Irregular Progressive Edge-Growth Tanner Graphs", IEEE Trans. on Inf. Theory, Vol. 51, No. 1, pp. 386–398, January 2005.

[9] A. Venkiah, D. Declercq and C. Poulliat, "Design of Cages with a Randomized Progressive Edge Growth Algorithm", *IEEE Commun. Letters*, vol. 12(4), pp. 301-303, April 2008.

[10] T. Verhoeff, "An updated table of minimum-distance bounds for binary linear codes," *IEEE Trans. Inform. Theory*, vol. IT-33, no. 5, pp. 665-680, Sept. 1987.

[11] DaVinci project, FP7 INFSO-ICT-216203, http://www.ict-davinci-codes.eu/...

[12] H. Wymeersch, H. Steendam, and M. Moeneclaey, "Log-domain decoding of LDPC codes overGF(q)," in Proc. IEEE Int.Conf. Commun., Paris, France, Jun. 2004, pp. 772-776