Tree Expectation Propagation for ML Decoding of LDPC Codes over the BEC

Luis Salamanca, *Student Member, IEEE*, Pablo M. Olmos, *Member, IEEE*, Juan José Murillo-Fuentes, *Senior Member, IEEE* and Fernando Pérez-Cruz, *Senior Member, IEEE*

***Abstract*—We propose a decoding algorithm for LDPC codes that achieves the maximum likelihood (ML) solution over the binary erasure channel (BEC). In this channel, the tree-structured expectation propagation (TEP) decoder improves the peeling decoder (PD) by processing check nodes of degree one and two. However, it does not achieve the ML solution, as the tree structure of the TEP allows only for approximate inference. In this paper, we provide the procedure to construct the structure needed for exact inference. This algorithm, denoted as generalized tree-structured expectation propagation (GTEP), modifies the code graph by recursively eliminating any check node and merging this information in the remaining graph. The GTEP decoder upon completion either provides the unique ML solution or a tree graph in which the number of parent nodes indicates the multiplicity of the ML solution. We also explain the algorithm as a Gaussian elimination method, relating the GTEP to other ML solutions. Compared to previous approaches, it presents an equivalent complexity, it exhibits a simpler graphical message-passing procedure and, most interesting, the algorithm can be generalized to other channels.**

***Index Terms*—ML decoding, LDPC codes, tree-structured expectation propagation, graphical models, binary erasure channel.**

I. Introduction

LOW-DENSITY PARITY-CHECK (LDPC) codes were proposed by Gallager in 1963 [2] as a remarkable near Shannon limit coding technique [3], [4]. The decoding of LDPC codes can be solved by message-passing algorithms over a graphical model [5], yielding linear time practical decoders [6]–[9]. Belief propagation (BP) is a message-passing algorithm typically considered for decoding LDPC codes, as it efficiently approximates posterior marginal probabilities [10]. In tree-like graphs, BP provides the maximum *a posteriori* (MAP) probability for each bit [10]–[12]. However, in graphs with cycles the BP is not able to achieve the MAP solution, or it may not even converge at all [8], [10].

In the asymptotic case, when we consider infinitely-large codes, density evolution (DE) [8] and extrinsic information transfer (EXIT) charts [13], [14] are used to predict the BP performance over LDPC codes. This prediction is used to optimize the LDPC degree distribution to maximize the BP decoding threshold, i.e. to approach capacity, [7], [15]–[19]. Ensembles of codes optimized in this way present highly irregular degree distributions, for which the convergence of the BP decoder to the asymptotic case is slow, i.e. large block sizes are needed [20]. Besides, they present irreducible asymptotic error floors because of their low minimum distance [9], [21]. Alternatively, we can overcome the error floor problem [22] if we consider spatially-coupled LDPC codes to approach capacity with BP [23]. However, the convergence to the asymptotic case is even slower, requiring extremely large code lengths (larger than 50000 bits) to perform optimally [24].

Regular LDPC codes under MAP decoding also approach capacity as the number of ones per column of the parity check matrix, namely the variable degree, increases [9], [25]. Even for fairly sparse graphs, the gap to capacity is negligible [23], e.g. the multiplicative gap to capacity for a rate-$1/2$ regular LDPC code with variable degree five is 10^{-3}. The BP fails to converge to this solution and, as the code density increases, its performance degrades rapidly. The Maxwell decoder [9] achieves the MAP solution by guessing some variables when the BP gets stuck. Nevertheless, the selection of these guessed variables and the order they are processed is not prespecified, and it may significantly affect the runtime complexity. In the context of Raptor codes, inactivation decoding [26], [27] also provides the MAP solution with reduced complexity thanks to the design of the codes, but it is not generalizable to other channels.

The expectation propagation (EP) [28] algorithm is a generalization of the BP, which uses a tractable approximation to perform inference in an intractable graphical model. The EP algorithm is equivalent to the BP when it uses a product of independent marginals as approximation. A natural extension of EP to improve the BP is to impose a tree structure over the given graph [29], [30] and apply EP to get a more accurate approximation for each marginal. This algorithm, hereafter referred to as tree-structured EP or TEP, allows capturing some of the information provided by the interactions between the variables in the graph. In a novel work, we have put forward the TEP decoder for LDPC codes over the BEC [31], [32], borrowing from the tree-structured EP algorithm. The TEP decoder is able to continue decoding when the BP gets stuck, since it solves relations between two variables. It is proven in [31], [32] that the TEP exhibits an improvement with respect to the performance of the BP, but not fully achieving the

This work was partially funded by Spanish government (Ministerio de Educación y Ciencia TEC2009-14504-C02-01/02, Consolider-Ingenio 2010 CSD2008-00010), and the European Union (FEDER).

L. Salamanca and J.J. Murillo-Fuentes are with the Dept. Teoría de la Señal y Comunicaciones, Escuela Superior de Ingenieros, Universidad de Sevilla, Camino de los Descubrimientos s/n, 41092 Sevilla, Spain. E-mail: {salamanca,murillo}@us.es

Pablo M. Olmos and F. Pérez-Cruz are with the Dept. Teoría de la Señal y Comunicaciones, Universidad Carlos III de Madrid, Avda. de la Universidad 30, 28911, Leganés (Madrid), Spain. E-mail: {olmos,fernando}@tsc.uc3m.es

Some preliminary results on some of these issues were presented at [1].

MAP solution. The TEP only considers relations between two variables to ensure that the degree of the check nodes does not increase. Hence the complexity of the algorithm is not compromised.

In this paper, we extend the TEP by exploiting relations between any set of variables. The resulting approach, the so called generalized TEP (GTEP), is a graph-based decoder that provides the maximum likelihood (ML) solution for LDPC codes over the BEC. If a unique solution is found, the ML and MAP solutions coincide, because we have absolute certainty about the probability of each bit. When the ML solution is not unique, the ML decoder should provide the set of equally likely sequences, while the MAP provides the posterior probabilities for the unknown bits. The ML solution provides more valuable information, because in the MAP solution we might have a significant chunk of erased bits with 50/50-chance, while the ML solution provides the exact number of possible chains and the exact bit values for them.

Over the erasure channel, the GTEP decoder removes a variable per iteration either revealing it or relating it to the remaining unknown variables in the graph. As a consequence, the GTEP algorithm either provides the unique ML solution or a directed acyclical graph (DAG), referred to as tree graph of relations, which contains the remaining unknown variables. The set of parent nodes indicates the multiplicity of the ML solution. The GTEP can be reinterpreted as a Gaussian elimination procedure, resulting in a complexity equivalent to that of the best strategy of the ML decoder in [33]. The GTEP algorithm for the BEC naturally provides a decoding with reduced complexity. Furthermore, it allows for an extension to other discrete channels, by exploiting the properties of the EP as inference method. For the TEP, we have presented some preliminary results in this line [34].

The rest of the paper is organized as follows. We present the general expectation propagation algorithm in Section II. We show in Section III that the EP for the BEC with standard approximations yields the peeling and TEP decoders. The GTEP decoder is developed in Section IV. We prove that the GTEP decoder achieves the ML solution in Section V, and we include in Section VI an analysis of its computational complexity. In Section VII, we study the performance of the GTEP for a regular ensemble. We end with some conclusions, in Section VIII.

II. Expectation Propagation

EP is an approximate algorithm to efficiently estimate marginal distributions in a factorized density function. EP imposes a tractable structure that it is optimized iteratively until it resembles the original density function, from which the marginal distributions are easily derived [28]. The original EP procedure only considers a product of single factors, which is similar to BP. In [29] the authors proposed a tree structure approximation, which provides more accurate solutions. We present this latter formulation in a different fashion from [29], to make it more amenable for LDPC channel decoding.

Let $\mathcal{C}$ be an LDPC code defined by its parity-check matrix $\mathbf{H}$. Given the received vector $\mathbf{Y}$, the aim of the bitwise-MAP decoder[1] is to choose each bit V_i to maximize:

$$p(V_i|\mathbf{Y}) = \sum_{\mathbf{V}\sim V_i} p(\mathbf{V}|\mathbf{Y}), \quad i = 1, \ldots, n, \tag{1}$$

where $\sum_{\mathbf{V}\sim V_i}$ denotes the sum over all variables in $\mathbf{V}$ except V_i. The density function $p(\mathbf{V}|\mathbf{Y})$ is obtained by Bayes' rule as:

$$p(\mathbf{V}|\mathbf{Y}) = \frac{p(\mathbf{Y}|\mathbf{V})p(\mathbf{V})}{p(\mathbf{Y})} \propto \prod_{i=1}^{n} p(Y_i|V_i) \prod_{j=1}^{m} \mathtt{C}_j(\mathbf{V}), \tag{2}$$

where $\mathtt{C}_j(\mathbf{V})$, for $j = 1, ..., m$, are the parity check constraints: $\mathtt{C}_j(\mathbf{V})$ is 1 if the parity check P_j is satisfied for $\mathbf{V}$, and 0 otherwise.

The marginalization in (1) has an exponential complexity with the code length n. Since the set of parity-check constraints is the intractable part of the factorized density function in (2), EP approximates these terms as multi-variable tree-like factor graphs [35]:

$$\mathtt{C}_j(\mathbf{V}) \approx W_j(\mathbf{V}) \triangleq \prod_{i=1}^{n} w_{i,j}(V_i, \mathbf{V}_{p_i}) \quad j = 1, \ldots, m, \tag{3}$$

where $w_{i,j}$ is a non-negative function and V_i and $\mathbf{V}_{p_i}$ are chosen such that (3) is tree-like. The full posterior probability in (2) is approximated by

$$q(\mathbf{V}) = \prod_{i=1}^{n} q_i(V_i|\mathbf{V}_{p_i}) = \frac{1}{Z} \prod_{i=1}^{n} p(Y_i|V_i) \prod_{j=1}^{m} W_j(\mathbf{V}), \tag{4}$$

where Z is a normalization constant and $q_i(V_i|\mathbf{V}_{p_i})$ represents the approximate conditional probability, i.e. $q_i(V_i|\mathbf{V}_{p_i}) \approx p(V_i|\mathbf{V}_{p_i}, \mathbf{Y})$. By using (3) in (4), this conditional probability yields:

$$q_i(V_i|\mathbf{V}_{p_i}) \propto p(Y_i|V_i) \prod_{j=1}^{m} w_{i,j}(V_i, \mathbf{V}_{p_i}). \tag{5}$$

For some variable nodes $\mathbf{V}_{p_i}$ might be missing, if only a single-variable factor $q_i(V_i)$ is needed to approximate them. EP computes $q(\mathbf{V})$ iteratively by minimizing the Kullback-Leibler divergence:

$$\begin{aligned} q(\mathbf{V}) &= \arg\min_{q(\mathbf{V})} D_{KL}\left(p(\mathbf{V}|\mathbf{Y})||q(\mathbf{V})\right) \\ &= \arg\min_{q(\mathbf{V})} \sum_{\mathbf{V}} p(\mathbf{V}|\mathbf{Y}) \log \frac{p(\mathbf{V}|\mathbf{Y})}{q(\mathbf{V})}. \end{aligned} \tag{6}$$

The minimization in (6) can be efficiently faced if both $q(\mathbf{V})$ and $p(\mathbf{V}|\mathbf{Y})$ belong to some exponential family [35]. In our scenario, we focus on discrete distributions and the optimization in (6) is accomplished by matching the moments between $q(\mathbf{V})$ and $p(\mathbf{V}|\mathbf{Y})$, iterating along the check node factors until convergence. In the ℓ-th iteration, EP performs the following two steps:

Step 1: It replaces one factor $W_{\mathtt{t}}(\mathbf{V})$ in (4) by the associated *true* term $\mathtt{C}_{\mathtt{t}}(\mathbf{V})$,

$$\widehat{p}(\mathbf{V}) = \mathtt{C}_{\mathtt{t}}(\mathbf{V}) q^{\ell/\mathtt{t}}(\mathbf{V}) \tag{7}$$

[1]For the sake of simplicity, hereafter we denote the bitwise-MAP decoder by just MAP decoder.

where

$$q^{\ell/\mathtt{t}}(\mathbf{V}) = \frac{q^{\ell}(\mathbf{V})}{W_{\mathtt{t}}(\mathbf{V})} = \prod_{i=1}^{n} p(Y_i|V_i) \prod_{\substack{j=1 \\ j\neq \mathtt{t}}}^{m} W_j(\mathbf{V}). \quad (8)$$

Step 2: It computes a new approximation $q^{\ell+1}(\mathbf{V})$ by matching the moments between with respect to $\hat{p}(\mathbf{V})$, which minimizes the KL divergence. Since they are discrete random variables, it reduces to the computation of the marginals,

$$q_i^{\ell+1}(V_i, \mathbf{V}_{p_i}) = \sum_{\mathbf{V}\sim V_i, \mathbf{V}_{p_i}} \widehat{p}(\mathbf{V}) \quad i = 1, \ldots, n. \quad (9)$$

The term $W_{\mathtt{t}}(\mathbf{V})$ is updated as $q^{\ell+1}(\mathbf{V})/q^{\ell/\mathtt{t}}(\mathbf{V})$. This information is used in following iterations [32]. For the BEC, $W_{\mathtt{t}}(\mathbf{V})$ simply yields the parity restriction between variables $(V_i, \mathbf{V}_{p_i})$, restraining the set of values that the involved variables can take.

Once term $W_{\mathtt{t}}(\mathbf{V})$ has been approximated, we return to *Step 1* and perform the optimization for another check node. When we introduce term $\mathtt{C}_{\mathtt{t}}(\mathbf{V})$ in (8), the graph for $\widehat{p}(\mathbf{V})$ might have loops. The marginalization in (9) can be readily computed using Pearl's cutset conditioning algorithm as proposed in [29], [36]. For the BEC this step is straightforward [32], as we later describe in Section III.

The tree-structured EP algorithm in [29] demands a particular factorization for the terms in (3) to be specified in advance. This factorization, or tree structure, might be amenable for some error patterns but useless for others. The GTEP algorithm, presented in Section IV, builds the tree on the fly for decoding LDPC codes over BEC, so it adapts to each error pattern to provide the ML solution with the least computational complexity. By iterating along the check nodes, the GTEP decoder for BEC defines a tractable factorization for the density function $q(\mathbf{V})$ in (4) in a simple and natural way. Unfortunately, for other channels, e.g. the binary-input memoryless symmetric (BMS) channel, the design of the factorization is not straight-forward [34].

III. Peeling and TEP decoder for decoding LDPC codes over the BEC

As discussed above, each particular factorizacion in (3) leads to different LDPC decoders. For example, if we use products of single factors, i.e. $w_{i,j}(V_i, \mathbf{V}_{p_i}) = w_{i,j}(V_i)$, the algorithm has the same solution as that of the standard BP procedure used for LDPC decoding. By including also pairwise factors, i.e. $w_{i,j}(V_i, \mathbf{V}_{p_i}) = w_{i,j}(V_i, V_{p_i})$, we have the TEP decoder in [31], [32]. In this section, we briefly review these two algorithms for BEC as a starting point for our GTEP decoder.

The PD [8], [20] is a graphical interpretation of the BP algorithm for decoding LDPC codes over the BEC [20]. This is usually described over the Tanner graph [37], [38] of the LDPC code. For an r-rate code, we have n variables nodes, $V_1, \ldots, V_n$, and $m = n \cdot (1 - r)$ parity checks, $P_1, \ldots, P_m$. The PD is initialized by removing all non-erased bits and complementing the parity of the check nodes connected to the removed variable nodes of value one. Then, at each iteration of the PD, we remove a check and decode a variable node by looking for a degree-one check node and copying its parity onto the variable it is connected to. This revealed variable can be removed from the graph, reversing the parity of the check node(s) it is connected to if its value is one. The PD finalizes whenever the transmitted codeword has been decoded (i.e. success), or there is no degree-one check node to process (i.e. failure). This interpretation is plausible, because the BP messages and values of variables in the graph move from complete uncertainty to perfect knowledge.

The TEP decoder [31], [32] can be understood as an extension of the PD that processes, each iteration, either a degree-one or a degree-two check node. A degree-two check node tells us that the variables connected to it are either equal, if the parity check is zero, or opposite, if the parity check is one. Therefore, we can remove a degree-two check node and one of the variables connected to it, and reconnect the edges of the deleted variable to the remaining one. The analysis of the TEP decoder is carried out in [31], [32], where the decoding threshold is shown to be equal to that of the BP. However, for finite length codes the TEP decoder is superior to the BP [39] with similar decoding complexity, i.e. linear in the code length $\mathcal{O}(n)$.

Processing a degree-two check node does not reveal a variable, it only relates two unknown variables. However, in some cases, after processing a degree-two check node, a degree-one check node is created. In Figure 1, we illustrate this scenario. The TEP decoder processes P_3 and removes V_2 from the Tanner graph, which is revealed once we decode V_1. V_1 inherits the connections of V_2, i.e. P_1, P_2 and P_4. Note that a double connection between a variable and a check node can be removed. Therefore, the TEP decoder creates a degree-one check node whenever the two variables connected to a degree-two check node also share a degree-three check node. In [31], [32], we show that the probability of this happening is negligible when we start decoding, but it grows as we process degree-one and degree-two check nodes, improving the BP performance for finite codes. Every time we process a degree-two check node, the remaining variable node either keeps or increases the number of connections, and the remaining check nodes either keep or decrease the numbers of connections. This is a desirable property, as it can be understood from the DE analysis [8], in which we prefer variables with more connections and factors with fewer connections to improve decoding.

IV. Generalized TEP decoder for the BEC

The GTEP decoder, as the PD and TEP decoders do, first removes all non-erased variables, flipping the parity of the check node(s) that were connected to the removed variable nodes of value one. Then, at each iteration, it removes a check and a variable node as follows:

Step 1: it selects a check node, P_q. Let l be its degree.

Step 2: it processes check node P_q:

1) if $l = 1$ it runs the PD.
2) if $l \geq 2$ it chooses one variable, V_i, out of the ones connected to P_q. It removes it along with P_q.

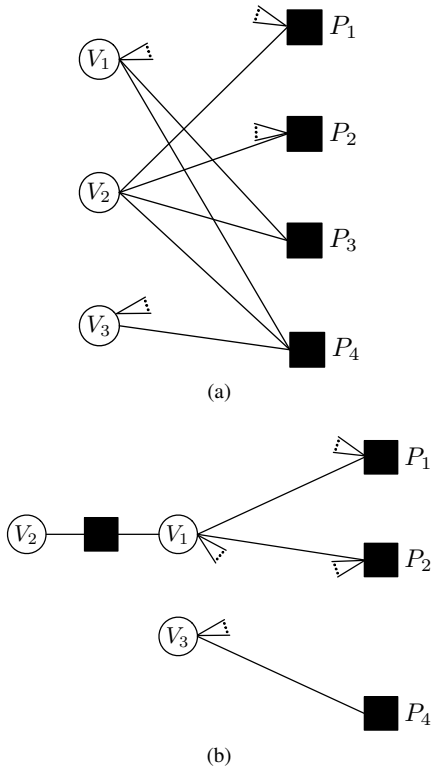

Fig. 1: In (a), we partly show the residual graph considered in an iteration of the TEP decoder. In (b), we illustrate the result when the parity-check node P_3 and the variable V_2 are processed by the TEP decoder, which creates a degree-one check that would not have been created by the BP decoder.

> The other parity-check node(s) connected to V_i are reconnected to all the variable(s) that were linked to P_q, denoted as $\mathbf{V}_{p_i}$, which become "reference" variable(s) of V_i. These parity-check node(s) are flipped if P_q is parity one.

Step 3: it goes to *Step 1* if there is any other check node to process, otherwise stop.

Choosing P_q to be processed is equivalent in the EP framework to recompute $q(\mathbf{V})$ by including $\mathtt{C}_q(\mathbf{V})$ in (7). Note that, with respect to the EP formulation in Section II, the GTEP does not fix a priori the set $\mathbf{V}_{p_i}$. It defines it on the fly as the others variables connected to P_q. Therefore, the parity restriction that we compute in (9) is unique, meaning that we always infer whether $(V_i, \mathbf{V}_{p_i})$ sum 0 or 1. After processing P_q, V_i is well defined as a function of $\mathbf{V}_{p_i}$. As a consequence, for each processed parity-check node of degree $l > 1$, we progressively construct a tree graph of relations. The parents of the removed variable V_i are the variables in the set $\mathbf{V}_{p_i}$.

When we process a check node with degree larger than two, the remaining check nodes might increase their degree. This is an undesirable consequence, because to decode erased variable nodes we need degree-one check nodes. Later, the matrix becomes denser and remaining variables share a large number of edges. Then, in the final steps of the decoder, after processing any check node, a lot of edges vanish. The GTEP finishes when all the parity-check nodes are processed. If the GTEP decodes all the parent nodes in the tree of relations, it successfully provides the transmitted codeword. If a set of d parent nodes are not decoded the tree provides the multiple ML solution, of cardinality 2^d.

We now illustrate through an example the different steps of the GTEP decoder, considering it selects a check node of lowest degree at each iteration. This assumption minimizes the algorithm complexity as we justify in Section VI. For simplicity, and without loss of generality, we assume the all-zero codeword has been transmitted. We start in Fig. 2a, where we have already processed all degree-one and degree-two check nodes and the PD and TEP decoders have failed to decode the correct word. The GTEP proceeds as follows:

1) It processes P_8 and removes V_8 along with it. The other check nodes connected to V_8, i.e. P_3 and P_7, are now connected to V_4 and disconnected from V_6, due to their double connection. V_4 and V_6 become "reference" variables of V_8. The remaining graph after we have processed this node is shown in Fig. 2b, where we also show the relation between V_4, V_6 and V_8.
2) It processes P_3, a degree-two check node created after processing P_8. Check P_3, along with V_2, are removed from the graph. The resulting graph is shown in Fig. 2c.
3) It continues with P_7 (another check node of degree 2), and deletes it from the graph with V_5, leading to Fig. 2d.
4) In Fig. 2d we have a degree-one check node, P_6, and when we run the PD decoder, we can decode V_7, V_1, V_3, V_4 and V_6, in this order. Once V_4 is decoded, we reveal V_5 and V_2, and once V_6 is decoded, we also reveal V_8.

In Fig. 3 we depict the tree graph of relations created after three iterations of the GTEP decoder for a different example. We have a received word with three check nodes to decode six variables and the decoder fails. The tree graph of relations has 3 parent variables. All the ML combinations can be obtained by setting the parent nodes independently to either 1 or 0. Hence, the multiplicity of the solution is 8.

V. ML DECODING FOR LDPC CODES OVER BEC

The BP, TEP and GTEP decoders for BEC can be reinterpreted as a Gaussian elimination procedure in which, in each iteration, a column of the parity check matrix is left with single nonzero entry. By extension of graph definitions, we say that a row is connected to a column if it has a one entry at the corresponding column. BP can be seen as a Gaussian elimination in which we only process columns that have at least one connected row of degree one, i.e., a row with a single nonzero entry. The TEP accounts also for rows of degree two and the GTEP is able to process any column, no matter the degree of the connected rows.

Theorem 1 (ML decoder): The GTEP algorithm for the BEC achieves the ML solution for LDPC decoding.

Proof: Let us denote by $\mathbf{H}_\epsilon^0$ the residual parity-check matrix once the non-erased bits (columns of the matrix) have been removed. The GTEP decoder performs linear transformations over matrix $\mathbf{H}_\epsilon^0$. By extension, $\mathbf{H}_\epsilon^\ell$ is the matrix after ℓ iterations.

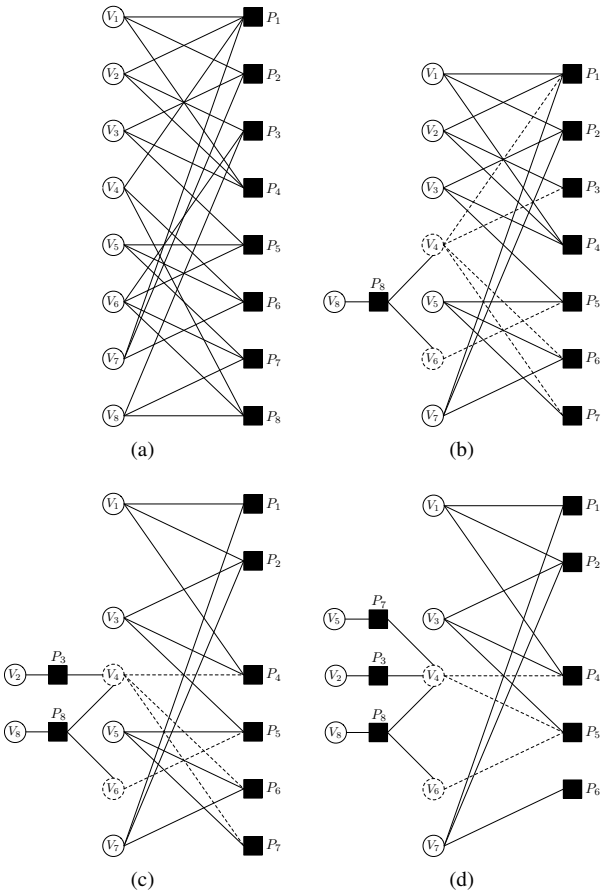

Fig. 2: We show the different iterations of the GTEP decoder for an illustrative example. On the righthand side of each subfigure we show the remaining graph and on the lefthand side we show the relations of the removed variables with the remaining variables in the graph. The dashed lines denote "reference" variables and the added edges in each iteration.

In each step, a row of the matrix is selected (i.e. a parity check) and one of its columns is processed as described in *Step 2* of the algorithm. Assume we have chosen the q-th row and the i-th column. *Step 2* is equivalent to add the q-th row to all the rows connected to the ith column. Note that this column ends with a single nonzero entry at its q-th position and thus we can conclude that the GTEP decoder performs Gaussian elimination. If the process successfully decodes, the residual matrix is of the form (after properly rearranging its columns and rows):

$$\mathbf{H}_{\epsilon}^{\infty} = \begin{bmatrix} \mathbf{I} \\ \mathbf{0} \end{bmatrix}, \tag{10}$$

then the solution is unique and each remaining variable (columns of the matrix) takes the values of the parity check it is connected to. This unique solution is the ML solution, because in the BEC we never commit an error [9]. If the GTEP does not decode, the residual matrix can be arranged to be of the form:

$$\mathbf{H}_{\epsilon}^{\infty} = \begin{bmatrix} \mathbf{I} & \mathbf{h}_{\epsilon} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}. \tag{11}$$

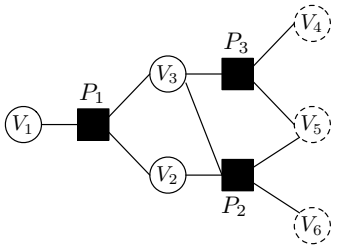

Fig. 3: Tree graph of relations

In (11) the ML solution has a multiplicity given by the number d of columns of $\mathbf{h}_{\epsilon}$, as any setting of the variables indicated by those columns fulfills the parity-checks. ■

VI. GTEP COMPLEXITY

The BP decoder exhibits a good performance with a computational cost linear with the number of variables. The BP diagonalizes the matrix by using just degree-one rows, taking the non-zero entry as pivot to perform Gaussian elimination. Hence, if the matrix $\mathbf{H}_{\epsilon}^{0}$ can be triangulated by rearranging its columns and rows, the BP successfully decodes.

As detailed in Section IV, every iteration the GTEP removes one variable, V_i, and a check node, P_q. Variable V_i is described as a function of the other variables connected to the parity check node, namely $\mathbf{V}_{p_i}$. As discussed in the former section, this step is done as an addition of rows, which causes matrix $\mathbf{H}_{\epsilon}^{\ell}$ to become dense. The complexity of adding a dense column to the others is of order quadratic with n, and the whole complexity becomes cubic.

To avoid unnecessary densification, we could process the matrix imposing that just a subset of the columns becomes dense, to easily diagonalize the remaining ones. The natural way for the GTEP to follow this strategy is to combine two simple ideas:

1) Find and process the row of $\mathbf{H}_{\epsilon}^{\ell}$ with the lowest non-zero degree and select the variable connected to it that present the highest degree. The other variables are labeled as reference variables.
2) Variables that are already reference should not be taken into account in the evaluation of the degree of the rows in $\mathbf{H}_{\epsilon}^{\ell}$, nor processed. These variables are already in the "dense" part of $\mathbf{H}_{\epsilon}^{\ell}$.

Therefore, once the row is selected according to these rules, we process one non-reference variable connected to the corresponding check node. The rest of connected variables are labeled as reference variables, if they were not. For instance, in Fig. 2.b V_4 and V_6 belong to the reference set and the dashed lines should not be counted to evaluate the degree of the check nodes. This figure has been rearranged in Fig. 4, in which we have placed the reference variables (dashed lines) to the right of the factors, and the non-reference variables to the left. The GTEP continues processing the check node with the lowest number of edges to the left. In the example, we could process either P_3 and P_7. Since their degree is one, no new reference variables are created, see Fig. 2.c. Note that if

the BP can successfully decode, the GTEP always chooses a degree-one check node and no reference variable is created.

When only reference variables are left, the GTEP continues applying Gaussian elimination to them, but now the columns are dense. Let us define α_{GTEP} such that $\alpha_{GTEP}n$ is the final number of reference variables. The complexity of the GTEP can be roughly approximated by $\mathcal{O}((\alpha_{GTEP}n)^3)$, i.e. the computational complexity of the Gaussian elimination of these variables.

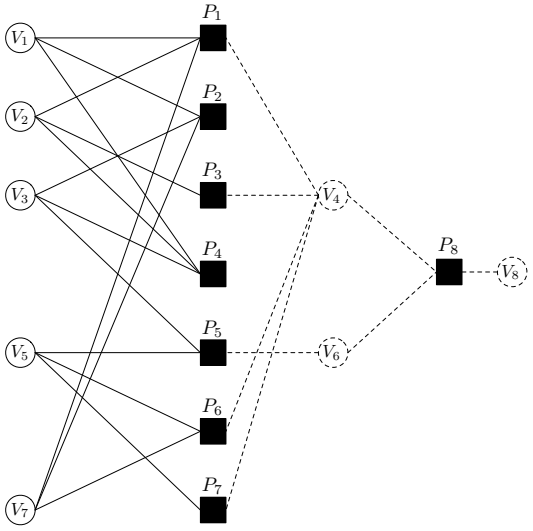

Fig. 4: Fig. 2.(b) rearranged, with the reference variables to the right and the other ones to the left. The GTEP processes the check node with the lowest number of edges to the left.

A. Complexity comparison

There have been several proposals in the literature to obtain efficient ML decoders for LDPC decoding over BEC [9], [27], [33], [40]. In this section we describe and compare them to the GTEP. After detailing the three algorithms A, B and C in [33], the other proposals are related to them. The GTEP decoder works similarly to procedure C in [33], with the great advantage of having a generalizable description for discrete channel based on the EP framework [34], as described in Section II. Besides, our method also recovers the multiplicity of the solution and provides a linear procedure to find all possible solutions through the tree graph of relations.

In [33] the authors include a classification of the procedures to perform ML decoding and a description of their complexities. In their work, the reference variables are a fraction α of columns declared as revealed in order to triangulate the remaining parity-check matrix, denoted as $\tilde{\mathbf{H}}$, to later solve the reference variables by Gaussian elimination. The computational cost of the triangularization of $\tilde{\mathbf{H}}$ is negligible compared to the $\mathcal{O}((\alpha n)^3)$ complexity of the Gaussian elimination of the reference variables, as these form a set of dense columns. Three procedures are described to choose the reference variables:

1) Procedure A: the most naive approach, where we randomly pick enough number of reference variables such that we save the gap between the erasure probability and the BP threshold for the given code, i.e. $\alpha_A \geqslant \epsilon - \epsilon_{BP}$.
2) Procedure B: we start the diagonalization process with $\tilde{\mathbf{H}} = \mathbf{H}^0_\epsilon$, and when it gets stuck, we declare each variable in $\tilde{\mathbf{H}}$ as reference variable with uniform probability δ. The authors investigate the limit $n \to \infty$ with $\delta \to 0^+$ to show that this procedure leads to $\alpha_B \leqslant \alpha_A$.
3) Procedure C: when the diagonalization gets stuck, we randomly select a row with degree $l \geq 2$ with probability $\delta\omega_l$, where δ is a constant. For each selected check node, $l-1$ of its neighbors are declared as reference variables and extracted from matrix $\tilde{\mathbf{H}}$. Through ω_l we can ensure that the lowest-degree rows are picked first, which leads to $\alpha_C \leqslant \alpha_B$.

In [33] expressions for α_A, α_B are α_C are only given in the limit $n \to \infty$. In this regime, Procedure C with some parameter setting is more efficient than A and B.

The Maxwell decoder, proposed in [9], proceeds as follows. When the BP gets stuck, it randomly selects one unknown variable and declares it as known, running again the BP algorithm. Hence, it can be equated to Procedure B.

Inactivation decoding (ID) was proposed in [26], [27] to perform ML decoding of Fountain and Raptor codes, extensively used in erasure data packet networks. Unlike standard LDPC codes, the rate in fountain codes is not fixed in advance. This flexibilility in the code rate can be exploited to attain excellent decoding properties. For instance, Raptors codes are designed to be decoded with up to one reference variable per check node, that is denoted as *inactivated variable*. The authors also state in [27] that, if more reference variables would be needed, the ID inactivates all except one of the variables connected to the check node of minimum degree. Thus, ID can be cast as Procedure C.

In [40] the authors present an ML algorithm where they proceed as Procedure B in [33] except for two main differences. First, they select as reference variables the ones with largest degrees. In the case of regular codes this poses no improvement, as the remaining variables out of the reference set has the same number of edges. For irregular codes, however, it should be taken into account, as the GTEP does. Second, they analyze the neighborhoods of the possible reference variables, to choose the best one. However, this search is focused on scenarios with a few number of reference variables, because they focus on improving the BP solution in the error floor region. No computational cost to find the optimal set is evaluated for the general case. Besides, given the best set of reference variables, the computational complexity is described in [40] as $\mathcal{O}(g^2_{\max}n)$, where $g_{\max}$ is related to the number of reference variables. They show that $g_{\max}$ is linear with the number of reference variables, and the complexity is of cubic order, as that of the procedures in [33].

From the discussion above, we can conclude that the GTEP algorithm shares the philosophy of Procedure C in [33]. However, while in Procedure C the row to process is randomly chosen through the sequence of probabilities $\delta\omega_l$ for $l \geq 2$, in GTEP we just process the one with lowest degree, resulting in a better suited implementation in the finite-length scenario. It is worthy to remark that, in the experimental section in [33],

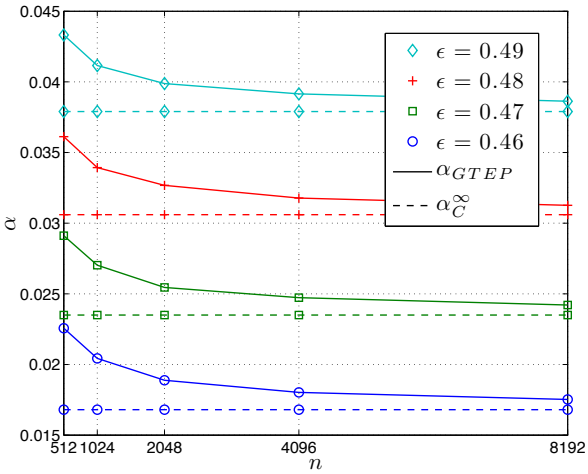

Fig. 5: Fraction of variables in the GTEP decoding that are reference ones, α_{GTEP}, for different values of ϵ. The limiting values for Procedure C in [33], α_C^∞, are also depicted.

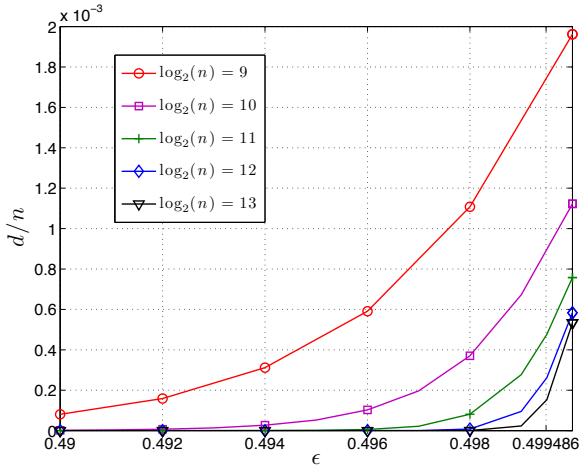

Fig. 6: We plot the average number of remaining parent variables, normalized by the number of variables, n, as a function of the erasure value of the channel, for the (5,10)-regular LDPC ensemble and different lengths of the codeword.

the authors set $\omega_2 = 1$ and negligible values to ω_l for $l \geq 3$, emulating the GTEP algorithm.

In Fig. 5 we have plotted the empirical mean value of α_{GTEP} for different code lengths of a (3,6)-regular LDPC ensemble and different erasure values. The horizontal lines indicate the α_C values provided in [33] for $n \to \infty$. We denote them by α_C^∞. In Fig. 5, we can observe that the value of α_{GTEP} decreases to α_C^∞ with the code size. Besides, for short code sizes $\alpha_{GTEP} n$ is fairly low, which results in a complexity of the Gaussian elimination of the reference variables lower or similar to the other (quadratic) operations of the decoding process. Finally, compared to the algorithm in [40], the two-steps search of the best reference variables could be addressed, however we believe its complexity would be its major drawback. We conclude that the GTEP naturally yields a ML decoder with equal or better complexity than the methods previously described.

VII. Analysis of the GTEP decoder for finite-length codes

In this section we focus on the (5,10)-regular LDPC ensemble, as it is a good example of LDPC code whose ML threshold is close to capacity, $\epsilon_{ML} = 0.499486$ [23], and far from the BP threshold ($\epsilon_{BP} \approx 0.34$).

A. Multiplicity of the ML solution

As stated in the previous sections, the codeword is decoded if all the parent variables of the tree graph of relations are revealed. On the contrary, the GTEP decoder provides a multiple ML solution whose multiplicity depends on the number of non-revealed parent nodes.

For the (5,10)-regular LDPC ensemble, we depict in Fig. 6 the curves for the normalized number of non-revealed parent variables, d/n, as a function of the erasure value of the channel. As the length of the codeword increases, the number of non-revealed parent variables for ϵ values below $\epsilon = 0.499486$ tends to 0, as the performance gets close to the asymptotic behavior.

B. Error rates

Finally, in Fig. 7, we depict the word error rate (WER) curves for the BP, TEP and GTEP decoders and different lengths of the (5,10)-regular LDPC code averaged over different realizations of the ensemble. These curves graphically illustrate how the proposed GTEP decoder achieves the ML solution. As n increases, the results tend to the asymptotic performance of the decoder. Thus, the waterfall of the WER curves takes place around a value of $\epsilon = 0.499486$, the ML threshold. For this ensemble, it has been shown that the TEP

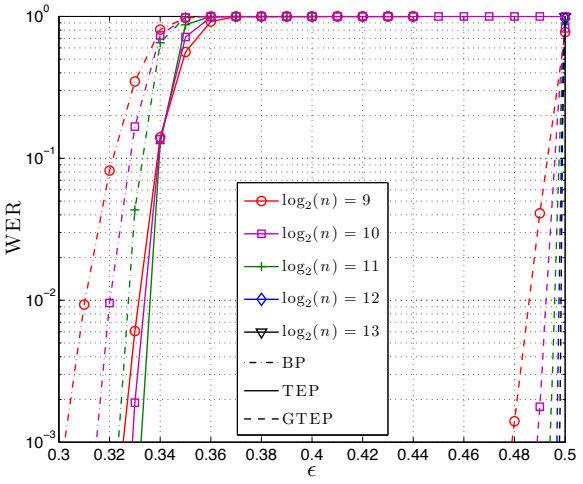

Fig. 7: WER performance for the BP (dash-dotted lines), the TEP (solid lines) and the GTEP (dashed lines) algorithms for the (5,10)-regular LDPC ensemble, and different lengths of the codeword.

threshold hardly improves the BP threshold, as it can be seen in the figure. Besides, we can observe the remarkable gain of the GTEP compared to the BP and the TEP decoders, where the performance of both algorithms significantly diminishes due to the low-girth cycles in these LDPC codes.

VIII. Conclusions

The linear complexity of the BP algorithm has made it the standard approach for LDPC decoding. However, the BP solution for regular LDPC ensembles does not achieve the channel capacity. Hence, in order to reach the capacity under BP decoding we are restricted to families of irregular sequences. However, these codes present more complex degree distributions and require much longer codewords than regular ensembles to perform optimally.

The TEP was presented as a tool to improve the BP decoding. The TEP is based on the tree-structured EP algorithm and it just focuses on relations between two variables. The TEP exhibits a better performance than the BP for finite length codes over the BEC, with no increase of the complexity. In this paper, we explore the application of tree-structured EP to the decoding of LDPC codes with no limit in the number of processed variables. For the BEC, we have shown that the resulting decoder, the GTEP, accomplishes the ML solution by fully processing the Tanner graph. Even if the GTEP decoder does not yield a unique codeword, it provides the set of possible ML solutions, and its multiplicity. Besides, the GTEP algorithm has a better or similar computational complexity compared to that of the ML decoders in the literature, such as the Maxwell decoder [9], the inactivation decoding [27] or the procedures by Burshtein *et al.* [33].

References

[1] L. Salamanca, P. Olmos, J. J. Murillo-Fuentes, and F. Pérez-Cruz, "MAP decoding for LDPC codes over the binary erasure channel," in *IEEE Information Theory Workshop (ITW)*, Oct. 2011, pp. 145 –149.

[2] R. G. Gallager, *Low Density Parity Check Codes*. MIT Press, 1963.

[3] D. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Transactions on Information Theory*, vol. 45, no. 2, pp. 399 –431, Mar. 1999.

[4] D. MacKay and R. Neal, "Near Shannon limit performance of low-density parity-check codes," *Electronics Letters*, vol. 32, no. 18, p. 1645, Aug. 1996.

[5] J. H. Kim and J. Pearl, "A computational model for causal and diagnostic reasoning in inference systems," 1983. [Online]. Available: http://hdl.handle.net/10203/17387

[6] S.-Y. Chung, J. Forney, G.D., T. Richardson, and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit," *IEEE Communications Letters*, vol. 5, no. 2, pp. 58 –60, Feb. 2001.

[7] T. Richardson, A. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 619 –637, Feb. 2001.

[8] T. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 599 –618, Feb. 2001.

[9] C. Measson, A. Montanari, and R. Urbanke, "Maxwell construction: The hidden bridge between iterative and maximum a posteriori decoding," *IEEE Transactions on Information Theory*, vol. 54, no. 12, pp. 5277 –5307, Dec. 2008.

[10] F. Kschischang, B. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 498 –519, Feb. 2001.

[11] N. Wiberg, "Codes and decoding on general graphs," Ph.D. dissertation, Department of Electrical Engineering Linköping University, 1996.

[12] S. Aji and R. McEliece, "The generalized distributive law," *IEEE Transactions on Information Theory*, vol. 46, no. 2, pp. 325 –343, Mar. 2000.

[13] S. Ten Brink, "Convergence behavior of iteratively decoded parallel concatenated codes," *IEEE Transactions on Communications*, vol. 49, no. 10, pp. 1727 –1737, Oct. 2001.

[14] A. Ashikhmin, G. Kramer, and S. ten Brink, "Extrinsic information transfer functions: model and erasure channel properties," *IEEE Transactions on Information Theory*, vol. 50, no. 11, pp. 2657 – 2673, Nov. 2004.

[15] M. Luby, M. Mitzenmacher, M. Shokrollahi, and D. Spielman, "Efficient erasure correcting codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 569 –584, Feb. 2001.

[16] A. Sanaei, M. Ramezani, and M. Ardakani, "Identical-capacity channel decomposition for design of universal LDPC codes," *IEEE Transactions on Communications on Communications*, vol. 57, no. 7, pp. 1972 – 1981, July 2009.

[17] S. Laendner and O. Milenkovic, "LDPC codes based on latin squares: Cycle structure, stopping set, and trapping set analysis," *IEEE Transactions on Communications on Communications*, vol. 55, no. 2, pp. 303 – 312, Feb. 2007.

[18] H. Saeedi and A. Banihashemi, "On the design of LDPC code ensembles for BIAWGN channels," *IEEE Transactions on Communications on Communications*, vol. 58, no. 5, pp. 1376 – 1382, May 2010.

[19] P. Oswald and A. Shokrollahi, "Capacity-achieving sequences for the erasure channel," *IEEE Transactions on Information Theory*, vol. 48, no. 12, pp. 3017 – 3028, Dec. 2002.

[20] T. J. Richardson and R. Urbanke, *Modern Coding Theory*. Cambridge University Press, Mar. 2008.

[21] Y. Han and W. Ryan, "Low-floor decoders for LDPC codes," *IEEE Transactions on Communications on Communications*, vol. 57, no. 6, pp. 1663 – 1673, June 2009.

[22] D. Mitchell, M. Lentmaier, and D. Costello, "AWGN channel analysis of terminated LDPC convolutional codes," in *Information Theory and Applications Workshop (ITA)*, Feb. 2011, pp. 1 –5.

[23] S. Kudekar, T. Richardson, and R. Urbanke, "Threshold saturation via spatial coupling: Why convolutional LDPC ensembles perform so well over the BEC," in *IEEE International Symposium on Information Theory Proceedings (ISIT)*, June 2010, pp. 684 –688.

[24] P. M. Olmos and R. Urbanke, "Scaling behavior of Convolutional LDPC ensembles over the BEC," in *IEEE International Symposium on Information Theory Proceedings (ISIT)*, Aug. 2011, pp. 1816 –1820.

[25] D. J. C. MacKay, *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003. [Online]. Available: http://www.cambridge.org/0521642981

[26] A. Shokrollahi, S. Lassen, and R. Karp, "Systems and processes for decoding chain reaction codes through inactivation," Digital Fountain Inc, Tech. Rep. U.S. Patent 6 856 263, Feb. 2005.

[27] A. Shokrollahi and M. Luby, *Raptor Codes*, ser. Foundations and trends in communications and information theory. Now Publishers Inc, 2011, vol. 6, no. 3-4.

[28] T. Minka, "Expectation propagation for approximate bayesian inference," in *UAI*, 2001, pp. 362–369.

[29] T. Minka and Y. Qi, "Tree-structured approximations by expectation propagation," in *Proceedings of the Neural Information Processing Systems Conference, (NIPS)*, 2003.

[30] Z. Ghahramani and M. I. Jordan, "Factorial hidden Markov models," *Machine Learning*, vol. 29, pp. 245–273, Nov. 1997.

[31] P. M. Olmos, J. J. Murillo-Fuentes, and F. Pérez-Cruz, "Tree-structure expectation propagation for decoding LDPC codes over binary erasure channels," in *IEEE International Symposium on Information Theory Proceedings (ISIT)*, June 2010, pp. 799 –803.

[32] ——, "Tree-structure expectation propagation for LDPC tree-structure expectation propagation for LDPC decoding over the BEC," *Submitted to IEEE Transactions on Information Theory*, 2012. [Online]. Available: http://arxiv.org/abs/1009.4287

[33] D. Burshtein and G. Miller, "Efficient maximum-likelihood decoding of LDPC codes over the binary erasure channel," *IEEE Transactions on Information Theory*, vol. 50, no. 11, pp. 2837 – 2844, Nov. 2004.

[34] L. Salamanca, P. M. Olmos, J. J. Murillo-Fuentes, and F. Pérez-Cruz, "Tree-structured expectation propagation for LDPC decoding in AWGN channels," in *Information Theory and Applications Workshop (ITA)*, Feb. 2012.

[35] M. J. Wainwright and M. I. Jordan, *Graphical Models, Exponential Families, and Variational Inference*. Foundations and Trends in Machine Learning, 2008.

[36] A. Becker, R. Bar-Yehuda, and D. Geiger, "Randomized algorithms for the loop cutset problem," *Journal of Artificial Intelligence Research*, vol. 12, no. 1, pp. 219–234, 2000.

[37] R. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on Information Theory*, vol. 27, no. 5, pp. 533 – 547, Sept. 1981.

[38] H.-A. Loeliger, "An introduction to factor graphs," *IEEE Signal Processing Magazine*, vol. 21, no. 1, pp. 28 – 41, Jan. 2004.
[39] P. M. Olmos, J. J. Murillo-Fuentes, and F. Pérez-Cruz, "Tree-structured expectation propagation for decoding finite-length LDPC codes," *IEEE Communications Letters*, vol. 15, no. 2, pp. 235 –237, Feb. 2011.
[40] H. Pishro-Nik and F. Fekri, "On decoding of low-density parity-check codes over the binary erasure channel," *IEEE Transactions on Information Theory*, vol. 50, no. 3, pp. 439 – 454, Mar. 2004.