Transactions Letters

On the UEP Capabilities of Several LDPC Construction Algorithms

Neele von Deetzen and Sara Sandberg

***Abstract*—This paper analyzes construction algorithms for low-density parity-check (LDPC) codes with respect to their unequal error protection (UEP) capabilities. We show that the choice of code construction algorithm highly affects the performance and UEP properties of LDPC codes with identical degree distributions. Our results provide an explanation to disagreements in earlier research.**

***Index Terms*—LDPC, unequal error protection, parity-check matrix, ACE, PEG.**

I. Introduction

UNEQUAL error protection (UEP) low-density parity-check (LDPC) codes that provide more protection for certain bits within the codeword are important for applications where the source bits have different sensitivities to errors. The desired UEP properties are a low bit-error rate (BER) or frame-error rate (FER) within one or several classes of bits, while the performance of the remaining classes should be comparable to non-UEP codes. Such codes can, for example, be constructed by an algebraic method based on Plotkin-type constructions [1]. However, since it is widely observed that the connection degree of a variable node affects the BER of its corresponding code bit, at least for a limited number of decoding iterations, it is typical to design the variable and/or check node degree distribution of the code in an irregular way using density evolution [2]–[4]. It should be noted though, that the results of papers on irregular UEP-LDPC codes disagree. For example, [4] shows significant UEP capabilities after 200 message-passing iterations, while [1] argues that no UEP gradation can be detected for irregular UEP-LDPC codes after 50 iterations. In this paper we explain the reasons behind the disagreeing results by analyzing different construction algorithms with respect to how the graph properties of the corresponding codes affect the UEP capabilities.

This paper focuses on LDPC codes, originally presented by Gallager in [5]. They exhibit a performance very close to the capacity for the binary symmetric memoryless channel [6]. LDPC codes are block codes with a sparse parity-check matrix H of dimension $(n-k)\times n$, where k and n are the lengths of the information word and the codeword, respectively, and $R = k/n$ denotes the code rate. An LDPC code can be represented by a bipartite graph, called Tanner graph [7], which facilitates the description of a decoding algorithm known as the message-passing algorithm [8]. The graph consists of two types of nodes, variable nodes and check nodes, which correspond to the bits of the codeword and to the parity-check constraints, respectively. A variable node is connected to a check node if the bit is included in the parity-check constraint. The number of a node's connections to other nodes is called the degree. We consider irregular LDPC codes with variable node and check node degree distributions defined by the polynomials [6] $\lambda(x) = \sum_{i=2}^{d_{v_{max}}} \lambda_i x^{i-1}$ and $\rho(x) = \sum_{i=2}^{d_{c_{max}}} \rho_i x^{i-1}$, where $d_{v_{max}}$ and $d_{c_{max}}$ are the maximum variable and check node degree of the code. The coefficients of the degree distributions describe the proportion of edges connected to nodes with a certain degree. UEP is usually obtained by assigning important bits to high-degree variable nodes and less important bits to the variable nodes with lower degrees. Good degree distributions are commonly computed by means of density evolution using a Gaussian approximation [9]. For a given codeword length and given degree distribution, the ensemble of codes is defined by random permutation of the edges in the graph. One instance of the ensemble (a specific code) is identified by a particular permutation. If the permutations are chosen randomly, all codes in an ensemble are equiprobable. However, to ensure good performance of the code, some instances are not allowed.

Once a degree distribution is obtained, a parity-check matrix H has to be constructed according to the degree distribution. Many construction algorithms have been developed and we consider five different algorithms for the construction of the parity-check matrix. All of the obtained codes belong to the same code ensemble. *Random construction*, following the approach of [6], was typically used a few years ago. We consider a random construction where only length-4 cycles between degree-2 variable nodes are avoided. However, several authors have suggested construction algorithms with better BER performance than the random construction, especially in the error-floor region, mainly by avoiding small cycles in the Tanner graph. The *progressive edge-growth (PEG) construction* algorithm is an efficient algorithm for the construction of parity-check matrices with large girth (the length of the shortest cycle in the tanner graph) by progressively connecting variable nodes and check nodes [10]. The *zigzag construction* algorithm connects the edges of degree-two variable nodes in a zigzag manner, according to [11]. The remaining edges may be connected in the same way as described for the random algorithm (zigzag-random) or according to the PEG algorithm (zigzag-PEG). The *approximate cycle extrinsic message degree (ACE) construction* algorithm lowers the error floor by emphasizing both the number of edges from variable nodes in

Paper approved by A. K. Khandani, the Editor for Coding and Information Theory of the IEEE Communications Society. Manuscript received November 19, 2008; revised July 8, 2009 and November 23, 2009.

N. von Deetzen was with the School of Engineering and Science, Jacobs University Bremen, Germany. She is now with Silver Atena Electronic Systems Engineering GmbH (e-mail: n.vondeetzen@silver-atena.de).

S. Sandberg is with the Department of Computer Science and Electrical Engineering, Luleå University of Technology, Sweden (e-mail: sara.sandberg@ltu.se).

Digital Object Identifier 10.1109/TCOMM.2010.101210.080614

a cycle to nodes in the graph that are not part of the cycle as well as the length of cycles [12]. The *PEG-ACE construction* algorithm is a generalization of the popular PEG algorithm, that is shown to generate good LDPC codes with short and moderate block lengths having large girth [13]. If the creation of cycles cannot be avoided while adding an edge, the PEG-ACE construction algorithm chooses an edge that creates the longest possible cycle with the best possible ACE constraint. In this paper we confirm by simulation that the design of an irregular variable node degree distribution provides UEP capability for a low number of message-passing iterations regardless of the construction algorithm used. However, the results also show that the choice of the construction algorithm is critical when good UEP properties are desired after a moderate or high number of iterations. UEP capability after many iterations is important since this enables considerably lower error rates than a low number of iterations. To ensure UEP capability of the code regardless of the choice of construction algorithm, the decoder must typically be interrupted after only 10 iterations, which results in performance losses.

II. Simulation Results

A. Ensemble Design

We consider the UEP-LDPC ensemble design proposed in [3], which is based on a hierarchical optimization of the variable node degree distribution for each protection class. The algorithm maximizes the average variable node degree within one class at a time while guaranteeing a minimum variable node degree as high as possible. The optimization can be stated as a linear programming problem and can, thus, be easily solved. To keep the overall performance of the UEP-LDPC code reasonably good, the search for UEP codes is limited to degree distributions whose convergence thresholds lie within a certain range ϵ of the minimum threshold of a code with the same parameters. We fix ϵ to 0.1 dB, which is shown in [3] to give a good trade-off between the performances of the protection classes.

The UEP-LDPC ensemble design algorithm is initialized with a maximum variable node degree $d_{v_{max}}$, the code rate R, and a check node degree distribution. The bits of the codeword are divided into N_c protection classes C^j according to their protection requirements where class C^1 contains the most protected bits and the last protection class contains all parity bits. We design a rate-1/2 UEP-LDPC code with $N_c = 3$ protection classes, $d_{v_{max}} = 30$ and $\rho(x) = 0.00749x^7 + 0.99101x^8 + 0.00150x^9$, which is found by numerical optimization in [6] to be a good check node degree distribution for $d_{v_{max}} = 30$. The proportions of the classes are chosen such that C^1 contains 20% of the information bits and C^2 contains 80%. Protection class C^3 contains all parity bits. Therefore, we are mainly interested in the performances of classes C^1 and C^2. The resulting variable node degree distribution is defined by the coefficients $\lambda_i^{(C^j)}$ which denote the fractions of edges incident to degree-i variable nodes of protection class C^j. The overall degree distribution is therewith given by $\lambda(x) = \sum_{j=1}^{N_c} \sum_{i=2}^{d_{v_{max}}} \lambda_i^{(C^j)} x^{i-1}$. Table I summarizes the optimized variable node degree distribution for the resulting UEP-LDPC code.

TABLE I
VARIABLE NODE DEGREE DISTRIBUTION OF THE UEP-LDPC ENSEMBLE.

C_1	C_2	C_3
$\lambda_{18}^{(C^1)} = 0.2521$	$\lambda_3^{(C^2)} = 0.0786$	$\lambda_2^{(C^3)} = 0.2130$
$\lambda_{19}^{(C^1)} = 0.0965$	$\lambda_4^{(C^2)} = 0.2511$	$\lambda_3^{(C^3)} = 0.0141$
$\lambda_{30}^{(C^1)} = 0.0946$		

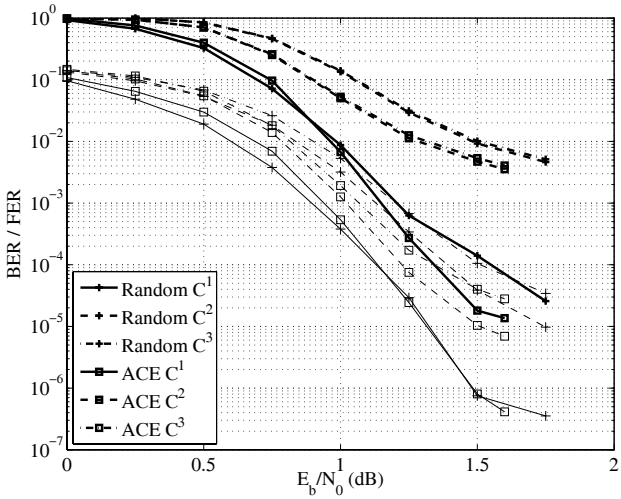

Fig. 1. FER and BER of the random code and the ACE code as a function of E_b/N_0, after 100 iterations. The bold curves show FER and the thin curves show BER. Both codes show good UEP capabilities, but the ACE code performs slightly better than the random code.

B. Performance Comparison

UEP-LDPC codes with length $n = 4096$ are constructed using the different construction algorithms. All codes belong to the ensemble described above. From each construction algorithm, we consider one code realization in the following. It should be noted that the differences in performance between several code realizations constructed with the same construction algorithm are small. We present simulation results for BPSK transmission over the AWGN channel. Fig. 1 shows the FER and the BER as a function of E_b/N_0 for the random and the ACE code after 100 decoder iterations. They both show good UEP properties, but the ACE code performs slightly better than the random code. We also see that the ACE code has a lower error-floor than the random code. Fig. 2 shows the FER and the BER for the zigzag-random and PEG-ACE code after 100 decoder iterations. The zigzag-random code shows moderate UEP capabilities, while the PEG-ACE code does not show any UEP at all in FER and very little in BER. Simulation results for the PEG code and the zigzag-PEG code are omitted here since they show almost exactly the same performance and UEP capabilities as the PEG-ACE code. For simplicity, we will summarize the construction algorithms into the following two groups: non-UEP algorithms and UEP-capable algorithms. The non-UEP construction algorithms are the PEG, the zigzag-PEG, and the PEG-ACE construction. The UEP-capable construction algorithms are the random, the ACE, and the zigzag-random construction.

For standard code design, i.e. without UEP, the PEG-ACE construction has been shown to lower the error-floor while the loss in the waterfall-region is minimal [13]. The results in Fig. 2 show the same behavior. Remarkably, the PEG-ACE

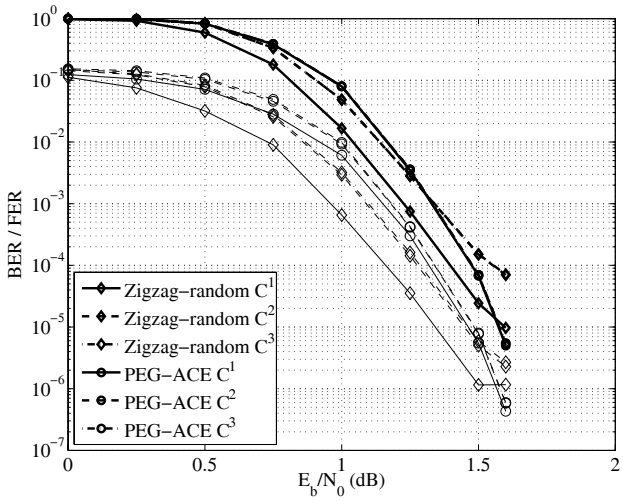

Fig. 2. FER and BER of the zigzag-random code and the PEG-ACE code as a function of E_b/N_0, after 100 iterations. The bold curves show FER and the thin curves show BER. The zigzag-random code shows moderate UEP capabilities, while the PEG-ACE code does not show any UEP at all in FER and very little in BER.

code shows almost no difference in performance between the classes. The PEG-ACE construction does not lower the error-floors of all classes compared to the random construction as may be expected, but it removes the UEP capability by improving C^2 and C^3 while degrading C^1. Also, the loss in the waterfall-region is slightly higher than shown for the standard code design, while the gain in the error-floor region is substantial since all classes have low error floors.

The results presented in this section suggest the use of the PEG-ACE code for high E_b/N_0. At $E_b/N_0 = 1.6$ dB, all classes of the PEG-ACE code have the same performance as the best class of the ACE code. However, for low E_b/N_0, the PEG-ACE code performs badly and the ACE code with UEP capability is a better choice.

III. Relevant Graph Properties

In this section, we present properties of the Tanner graph which are relevant for the UEP behavior of the code. These properties concern the amount of connections between variable nodes of different protection classes.

A. Connections Between Protection Classes

Since the degree distributions $\lambda(x)$ and $\rho(x)$ are equal for all codes, we investigate how the incident variable nodes of a check node are spread between the classes. Generally, a check node degree distribution may be defined from the node's perspective as

$$\tilde{\rho}(x) = \sum_{i=2}^{d_{c_{max}}} \tilde{\rho}_i x^{i-1} \ .$$

The coefficients $\tilde{\rho}_i$ correspond to the fraction of degree-i check nodes. In order to account for connections to different protection classes, we define detailed check node degree distributions for the protection classes C^j,

$$\tilde{\rho}^{(C^j)}(x) = \sum_{i=0}^{d_{c_{max}}} \tilde{\rho}_i^{(C^j)} x^{i-1} \ , \quad j = 1 \dots N_c \ . \qquad (1)$$

The coefficients $\tilde{\rho}_i^{(C^j)}$ correspond to the fraction of check nodes with i edges connected to class-C^j variables nodes. Note that i is not the overall degree of the check nodes but only the number of edges which are connected to class-C^j variable nodes. For example, $\tilde{\rho}_4^{(C^1)}$ is the fraction of all check nodes with exactly 4 edges connected to C^1, regardless of the connections of the remaining edges to the other classes. By definition we have $\sum_{i=0}^{d_{c_{max}}} \tilde{\rho}_i^{(C^j)} = 1, \ j = 1, \dots, N_c$. This detailed check node degree distribution is similar to the detailed representation described in [14], but we consider the degree distribution from the node's perspective while [14] considers the edge's perspective. The representation in [14] is also more detailed than necessary for our purpose since it defines connections to nodes of certain degrees instead of certain protection classes. Table II presents the coefficients of the detailed check node degree distributions for the ACE, the zigzag-random, the PEG-ACE, and a modified PEG-ACE code that will be discussed in Section IV. Note that the maximum check node degree of all codes is $d_{c_{max}} = 10$.

Most of the coefficients of the ACE code are non-zero for all three protection classes, while the PEG-ACE code has only a few non-zero coefficients. That is, the PEG-ACE code only has a few different types of check nodes, and the numbers of connections to the protection classes are very similar for all nodes: The PEG-ACE coefficients $\tilde{\rho}_4^{(C^1)}$, $\tilde{\rho}_3^{(C^2)}$ and $\tilde{\rho}_2^{(C^3)}$ are all large, which shows that most check nodes have 4 edges connected to C^1, 3 edges to C^2, and 2 edges to C^3. This means that the variable nodes of a protection class are generally well connected to other protection classes through the check nodes. In contrast, the ACE code exhibits many different kinds of check nodes. Some of the check nodes are mainly connected to one protection class, having only one or two edges going to other protection classes. There even exist check nodes having 10 edges to a protection class, which means that they are solely connected to this class. This property seems to be important for the capability of providing UEP. With more non-zero coefficients the classes are more isolated and the propagation of messages between the classes will be slower. If most check nodes have several edges to all protection classes, reliable and unreliable messages from different classes may proceed to other classes more easily and affect their performance.

The detailed check node degree distributions of the zigzag-random code have few non-zero coefficients for C^3, while the distributions for C^1 and C^2 are similar to the ACE code. Many check nodes have two edges connected to C^3, while the number of edges to the other classes vary. The number of connections between the classes is therefore higher than for the ACE code, but lower than for the PEG-ACE code. This agrees with the UEP capabilities, since the zigzag-random code shows less UEP than the ACE code and more than the PEG-ACE code. The detailed distributions of the other codes are omitted here since the distribution of the random code is very similar to that of the ACE code and all non-UEP codes have almost the same detailed check node degree distributions. In order to visualize the observations, Fig. 3 shows the detailed check node degree distributions of the ACE, the zigzag-random, and the PEG-ACE code.

TABLE II
DETAILED CHECK NODE DEGREE DISTRIBUTIONS.

	ACE			Zigzag-random			PEG-ACE			Modified PEG-ACE		
	C^1	C^2	C^3	C^1	C^2	C^3	C^1	C^2	C^3	C^1	C^2	C^3
$\tilde{\rho}_0^{(C^j)}$	0.0410	0.0259	0	0.0425	0.0615	0	0	0	0	0	0.0371	0
$\tilde{\rho}_1^{(C^j)}$	0.0527	0.1216	0.4819	0.0566	0.1426	0.0420	0	0.0054	0	0.0366	0.1514	0.4829
$\tilde{\rho}_2^{(C^j)}$	0.1074	0.2358	0.2427	0.1089	0.2319	0.8784	0.0005	0.1211	0.9575	0.1753	0.2334	0.2251
$\tilde{\rho}_3^{(C^j)}$	0.1480	0.2769	0.1465	0.1475	0.2236	0.0757	0.1050	0.7783	0.0425	0.1621	0.2510	0.1465
$\tilde{\rho}_4^{(C^j)}$	0.2393	0.2129	0.0645	0.2124	0.1426	0.0034	0.7998	0.0947	0	0.2471	0.1665	0.0591
$\tilde{\rho}_5^{(C^j)}$	0.2109	0.0859	0.0352	0.2315	0.1069	0.0005	0.0942	0.0005	0	0.2002	0.0767	0.0850
$\tilde{\rho}_6^{(C^j)}$	0.1445	0.0303	0.0161	0.1426	0.0566	0	0.0005	0	0	0.1294	0.0508	0.0015
$\tilde{\rho}_7^{(C^j)}$	0.0503	0.0088	0.0054	0.0557	0.0317	0	0	0	0	0.0444	0.0269	0
$\tilde{\rho}_8^{(C^j)}$	0.0059	0.0020	0.0039	0.0024	0.0024	0	0	0	0	0.0049	0.0064	0
$\tilde{\rho}_9^{(C^j)}$	0	0	0.0029	0	0	0	0	0	0	0	0	0
$\tilde{\rho}_{10}^{(C^j)}$	0	0	0.0010	0	0	0	0	0	0	0	0	0

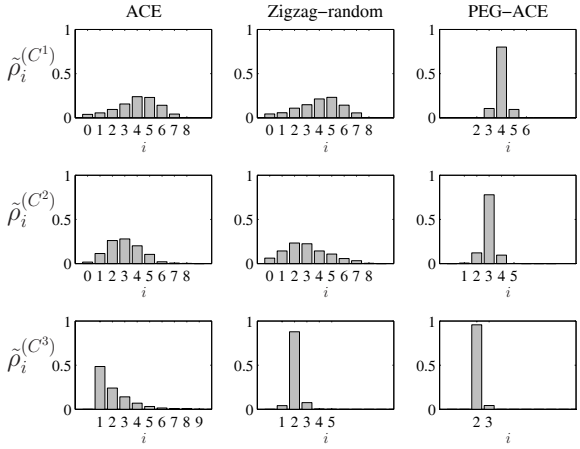

Fig. 3. Detailed check node degree distributions of the ACE, the zigzag-random, and the PEG-ACE code. The UEP-capable ACE code has many non-zero coefficients, while the non-UEP PEG-ACE code has one large coefficient in each class, which leads to more connections between the classes.

B. Detailed Mutual Information Evolution

The effect of the number of connections between the classes on the performance may be analyzed by calculating the mutual information (MI) functions of the different codes. By calculating the theoretical MI functions, the effect of the number of connections is isolated from effects due to cycles, trapping sets, and codeword length, since the calculation is based on the corresponding cycle-free graph. Typically, the MI functions are calculated from the degree distributions $\lambda(x)$ and $\rho(x)$ of a code. However, in our case all codes have the same overall degree distributions $\lambda(x)$ and $\rho(x)$. To observe the differences between codes constructed by the various algorithms, a detailed computation of MI may be performed by considering the edge-based MI messages traversing the graph instead of node-based averages. This has been done for protographs in [15]. We follow the same approach, but use the parity-check matrix instead of the protograph base matrix. See [16], [17] for more details on MI analysis.

Let I_{Av} be the *a priori* MI between one input message and the codeword bit associated to the variable node. I_{Ev} is the extrinsic MI between one output message and the codeword bit. Similarly on the check node side, we define I_{Ac} (I_{Ec}) to be the *a priori* (extrinsic) MI between one check node input (output) message and the codeword bit corresponding to the variable node providing (receiving) the message. The evolution is initialized by the MI between one received message and the corresponding codeword bit, denoted by I_{ch}, which corresponds to the channel capacity. For the AWGN channel, it is given by $I_{ch} = J(\sigma_{ch})$, where

$$\sigma_{ch}^2 = 8R\frac{E_b}{N_0} \tag{2}$$

and E_b/N_0 is the signal-to-noise ratio at which the analysis is performed. The function $J(\cdot)$ is defined by

$$J(\sigma) = 1 - \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-\sigma^2/2)^2}{2\sigma^2}} \log_2(1+e^{-y})dy \tag{3}$$

and computes the MI based on the noise variance. For a variable node with degree d_v, the extrinsic MI between the s-th output message and the corresponding codeword bit is [15]

$$I_{Ev|s} = J\left(\sqrt{\sum_{l=1,l\neq s}^{d_v} [J^{-1}(I_{Av|l})]^2 + [J^{-1}(I_{ch})]^2}\right), \tag{4}$$

where $I_{Av|l}$ is the *a priori* MI of the message received by the variable node on its l-th edge. The extrinsic MI for a check node with degree d_c may be written as

$$I_{Ec|s} = 1 - J\left(\sqrt{\sum_{l=1,l\neq s}^{d_c} [J^{-1}(1-I_{Ac|l})]^2}\right), \tag{5}$$

where $I_{Ac|l}$ is the *a priori* MI of the message received by the check node on its l-th edge. Note that the MI functions are subject to the Gaussian approximation (see [9]) and are not exact.

The *a posteriori* MI of the check nodes (denoted by I_{APPc}) at $E_b/N_0 = 0.7$ dB is shown in Fig. 4 for the ACE and the PEG-ACE code. An average I_{APPc} of each class is calculated as an average I_{APPc} of all edges incident to variable nodes of the corresponding class. The figure shows that the average I_{APPc} of all classes are almost equal for the PEG-ACE code, while they differ for the ACE code. This behavior may

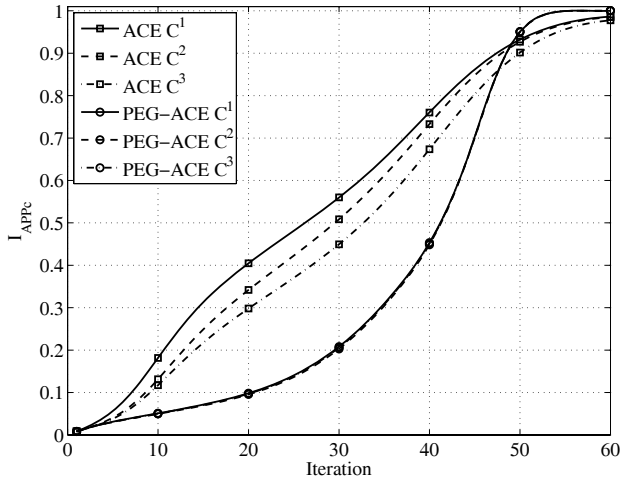

Fig. 4. Average check node *a posteriori* MI (I_{APPc}) as a function of the number of decoder iterations at $E_b/N_0 = 0.7$ dB. The average I_{APPc} of all classes are almost equal for the PEG-ACE code, while they differ for the ACE code.

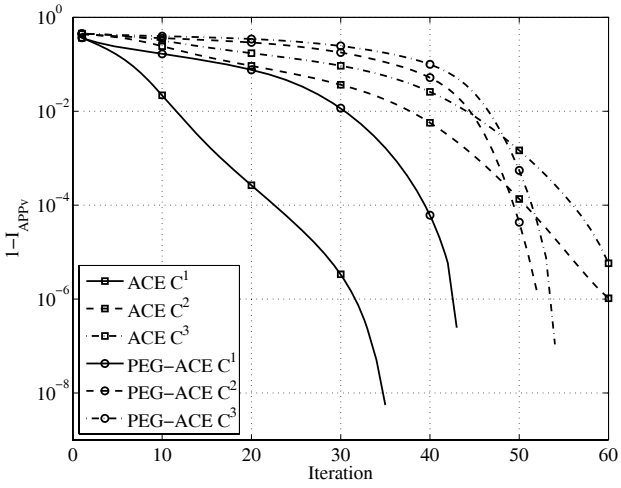

Fig. 5. Distance of average variable node *a posteriori* MI (I_{APPv}) to the maximum MI, as a function of the number of decoder iterations, at $E_b/N_0 = 0.7$ dB. Protection class C^1 of the ACE code converges faster than that of the PEG-ACE code, while the other classes take more iterations to converge for the ACE code than for the PEG-ACE code.

be explained by the connections of the check nodes to the different protection classes described by the detailed check node degree distributions in Table II. For the PEG-ACE code, almost all check nodes are connected to 4 variable nodes from C^1, 3 variable nodes from C^2 and 2 variable nodes from C^3. Even if the extrinsic information from the variable nodes ($I_{Ev} = I_{Ac}$) differs (due to the irregular variable node degree distribution), almost all check nodes combine the same number of nodes from each class. This gives almost equal I_{Ec} for all check nodes, which means that the performance of different classes will be averaged over the whole codeword in this step. For the ACE code on the other hand, the number of variable nodes from different classes that are connected to a check node differs much more. This allows for some check nodes with higher MI than others and the difference in MI of the check nodes will increase the UEP capability of the code.

Fig. 5 shows how close the average variable node *a posteriori* MI (denoted by I_{APPv}) is to its maximum, as a function of the number of decoder iterations. It is shown in [16] that small differences in MI may lead to significant differences in BER for MI values near 1. Therefore, the figure shows the distance of the MI to its maximum value, i.e. $1 - I_{APPv}$, on a logarithmic scale. Note that the convergence speeds of the protection classes are farther apart for the ACE algorithm. Protection class C^1 of the ACE code converges faster than that of the PEG-ACE code, while the other classes take more iterations to converge for the ACE code than for the PEG-ACE code. Theoretically, the MI may approach 1 arbitrarily close and there will still be UEP. The actual amount of UEP depends on the convergence speeds of the classes, [3]. In practice, the accuracy is limited by the numerical precision of the computations and, for example, approximations of the J-function. However, using the approximation of the error probability based on mutual information from [16]

$$P_b \approx \frac{1}{2}\,\mathrm{erfc}\left(\frac{\sqrt{8R\frac{E_b}{N_0} + J^{-1}(I_{Av})^2 + J^{-1}(I_{Ev})^2}}{2\sqrt{2}}\right), \quad (6)$$

the error probabilities of the protection classes may be estimated. Note that the results are not exact for finite-length codes and become more inexact with lower error rates. Nevertheless, we observe significant differences in BER between the classes if their MI values are close to 1 and only differ in the fifth decimal position. Remember that for all codes we consider here, there will be some UEP capability strictly depending on the variable node irregularity of the LDPC code. In the first iteration, all I_{Ec} will be almost equal (due to the concentrated check node degree distribution, i.e., a majority of the check nodes have the same degree) and the only difference in I_{APPv} between the classes will depend on the variable node degrees.

The above discussion shows that the number of edges from a check node to variable nodes of different classes, defined by the detailed check node degree distribution in Table II plays an important role to the UEP capability. However, in comparing the theoretical MI from above with the true values obtained by measuring the actual decoding LLRs, we observe deviations. The measured MI values are lower than the theoretical ones which is due to finite-length issues such as cycles and trapping sets. Nevertheless, the UEP properties are valid for both theoretical and measured observations.

IV. Modified PEG-ACE Construction with Increased UEP Capability

The above sections discussed the number of edges from check nodes to the protection classes and differences between the UEP codes and the non-UEP codes were found. In order to verify that the presented argument indeed is the reason for the differences in UEP capability, we modify the (non-UEP) PEG-ACE construction algorithm to yield codes with a similar detailed check node degree distribution as the UEP-capable ACE code. We further demonstrate that codes constructed by the modified PEG-ACE algorithm have good UEP capabilities.

The algorithm is modified in such a way that it only connects a variable node to check nodes which do not violate certain detailed check node degree distributions $\tilde{\rho}^{(C^j)}(x)$. Thereby, the modified PEG-ACE code can be forced to have

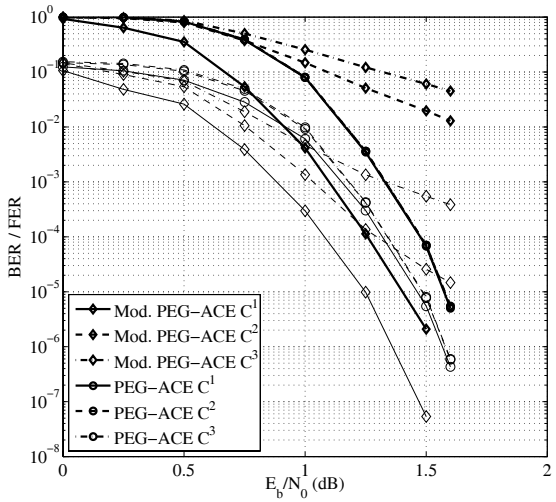

Fig. 6. FER and BER of the modified PEG-ACE code in comparison to the original PEG-ACE code after 100 iterations. The bold curves show FER and the thin curves show BER. The modification of the PEG-ACE code has increased its UEP capability significantly.

a detailed check node degree distribution similar to the ACE code instead of its original distribution. By doing so, the detailed check node degree distribution given in the three right-most columns of Table II is obtained. Notice that this is very similar to the detailed distribution of the ACE code. The modified algorithm explained above is called *modified PEG-ACE* in the following.

Fig. 6 shows the FER and BER of the modified PEG-ACE code in comparison to the original PEG-ACE code. The figure shows that by changing the detailed check node degree distribution of the original PEG-ACE code while maintaining the overall degree distributions, it is possible to enhance its UEP capability significantly. Instead of equal error rates for all classes, the modification has improved the BER of C^1 significantly, while slightly degrading C^2 and C^3. The modified PEG-ACE code shows even more UEP capability than the ACE code.

With the modified PEG-ACE algorithm, it is possible to enhance the UEP capability of a code by modifying its detailed check node degree distribution. Optimization of the detailed check node degree distribution is outside the scope of this paper. However, the detailed check node degree distribution may be used as a tool to move between codes with good UEP capability but lower overall performance and codes with good overall performance but less UEP capability.

V. Conclusions

We have compared the UEP capabilities of several construction algorithms. Results show that the considered algorithms may be divided into two groups: UEP-capable codes are those constructed by the random, the ACE, and the zigzag-random construction algorithm, while the non-UEP codes are constructed by the PEG, the PEG-ACE and the zigzag-PEG construction algorithm. Note that the authors of [4] used the random construction and showed good UEP properties while the authors of [1] used the PEG algorithm and could observe no UEP after 50 iterations. These findings explain why earlier literature on irregular UEP-LDPC codes show disagreeing results.

Analysis of the parity-check matrices of the different codes shows that the amount of connections between the different classes is higher for the non-UEP codes than for the UEP-capable codes. By introducing the concept of a detailed check node degree distribution, we have shown that the number of edges from a check node to variable nodes of different classes affects the UEP capability. Furthermore, the PEG-ACE construction algorithm has successfully been modified in order to construct good UEP-capable codes.

VI. Acknowledgments

The authors would like to thank Prof. James LeBlanc, Luleå University of Technology, Sweden, for motivating ideas, discussions, and support.

References

[1] V. Kumar and O. Milenkovic, "On unequal error protection LDPC codes based on Plotkin-type constructions," *IEEE Trans. Commun.*, vol. 54, pp. 994-1005, June 2006.

[2] L. Sassatelli, W. Henkel, and D. Declercq, "Check-irregular LDPC codes for unequal error protection under iterative decoding," in *Proc. 4th Int. Symp. Turbo Codes Related Topics 2006*, Apr. 2006.

[3] C. Poulliat, D. Declercq, and I. Fijalkow, "Enhancement of unequal error protection properties of LDPC codes," *EURASIP J. Wireless Commun. Netw.*, vol. 2007, pp. Article ID 92659, 9 pages, 2007. doi:10.1155/2007/92659.

[4] N. Rahnavard, H. Pishro-Nik, and F. Fekri, "Unequal error protection using partially regular LDPC codes," *IEEE Trans. Commun.*, vol. 55, pp. 387-391, Mar. 2007.

[5] R. Gallager, "Low-density parity-check codes," *IRE Trans. Inf. Theory*, vol. 8, pp. 21-28, Jan. 1962.

[6] T. Richardson, M. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 47, pp. 619-637, Feb. 2001.

[7] M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inf. Theory*, vol. 27, pp. 533-547, Sep. 1981.

[8] F. Kschischang, B. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inf. Theory*, vol. 47, pp. 498-519, Feb. 2001.

[9] S.-Y. Chung, T. Richardson, and R. Urbanke, "Analysis of sum-product decoding of low-density parity-check codes using a Gaussian approximation," *IEEE Trans. Inf. Theory*, vol. 47, pp. 657-670, Feb. 2001.

[10] X.-Y. Hu, E. Eleftheriou, and D. M. Arnold, "Regular and irregular progressive edge-growth Tanner graphs," *IEEE Trans. Inf. Theory*, vol. 51, pp. 386-398, Jan. 2005.

[11] X.-Y. Hu, E. Eleftheriou, and D. M. Arnold, "Progressive edge-growth Tanner graphs," in *Proc. IEEE Global Telecommun. Conf. 2001*, vol. 2, pp. 995-1001, Nov. 2001.

[12] T. Tian, C. Jones, D. Villasenor, and R. Wesel, "Selective avoidance of cycles in irregular LDPC code construction," *IEEE Trans. Commun.*, vol. 52, pp. 1242-1247, Aug. 2004.

[13] D. Vukobratovic and V. Senk, "Generalized ACE constrained progressive edge-growth LDPC code design," *IEEE Commun. Lett.*, vol. 12, pp. 32-34, Jan. 2008.

[14] K. Kasai, T. Shibuya, and K. Sakaniwa, "Detailedly represented irregular low-density parity-check codes," *IEICE Trans. Fundamentals*, vol. E86-A, pp. 2435-2444, Oct. 2003.

[15] G. Liva and M. Chiani, "Protograph LDPC codes design based on EXIT analysis," in *Proc. IEEE GLOBECOM 2007*, pp. 3250-3254, Nov. 2007.

[16] S. ten Brink, "Convergence behavior of iteratively decoded parallel concatenated codes," *IEEE Trans. Commun.*, vol. 49, pp. 1727-1737, Oct. 2001.

[17] S. ten Brink, G. Kramer, and A. Ashikhmin, "Design of low-density parity-check codes for modulation and detection," *IEEE Trans. Commun.*, vol. 52, pp. 670-678, Apr. 2004.