Hindawi Publishing Corporation
EURASIP Journal on Wireless Communications and Networking
Volume 2010, Article ID 423989, 8 pages
doi:10.1155/2010/423989

Research Article

Improved Design of Unequal Error Protection LDPC Codes

Sara Sandberg

Department of Computer Science and Electrical Engineering, Luleå University of Technology, SE-971 87 Luleå, Sweden

Correspondence should be addressed to Sara Sandberg, sara.sandberg@ltu.se

Received 7 September 2010; Accepted 9 November 2010

Academic Editor: Richard Kozick

We propose an improved method for designing unequal error protection (UEP) low-density parity-check (LDPC) codes. The method is based on density evolution. The degree distribution with the best UEP properties is found, under the constraint that the threshold should not exceed the threshold of a non-UEP code plus some threshold offset. For different codeword lengths and different construction algorithms, we search for good threshold offsets for the UEP code design. The choice of the threshold offset is based on the average a *posteriori* variable node mutual information. Simulations reveal the counter intuitive result that the short-to-medium length codes designed with a suitable threshold offset all outperform the corresponding non-UEP codes in terms of average bit-error rate. The proposed codes are also compared to other UEP-LDPC codes found in the literature.

1. Introduction

In many communication scenarios, such as wireless networks and transport of multimedia data, sufficient error protection is often a luxury. In these systems, it may be wasteful or even infeasible to provide uniform protection for all information bits. Instead, it is more efficient to protect the most important information more than the rest, by using a channel code with unequal error protection (UEP). This implies improving the performance of the more important bits by sacrificing some performance of the less important bits. This paper focuses on the design of UEP low-density parity-check (LDPC) codes with improved average bit-error rate (BER).

Several methods for designing UEP-LDPC codes have been presented, [1–11]. The irregular UEP-LDPC design schemes described in [1–7] are based on the irregularity of the variable and/or check node degree distributions. These schemes enhance the UEP properties of the code through density evolution methods. Vasic et al. proposed a class of UEP-LDPC codes based on cyclic difference families, [8]. In [9], UEP capability is achieved by a combination of two Tanner graphs of different rates. The UEP-LDPC codes presented in [10] are based on the algebraic Plotkin construction and are decoded in multiple stages. UEP may also be provided by nonbinary LDPC codes, [11].

In this work, we consider the flexible UEP-LDPC code 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 average 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. In the following, we call ϵ the threshold offset.

In the latest years, much effort has been spent on construction algorithms for short-to-medium-length LDPC codes, [12–14]. However, these algorithms rely on degree distributions optimized for infinitely long codes and focus on constructing LDPC code graphs with a small number of short cycles, thereby improving the performance in the error-floor region for short LDPC codes. In the design proposed here, we optimize the threshold offset given the construction algorithm used to specify the parity-check matrix of the code. The improved UEP codes have an average performance that is better than the corresponding non-UEP codes (with $\epsilon = 0$ dB), which may seem counter-intuitive. Typically, the reduced error rate of the most protected class

is compensated for by an increased error rate of the other classes. Nonetheless, we show that with a good choice of the threshold offset and for several common code construction algorithms, the performance of the UEP code is significantly better than the performance of the corresponding non-UEP code. However, for long codes and a high number of decoder iterations, the UEP code design reduces the performance since the UEP codes have worse thresholds than the non-UEP codes. The performance for different values of the threshold offset is found in [3], which shows that a threshold offset of 0.1 dB is a good choice. This is true for the random construction used in [3], but it is not noted that the best choice of the threshold offset varies for different construction algorithms.

Some intuition as to why UEP code design may increase the average performance can be gained by considering irregular LDPC codes, not designed for UEP, and their advantages compared to regular LDPC codes, [15]. In irregular codes, variable nodes with high degree typically correct their value quickly and these nodes can then help to correct lower degree variable nodes. Therefore, irregular graphs may lead to a wave effect, where the highest degree nodes are corrected first, then the nodes with slightly lower degree, and so on. The more irregular a code is, that is, the higher the maximum variable node degree, the faster the correction of the high degree variable nodes. There are reasons to believe that the code with the best threshold under an appropriate constraint on the allowed number of iterations, that is, a code with fast convergence, yields the best performance for finite-length codes also when the number of iterations is high, [16]. UEP code design is another way to achieve the differentiation between nodes that may lead to a wave effect and fast convergence. By allowing the code to have a worse threshold (as is the case in the UEP code design we consider), more differentiation between nodes in different classes can be achieved. It should also be noted that there is a trade-off between the maximum variable node degree and the codeword length. The maximum variable node degree should be lower for a short code to reduce the number of harmful cycles involving variable nodes of low degree. The wave effect achieved by the UEP design is accomplished without increasing the maximum variable node degree.

Let us recall some basic notation of LDPC codes. The sparse parity-check matrix H has dimension $(n-k)\times n$, where k and n are the lengths of the information word and the codeword, respectively. We consider irregular LDPC codes with edge-based variable node and check node degree distributions defined by the polynomials [16] $\lambda(x) = \sum_{i=2}^{d_{v\max}} \lambda_i x^{i-1}$ 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, respectively. For UEP codes, we divide the variable nodes into several protection classes $(C^1, C^2, \ldots, C^{Nc})$ with degrading level of protection. The resulting variable node degree distribution is defined by the coefficients $\lambda_i^{(Cj)}$, which denote the fractions of edges incident to degree-i variable nodes of protection class C^j. The overall degree distribution is given by $\lambda(x) = \sum_{j=1}^{Nc} \sum_{i=2}^{d_{v\max}} \lambda_i^{(Cj)} x^{i-1}$. In the following, we distinguish between code design, by which we mean the design of degree distributions that describe a code ensemble, and code construction, by which we mean the construction of a specific code realization (described by a parity-check matrix).

2. Design of Finite-Length UEP Codes

In [16], Richardson et al. state that for short LDPC codes, it is not always best to pick the degree distribution pair with the best threshold. Instead, it can be advantageous to look for the best possible threshold under an appropriate constraint on the allowed number of iterations. In this paper, we show that by searching among degree distributions designed for UEP with worse threshold than the corresponding non-UEP degree distributions, we may find degree distributions with significantly lower error rates for a finite length than the degree distributions with the best possible threshold. Well-designed UEP codes have faster convergence for all protection classes and thereby better performance for finite-length LDPC codes.

2.1. Detailed Mutual Information Evolution. An appropriate method for analyzing UEP codes is needed to choose a good value for the threshold offset ϵ, without relying on time-consuming error rate simulations. We consider the theoretical mutual information (MI) functions, which are typically calculated from the degree distributions $\lambda(x)$ and $\rho(x)$ of a code. However, different LDPC codes with the same degree distributions can have very different UEP properties, [17]. The differences depend on how different protection classes are connected, which in turn depends on the code construction algorithm used to place the edges in the graph according to the given degree distributions. To observe the differences also between codes with equal degree distributions, 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 [18]. We follow the same approach, but use the parity-check matrix instead of the protograph base matrix. See [19, 20] for more details on MI analysis. The detailed MI evolution is described in detail in the appendix. In the following, we use the average *a posteriori* variable node MI denoted by $I_{\mathrm{APP}v}$ (calculated for each variable node in step (5) of the MI analysis in the appendix) to compare the convergence rates of different LDPC codes.

2.2. Design Procedure. For simplicity, a good value of ϵ is found through an exhaustive search in a region of typical values. In the following section, we show that there is only a small difference in MI and BER for similar values of ϵ. It is therefore reasonable to consider only a few values of ϵ in the search and the best value among these is likely to give a BER result very close to what could be achieved by a more thorough search. For each value of ϵ in the range of the search, three steps must be performed.

(1) Design a UEP code following [3] for the ϵ under consideration, keeping $\rho(x)$, $d_{v\max}$, and the proportions

between the protection classes fixed. This step results in subdegree distributions for each protection class.

(2) Construct a parity-check matrix using an appropriate code construction algorithm.

(3) Calculate the detailed MI evolution for a given E_b/N_0 and a maximum number of decoder iterations. The code with the highest average $I_{\text{APP}v}$ has the best overall performance within this family of codes.

The value of the threshold offset ϵ is optimized for a specific E_b/N_0, which means that a code that is optimized for low E_b/N_0 may perform worse for high E_b/N_0, and vice versa. For UEP codes, the proportions between the protection classes also affect the UEP properties of a code and thereby are also the best choice of ϵ. In our simulations we have seen that with 20% of the information bits in the most protected class C^1 and 80% in a less protected class C^2, good performance is achieved for rate 1/2 codes of different lengths. We therefore omit further investigations of the effect of different proportions between the protection classes.

3. Design Examples

We design UEP codes of lengths $n = 2048$ and $n = 8192$. All codes are designed using the check node degree distributions given in [16, Table II]. The performance of any UEP code is compared to the performance of a non-UEP code with the variable node degree distribution that gives the best threshold, also tabulated in [16, Table II]. A maximum of 100 decoder iterations is allowed. Except for the variable node degree distribution, the UEP codes and the corresponding non-UEP code have the same parameters. We consider only rate-1/2 LDPC codes. All UEP codes presented in this section have 20% of the information bits in C^1 and the remaining 80% in C^2. A third protection class contains all parity bits. We first focus on design of generalized ACE constrained progressive edge-growth (PEG) codes [14] (in the following denoted by PEG-ACE codes) in Section 3.1. The random construction and the PEG construction algorithm [12] are considered in Section 3.2.

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 [12]. The *approximate cycle extrinsic message degree (ACE) construction* algorithm lowers the error floor by emphasizing both the number of edges from variable nodes in a cycle to nodes in the graph that are not part of the cycle as well as the length of cycles [13]. 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 [14]. 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.

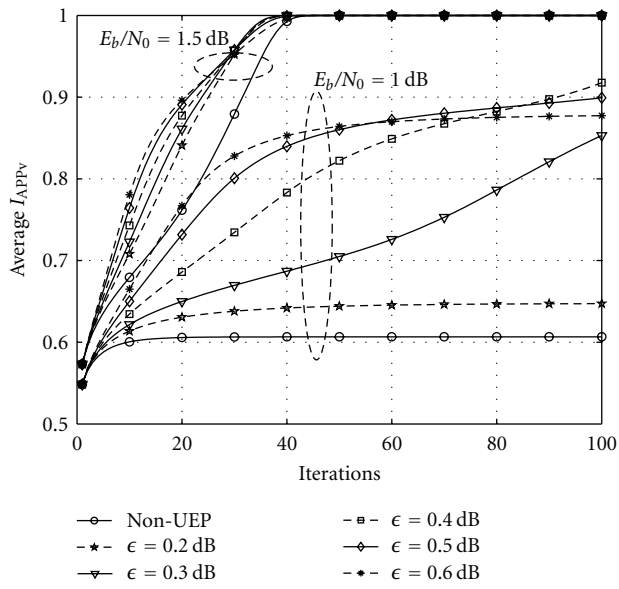

Figure 1: Average *a posteriori* variable node MI as a function of decoder iterations for six different PEG-ACE codes with $n = 2048$ and $d_{v\max} = 20$. For a low number of iterations, a large ϵ gives fast convergence for both E_b/N_0 shown. For a larger number of iterations (around 60), $\epsilon = 0.5$ dB gives the highest MI at $E_b/N_0 = 1$ dB, and $\epsilon = 0.3$ dB gives the highest MI at $E_b/N_0 = 1.5$ dB.

3.1. Optimization of the Threshold Offset for PEG-ACE Codes. Six different PEG-ACE codes with varying ϵ are designed and constructed according to the design procedure in Section 2.2 Non-UEP codes correspond to $\epsilon = 0$ dB. These codes have length $n = 2048$ and allowed maximum variable node degree $d_{v\max} = 20$. Figure 1 shows the average *a posteriori* variable node MI as a function of decoder iterations at two different E_b/N_0. For a low number of iterations, a large ϵ gives fast convergence for both E_b/N_0 shown. This implies that for applications where only a small number of decoder iterations are allowed, a large ϵ yields the best performance. The average $I_{\text{APP}v}$ at $E_b/N_0 = 1$ dB is maximized by $\epsilon = 0.5$ dB. After 100 iterations, $\epsilon = 0.4$ dB gives the highest average $I_{\text{APP}v}$, but simulations show that the code with $\epsilon = 0.5$ dB outperforms the code with $\epsilon = 0.4$ dB at low E_b/N_0. At $E_b/N_0 = 1.5$ dB, $\epsilon = 0.3$ dB maximizes the average $I_{\text{APP}v}$. The variable node degree distributions for the PEG-ACE codes with $\epsilon = 0.3$ dB and $\epsilon = 0.5$ dB are tabulated in Table 1. The random code with $\epsilon = 0.1$ dB will be considered in Section 3.2.

Figure 2 shows the BER performance of the two protection classes containing information bits for $\epsilon = 0.3$ dB and $\epsilon = 0.5$ dB. The BER of the non-UEP code is shown for comparison. The figure shows that the code with $\epsilon = 0.5$ dB performs well for low E_b/N_0 as expected, while the code with lower ϵ has less UEP capability but better average performance at high E_b/N_0. The average BER is just slightly lower than the BER of C^2, since the average BER is calculated from the BERs of the two protection classes containing information bits, scaled with the proportions of the classes.

Table 1: Variable node degree distributions for two PEG-ACE codes ($\epsilon = 0.3$ dB and $\epsilon = 0.5$ dB) and the random $\epsilon = 0.1$ dB code.

	PEG-ACE $\epsilon = 0.3$ dB			PEG-ACE $\epsilon = 0.5$ dB			Random $\epsilon = 0.1$ dB		
	C^1	C^2	C^3	C^1	C^2	C^3	C^1	C^2	C^3
λ_2			0.2071			0.2125			0.2332
λ_3		0.2288	0.0496		0.2882	0.0415		0.1143	0.0105
λ_4		0.0792						0.2319	
λ_{17}							0.3749		
λ_{18}	0.3772						0.0353		
λ_{19}	0.0581			0.4268					
λ_{20}				0.0310					

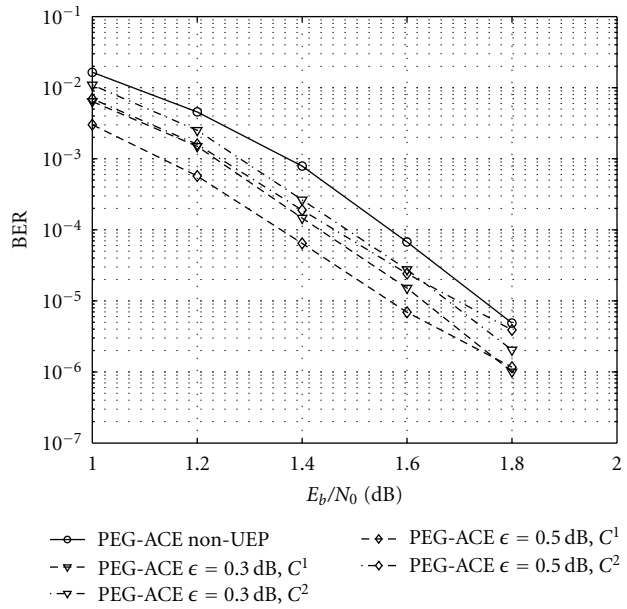

Figure 2: BER performance of three PEG-ACE codes with length $n = 2048$ and allowed $d_{v\max} = 20$. The average BER is shown for the non-UEP code, while the BERs of both C^1 and C^2 are shown for the UEP codes. Both classes of the UEP codes perform better than the non-UEP code.

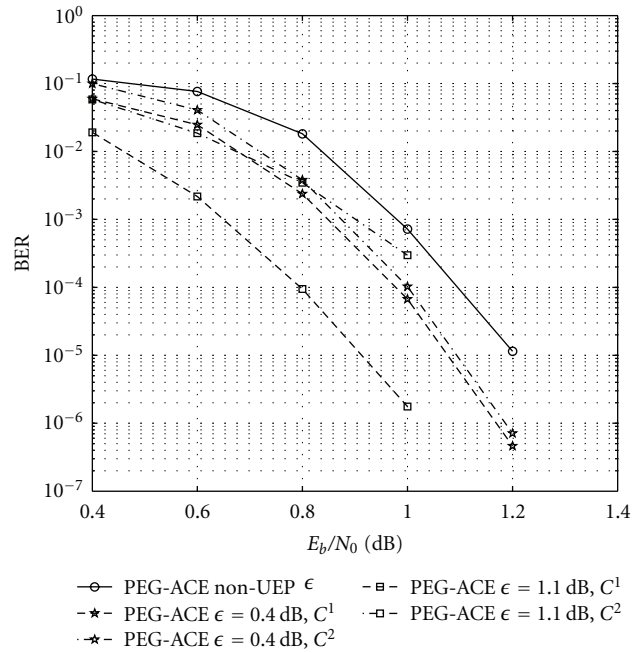

Figure 3: BER performance of non-UEP and UEP codes of length $n = 8192$ and $d_{v\max} = 30$. Both classes of the two UEP codes have lower BER than the non-UEP code. The code optimized for a low E_b/N_0 (0.5 dB) with $\epsilon = 1.1$ dB performs best at low E_b/N_0, while the code optimized for high E_b/N_0 (1 dB) with $\epsilon = 0.4$ dB has a lower average BER at E_b/N_0 above 0.8 dB.

Note that both classes of these UEP codes perform better than the comparable non-UEP code. In addition, the UEP codes offer a small difference in BER between the classes.

Figure 3 shows the performance of three PEG-ACE codes of length $n = 8192$ and allowed maximum variable node degree $d_{v\max} = 30$. The non-UEP code is compared to a code optimized for low E_b/N_0 (which gives $\epsilon = 1.1$ dB) and a code optimized for high E_b/N_0 (which gives $\epsilon = 0.4$ dB). Both classes of the two UEP codes perform better than the non-UEP code. At $E_b/N_0 = 1.2$ dB, the UEP code with $\epsilon = 0.4$ dB has an average BER which is around one magnitude less than the non-UEP code.

3.2. Code Design for Other Construction Algorithms. For given degree distributions, it has been shown that the choice of the construction algorithm strongly affects the UEP properties of the LDPC code, [17]. For codes with little inherent UEP (e.g., PEG and PEG-ACE codes), the threshold offset ϵ needs to be large to yield a code with good UEP capability. On the other hand, for codes with significant inherent UEP (e.g., randomly constructed codes), a high ϵ may make the less protected classes so badly protected that a wave effect does not occur. Figure 4 shows the performance of codes of length $n = 2048$ constructed by three different algorithms: the random construction, only avoiding cycles of length 4, the PEG construction algorithm, and the PEG-ACE construction algorithm. The figure shows the BER of the non-UEP codes as well as the BER of classes C^1 and C^2 of the UEP codes. The variable node degree distributions of the UEP codes are given in Table 1. Note that the PEG $\epsilon = 0.5$ dB

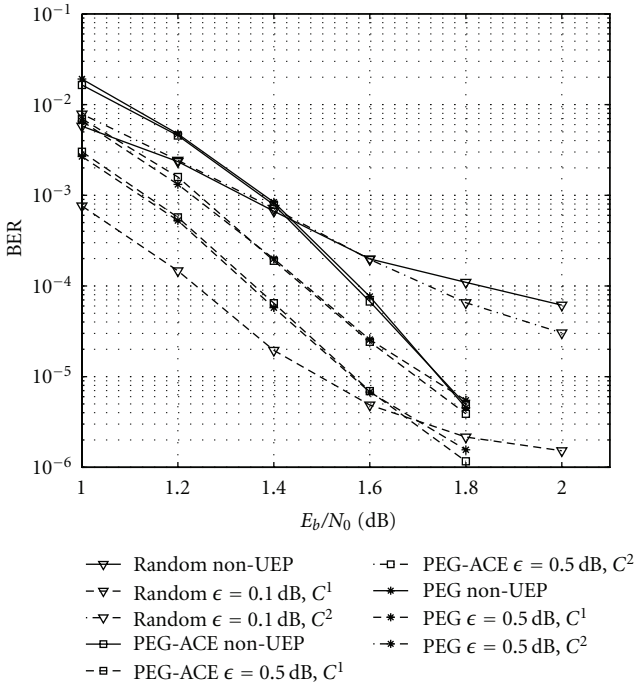

FIGURE 4: BER performance of non-UEP and UEP codes, constructed by three different construction algorithms. All codes have $n = 2048$ and allowed $d_{v\max} = 20$. The random code has good UEP capability already for $\epsilon = 0.1$ dB, while the PEG code and the PEG-ACE code have less UEP capability for $\epsilon = 0.5$ dB. All UEP codes have better average performance than the corresponding non-UEP codes, except for the random code at low E_b/N_0.

code has the same variable node degree distribution as the PEG-ACE $\epsilon = 0.5$ dB code.

For the random code, MI calculations at $E_b/N_0 = 1$ dB show that the non-UEP code gives the highest MI, except at very few decoder iterations. At $E_b/N_0 = 1.5$ dB, the code with $\epsilon = 0.1$ dB is slightly better than other choices of ϵ after around 50 iterations. Thereby it can be assumed that the non-UEP random code will perform better at low E_b/N_0, and the UEP random code with $\epsilon = 0.1$ dB will perform better at high E_b/N_0. This is confirmed by the results shown in Figure 4. For the PEG and PEG-ACE code, the threshold offset $\epsilon = 0.5$ dB used to design the codes gives the maximum MI at $E_b/N_0 = 1$ dB. For both codes, $\epsilon = 0.3$ dB gives higher MI at $E_b/N_0 = 1.5$ dB, but for the PEG code there is only a slight difference in MI between these two values of ϵ.

The figure shows that both the PEG and the PEG-ACE UEP codes have significantly better performance than the corresponding non-UEP codes at low E_b/N_0. Remember that the PEG-ACE code with $\epsilon = 0.3$ dB performs better for high E_b/N_0. The PEG and PEG-ACE construction algorithms result in codes with little inherent UEP and the UEP capabilities gained by the relatively high ϵ give faster convergence of the codes. However, the random UEP code has only a slightly lower average BER than the non-UEP code at high E_b/N_0. Note that the average performance of random codes with $\epsilon > 0.1$ dB is worse than for the non-UEP code. That is, for the random code with much inherent UEP there is not as much to gain by the UEP code design as for the PEG and the PEG-ACE codes, which have little inherent UEP.

These results show as expected that a PEG or PEG-ACE code should typically be chosen instead of a random code, even if the application benefits from UEP. For a large range of E_b/N_0, the most protected class of the PEG and PEG-ACE code has only slightly worse performance than the random code, while the average BER is much higher for the random code. This paper shows that we can improve both the average BER performance and the UEP capability by optimizing the threshold offset.

3.3. Comparison to Other UEP-LDPC Codes. UEP-LDPC codes of similar length and rate as the codes presented here can be found in [6, 7]. Ma and Kwak [6] propose a partially regular code design, where all variable nodes in one protection class have the same degree. Good variable node degrees for each class are found through density evolution using the Gaussian approximation. Figures 5 and 6 show the performance of the code designed by Ma and Kwak together with the performance of two PEG-ACE codes (also shown in Figure 2). All these codes have $n = 2048$ and $d_{v\max} = 20$. The code designed by Ma and Kwak has 12.5% of the information bits in C^1 and 87.5% in C^2, compared to the codes proposed in this paper where 20% of the information bits belong to C^1. Note that Ma and Kwak [6] present the performance after only 10 decoder iterations, so in Figure 5 we compare their code to the PEG-ACE codes after 10 iterations. Figure 6 demonstrates the performance of the same codes after 100 iterations. The performance of the Ma and Kwak code for 10 iterations is taken directly from [6]. To find the performance after 100 iterations, we have run simulations with a code constructed according to the specifications given in [6]. It has been verified that our simulations give the same result as shown in [6] after 10 iterations.

Figure 5 shows that after only 10 decoder iterations, the PEG-ACE $\epsilon = 0.5$ dB code performs best at low E_b/N_0. This is in accordance with Figure 1, which demonstrates that the code with the highest ϵ has the best average MI when the number of decoder iterations is low. However, at high E_b/N_0, the code proposed by Ma and Kwak has a lower average BER (C^2 performs better). The PEG-ACE $\epsilon = 0.3$ dB code performs almost the same as the Ma and Kwak code, except at high E_b/N_0, where C^2 of the PEG-ACE code has worse performance.

After 100 decoder iterations, see Figure 6, the PEG-ACE $\epsilon = 0.5$ dB code performs well at low E_b/N_0 compared to the code designed by Ma and Kwak. Up to $E_b/N_0 = 1.5$ dB, there is a gain of around 0.13 dB for C^1 and around 0.1 dB for C^2 for the different BERs. At $E_b/N_0 = 1.8$ dB, the Ma and Kwak code performs slightly better than the PEG-ACE $\epsilon = 0.5$ dB code. The PEG-ACE $\epsilon = 0.3$ dB code has a little bit lower BERs than the Ma and Kwak code at all E_b/N_0.

Note that there is a significant difference in BER between C^1 and C^2 after 10 iterations, while the difference is much smaller after 100 iterations. This is typical for codes constructed using the PEG or PEG-ACE algorithm [17].

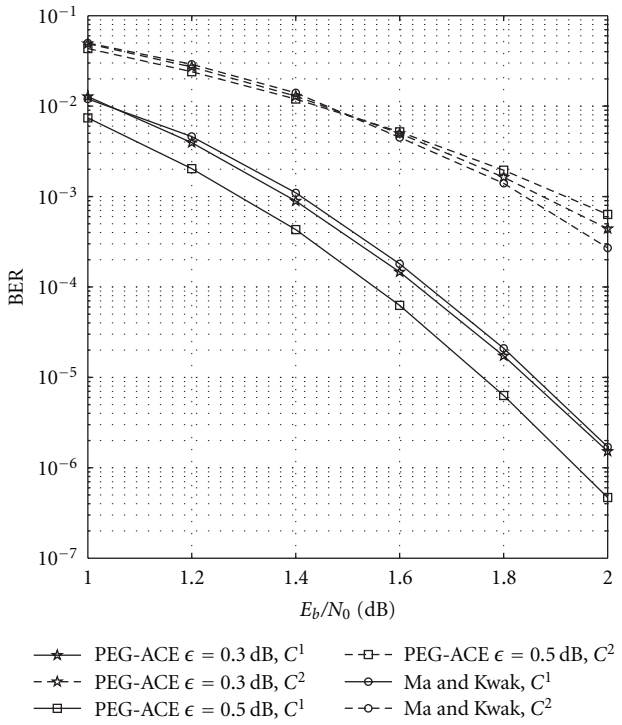

FIGURE 5: BER performance of the PEG-ACE $\epsilon = 0.5$ dB code and the code designed by Ma and Kwak [6] after 10 decoder iterations. Both codes have $n = 2048$ and $d_{v\max} = 20$.

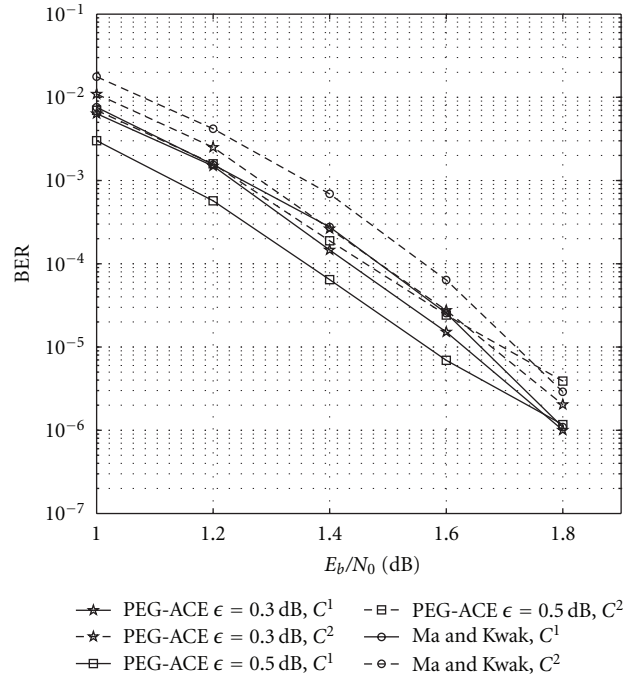

FIGURE 6: BER performance of two PEG-ACE codes and the code designed by Ma and Kwak [6] after 100 decoder iterations. Both codes have $n = 2048$ and $d_{v\max} = 20$. The PEG-ACE $\epsilon = 0.3$ dB code performs slightly better than the code designed by Ma and Kwak for both classes. The PEG-ACE $\epsilon = 0.5$ dB code performs much better than the Ma and Kwak code for low E_b/N_0, but C^2 has slightly higher BER at high E_b/N_0.

However, if 100 iterations are allowed, both classes have a lower BER than the most protected class have after only 10 iterations. Thus, if a reasonably high number of iterations can be allowed in terms of time and complexity, it is better to run many iterations even if the difference in error protection between the classes is reduced.

Yang et al. [7] present simulation results for an irregular UEP-LDPC code of length $n = 10000$ and $d_{v\max} = 11$, which may be compared to the PEG-ACE codes proposed here with $n = 8192$ and $d_{v\max} = 30$. Figure 7 shows the performance of these codes. Note that Yang et al. divide the information bits into three different protection classes, 1485 bits in C^1, 307 bits in C^2, and 3208 bits in C^3. The PEG-ACE $\epsilon = 0.4$ dB code performs best for high E_b/N_0. At $E_b/N_0 = 1.2$ dB, C^2 of the PEG-ACE $\epsilon = 0.4$ dB code has a lower BER than C^1 of the code designed by Yang et al. At low E_b/N_0, the PEG-ACE $\epsilon = 0.4$ dB code has similar performance as the code designed by Yang et al. The PEG-ACE $\epsilon = 1.1$ dB code has very good performance at low E_b/N_0, and at E_b/N_0 up to 0.7 dB the average BER is lower than the BER of the most protected class in the code designed by Yang et al.

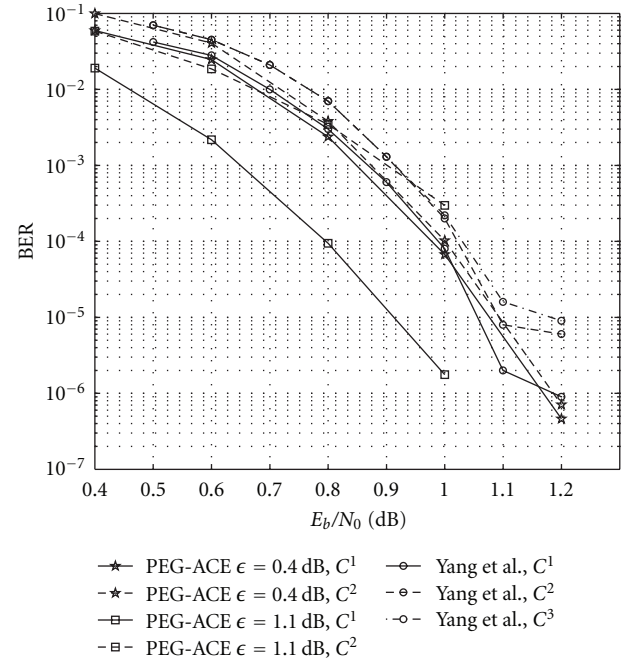

FIGURE 7: BER performance of two PEG-ACE codes with $n = 8192$ and $d_{v\max} = 30$ and the code designed by Yang et al. [7] with $n = 10000$ and $d_{v\max} = 11$.

4. Conclusions

We have proposed an improved design algorithm for UEP-LDPC codes, resulting in codes with reduced average BER. The algorithm searches for good threshold offsets for the UEP design, given different codeword lengths and different construction algorithms. The choice of the threshold offset

is based on the average *a posteriori* variable node MI of the codes. Simulations show that the codes designed with a suitable threshold offset all outperform the corresponding non-UEP codes in terms of average BER. We show that the average BER is reduced by up to an order of magnitude by the proposed code design.

Appendix

Detailed MI Evolution

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 $I_{\mathrm{ch}} = J(\sigma_{\mathrm{ch}})$, where

$$\sigma_{\mathrm{ch}}^2 = 8R\frac{E_b}{N_0} \tag{A.1}$$

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

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

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 [18]

$$I_{Ev|s} = J\left(\sqrt{\sum_{l=1,l\neq s}^{d_v} \left[J^{-1}\left(I_{Av|l}\right)\right]^2 + \left[J^{-1}\left(I_{\mathrm{ch}}\right)\right]^2}\right), \tag{A.3}$$

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

$$I_{Ec|s} = 1 - J\left(\sqrt{\sum_{l=1,l\neq s}^{d_c} \left[J^{-1}\left(1-I_{Ac|l}\right)\right]^2}\right), \tag{A.4}$$

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 [21]) and are not exact.

The following algorithm describes the MI analysis of a given parity-check matrix. We denote element (i, j) of the parity-check matrix by $h_{i,j}$.

(1) Initialization

$$I_{\mathrm{ch}} = J(\sigma_{\mathrm{ch}}). \tag{A.5}$$

(2) Check to variable update

(a) For $i = 1,\ldots,n-k$ and $j = 1,\ldots,n$, if $h_{i,j} = 1$, calculate

$$I_{Ev}(i,j) = J\left(\sqrt{\sum_{s\in\mathcal{C}_j, s\neq i} \left[J^{-1}\left(I_{Av}(s,j)\right)\right]^2 + \left(J^{-1}(I_{\mathrm{ch}})\right)^2}\right), \tag{A.6}$$

where $\mathcal{C}_j$ is the set of check nodes incident to variable node j.

(b) If $h_{i,j} = 0$, $I_{Ev}(i,j) = 0$.

(c) For $i = 1,\ldots,n-k$ and $j = 1,\ldots,n$, set $I_{Ac}(i,j) = I_{Ev}(i,j)$.

(3) Variable to check update

(a) For $i = 1,\ldots,n-k$ and $j = 1,\ldots,n$, if $h_{i,j} = 1$, calculate

$$I_{Ec}(i,j) = 1 - J\left(\sqrt{\sum_{s\in\mathcal{V}_i, s\neq j} \left[J^{-1}\left(1-I_{Ac}(i,s)\right)\right]^2}\right), \tag{A.7}$$

where $\mathcal{V}_i$ is the set of variable nodes incident to check node i.

(b) If $h_{i,j} = 0$, $I_{Ec}(i,j) = 0$.

(c) For $i = 1,\ldots,n-k$ and $j = 1,\ldots,n$, set $I_{Av}(i,j) = I_{Ec}(i,j)$.

(4) *A posteriori* check node MI

For $i = 1,\ldots,n-k$, calculate

$$I_{\mathrm{APP}c}(i) = 1 - J\left(\sqrt{\sum_{s\in\mathcal{V}_j} \left[J^{-1}\left(1-I_{Ac}(i,s)\right)\right]^2}\right). \tag{A.8}$$

(5) *A posteriori* variable node MI

For $j = 1,\ldots,n$, calculate

$$I_{\mathrm{APP}v}(j) = J\left(\sqrt{\sum_{s\in\mathcal{C}_j} \left[J^{-1}\left(I_{Av}(s,j)\right)\right]^2 + \left[J^{-1}\left(I_{\mathrm{ch}}\right)\right]^2}\right). \tag{A.9}$$

(6) Repeat (2)–(5) until $I_{\mathrm{APP}v} = 1$ for $j = 1,\ldots,n$.

References

[1] N. Rahnavard and F. Fekri, "Unequal error protection using low-density parity-check codes," in *Proceedings of the IEEE International Symposium on Information Theory (ISIT '04)*, p. 449, July 2004.

[2] N. Rahnavard, H. Pishro-Nik, and F. Fekri, "Unequal error protection using partially regular LDPC codes," *IEEE Transactions on Communications*, vol. 55, no. 3, pp. 387–391, 2007.

[3] C. Poulliat, D. Declercq, and I. Fijalkow, "Enhancement of unequal error protection properties of LDPC codes," *Eurasip Journal on Wireless Communications and Networking*, vol. 2007, Article ID 92659, 9 pages, 2007.

[4] L. Sassatelli, W. Henkel, and D. Declercq, "Check-irregular LDPC codes for unequal error protection under iterative decoding," in *Proceedings of the 4th International Symposium on Turbo Codes & Related Topics*, April 2006.

[5] H. Pishro-Nik, N. Rahnavard, and F. Fekri, "Nonuniform error correction using low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 51, no. 7, pp. 2702–2714, 2005.

[6] P. Ma and K. S. Kwak, "Unequal error protection low-density parity-check codes design based on gaussian approximation in image transmission," in *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC '09)*, pp. 1–6, April 2009.

[7] X. Yang, D. Yuan, P. Ma, and M. Jiang, "New research on unequal error protection (UEP) property of irregular LDPC codes," in *Proceedings of the 1st Consumer Communications and Networking Conference (CCNC '04)*, pp. 361–363, January 2004.

[8] B. Vasic, A. Cvetkovic, S. Sankaranarayanan, and M. Marcellin, "Adaptive error protection low-density parity-check codes for joint source-channel coding schemes," in *Proceedings of the IEEE International Symposium on Information Theory (ISIT '03)*, pp. 267–267, July 2003.

[9] N. Rahnavard and F. Fekri, "New results on unequal error protection using LDPC codes," *IEEE Communications Letters*, vol. 10, no. 1, pp. 43–45, 2006.

[10] V. Kumar and O. Milenkovic, "On unequal error protection LDPC codes based on plotkin-type constructions," *IEEE Transactions on Communications*, vol. 54, no. 6, pp. 994–1005, 2006.

[11] A. Goupil and D. Declercq, "UEP non-binary LDPC codes: a promising framework based on group codes," in *Proceedings of the IEEE International Symposium on Information Theory (ISIT '08)*, pp. 2227–2231, July 2008.

[12] X. Y. Hu, E. Eleftheriou, and D. M. Arnold, "Regular and irregular progressive edge-growth tanner graphs," *IEEE Transactions on Information Theory*, vol. 51, no. 1, pp. 386–398, 2005.

[13] T. Tian, C. R. Jones, J. D. Villasenor, and R. D. Wesel, "Selective avoidance of cycles in irregular LDPC code construction," *IEEE Transactions on Communications*, vol. 52, no. 8, pp. 1242–1247, 2004.

[14] D. Vukobratović and V. Šenk, "Generalized ACE constrained progressive edge-growth LDPC code design," *IEEE Communications Letters*, vol. 12, no. 1, pp. 32–34, 2008.

[15] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Improved low-density parity-check codes using irregular graphs," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 585–598, 2001.

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

[17] N. Von Deetzen and S. Sandberg, "On the UEP capabilities of several LDPC construction algorithms," *IEEE Transactions on Communications*, vol. 58, no. 11, pp. 3041–3046, 2010.

[18] G. Liva and M. Chiani, "Protograph LDPC codes design based on EXIT analysis," in *Proceedings of the 50th Annual IEEE Global Telecommunications Conference (GLOBECOM '07)*, pp. 3250–3254, November 2007.

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

[20] S. Ten Brink, G. Kramer, and A. Ashikhmin, "Design of low-density parity-check codes for modulation and detection," *IEEE Transactions on Communications*, vol. 52, no. 4, pp. 670–678, 2004.

[21] S. Y. Chung, T. J. Richardson, and R. L. Urbanke, "Analysis of sum-product decoding of low-density parity-check codes using a Gaussian approximation," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 657–670, 2001.