On the Design of Generalized LDPC Codes with Component BCJR Decoding

Yanfang Liu†, Pablo M. Olmos*, and David G. M. Mitchell†
†Klipsch School of Electrical and Computer Engineering, New Mexico State University, Las Cruces, USA
{viviliu,dgmm@nmsu.edu}
*Universidad Carlos III de Madrid & Gregorio Marañón Health Research Institute, Madrid, Spain
{olmos@tsc.uc3m.es}

GLOBECOM 2020 - 2020 IEEE Global Communications Conference | 978-1-7281-8298-8/20/$31.00 ©2020 IEEE | DOI: 10.1109/GLOBECOM42002.2020.9322143

***Abstract*—Generalized low-density parity-check (GLDPC) codes, where the single parity-check (SPC) nodes are replaced by generalized constraint (GC) nodes, are known to offer a reduced gap to capacity when compared with conventional LDPC codes, while also maintaining linear growth of minimum distance. However, for certain classes of practical GLDPC codes, there remains a gap to capacity even when utilizing blockwise decoding algorithm at GC nodes. In this work, we propose to optimize the design of GLDPC codes where the GC nodes are decoded with a trellis-based bit-wise Bahl-Cocke-Jelinek-Raviv (BCJR) component decoding algorithm. We analyze the asymptotic threshold behavior of GLDPC codes and determine the optimal proportion of the GC nodes in the GLDPC Tanner graph. We show significant performance improvements compared to existing designs with the same order of decoding complexity.**

***Index Terms*—Generalized low-density parity-check codes, BCJR decoding, trellis of linear block codes**

I. Introduction

As first proposed by Tanner in 1980's [1], generalized low-density parity check (GLDPC) codes have been shown to provide both good minimum distance and low decoding complexity [2], [3], [5]. In contrast to conventional low-density parity-check (LDPC) codes, the single parity-check (SPC) nodes in the GLDPC Tanner graph are replaced with generalized constraint (GC) nodes. The code associated with each GC node is called a component code. In the GLDPC literature, Hamming codes, Hadamard codes, BCH codes, convolutional codes, and expurgated random codes have been used as component codes [2]–[5]. With careful selection of the component codes, GLDPC codes have been shown to be asymptotically good [2], [6] and possess excellent iterative decoding thresholds [3], [5]. Consequently, many advantages, e.g., improved performance in noisy channels, low error floor and fast convergence speed, are promised [3], [7].

In a recent paper [8], it was shown that upon selecting a specific class of component codes, the degree distribution (DD) of a GLDPC code ensemble can be optimized for good thresholds while maintaining linear growth of minimum distance. Moreover, by employing a probabilistic description of the decoding capabilities at GC nodes, the authors managed to analyze the trade-off between coding rate and iterative decoding threshold as they increase the proportion ν of GC nodes in the GLDPC Tanner graph. Although the thresholds of designed GLDPC code ensembles are improved when compared to conventional LDPC code ensembles, a gap to capacity remains even with the optimal proportion of GC nodes and blockwise decoding of component codes.

In order to improve the decoding performance and decrease this threshold gap, we propose to optimize GLDPC code design where component GC nodes are decoded with the Bahl-Cocke-Jelinek-Raviv (BCJR) decoding algorithm. The BCJR decoding at GC nodes allows component decoders to correct errors that are otherwise uncorrectable with the blockwise decoding algorithm of [8]; however, the proportion, type, and location of GC nodes must be optimized accordingly.

Using exemplary $(2, 6)$- and $(2, 7)$-regular GLDPC codes, which have $(6, 3)$ shortened Hamming and $(7, 4)$ Hamming as constraint codes, respectively, we begin by examining the asymptotic threshold behavior of GLDPC codes and determining the optimal proportion of the GC nodes in the GLDPC Tanner graph. Significant threshold improvements are observed when compared to the results of [8], [9]. We then expand the approaches of [10] and [11] to derive a suitable trellis and BCJR rules for the constraint Hamming codes, and present simulation results confirming the robust error control performance predicted by the asymptotic analysis. We also compare the decoding performance of a $(2, 6)$-regular GLDPC code with an optimized proportion of GC nodes, BCJR component decoding, and an outer code to match the rates of the 5G candidates proposed in [12], [13]. Finally, we investigate the decoding complexity of the proposed schemes and show the same order of complexity compared to the technique of [9].

II. Asymptotic Analysis of BCJR Decoded GC Nodes in GLDPC Codes

In this section, we determine the optimal proportion of GC nodes in GLDPC code ensembles via an asymptotic analysis over the binary erasure channel (BEC).

A. GLDPC Codes and Code Ensembles

We represent a GLDPC code ensemble as shown in Fig. 1, where the Tanner graph has n variable nodes (coded bits), illustrated by white circles, that are connected by edges to c constraint nodes, illustrated by plus squares. Among the c parity-check nodes, a proportion $\nu \in [0, 1]$ correspond to GC nodes (shaded plus squares) while the rest correspond to SPC nodes (white plus squares).

978-1-7281-8298-8/20/$31.00 ©2020 IEEE

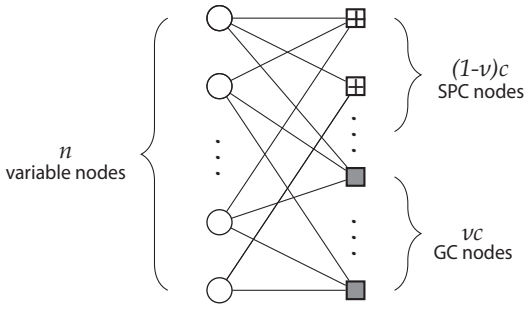

Fig. 1. Tanner graph of a GLDPC code.

An *ensemble* of GLDPC codes of length n is defined by a DD and proportion of GC nodes ν.[1] Note that the ensemble contains all codes of length n where the fraction ν of GC nodes are randomly placed, the edges are randomly assigned (according to the given DD), and the edge ordering (ordering code bits for a component code) is randomly assigned. The *design rate* of the GLDPC code ensemble is $R(\nu) = R - \nu(1-R)(1-m)$, where m is the number of linearly independent parity-check in the component code, and R is the original LDPC code rate [8].

B. Peeling Decoding Analysis

For the BEC, *peeling decoding* (PD) algorithms are employed to iteratively decode graph-based codes (such as LDPC codes and GLDPC codes) [19]. For LDPC codes, PD is initialized by removing all known variable nodes and attached edges from the Tanner graph and then the algorithm iteratively removes any remaining degree 1 check nodes (the degree of a node is the number of edges connected with this node), the edges adjacent with the removed check nodes, and connected variable nodes (now known). If all of the variable nodes are removed from the graph, it corresponds to a decoding *success*, otherwise it is a decoding *failure*. A sequence of *residual graphs* whose evolution can be predicted by a set of differential equations is produced by this decoding process [19]. The *threshold* of the LDPC code ensemble is given by the biggest channel parameter ϵ of the BEC which ensures a decoding success.

For GLDPC codes, the initialization of PD is same as the LDPC codes; however, the PD of GLDPC codes will then iteratively remove any degree 1 SPC nodes *as well as* GC nodes with input erasure patterns that are correctable, the edges adjacent with the removed check nodes (SPC or GC), and the connected variable nodes. Given the minimum distance, decoding capability, the proportion of GC nodes and DD of the GLDPC graph, the differential equation method was extended in [8] to predict the asymptotic evolution and the corresponding thresholds of GLDPC code ensembles when applying decoding algorithm for the GC nodes. In the remainder of this section, the asymptotic evolution of GLDPC codes is analyzed when applying bit-wise BCJR decoding to the GC nodes.

C. BCJR Component Decoding

To illustrate the difference between the technique in [8] and BCJR[2] decoding of GC nodes, we give a specific example where the component codes are chosen to be the $(7, 4, d_{\min} = 3)$ Hamming code, which has generator matrix

$$\mathbf{G}_{74} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{g}_1 \\ \mathbf{g}_2 \\ \mathbf{g}_3 \\ \mathbf{g}_4 \end{bmatrix}, \quad (1)$$

where $d_{\min}$ represents the minimum distance of the component code. Assume that the all zero codeword is transmitted, then any pattern of one erasure or two erasures and 80% of the three erasure patterns can be corrected by both BCJR and the decoding technique in [8]. However, for a received erasure pattern with four erasures, such as $[? \; ? \; ? \; 0 \; 0 \; 0 \; ?]$, the technique of [8] would decide this GC node is "uncorrectable" but BCJR decoders can infer that the second bit must be a 0 given that there are only two codewords of the $(7, 4)$ Hamming code that has three 0s in the 4th, 5th, and 6th positions ($[0 \; 0 \; 0 \; 0 \; 0 \; 0 \; 0]$ and $[1 \; 0 \; 1 \; 0 \; 0 \; 0 \; 1]$), both of which have the second bit equal to 0. We refer to this capability of the GC node as *single bit decodable*.

Although BCJR component decoding can not correct the whole block (at this time), this improved error correcting capability can increase the thresholds of designed GLDPC code ensembles via the iterative procedure. For this component code, we calculate that the fraction of single bit decodable $d_{\min} + 1$ erasure patterns is 0.75 (BCJR can not partially or completely decode any erasure pattern beyond $d_{\min} + 1$ erasures). Following [8], we incorporate this extra decoding capability of the BCJR algorithm compared to the technique of [8] into the asymptotic analysis by using a probabilistic description of the peeling decoder in which, every time a degree $(d_{\min}+1)$ GC node is created, then it is tagged as single bit decodable with probability 0.75. This description becomes accurate for large block lengths, as long as we assume code bits are labeled at random at every GC node, and it allows one to incorporate the BCJR decoding model into the differential equation method. A similar probabilistic description of BCJR decoding can be suitably obtained for any GC node (code) based on correctable or partially correctable error patterns and incorporated into the analysis of [8].

In the following, we note the modifications that must be made to incoporate the improved decoding capability of BCJR decoding to the method of [8].[3] In order to characterize the DD evolution of the residual Tanner graph of $(2, 7)$-regular GLDPC codes with BCJR component decoding, we first introduce the notations needed for the differential equation

[1]The DD of an LDPC (or a GLDPC) code ensemble is characterized by polynomials $\lambda(x)$ and $\rho(x)$, where the coefficients of the polynomials correspond to the fraction of edges in the graph connected to nodes of a certain degree, see [21].

[2]For a BEC, we implement BCJR by exact bit marginalization (bit-wise MAP decoding).

[3]Due to space constraints, we refer the reader to [8] for full details of the analysis.

method. We say any edge connected to a degree i variable node has *left degree* i, $i \in \{1, \ldots, J\}$, where J is the maximum degree of any variable node in the Tanner graph. Similarly, any edge connected to a degree j SPC (resp. GC) node has *right SPC (resp. GC) degree* j, $j \in \{1, \ldots, K\}$, where K is the maximum degree of any check node in the Tanner graph. Let $L_i^{(\ell)}$ represent the number of edges that have left degree i at iteration ℓ, let $R_{pj}^{(\ell)}$ represent the number of edges have right SPC degree j, and let $R_{cj}^{(\ell)}$ represent the number of edges with right GC degree j at iteration ℓ. For $j = d_{\min}$ (respectively $d_{\min}+1$), we split $R_{cj}^{(\ell)}$ into two terms, $\hat{R}_{cj}^{(\ell)}$ and $\bar{R}_{cj}^{(\ell)}$, where $\hat{R}_{cj}^{(\ell)}$ denotes the number of edges with right GC degree j connected to GC nodes tagged as *decodable* (resp. single bit decodable), and $\bar{R}_{cj}^{(\ell)}$ denotes the number of edges with right GC degree j connected to GC nodes tagged as *not-decodable* (both values of j). Clearly, we have $R_{cj}^{(\ell)} = \hat{R}_{cj}^{(\ell)} + \bar{R}_{cj}^{(\ell)}$. The main difference to the method in [8] is that, when the decoder selects, at random, a degree-4 check node, there is 0% probability this check node can be decoded with the decoding technique in [8]; however, even though the BCJR decoder cannot decode the GC node either, we now have a 75% probability that one of the edges will be removed from the residual graph (correcting the variable node adjacent to the edge).[4] By structure of the $(7, 4, 3)$ Hamming code if, during an iteration, a not-decodable degree 4 check node has one edge removed (via some other decodable check node), it becomes a decodable degree 3 check node for the next iteration, however, if a single bit decodable degree 4 check node has one edge removed, we compute that there is a probability of 75% that it becomes a decodable degree 3 check node.

D. Threshold Results for Optimized GLDPC Code Ensembles

To compare the performance of the decoding technique in [8] and BCJR decoding of GC nodes, we plot the threshold vs. coding rate of $(2, 6)$-regular GLDPC and $(2, 7)$-regular GLDPC code ensembles as a function of the fraction ν of GC nodes in Fig. 2, where the channel capacity is indicated by the black line.[5] For each code ensemble, the leftmost point is the original $(2, 6)$-regular LDPC or $(2, 7)$-regular LDPC code. As the proportion ν of GC nodes increases, the thresholds of the designed GLDPC codes increase and the coding rate decreases. When ν is small, the improved thresholds do not compensate for the rate loss, which causes the additive gap to capacity to be worse (larger) than the original LDPC codes. As ν increases, the gap to capacity ("Gap" in Fig. 2) decreases, but the thresholds of the BCJR and the decoding technique in [8] are approximately the same because there are too few GC nodes that can combine during the iterative process to collectively correct block errors. As the proportion ν of the GC nodes becomes sufficiently large the thresholds obtained by component BCJR decoding exceed the thresholds obtained by [8] and approach capacity. Note that the gap to capacity is greatly reduced. The optimal proportion of GC nodes was determined to be $\nu^* = 0.75$ for both $(2, 6)$-regular GLDPC and $(2, 7)$-regular GLDPC code ensembles.[6] As ν increases beyond ν^*, the additional error correcting capability again does not compensate for the rate loss and the gap to capacity increases, but the improvement over the decoding technique in [8] is maintained.

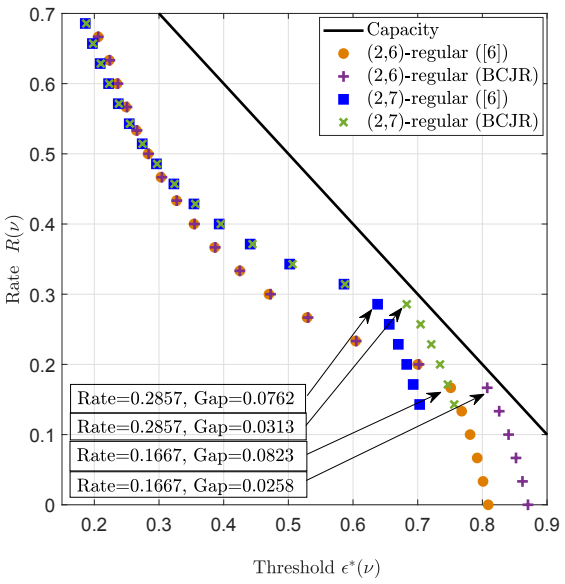

Fig. 2. Threshold ϵ^* vs. coding rate R as a function of the proportion ν of GC nodes for (J, K)-regular GLDPC codes with the decoding technique in [8] and BCJR decoding at GC nodes.

III. GLDPC Codes with Component Hamming Code Trellises

In order to adopt BCJR decoding at the GC nodes for an additive white gaussian noise (AWGN) channel, we require an appropriate trellis representation of the component code.

A. A Trellis-Oriented Generator Matrix

The BCJR algorithm enables MAP decoding of codes defined on trellises [14]. This decoding technique was extended to linear block codes in [10]. Forney introduced another construction of a trellis for linear block codes in [15] and showed that it is most convenient to construct trellises from a generator matrix in "trellis-oriented" form. This was subsequently shown to minimize the number of vertices [16], and as such, the Forney trellis is also called the *minimal trellis* of the code and the corresponding generator matrix is said to have *minimal-span* [17].

We transform the generator matrix of Hamming codes to be trellis-oriented by performing elementary row operations in

[4]We remind the reader that, in PD, as edges are removed the "degree" of the check node reduces. This is equivalent to more inputs being known. For example, a GC node with 7 inputs, 4 of which corresponds to erasures, would be referred to as a degree 4 GC node.

[5]The probabilistic description for the decoding technique in [8] and BCJR component decoding of shortened $(6, 3)$ Hamming codes was obtained in a similar way to the $(7, 4)$ Hamming codes.

[6]We remind the reader that this technique can similarly be applied to determine ν^* for other code ensembles and component codes.

order to minimize the complexity of a trellis given particular linear block code, measured as the number of branches per section [15]. To determine the minimal-span matrix, we begin with the definitions and notations of *index* and *span* that are used to draw the trellis of linear block codes. Define a nonzero binary n-tuple $\mathbf{x} = [x_1, x_2, \ldots, x_n]$. Let $L(\mathbf{x})$ denote the *left index* of $\mathbf{x}$, the smallest integer i for $x_i \neq 0$, and let $R(\mathbf{x})$ denote the *right index* of $\mathbf{x}$, the largest integer i for $x_i \neq 0$. Let *span*$(\mathbf{x})$ denote the *span* of $\mathbf{x}$, *span*$(\mathbf{x}) = [L(\mathbf{x}), L(\mathbf{x}) + 1, \ldots, R(\mathbf{x})]$. The corresponding *span-length* of $\mathbf{x}$ is computed as the number of elements in *span*$(\mathbf{x})$. A nonzero vector $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ is *active* at *coordinate* i if $L(\mathbf{x}) < i$ and $R(\mathbf{x}) > i$. With these definitions in hand, we can now define the minimal-span matrix such that the span-length of every row in the matrix has the possible smallest value.

Example 1: We demonstrate our approach using the $(7, 4)$ Hamming code with generator matrix (1) as the component codes in a GLDPC code ensemble. The span of row 1 is *span*$(\mathbf{g}_1) = [1, 2, 3]$, with $L(\mathbf{g}_1) = 1$, $R(\mathbf{g}_1) = 3$, and span-length 3. The span-length of the rows of $\mathbf{G}_{74}$ are $3, 5, 5$ and 7, respectively. By applying elementary row operations on (1), the transformed trellis[7]-oriented generator matrix of the $(7, 4)$ Hamming code is

$$\mathbf{G}'_{74} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{2}$$

The span-length of the rows of $\mathbf{G}'_{74}$ are $3, 4, 4$, and 4, respectively. $\mathbf{G}'_{74}$ is a minimal-span matrix. □

Let $\mathbf{G}$ be a minimal-span $k \times n$ generator matrix with rows $\mathbf{x}_1, \ldots, \mathbf{x}_k$ and let $\mathrm{S} = \{\mathrm{S}_1, \ldots, \mathrm{S}_k\}$ represent a set of *row covers* of $\mathbf{G}$. Here, each S_j is a discrete interval $[L_j, L_{j+1}, \ldots, R_j]$ containing the span of $\mathbf{x}_j$. Also let $\lambda(\mathbf{u})$ be the span of the ith column of $\mathbf{G}'_{74}$. For a given set of row covers S, the sets A_i are defined as $A_i = \{j | i \in \mathrm{S}_j\}$, where $i = 1, 2, \ldots, n$. In other words, A_i denotes which rows of $\mathbf{G}$ are "S-active" at coordinate i. Similarly, let B_i represent which rows of $\mathbf{G}$ are "S-active" at depth i, where $B_0 = B_n = \emptyset, B_i = A_i \cap A_{i+1}$, for $i = 1, 2, \ldots, i-1$. The cardinalities of A_i and B_i are denoted by α_i and β_i, respectively. Now let $\mathbf{u} = [u_1, u_2, \ldots, u_k]$ be a binary k-tuple, where $k = 2^{\alpha_i}$. We define *init*$(\mathbf{u})$ and *fin*$(\mathbf{u})$ as the vector of length $|B_{i-1}|$ (respectively $|B_i|$) containing the ordered entries of $\mathbf{u}$ with indices in B_{i-1} (respectively B_i). Given these definitions, to describe the edge sets $E_{i-1,i}$, $i = 1, 2, \ldots, 7$, a sequence of tables can be obtained and pieced together into a graphical representation of the trellis.

Example 1 (cont.): Table I gives the values of A_i, B_i, α_i, and β_i computed for generator matrix $\mathbf{G}'_{74}$ of the $(7, 4)$ Hamming code (2) which is minimal-span. Using the technique proposed in [10], we obtain edge sets $E_{i-1,i}$, for $i = 1, \ldots, 7$, which for completeness and reproducibility of results are given in the sequence of tables in Appendix A.

[7]Any generator matrix can used, but $\mathbf{G}'_{74}$ is desirable since it has minimal-span.

TABLE I
VALUES OF A_i, B_i, α_i, AND β_i FOR THE $(7, 4)$ HAMMING CODE (2)

i	A_i	B_i	α_i	β_i
0	$\emptyset$	$\emptyset$	0	0
1	{1}	{1}	1	1
2	{1,2}	{1,2}	2	2
3	{1,2,3}	{2,3}	3	2
4	{2,3,4}	{2,3,4}	3	3
5	{2,3,4}	{3,4}	3	2
6	{3,4}	{4}	2	1
7	{4}	$\emptyset$	1	0

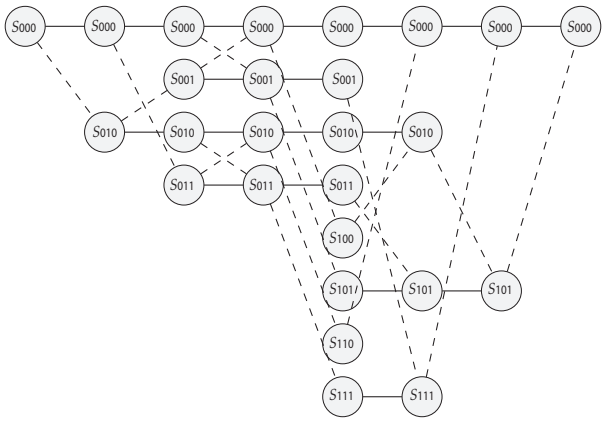

Fig. 3. Trellis of the $(7, 4)$ Hamming code derived from $\mathbf{G}'_{74}$.

The corresponding trellis is shown in Fig. 3, where the paths through the tree correspond to codewords in the component code. Solid and dashes lines correspond to 0 and 1 outputs, respectively. The states S are labeled with subscripts corresponding to the outputs. Based on the trellis, we derive a branch metric, forward metric, and backward metric of the $(7, 4)$ Hamming code on AWGN channel under BPSK modulation following the standard BCJR rules [18]. This provides soft a posteriori probability (APP) outputs for the GC nodes in GLDPC codes.

IV. NUMERICAL RESULTS AND DISCUSSION

A. Simulation Results

Using the optimal proportion ν^* of GC nodes that was determined using the asymptotic analysis outlined in Section II, we compare the BER performance on the BPSK modulated AWGN channel with BCJR and decoding technique [8] component decoding of the rate $R = 1/12$ $(2, 6)$-regular and $(2, 7)$-regular GLDPC coding schemes proposed in [9]. The location of GC nodes were selected randomly in the code construction. These rate $R = 1/12$ codes have 40 information bits and block length 480 bits (typical parameters proposed for ultra-reliable low-latency communication (URLLC)). Like [9], we also apply an (79, 40) outer code that corrects up to 15 errors for comparison. From Fig. 4, we observe that the simulation results shows that gains of approximately 2dB at

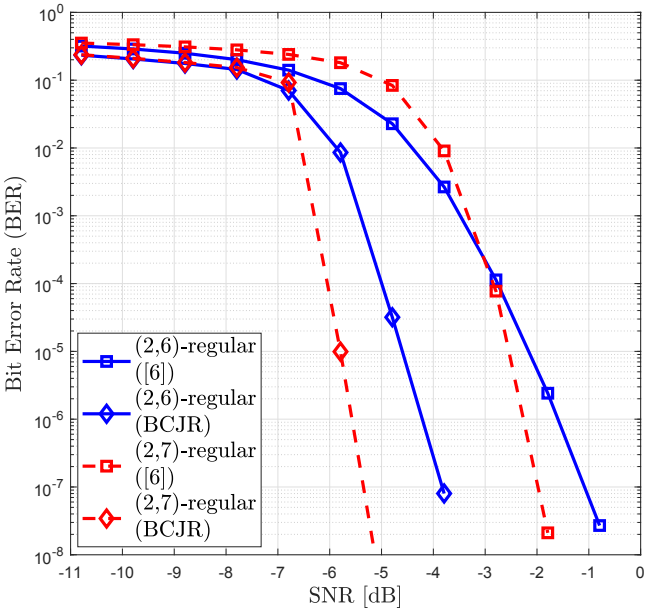

Fig. 4. BER performance of the GLDPC codes has $R = 1/12$, 40 information bits and outer code proposed in [9] on AWGN channel with BPSK modulation and BCJR or decoding technique [8] component decoding of GC nodes.

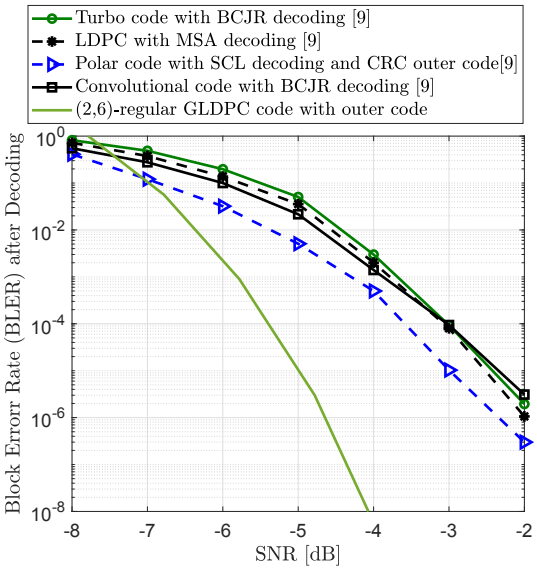

Fig. 5. BLER over an AWGN channel with QPSK modulation for the the $(2, 6)$-regular GLDPC code with outer code correct up to 15 errors and different decoding algorithms.

a BER of 10^{-6} when using BCJR for component GC nodes decoding of the (2,6)-regular GLDPC when compared to the decoding technique [8] component decoded code. For the case of the (2,7)-regular GLDPC code, approximately 3dB is gained at a BER of 10^{-6}.

Fig. 5 shows the block error rate (BLER) decoding performance of some 5G URLLC candidates under QPSK modulation [12], [13], including turbo codes with BCJR decoding, LDPC codes with MSA decoding and offset MSA decoding, polar codes with successive cancellation list (SCL) decoding and a CRC outer code, and convolutional codes with BCJR decoding. Also shown is the BLER performance of the (2,6)-regular GLDPC code with BCJR component decoding, optimized proportion ν of GC nodes, and the outer code to rate match. We observe that approximately 2dB is gained at a BER of 10^{-6} compared to the polar codes with SCL decoding and a CRC outer code, at the cost of some additional decoding complexity.

B. Decoding Complexity

Component BCJR decoding of the GC nodes requires computation of the branch metric, forward metric, backward metric, and log-likelihood ratio (LLR) values. The decoding complexity will also depend on the trellis realization. The numbers given in this section are determined for the minimal-span matrices of shortened $(6, 3)$ Hamming code and $(7, 4)$ Hamming code, respectively. Following a standard BCJR implementation [11], we require $4K$ operations to determine the branch metric for a $(2, K)$-regular GLDPC code. The computation of the forward metric and the backward metric for a $(2, 6)$-regular GLDPC code both require 25 multiplications and 7 additions, and the computation of the LLR values for variable node requires 66 additions/subtractions. Hence, to update every GC node for a $(2, 6)$-regular GLDPC code requires $4K + 32 \times 2 + 66 = 134$ additions/subtractions (or $4K + 48 \times 2 + 101 = 225$ additions/subtractions for a $(2, 7)$-regular GLDPC code). There are also $K+2$ exponential and K log computations for a $(2, K)$-regular code. In total, the BCJR component decoding complexity for the $(2, 6)$-regular GLDPC code simulated in Section IV-A is 21567 additions/subtractions + 14 exponential/log operations (or 26420 additions/subtractions + 16 exponential/log operations for the $(2, 7)$-regular GLDPC code), whereas the others' decoding complexity is 16287 and 37520 additions/subtractions, respectively [9]. Therefore, BCJR component decoding can be seen to have the same order of complexity as [8] for these codes, yet results in significantly improved performance.

V. Conclusion

In this paper, we presented an asymptotic analysis of GLDPC code ensembles with component BCJR decoding on the BEC and determined the optimal proportion of GC nodes to minimize the threshold gap to capacity. The obtained thresholds were shown to be close to capacity and significantly improved over the component decoding method of [8]. We then designed the minimal-span trellis for component BCJR decoding of Hamming codes and constructed GLDPC codes suitable for URLLC. Simulation results were shown to confirm the expected performance improvement from the asymptotic analysis over the component decoding method of [8] with similar overall decoding complexity. The results also show

competitive performance versus alternative schemes proposed for URLLC.

VI. Acknowledgment

This work has been funded in part by the National Science Foundation under Grant Nos. ECCS-1710920 and OIA-1757207.This work has been funded in part by the Spanish Government under Grant TEC2016-78434-C3-3-R, in part by Comunidad de Madrid under Grants IND2017/TIC-7618, IND2018/TIC-9649, and Y2018/TCS-4705, and in part by the European Research Council (ERC) through the European Union Horizon 2020 research and innovation program under Grant 714161.

Appendix

The sequence of tables used to generate the trellis of the $(7, 4)$ Hamming code are given as follows:

$E_{0,1}$

$\mathbf{u}$	*init*$(\mathbf{u})$	*fin*$(\mathbf{u})$	$\lambda(\mathbf{u})$
$[0]$	$\emptyset$	$[0]$	$[0]$
$[1]$	$\emptyset$	$[1]$	$[1]$

$E_{1,2}$

$\mathbf{u}$	*init*$(\mathbf{u})$	*fin*$(\mathbf{u})$	$\lambda(\mathbf{u})$
$[0, 0]$	$[0]$	$[0, 0]$	$[0]$
$[0, 1]$	$[0]$	$[0, 1]$	$[1]$
$[1, 0]$	$[1]$	$[1, 0]$	$[1]$
$[1, 1]$	$[1]$	$[1, 1]$	$[0]$

$E_{2,3}$

$\mathbf{u}$	*init*$(\mathbf{u})$	*fin*$(\mathbf{u})$	$\lambda(\mathbf{u})$
$[0, 0, 0]$	$[0, 0]$	$[0, 0]$	$[0]$
$[0, 0, 1]$	$[0, 0]$	$[0, 1]$	$[1]$
$[0, 1, 0]$	$[0, 1]$	$[1, 0]$	$[1]$
$[0, 1, 1]$	$[0, 1]$	$[1, 1]$	$[0]$
$[1, 0, 0]$	$[1, 0]$	$[0, 0]$	$[1]$
$[1, 0, 1]$	$[1, 0]$	$[0, 1]$	$[0]$
$[1, 1, 0]$	$[1, 1]$	$[1, 0]$	$[0]$
$[1, 1, 1]$	$[1, 1]$	$[1, 1]$	$[1]$

$E_{4,5}$

$\mathbf{u}$	*init*$(\mathbf{u})$	*fin*$(\mathbf{u})$	$\lambda(\mathbf{u})$
$[0, 0, 0]$	$[0, 0, 0]$	$[0, 0]$	$[0]$
$[0, 0, 1]$	$[0, 0, 1]$	$[0, 1]$	$[1]$
$[0, 1, 0]$	$[0, 1, 0]$	$[1, 0]$	$[1]$
$[0, 1, 1]$	$[0, 1, 1]$	$[1, 1]$	$[0]$
$[1, 0, 0]$	$[1, 0, 0]$	$[0, 0]$	$[1]$
$[1, 0, 1]$	$[1, 0, 1]$	$[0, 1]$	$[0]$
$[1, 1, 0]$	$[1, 1, 0]$	$[1, 0]$	$[0]$
$[1, 1, 1]$	$[1, 1, 1]$	$[1, 1]$	$[1]$

$E_{5,6}$

$\mathbf{u}$	*init*$(\mathbf{u})$	*fin*$(\mathbf{u})$	$\lambda(\mathbf{u})$
$[0, 0]$	$[0, 0]$	$[0]$	$[0]$
$[0, 1]$	$[0, 1]$	$[1]$	$[1]$
$[1, 0]$	$[1, 0]$	$[0]$	$[1]$
$[1, 1]$	$[1, 1]$	$[1]$	$[0]$

$E_{6,7}$

$\mathbf{u}$	*init*$(\mathbf{u})$	*fin*$(\mathbf{u})$	$\lambda(\mathbf{u})$
$[0]$	$[0]$	$\emptyset$	$[0]$
$[1]$	$[1]$	$\emptyset$	$[1]$

$E_{3,4}$

$\mathbf{u}$	*init*$(\mathbf{u})$	*fin*$(\mathbf{u})$	$\lambda(\mathbf{u})$
$[0, 0, 0]$	$[0, 0]$	$[0, 0, 0]$	$[0]$
$[0, 0, 1]$	$[0, 0]$	$[0, 0, 1]$	$[1]$
$[0, 1, 0]$	$[0, 1]$	$[0, 1, 0]$	$[0]$
$[0, 1, 1]$	$[0, 1]$	$[0, 1, 1]$	$[1]$
$[1, 0, 0]$	$[1, 0]$	$[1, 0, 0]$	$[1]$
$[1, 0, 1]$	$[1, 0]$	$[1, 0, 1]$	$[0]$
$[1, 1, 0]$	$[1, 1]$	$[1, 1, 0]$	$[1]$
$[1, 1, 1]$	$[1, 1]$	$[1, 1, 1]$	$[0]$

References

[1] R. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inf. Theory*, vol. 27, no. 5, pp. 533–547, 1981.

[2] N. Miladinovic and M. Fossorier, "Generalized LDPC codes and generalized stopping sets," *IEEE Trans. Comm.*, vol. 56, no. 2, Feb. 2008.

[3] G. Liva, W. Ryan, and M. Chiani, "Quasi-cyclic generalized LDPC codes with low error floors," *IEEE Trans. Comm.*, vol. 56, no. 1, pp. 49–57, Feb. 2008.

[4] M.U. Farooq, M. Saeedeh, L. Michael, "Generalized LDPC Codes with Convolutional Code Constraints" *IEEE Int. Sym. on Inf. Theory (ISIT)*, Los Angeles, USA, June 2020, pp. 479-484.

[5] E. Paolini, M. P. C. Fossorier and M. Chiani, "Generalized and doubly generalized LDPC codes with random component codes for the binary erasure channel," *IEEE Trans. Inf. Theory*, vol. 56, no. 4, pp. 1651–1672, Apr. 2010.

[6] M.F. Flanagan, E. Paolini, M. Chiani, M.P. C. Fossorier, "On the growth rate of the weight distribution of irregular doubly generalized LDPC codes," *IEEE Trans. Inf. Theory*, vol. 57, no. 6, pp.3721–3737, 2011.

[7] D. Mitchell, M. Lentmaier, and D. J. Costello, "On the minimum distance of generalized spatially coupled LDPC codes," in *Proc. IEEE Int. Symp. on Inf. Theory*, Istanbul, Turkey, July 2013, pp. 1874–1878.

[8] Y. Liu, P. M. Olmos, and T. Koch, "A probabilistic peeling decoder to efficiently analyze generalized LDPC codes over the BEC," *IEEE Trans. Inf. Theory*, vol. 65, no. 8, pp. 4831–4853, 2019.

[9] Y. Liu, P. M. Olmos, and D. G. Mitchell, "Generalized LDPC Codes for Ultra Reliable Low Latency Communication in 5G and Beyond," *IEEE Access*, vol. 6, pp. 72 002–72 014, 2018.

[10] R. J. McEliece, "On the BCJR trellis for linear block codes," *IEEE Trans. Inf. Theory*, vol. 42, no. 4, pp. 1072–1092, 1996.

[11] A. Banerjee. (2016) BCJR on AWGN. [Online]. Available: https://nptel.ac.in/courses/117104120/9

[12] Gamage, Heshani and Rajatheva, Nandana and Latva-Aho, Matti, "Channel coding for enhanced mobile broadband communication in 5G systems", *IEEE European conference on networks and communications (EuCNC), Oulu, Finland, June*, pp 1–6, 2017.

[13] Sybis, Michal and Wesolowski, Krzysztof and Jayasinghe, Keeth and Venkatasubramanian, Venkatkumar and Vukadinovic, Vladimir, "Channel coding for ultra-reliable low-latency communication in 5G systems," *IEEE 84th vehicular technology conference (VTC-Fall), Montreal, Canada. Sep.*, pp. 1–5, 2016.

[14] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate (corresp.)," *IEEE Trans. Inf. Theory*, vol. 20, no. 2, pp. 284–287, 1974.

[15] G. D. Forney, "Coset codes. II. Binary lattices and related codes," *IEEE Trans. Inf. Theory*, vol. 34, no. 5, pp. 1152–1187, 1988.

[16] D. J. Muder, "Minimal trellises for block codes," *IEEE Trans. Inf. Theory*, vol. 34, no. 5, pp. 1049–1053, 1988.

[17] F. R. Kschischang and V. Sorokine, "On the trellis structure of block codes," *IEEE Trans. Inf. Theory*, vol. 41, no. 6, pp. 1924–1937, 1995.

[18] S. Lin and D. J. Costello, *Error control coding,* Second Edition, Prentice Hall, 2001.

[19] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Efficient erasure correcting codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 569–584, 2001.

[20] Nicholas C. Wormald, "Differential equations for random processes and random graphs," *Annals of Applied Probability*, vol. 5, no. 4, pp. 1217–1235, 1995.

[21] Luby, Michael G., Michael Mitzenmacher, Mohammad Amin Shokrollahi, and Daniel A. Spielman. "Improved low-density parity-check codes using irregular graphs." *IEEE Trans. on Inf. Theory*, vol. 47, no. 2, pp. 585-598, 2001.