A Rate-Compatible Puncturing Scheme for Finite-Length LDPC Codes

Reza Asvadi, *Member, IEEE,* and Amir H. Banihashemi, *Senior Member, IEEE*

unlocked

***Abstract*—In this paper, we propose a rate-compatible puncturing scheme for finite-length low-density parity-check (LDPC) codes over the additive white Gaussian noise (AWGN) channel. The proposed method is applicable to any LDPC mother code, both regular and irregular, and constructs punctured codes which perform well in both the waterfall and the error floor regions for a wide range of code rates. The scheme selects code bits to be punctured one at a time and based on a sequence of criteria. An important selection criterion is the number of short cycles with low approximate cycle extrinsic message degree (ACE) in which a candidate bit node participates. Simulation results demonstrate that the ACE measure, which is most often the determining criterion in the final selection of the puncturing candidates, plays an important role in improving the performance of the codes in both the waterfall and the error-floor regions. These results also demonstrate that the proposed scheme is superior to the existing puncturing methods, particularly when a wide range of code rates is desirable.**

***Index Terms*—Low-density parity-check (LDPC) codes, rate-compatible LDPC codes, rate-compatible puncturing, adaptive rate coding, finite-length LDPC codes, hybrid ARQ schemes, error floor, approximate cycle extrinsic message degree (ACE).**

I. Introduction

CHANNEL coding schemes with multiple code rates have many applications such as wireless communications over time-varying channels where the lower rate codes are used for poor channel conditions. *Rate-compatibility* is a desirable property for such coding schemes which allows a single pair of encoder/decoder to be used for the whole set of codes. A common simple technique for constructing rate-compatible (RC) codes is *puncturing*, where the higher rate codes are obtained by puncturing the parity bits of a low-rate code, referred to as the *mother code*. The puncturing is performed in a rate-compatible fashion where the set of punctured bits of each code is the subset of the punctured bits of the next higher rate code in the sequence. In this work, we are interested in designing rate-compatible low-density parity-check (RC-LDPC) codes using puncturing.

The design of RC-LDPC codes has been an active area of research [1]–[6]. In particular, rate-compatible puncturing schemes for *finite-length* LDPC codes have been designed in [2]–[5]. In [2], the concept of *recovery tree* $T(v)$ for a bit node v is introduced. This tree is the shallowest tree subgraph[1] of the code's Tanner graph (TG) whose root is in v, and whose leaves are all unpunctured bit nodes.[2] In addition, the subgraph must satisfy the extra condition that all the neighboring bit nodes of every check node in the subgraph are also part of the subgraph. By definition, the recovery tree is the tree subgraph with the above properties that has the smallest number of unpunctured nodes. If two nodes w_1 and w_2 are connected by an edge in $T(v)$ and w_1 is closer to v than w_2, we call w_1 the *parent* of w_2, and w_2 a *child* of w_1. If the depth of $T(v)$ is two, i.e., if there exists a check node neighbor of v whose children in $T(v)$ are all unpunctured nodes, following the terminology of [2], we refer to v as being one-step-recoverable (1-SR). Similarly, node v is called m-SR if the depth of $T(v)$ is $2m$. The puncturing scheme of [2] is to first puncture as many 1-SR bits as possible followed by as many 2-SR bits as possible and so on. To decide between m-SR bits with the same value of m, the priority is given to the bit with the smallest number of unpunctured nodes in its recovery tree. In [3], the puncturing scheme of [2] was modified and improvements in performance were reported for RC-LDPC codes obtained from two irregular mother codes. A puncturing scheme similar to that of [2] was also proposed in [4], where the punctured bits were selected far apart in the code's TG to reduce the performance degradation. Ideas similar to those in [2] and [3] were also used in [5] to develop a puncturing algorithm, referred to as *progressive node puncturing (PNP)*, for protograph-based LDPC codes.

In this paper, we propose a scheme that punctures the code bits one at a time and through a two-phase process. In the first phase, potential candidates for puncturing are identified. These are the code bits that are neighbors to check nodes with the minimum number of adjacent punctured bit nodes, and in addition are likely to provide reliable information to their bit-node neighbors in earlier stages of iterative decoding. In the second phase, a bit node is selected from the candidate set obtained in phase 1 using a hierarchy of criteria. The final, and often determining, criterion in the second phase is to select a bit node that participates in the smallest number of short cycles with low *approximate cycle extrinsic message degree (ACE)* [7]. The proposed algorithm can be applied to any parity-check matrix (any TG) of an arbitrary LDPC mother code as it does not assume any special structure for the code or the parity check matrix.

Manuscript received August 8, 2012. The associate editor coordinating the review of this letter and approving it for publication was L. Dolecek.

R. Asvadi is with the Center for Wireless Communications (CWC), Department of Communications Engineering, University of Oulu, P.O. Box 4500, FI-90014, Oulu, Finland (e-mail: rasvadi@ee.oulu.fi).

A. H. Banihashemi is with the Department of Systems and Computer Engineering, Carleton University, Ottawa, ON K1S 5B6, Canada (e-mail: ahashemi@sce.carleton.ca).

Digital Object Identifier 10.1109/LCOMM.2012.112812.121785

[1]The depth of a tree is defined at the length of the longest path between the root and the leaves of the tree.

[2]It should be noted that in the context of [2], as well as in that of this paper, a recovery tree may not be in fact a tree as defined in graph theory, i.e., a recovery tree may contain multiple copies of the same node, which implies the existence of cycles in the graph theoretical sense. By the construction of the recovery tree, however, as we grow the tree, we include all the neighboring nodes of a check node except for its parent, or a check node neighbor of a punctured variable node other than its parent, in the next layer of the tree. This is performed without worrying about the possibility that copies of the same nodes may have already appeared in the tree. These multiple copies of the same node are treated as different nodes and thus the term "recovery *tree*."

II. LDPC Codes and Tanner Graph Properties Related to Puncturing

Consider a binary LDPC code $\mathcal{C}$ represented by a Tanner graph $G = (V_b \cup V_c, E)$, where $V_b = \{v_1, \ldots, v_n\}$ and $V_c = \{c_1, \ldots, c_m\}$ are the sets of variable nodes and check nodes, respectively, and E is the set of edges. In the Tanner graph G, two nodes v_j and c_i are *adjacent* if $\{v_j, c_i\} \in E$. The set consisting of all nodes that are adjacent to node w is called w's *neighbors*, and is denoted by $\mathcal{N}(w)$. Two variable nodes are said to be neighbors if they have a common adjacent check node. The set of all neighboring variable nodes of a variable node v_j is called the *border* of v_j and is denoted by $\mathcal{B}(v_j) = \{v : \mathcal{N}(v) \cap \mathcal{N}(v_j) \neq \emptyset\}$.

We use $d(w)$ to denote the degree of a node w. For each cycle ξ of G, the *approximate cycle extrinsic message degree* (ACE) $ACE(\xi)$ is calculated by $\sum_{v \in \xi}(d(v) - 2)$, where the summation is over all the variable nodes v in ξ [7]. The ACE is a simple metric for measuring the connectivity of a cycle to the rest of the graph, and implies how harmful the cycle is to the iterative decoding process, i.e., the smaller the ACE, the more harmful the cycle.

There are a number of puncturing schemes for finite-length codes in the literature that outperform random puncturing [1]–[5]. These schemes, that are collectively referred to as *intentional puncturing schemes*, select the punctured bits based on a deterministic algorithm that aims at optimizing a certain cost function reflecting the performance of the punctured codes. Our goal in this work is also to devise an intentional puncturing scheme. For this, in the following, we discuss some properties of the TG of the mother code related to the performance metrics that we use in our design. Some of these metrics have already been used in the existing designs. In such cases, we stay loyal to the original notation.

At any stage of the puncturing process, we denote the punctured subset of V_b by P. We use the notation $F(c)$ to denote the number of punctured variable nodes adjacent to the check node c, i.e., $F(c) = |\{v : v \in \mathcal{N}(c) \cap P\}|$. For an unpunctured variable node v, we use the notation $H(v)$ to denote the number of punctured nodes at the border of v, i.e., $H(v) = |\mathcal{B}(v) \cap P|$. Both metrics $F(c)$ and $H(v)$ are used in [5] as part of the criteria for puncturing. Clearly, the larger the value of $F(c)$, the smaller the chances that check node c can provide reliable information to its adjacent neighbors in earlier stages of iterative decoding. Similarly, the larger the value of $H(v)$, the smaller the likelihood that node v receives reliable information from its neighboring check nodes.

Similar to the definition of a recovery tree for a punctured variable node given in [2], we define the *recovery tree* $T(c)$ *of a check node* c as follows: Consider the shallowest tree subgraph of the mother code's TG G that is rooted at c and has unpunctured variable nodes as its leaves. The tree must also satisfy the condition that if a check node belongs to the tree, so must do all its variable node neighbors. Fig. 1 is an example of such a tree. The next step is to prune this tree at its punctured variable nodes, whenever possible, to obtain the smallest tree subgraph of G rooted at c with unpunctured variable nodes at the leaves. The resulting subgraph is $T(c)$. In Fig. 2, the process of pruning is explained for the tree of

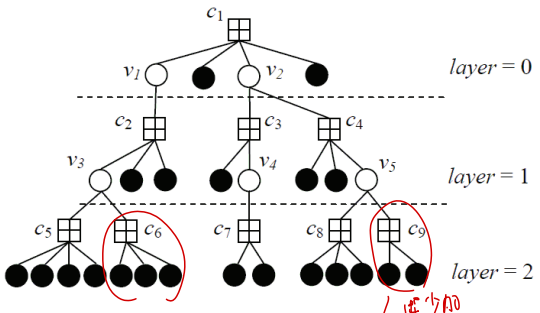

Fig. 1. An example of the construction of the initial tree used to obtain the recovery tree of a check node: ⊞ = a check node, ◯ = a punctured node, and ● = an unpunctured node.

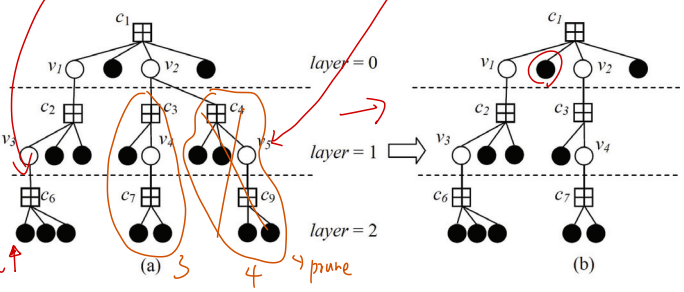

Fig. 2. Pruning procedure is performed at two stages on the tree of Fig. 1 to obtain the recovery tree $T(c_1)$.

Fig. 1 in two steps. In the first step, the branches at layer 2 of the tree connected to nodes v_3 and v_5 are pruned. This is followed by pruning the branches at layer 1 connected to node v_2. The resulting graph is $T(c_1)$.[3]

We use the notation $U(c)$ to denote the number of unpunctured variable nodes in $T(c)$. As an example, for the $T(c_1)$ of Fig. 2, we have $U(c_1) = 10$. Intuitively, the smaller the value of $U(c)$, the higher the reliability of messages that c can pass to its neighboring variable nodes. An analytical result related to this statement was also proved in [2], where it was shown that the smaller the number of unpunctured variable nodes in the recovery tree of a punctured node, the lower the recovery error probability of the node.

Notation $K(v)$ is used to denote the sum of the number of unpunctured variable nodes in the recovery trees $T(c)$ of all the neighboring check nodes c of variable node v, i.e., $K(v) = \sum_{c \in \mathcal{N}(v)} U(c)$. A smaller value of $K(v)$ implies that more reliable messages would be received by v through its neighboring check nodes.

Short cycles with low ACE are known to deteriorate the performance of iterative decoding algorithms [7]. It is thus desirable to avoid puncturing the variable nodes of such cycles as this will further weakens the overall reliability of such structures, making them even more vulnerable to noise and errors. To evaluate the extent in which different variable nodes participate in short cycles with low ACE values, we define *ACE score spectrum* of a variable node as follows: Let $C(\ell, \eta)$ denote the set of cycles of length ℓ with ACE value η. We define $C_{(ACE \leq \eta_{max})}(\ell)$ as the set of cycles in the union of

[3]The concept of check node recovery tree and the pruning process described above have also been used in [3] as part of their puncturing design process. The application of these concepts however is different in our design approach compared to that of [3].

sets $C(\ell,\eta)$ with $\eta \le \eta_{max}$. We then define the *ACE score* $S^v_{\eta_{max}}(\ell)$ of a variable node v corresponding to cycles of length ℓ with ACE values $\eta \le \eta_{max}$ as the number of cycles in $C_{(ACE\le\eta_{max})}(\ell)$ in which node v participates. The ACE score spectrum $\mathbf{S}^v_{\eta_{max}}(\ell_{max})$ of depth ℓ_{max} of node v corresponding to cycles with ACE values $\eta \le \eta_{max}$ is then defined as the $\ell_{max}/2$-tuple $(S^v_{\eta_{max}}(2), S^v_{\eta_{max}}(4), \ldots, S^v_{\eta_{max}}(\ell_{max}))$.

III. The Proposed Puncturing Algorithm

We consider a systematic LDPC mother code with rate r_0 and block length n_0. We only puncture the parity bits, denoted by V_b^P, to obtain the sequence of RC-LDPC codes with rates $r_m > \cdots > r_1 > r_0$.[4] The required number of punctured nodes, N_k, to obtain the code rate $r_k, k = 1, \ldots, m$, is given by

$$N_k = \lfloor \frac{n_0(r_k - r_0)}{r_k} \rfloor. \quad (1)$$

The proposed puncturing scheme starts by puncturing N_1 parity bits, one at a time, to obtain the first code with rate r_1 in the sequence. It then continues by puncturing $N_2 - N_1$ more parity bits, for a total of N_2 punctured bits, to obtain the second code with rate r_2 in the sequence. This process continues until N_m parity bits for the code with rate r_m are punctured. To select each punctured bit, the algorithm goes through two phases. In phase one, among all check nodes with minimum $F(c)$, check nodes with lowest $U(c)$ are selected, and are denoted by $CandidateCheck$. Then, as the candidate set for puncturing, all the unpunctured variable nodes which are neighbors to $CandidateCheck$ are selected. These variable nodes are denoted by $CandidateVar$. We thus have $CandidateVar = \{v : v \in \mathcal{N}(c), \forall c \in CandidateCheck\} \cap V_b^P \setminus P$.

In the phase two of the algorithm, a parity bit v is selected from the set $CandidateVar$ based on the following hierarchy of criteria: i) minimizing $H(v)$, ii) minimizing $d(v)$, iii) minimizing $K(v)$. If there are more than one unpunctured parity bits with the same minimum values of $H(v)$, $d(v)$, and $K(v)$, for each such parity bit v, we find the ACE score spectrum $\mathbf{S}^v_{\eta_{max}}(\ell_{max})$, for given values of η_{max} and ℓ_{max}, and select the bit that has the smallest ACE score $S^v_{\eta_{max}}(4)$. If there are more than one bits with the same $S^v_{\eta_{max}}(4)$, we then select the one with the smallest $S^v_{\eta_{max}}(6)$. We continue this process by going through the components of the ACE score spectrum in the increasing order of their indices until a parity bit is selected. If there is a tie at the end of this process, i.e., if there are a number of bits with the same minimal ACE score spectrum, we select one at random. Our simulation results show that by selecting ℓ_{max} to be about 20, the likelihood that we need to select a bit randomly is very low. These results also demonstrate that, for the ACE score spectrum to accurately represent the performance of the code, one should select $\eta_{max} \le 4$.

The details of the proposed puncturing algorithm is given in Algorithm 1, where $\mathcal{C}(r_k), k = 0, \ldots, m$, denotes the code with rate r_k in the sequence. Also, $UPset_k$ and P_k denote the set of unpunctured parity bits of $\mathcal{C}(r_{k-1})$ and the number of punctured variable nodes required to obtain $\mathcal{C}(r_k)$ from $\mathcal{C}(r_{k-1})$, respectively. We also define $Pset_k = V_b^P \setminus UPset_k$ which denotes the punctured nodes of $\mathcal{C}(r_{k-1})$, and thus $|Pset_k| = N_{k-1}$. Moreover, $P_k = N_k - N_{k-1}$, where N_k and N_{k-1} are calculated by (1).

Algorithm 1 Proposed Puncturing Algorithm

Inputs: Parity-check matrix (Tanner graph) of $\mathcal{C}(r_0)$; sequence of rates $r_1, \ldots, r_m$; η_{max}; ℓ_{max}.

1) Initialization: Set $k = 1$.
2) Find the set $UPset_k$, and calculate P_k by (1). Also, for all $c \in V_c$, calculate $F(c)$ and $U(c)$, and set $counter = 0$.
3) Find set $\Psi = \{c^* \in V_c : F(c^*) \le F(c), \forall c \in V_c\}$, and then set $CandidateCheck = \{c^* \in \Psi : U(c^*) \le U(c), \forall c \in \Psi\}$.
4) $CandidateVar = \{v : v \in \mathcal{N}(c), \forall c \in CandidateCheck\} \bigcap UPset_k$.
5) $\Gamma = \{v^* \in CandidateVar : H(v^*) \le H(v), \forall v \in CandidateVar\}$. If $|\Gamma| = 1$, $v_p = \Gamma$ and go to Step 10.
6) $\Delta = \{v^* \in \Gamma : d(v^*) \le d(v), \forall v \in \Gamma\}$. If $|\Delta| = 1$, $v_p = \Delta$ and go to Step 10.
7) $\Theta = \{v^* \in \Delta : K(v^*) \le K(v), \forall v \in \Delta\}$. If $|\Theta| = 1$, $v_p = \Theta$ and go to Step 10.
8) Set $\ell = 4$, and $\Lambda = \Theta$;
   (a) $\Lambda = \{v^* \in \Lambda : S^{v^*}_{\eta_{max}}(\ell) \le S^v_{\eta_{max}}(\ell), \forall v \in \Lambda\}$;
   (b) If $|\Lambda| = 1$, $v_p = \Lambda$ and go to Step 10;
   (c) If $\ell < \ell_{max}$, $\ell = \ell + 2$, go to Step 8(a).
9) Choose v_p at random from Λ.
10) Puncture v_p, set $counter++$, $UPset_k = UPset_k \setminus v_p$, and $Pset_k = Pset_k \bigcup v_p$.
11) If $counter < P_k$, then $\forall c \in \mathcal{N}(v_p)$, set $F(c) = F(c) + 1$, and $\forall c \in \{c : v_p \in T(c)\}$, update $U(c)$. Also, $\forall v \in \mathcal{B}(v_p)$, set $H(v) = H(v) + 1$, and Go to Step 3. Otherwise, if $k < m$, then set $k++$, and go to Step 2.
12) Stop.

IV. Simulation Results

In this section, we present results on puncturing two LDPC mother codes of rate $r_0 = \frac{1}{2}$ and block length $n_0 = 1024$, one regular and the other irregular. Simulations are performed with binary phase shift keying (BPSK) modulation over the AWGN channel. Notations E_b and N_0 are used to denote the average energy per information bit and the one-sided power spectral density of the AWGN. Belief propagation (sum-product) algorithm with the maximum number of iterations 50 is used for decoding, and for each simulation point, 100 block errors are generated.

We first consider a randomly constructed $(3, 6)$-regular LDPC code of girth 6, and use the proposed scheme to obtain a rate-compatible sequence of codes with rates 0.5, 0.6, 0.7 and 0.8. In the proposed algorithm, we use $\ell_{max} = 14$ and $\eta_{max} = 4$. For each of the three rates 0.6, 0.7, and 0.8, the number of punctured bits determined by the four criteria $H(v)$, $d(v)$, $K(v)$ and ACE score spectrum are respectively, $(0, 0, 0, 170)$, $(2, 0, 6, 284)$, and $(6, 0, 44, 334)$. This demonstrates the significant role that the ACE score spectrum plays in the proposed puncturing scheme. Clearly, in this example,

[4] In principle, the proposed puncturing scheme is also applicable to non-systematic codes. We however assume a systematic mother code and only puncture parity bits, similar to [2] and [3], for fair and simple comparisons.

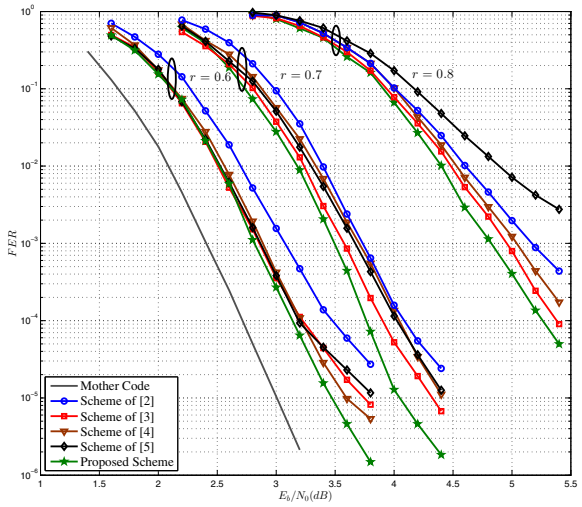

Fig. 3. FER performance of RC-LDPC codes with rates 0.5, 0.6, 0.7 and 0.8, constructed by different puncturing schemes applied to the $(3, 6)$-regular LDPC code with length 1024.

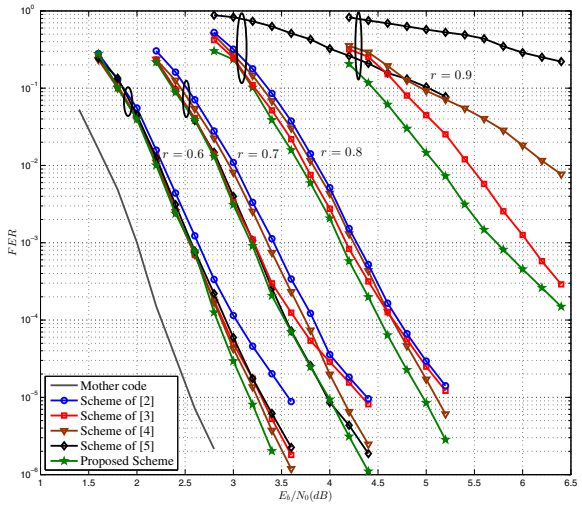

Fig. 4. FER performance of RC-LDPC codes constructed by the application of different puncturing schemes to the irregular LDPC code with length 1024.

since the mother code is regular, no node is punctured based on the $d(v)$ criterion.

The frame error rate (FER) curves of the punctured codes are shown in Figure 3. In the same figure, the FER curves for the puncturing schemes of [2]–[5] are provided for comparison.[5] As can be seen, at all punctured code rates, the proposed algorithm performs at least as good as the best of the four existing algorithms in the waterfall region, and outperforms all four schemes in the error-floor region. Bit error rate (BER) curves for all the schemes are similar to those of FER in terms of the relative performance, and are thus omitted.

[5]To implement the puncturing scheme of [5], we incorporate criterion C in the PNP algorithm, as described in [5]. Also, for each rate, we repeat the algorithm 1000 times with different random seeds, and select the puncturing pattern whose Tanner graph has the maximum puncturing score.

As the second example, we consider a mother code that is constructed by the progressive-edge-growth (PEG) algorithm [8] based on an optimized degree distribution taken from [3][Code 1 of Table II]: $\lambda(x) = 0.27236x + 0.23771x^2 + 0.07091x^3 + 0.41902x^9$, and $\rho(x) = 0.66x^6 + 0.34x^7$. The girth of the Tanner graph of the code is 6. The proposed method with $\eta_{max} = 2$ and $\ell_{max} = 22$ is applied to the mother code to obtain RC-LDPC codes at rates 0.6, 0.7, 0.8, and 0.9. In the Tanner graph of the mother code, all cycles of length $\ell \leq 12$ have ACE values larger than 2, and thus in the ACE score spectrum only the components corresponding to cycles with length $\ell \geq 14$ are nonzero. For each of the four rates 0.6, 0.7, 0.8, and 0.9, the number of punctured bits determined by the four criteria $H(v)$, $d(v)$, $K(v)$ and ACE score spectrum are respectively, $(0, 0, 1, 169)$, $(2, 1, 6, 283)$, $(2, 1, 28, 353)$, and $(6, 1, 68, 380)$. These results show that, similar to the regular case, for all rates, a large fraction of the punctured nodes are determined by the ACE score spectrum. FER results of the punctured codes are shown in Figure 4, along with the FER curves for the puncturing schemes of [2]–[5]. (Note that rate 0.9 is not achievable by the method of [2].) The comparison between these results demonstrates the superior performance of the proposed construction, particularly at higher rates. BER curves provide a similar conclusion and are thus omitted.

ACKNOWLEDGMENT

The authors wish to thank Mehdi Karimi for providing them with the software for finding low-ACE cycles.

REFERENCES

[1] M. R. Yazdani and A. H. Banihashemi, "On construction of rate-compatible low-density parity-check codes," *IEEE Commun. Lett.*, vol. 8, pp. 159–161, Mar. 2004.

[2] J. Ha, J. Kim, D. Klinc, and S. W. McLaughlin, "Rate-compatible punctured low-density parity-check codes with short block lengths," *IEEE Trans. Inf. Theory*, vol. 52, pp. 728–738, Feb. 2006.

[3] H. Y. Park, J. W. Kang, K. S. Kim, and K. C. Whang, "Efficient puncturing method for rate-compatible low-density parity-check codes," *IEEE Trans. Wireless Commun.*, vol. 6, pp. 3914–3919, Nov. 2007.

[4] B. N. Vellambi and F. Fekri, "Finite length rate-compatible LDPC codes: a novel puncturing scheme," *IEEE Trans. Commun.*, vol. 57, pp. 297–301, Feb. 2009.

[5] M. El-khamy, J. Hou, and N. Bhushan, "Design of rate-compatible structured LDPC codes for hybrid ARQ applications," *IEEE J. Sel. Areas Commun.*, vol. 27, no. 6, pp. 965–973, Aug. 2009.

[6] H. Saeedi, H. Pishro-Nik, and A. H. Banihashemi, "Successive maximization for the systematic design of universally capacity approaching rate-compatible sequences of LDPC code ensembles over binary-input output-symmetric memoriless channels," *IEEE Trans. Commun.*, vol. 59, pp. 1807–1819, July 2011.

[7] T. Tian, C. Jones, J. D. Villasenor, and R. D. Wesel, "Selective avoidance of cycles in irregular LDPC code construction," *IEEE Trans. Commun.*, vol. 52, pp. 1242–1248, Aug. 2004.

[8] 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.