A Puncturing Scheme for Low-Density Parity-Check Codes Based on 1-SR Nodes

Lijun Zhang
School of Electr. and Inform. Eng.
Beijing Jiaotong Univ.
Beijing, 100044, China
Email: ljzhang@bjtu.edu.cn

Fuli Ma
Center for Space Science and Applied Research
Chinese Academy of Sciences
Beijing, 100190, China
Email: mafuli007@nssc.ac.cn

L. L. Cheng
Dept. of Electronic Engineering
City Univ. of Hong Kong
Hong Kong, China
Email: itacheng@cityu.edu.hk

***Abstract*—A rate-compatible puncturing scheme for LDPC codes is proposed based on the fact that a one-step recoverable (1-SR) node is more reliable than a k-SR node ($k > 1$) and a 1-SR node with more survived check nodes can be recovered more reliably in a recovery tree, so the scheme tries to obtain 1-SR nodes which have multiple survived check nodes as many as possible. Simulation results show that the puncturing scheme has better performance than both the random puncturing scheme and the novel method in [5].**

***Index Terms*—LDPC codes; rate-compatible; puncturing; 1-SR node.**

I. Introduction

In time-varying channels with channel state information (CSI), multiple rates coding scheme is preferred to a single rate coding scheme. Rate-compatible (RC) is defined as a family of punctured codes derived from a single mother code, and the punctured bits in a lower rate code are also punctured in a higher rate code [1]. RC puncturing coding can avoid introducing several different encoder-decoder pairs and be preferred in type-II hybrid automatic-repeat-request (HARQ) protocols, which can increase the overall throughput in the presence of impairments [2]. The performance of a punctured code is determined by the puncturing pattern which distributes the locations of the punctured bits. The punctured bits should be chosen carefully to form the best puncturing pattern, thus minimizing the gap between the dedicated and punctured codes.

The design and analysis of puncturing for LDPC codes have been extensively studied [3–5]. In [3, 4], Ha *et al.* proposed to puncture finite-length LDPC codes based on their classification of punctured nodes into k-step recoverable (k-SR) nodes, where k is a positive integer. The authors claimed that, on the average, the magnitudes of the average log-likelihood ratios (LLRs) for k-SR nodes decrease with increasing k. Based on this claim, they presented a greedy algorithm selecting the bits to be punctured. The main disadvantage of this approach is the greedy selection of bits to be punctured. By this mode of selection, the reliability of the $(k + 1)$-SR nodes in some sense critically depends on the reliability of the i-SR nodes for $i \leq k$. Vellambi *et al.* proposed a scheme based on the observation that the performance degradation can be mitigated if the punctured bits are far apart from each other in the Tanner graph of the code [5]. Later in [6], the number of survived check nodes (SCNs) of a k-SR node was considered based on the fact that a k-SR node with more SCNs can be recovered more reliably. However, it can be applied only to the LDPC code with a dual-diagonal block structure. El-Khamy *et al.* proposed a puncturing scheme named progressive node puncturing (PNP) [2]. In the traditional puncturing schemes, the low rate codes in the RC family may perform well, but the higher rate codes will suffer from error floors. The PNP algorithm carefully assigns the puncturing pattern such that the further puncturing of the code can produce good high rate codes, but it is only applicable to some special LDPC codes and not general.

An ideal puncturing pattern should build such a Tanner graph where the number of 1-SR nodes is maximized and all of them have as many SCNs as possible. Based on 1-SR nodes and the aforementioned ideas, a new puncturing scheme for LDPC codes is proposed. In the scheme, two key factors, namely, the number of 1-SR nodes and the number of SCNs, are considered simultaneously. It is promising to obtain the potentially ideal puncturing pattern with the method.

II. Basic Concepts

An LDPC code is a linear block code given by the null space of an $m \times n$ sparse matrix $\mathbf{H}$, termed as parity-check matrix. If $\mathbf{H}$ has constant column weight g and row weight r, where $r = gn/m$ and $g \ll m$, the code is a regular LDPC code; otherwise, an irregular LDPC code. The component '1' in $\mathbf{H}$ is represented by a connection or an edge between a check node and a variable node in the corresponding bipartite graph where no edges connect two nodes of the same type. Such a graph is also called a Tanner graph [7]. A parity-check matrix or the corresponding Tanner graph can uniquely define an LDPC code.

For RC puncturing LDPC codes, a punctured node will be recovered correctly with a reliable message if it has more neighboring check nodes and each of the check nodes has reliable neighbors. A punctured variable node is called one-step recoverable (1-SR), if it has at least one check node neighbor whose remaining variable node neighbors are all unpunctured [4]. Such check nodes are called survived check nodes, namely, SCNs. Once a punctured variable node receives a non-zero LLR message from its neighbors, the node has an opportunity to be recovered, no matter the LLR value is correct or not. Obviously, a 1-SR variable node is guaranteed to be recovered after the first iteration. Similarly, a k-SR variable node can be recovered in the k-th iteration. As shown in Fig. 1, variable node v_1 is 1-SR, since it can be recovered from the SCN c_1 in the first iteration. Once v_1 is recovered, it makes the v_2 recoverable by its SCN c_2 in the second iteration, so v_2 is called a 2-SR node.

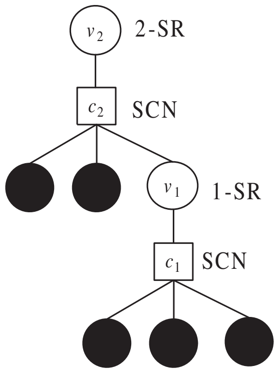

Fig. 1. Recovery tree of a 2-SR node. The squares for SCN, the filled circles for unpunctured variable nodes, and the unfilled circles for punctured variable nodes.

III. Puncturing Algorithm

A. Analysis of multiple SCNs

1) Over binary erasure channel: Suppose a punctured LDPC code is transmitted over BEC with an erasure probability of $p < 1$. As shown in Fig. 2, the number of SCNs for the punctured node v_1 is t, then v_1 can be recovered from any c_i, where $1 \leq i \leq t$. The probability of v_1 recovered from c_i is

$$(1-p)^{r-1}.$$

If v_1 receives non-zero information from c_i, $1 \leq i \leq t$, it is said to be recoverable. So the total recovered probability is formulated as

$$P_r(v_1) = 1 - [1-(1-p)^{r-1}]^t. \tag{1}$$

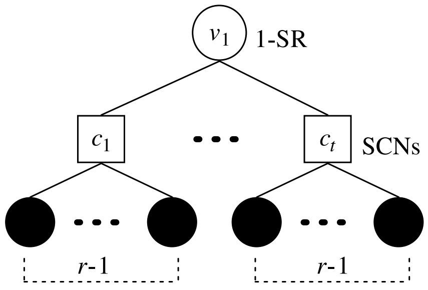

Fig. 2. One punctured node with multiple SCNs. The squares for SCN, the filled circles for unpunctured variable nodes, and the unfilled circles for punctured variable nodes.

Due to the fact $1-(1-p)^{r-1} < 1$, we can see that as the number of SCNs t increases, $[1-(1-p)^{r-1}]^t$ will decrease, and the recovered probability will become larger. The more SCNs, the more likely the punctured node can be recovered.

2) Over binary-input AWGN (BI-AWGN) channel: Suppose the LDPC codes are decoded by sum-product algorithm (SPA) [8]. In the decoding process, the message updates include two stages. In stage one,

$$w = u_0 + \sum_{j=1}^{g-1} u_j, \tag{2}$$

where u_j for $j = 1, 2, \cdots, g-1$, are the incoming LLRs from the $g-1$ neighbors of a variable node, w is the message sent to the remaining neighbor, and the LLR message from the channel is denoted by u_0.

In stage two,

$$\tanh\frac{u}{2} = \prod_{i=1}^{r-1} \tanh\frac{w_i}{2}, \tag{3}$$

where w_i for $i = 1, 2, \cdots, r-1$, are the incoming LLRs from the $r-1$ neighbors of a check node, and u is the message sent to the remaining neighbor.

In order to decode the punctured LDPC codes, the only modification in the SPA is to initialize the LLRs of the punctured nodes with zero, i.e., u_0 is zero in (2). If this node receives non-zero LLR from a neighbor, it will send non-zero LLRs to other neighbors. The necessary condition for a punctured node to be recovered is that it must be a neighbor of at least a check node who can relay non-zero LLR message to the punctured node.

With density evolution (DE) using Gaussian approximation, the mean LLR values from variable nodes can be represented as [9]

$$\gamma_w^{(l)} = \gamma_{u_0} + (g-1)\gamma_u^{(l-1)}, \tag{4}$$

where l represents the l-th iteration, γ_{u_0}, γ_u and γ_w are means of messages u_0, u and w, respectively.

Based on the derivation of the previous research in [4], suppose that from 1-SR to k-SR nodes in a recovery tree of variable v, all the punctured nodes have only one SCN. Let c_i be a survived check node, then the mean LLR received by c_i is

$$\gamma_{u,i} = \phi^{-1}(1 - [1-\phi(\gamma_{u_0})]^{S(c_i)}), \tag{5}$$

where $S(c_i)$ denotes the number of unpunctured nodes under c_i in the recovery tree of v and $\phi(x)$ is defined as [9]

$$\phi(x) = \begin{cases} 1 - \frac{1}{\sqrt{4\pi x}} \cdot \\ \int_{-\infty}^{\infty} \tanh\frac{t}{2} \exp\left(-\frac{(t-x)^2}{4x}\right) \mathrm{d}t, & \text{for } x > 0 \\ 1, & \text{for } x = 0 \end{cases}, \tag{6}$$

The LLR value which a variable node v receives is the sum of the incoming LLR values from all the SCNs of v, i.e.,

$$\gamma_u(v) = \sum_{i=1}^{|N_{sc}(v)|} \gamma_{u,i}, \tag{7}$$

where $N_{sc}(v)$ is the set of all the neighboring SCNs of v.

In BI-AWGN channel, assuming the independent identical distribution and symmetry condition, the error probability of a node v is [4]

$$P_e(v) = Q\left(\frac{\gamma_u(v)}{\sqrt{\sigma^2(v)}}\right) = Q\left(\sqrt{\frac{\gamma_u(v)}{2}}\right), \tag{8}$$

where $\sigma^2(v)$ is the variance of LLR value from node v. Because Q-function is monotonic decrease, (7) and (8) suggest that multiple SCNs of a punctured node v make the LLR value sent from v bigger and evolve faster than that with a single SCN. Moreover, the error probability of a punctured node v can be decreased by multiple SCNs.

Based on the above discussion, it can be seen that in both BEC and BI-AWGN channels, the recovered probability will increase with the number of SCNs. In the following puncturing algorithm, the number of SCNs is taken as an important factor.

B. Algorithm description

Puncturing of the mother code can be performed either randomly or according to carefully chosen locations of punctured bits optimized for a given parity-check matrix. The random puncturing manner is easy to implement, but the performance of the codes is poor always. So a puncturing pattern should be designed carefully to mitigate the performance degradation due to rate increment.

The puncturing problem can be formulated as: for a given code C_0 with rate R_0, its child codes $C_1, C_2, \cdots, C_k$ have code rates $R_1, R_2, \cdots, R_k$, respectively, and $R_1 \leq R_2 \leq \cdots \leq R_k$. Assume the length of the mother code is N, in order to get the target code with code rate R_t,

$$\tau_t = \left\lfloor \frac{N(R_t - R_0)}{R_t} \right\rfloor \tag{9}$$

nodes in total should be punctured.

In the forthcoming algorithm, the number of 1-SR nodes with multiple SCNs will be maximized. Notations are listed below for making expressions concise.

- $Remain\_c$: the set of check nodes which neighbor no punctured nodes,
- $Candidate$: the set of variable nodes from which the punctured nodes can be chosen,
- All_sc: the set of all SCNs,
- P_v_sc: the set of SCNs for punctured node v,
- $Index$: the set of all the possible punctured nodes, and the first τ_t nodes are punctured to get the target rate R_t,
- $Fixed$: the set of unpunctured nodes,
- d_{v_p}: the number of SCNs for variable node v,
- v_{v_p}: the number of SCNs that a punctured node v neighbors,
- $Candidate\_c$: the set of SCNs for the variable node who has the maximum d_{v_p}.
- $N(x)$: the set of neighbors of node x.

For a given $m \times n$ parity-check matrix $\mathbf{H}$, the algorithm can be described as follows.

Algorithm 1 Proposed

```
1:  initialize $Remain\_c = \{c_1, c_2, \cdots, c_m\}$, $All\_sc = \emptyset$, $Fixed = \emptyset$, and $Index = \emptyset$;
2:  $Candidate = \bigcup_{x \in Remain\_c} N(x) \backslash \bigcup_{y \in All\_sc} N(y)$;
3:  choose $v_1$ at random from $Candidate$, set $Index = Index \bigcup \{v_1\}$, $All\_sc = All\_sc \bigcup P_{v_1}\_sc$, and $Remain\_c = Remain\_c \backslash All\_sc$;
4:  if $Candidate == \emptyset$ then
5:      continue;
6:  else
7:      go back to line 2;
8:  end if
9:  if $Remain\_c \neq \emptyset$ then
10:     continue;
11: else
12:     go to line 21;
13: end if
14: $Candidate = \bigcup_{x \in Remain\_c} N(x) \backslash Fixed$;
15: choose one variable node $v$ at random from $Candidate$ with the minimum $d_{v\_p}$, set $Index = Index \bigcup \{v\}$, and $All\_sc = All\_sc \bigcup P_v\_sc$;
16: if the check node $c$ is the only SCN of punctured node $v$ then
17:     set $Fixed = Fixed \bigcup N(c)$, $Remain\_c = Remain\_c \backslash All\_sc$, and go back to line 9;
18: else
19:     continue;
20: end if
21: add the SCNs of the punctured nodes which have the maximum $v_{v\_p}$ into $Candidate\_c$, and update $Candidate = \bigcup_{x \in Candidate\_c} N(x) \backslash Fixed$;
22: choose one variable node $v$ at random from $Candidate$ with the minimum $d_{v\_p}$;
23: set $Index = Index \bigcup \{v\}$, and $All\_sc = All\_sc \bigcup P_v\_sc$;
24: if $|Fixed| == n$ then
25:     stop;
26: else
27:     go back to line 21;
28: end if
```

The algorithm receives a parity-check matrix of an LDPC code and outputs the set of the punctured nodes. At the beginning, we initialize the $Remain\_c$ with all the rows and set All_sc to empty set. There are multiple variable nodes with almost the same attributes in $Candidate$, and they cannot be distinguished by only one criterion. For simplicity, we select one variable node v at random from $Candidate$ as the punctured node and include all of its SCNs in All_sc and P_v_sc. Line 1 will continue until $\bigcup_{x\in All\_sc} N(x)$ contains all variable nodes.

Lines 9 through 13 check if $Remain\_c$ is empty. If it is not empty, there must be some check nodes that neighbor no punctured nodes. Then we can find more 1-SR nodes on these check nodes. Before the line 14 is involved, the number of SCN of all the punctured nodes in $Index$ equals the column weight of $\mathbf{H}$ matrix.

Lines 14 through 20 will choose the variable node that has the minimum d_{v_p}, which means the new selected punctured node will affect the least number of SCNs in All_sc, so there will be more 1-SR nodes. The number of SCNs for the punctured nodes is also checked. If there are some punctured nodes which neighbor only one SCN, then the neighbors of the only SCN will be added into $Fixed$ to ensure each punctured node has at least one SCN.

When lines 21 through 23 are executed, it means all check nodes neighbor at least one punctured node. When a new punctured node is chosen, it will inevitably affect the number of SCNs which belong to those punctured nodes previously chosen. In order to minimize the effect and balance the number of SCNs for each punctured node, the punctured nodes with the most SCNs are used to update $Candidate\_c$, and then $Candidate$. The next new variable nodes with minimum d_{v_p} will be selected from $Candidate$ for puncturing.

C. An example

To explain the algorithm more clearly, an example is given below.

Shown in Fig. 3 is a Tanner graph, which illustrates the connection of variable nodes and check nodes. The puncturing scheme includes three steps.

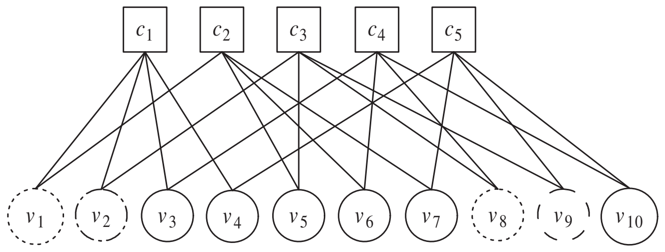

Fig. 3. The puncturing order of variable nodes, variable nodes with dotted border for the punctured nodes in the fist step, dashed border for nodes in the second step and dot-dashed border for nodes in the third step.

In the first step, v_1 and v_8 are chosen as the punctured nodes, as they have the same number of SCNs which equals the column weight of $\mathbf{H}$ matrix. Their SCNs are added into both All_sc and P_v_sc, where $v = v_1,\ v_8$. If one more variable node is punctured, the number of SCNs for v_1 and v_8 will decrease. According to line 4 in the algorithm, the first step ends.

At the beginning of the second step, the check nodes which neighbor no punctured nodes are added into $Remain\_c$. Then the variable node in $Candidate$ which neighbors the least number of check nodes in All_sc will be chosen as the next punctured node. As shown in the figure, v_9 is selected. Because v_8 and v_9 have only one SCN each, namely, c_4 and c_5, the variable nodes these two SCNs neighbor will be added into $Fixed$, i.e., v_3, v_4, v_6, v_7 and v_{10}. Once the $Remain\_c$ becomes empty, each check node connects at least one punctured node, and the second step ends.

During the third step, only v_2 and v_5 can be chosen as the punctured nodes, because all the other variable nodes are in $Fixed$. Suppose v_2 is chosen as the next punctured node, v_5 will be included in $Fixed$, for the existence of SCN c_2. Now, all the unpunctured nodes belong to $Fixed$, and the puncturing algorithm ends.

As to the complexity, it should be noted that the complexity of our algorithm is a little higher than that in [5] because of more comparison operations. We do not decide to do further analysis for the complexity, as the puncturing procedure runs once and operates offline. Once the optimal puncturing pattern is determined, the puncturing algorithm will no longer be in action, and has nothing to do with encoding and decoding.

IV. Simulation Results

Arbitrary rates can be achieved from a low rate mother code via puncturing, but the performance will degrade significantly when the rate approaches one. In [3], authors evaluated the performance of several punctured LDPC codes and optimized the puncturing pattern to get the best performance. However, their simulation results show that for high rate LDPC codes the performance degrades. The phenomenon has been explained by the threshold effect of punctured codes discussed in [10].

The algorithm in [4] is a very classic RC puncturing algorithm proposed in 2006. Based on this algorithm, many new RC puncturing schemes which have better performance are proposed, and the algorithm in [5] is typical which will be called *novel* in the sequel. We compare the performance of punctured codes constructed by the proposed puncturing algorithm with a random puncturing scheme and the *novel* scheme over AWGN channel at various code rates. The bit error rate (BER) of each code is evaluated by observing 100 erroneous words at each E_b/N_0 using binary phase shift keying (BPSK) modulation. The standard SPA is used for decoding, and the maximum number of iteration is set to 50.

Two 1/2-rate (3, 6) regular LDPC codes, one constructed by progressive edge-growth (PEG) algorithm with length 1008 [11], and the other constructed by MacKay with length 816 [12], are used as the mother codes, respectively. By puncturing variable nodes, the punctured codes with rates 0.6, 0.65, 0.7 and 0.75 are obtained.

As shown in Fig. 4 and Fig. 5, the proposed algorithm outperforms the random scheme significantly, and also performs slightly better than the *novel* scheme at rate 0.6. In Fig. 4, at BER of 10^{-5}, the proposed scheme for PEG codes can get coding gains of 0.1 dB and 0.4 dB in comparison with the random scheme and the *novel* scheme, respectively. Similar behavior can be observed in Fig. 5. The corresponding coding gains for MacKay codes are 0.2 dB and 0.5 dB, respectively.

The advantage of the proposed scheme becomes more obvious with the increase of the coding rates. At rate 0.75, the proposed scheme exceeds the *novel* scheme by 0.6 dB and 0.3 dB for PEG code and MacKay code, respectively.

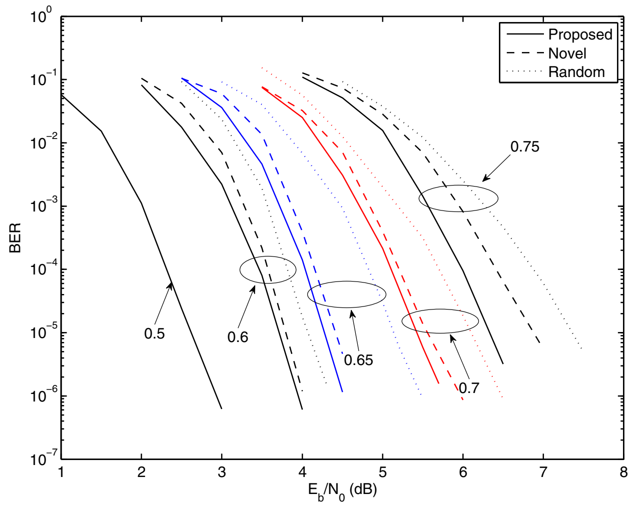

Fig. 4. The BER performance of the PEG codes with rates 0.6, 0.65, 0.7, and 0.75 by the proposed scheme, random scheme, and the *novel* scheme.

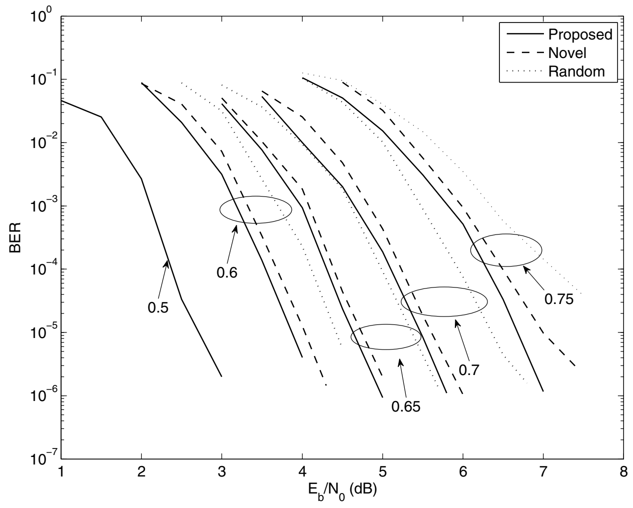

Fig. 5. The BER performance of the MacKay codes with rates 0.6, 0.65, 0.7, and 0.75 by the proposed scheme, random scheme, and the *novel* scheme.

At low rates, the proposed algorithm ensures a large number of k-SR nodes for small k. At high rates, the new punctured nodes inevitably destroy the existed SCNs of those previous punctured nodes. In order to minimize the effect of the new punctured nodes, as mentioned above, the proposed algorithm chooses those variable nodes for puncturing who connect to the previously punctured nodes via maximal number of the existed SCNs, and tries to make an even distribution of SCNs for the punctured nodes. The efficiency of the two measures is verified by Table I. It can be seen that not only the number of 1-SR nodes but also the number of SCNs of the proposed algorithm have advantages over those of the *novel* scheme.

TABLE I
THE STATISTICAL RESULTS FOR THE NUMBER OF 1-SR NODES AND THE AVERAGE NUMBER OF SCNS IN THE PROPOSED (P) SCHEME AND THE *novel* (*N*) SCHEME .

schemes and rates	(1008, 504) code[1]			(816, 408) code[2]		
	τ_t	1-SR	SCNs	τ_t	1-SR	SCNs
P 0.6	168	168	2.62	136	136	2.58
N 0.6	168	168	2.30	136	136	2.27
P 0.65	232	220	1.68	188	170	1.70
N 0.65	232	210	1.50	188	165	1.53
P 0.7	288	205	1.11	233	161	1.12
N 0.7	288	199	1.01	233	159	1.03
P 0.75	336	199	0.80	272	155	0.78
N 0.75	336	188	0.70	272	146	0.65

[1] PEG code.
[2] MacKay code.

It should be noted that the maximal achievable rate for the proposed puncturing algorithm is a little lower than that for the *novel* scheme. In above example, for the mother code of rate 0.5, the maximum code rate obtained is nearly 0.9 with the proposed scheme, while the corresponding rate is 0.92 or even higher with the *novel* scheme. Furthermore, there should exist some tradeoff between puncturing range and error performance, but we are still working on the relationship.

V. Conclusion

1-SR nodes and their SCNs play important roles in the RC LDPC codes. Theoretical analysis shows that more k-SR nodes for low k and multiple SCNs can effectively help to reduce the error probability. The new scheme based on the maximization of 1-SR nodes and the corresponding SCNs can get more 1-SR nodes and SCNs than the *novel* scheme in [5] can do, and gain a clear advantage over those existed schemes in a wide range of coding rates in AWGN channel.

Acknowledgment

This work is supported by the fundamental research funds for the central universities.

References

[1] J. Li and K. Narayanan, "Rate-compatible low density parity check codes for capacity-approaching ARQ scheme in packet data communications," in *Proc. IEEE International Conference on Communications (ICC'02)*, Virgin Islands, Nov. 2002, pp. 201–206.

[2] 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.

[3] J. Ha and S. W. McLaughlin, "Optimal puncturing distribution for rate compatible low density parity check code,"

in *Proc. IEEE International Symposium on Information Theory (ISIT'03)*, Sep. 2003, p. 233.

[4] J. Ha *et al.*, "Rate-compatible punctured low-density parity-check codes with short block lengths," *IEEE Trans. Inf. Theory*, vol. 52, no. 2, pp. 728–738, Feb. 2006.

[5] B. N. Vellambi and F. Fekri, "Finite-length rate-compatible LDPC codes: a novel puncturing scheme," *IEEE Trans. Commun.*, vol. 57, no. 2, pp. 297–301, Feb. 2009.

[6] E. Choi, S. Suh, and J. Kim, "Rate-compatible puncturing for low-density parity-check codes with dual-diagonal parity structure," in *Proc. IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'05)*, Sep. 2005, pp. 2642–2646.

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

[8] R. G. Gallager, "Low density parity check codes," *IRE Trans. Inform. Theory*, vol. IT-18, pp. 21–28, Jan. 1962.

[9] 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 Trans. Inf. Theory*, vol. 47, no. 2, pp. 657–670, Feb. 2001.

[10] H. Pishro-Nik and F. Fekri, "Results on punctured low-density parity-check codes and improved iterative decoding techniques," *IEEE Trans. Inf. Theory*, vol. 53, no. 2, pp. 599–614, Feb. 2007.

[11] X. Y. Hu, E. Eleftheriou, and D. M. Arnold, "Regular and irregular progressive edge-growth Tanner graphs," *IEEE Trans. Inf. Theory*, vol. 51, no. 1, pp. 386–398, Jan. 2005.

[12] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electron. Lett.*, vol. 32, no. 18, pp. 1645–1646, Aug. 1996.