Done

A Non-Greedy Puncturing Method for Rate-Compatible LDPC Codes

Lin Zhou, Wei-Cheng Huang, Rui Zhao, and Yu-Cheng He

Abstract: **In this paper, we propose a non-greedy puncturing method for constructing binary rate-compatible low-density parity-check codes. First, we show that the recovery error probability for a punctured variable node can be reduced by allocating a small number of unpunctured variable nodes in the recovery tree. Then, we redefine the recovery tree according to the principle of iterative decoding with the sum-product algorithm. The proposed non-greedy puncturing algorithm lies mainly in the dynamical minimization of the number of unpunctured variable nodes in the redefined recovery tree for the punctured variable nodes. Finally, simulation results show that the proposed puncturing algorithm outperforms the existing best puncturing algorithms in bit error rate performances of iterative decoding.**

Index Terms: **Low-density parity-check codes (LDPC), puncturing, rate-compatible.**

I. INTRODUCTION

RATE design for channel coding is indispensable to rate-adaptive communications over time-varying wireless channels [1]. Generally stated, low-rate codes can guarantee the transmission reliability under poor channel conditions, whereas high-rate codes can improve the transmission efficiency under good channel conditions. In conventional rate-adaptive channel coding schemes, multiple encoder and decoder pairs with different code rates are deployed to match channel conditions in a switching mode, which yields unacceptable implementation costs [2]. Rate-compatible low-density parity-check (LDPC) codes, referred to as RC-LDPC codes [3], can provide a way of low-cost implementation and capacity-approaching performances for practical rate-adaptive communications.

Using rate-compatible techniques, a family of RC-LDPC codes can be constructed based on a mother LDPC code and decoded by only a single decoder for the mother code. Typically, rate-compatible techniques include puncturing [1]–[14], shortening [15], [16], and protograph extension [17]–[20]. Puncturing is the most popular technique for constructing RC-LDPC codes with high-rate codes as subsets of lower-rate codes. From the point of view of codes on graphs, each low-rate code has

2017

a set of punctured variable nodes as a subset of the punctured variable nodes for higher-rate codes.

Recently, puncturing techniques for RC-LDPC codes have attracted much attention over the years. A non-greedy algorithm was proposed in [1] based on the selection of puncturing pattern with the punctured variable nodes keeping enough large distances. However, this method suffers from a limitation on the highest rate. A greedy algorithm was proposed in [2] that partitions the punctured variable nodes into groups of K-step-recoverable (K-SR) nodes in the recovery tree. It was shown that large K values yield less reliable recovery messages at the K-SR nodes. Thus, the greedy algorithm was aimed at achieving the largest group of K-SR nodes with the smallest K value. In [4], the progressive-edge-growth (PEG) algorithm in [21] was employed to construct low-rate codes, while random puncturing was instead employed for high-rate codes. In [7], the pruned recovery tree was constructed using a cost function to measure the importance of variable nodes, and a stopping set check algorithm was incorporated into the puncturing algorithm. In [10], a progressive puncturing algorithm was developed under similar criterions to that in [2]. In [11] and [12], the recoverability of the punctured variable nodes with unavoidable short cycles was improved by introducing approximate cycle extrinsic (ACE) message degrees. In [13], the ACE-based puncturing scheme was further investigated under three criterion: ensuring the recoverability of punctured variable nodes, minimizing the number of completely punctured cycle trapping sets, and minimizing the number of punctured variable nodes in short cycles. In [14], a criterion of absorbing set class was considered instead of the ACE criterion for the puncturing scheme in [12].

VN degree ↑
→收敛快
CN degree ↓
reliable ↑

It is well-known that in iterative decoding of irregular LDPC codes, ☆the variable nodes with high degree can quickly converge to their correct values, and this improves in turn the convergence of the variable nodes with lower degrees due to message passing. On the other hand, ☆the check nodes with low degrees can provide more reliable likelihood messages than those with higher degrees. Therefore, under various criterions of puncturing, it is also expected to puncture the variable nodes with the smallest degrees and maintain a least number of punctured variable nodes involved in any check node. However, to obtain high-rate codes, the variable nodes with higher degrees may have to be punctured, and meanwhile, some check nodes may have to be connected with more punctured variable nodes. A check node is said to be "dead" if it neighbors to one or more unrecoverable punctured variable nodes, otherwise it is said to be "survived".

In this paper, we propose a non-greedy puncturing method for constructing RC-LDPC codes. By non-greedy, we mean that puncturing may not be constrained only by the minimization of recovering iterations and the minimization of degree distribu-

Manuscript received December 1, 2015; approved for publication by Shigeaki Ogose, Division II Editor, October 18, 2016.

This work was supported in part by the grants from the National Natural Science Foundation of China (61302095, 61401165), the Natural Science Foundation of Fujian Province of China (2014J05076, 2014J01243, 2015J01262), and the Science Foundation of National Huaqiao University (12BS219, 13Y0384).

The authors are with the Xiamen Key Laboratory of Mobile Multimedia Communications, National Huaqiao University, Xiamen, China, while Lin Zhou and Yu-Cheng He are also members of the State Key Laboratory of Integrated Services Networks, Xidian University, Xi'an, China, emails: linzhou@hqu.edu.cn, vinson.wc.h@gmail.com, {rzhao, yucheng.he}@hqu.edu.cn.

Digital object identifier: 10.1109/JCN.2017.000006

tions for punctured variable nodes. Instead, we explore a metric of the recovery tree that significantly affects the recovery error probability.

The remainder of the paper is organized as follows. In Section II, we show that the number of the unpunctured variable nodes in the recovery tree plays a key role in the recovery error probability for a punctured variable node. Then, in Section III, we redefine the recovery tree which is fully expanded according to the computation tree and reflects the essence of the iterative sum-product algorithm (SPA). The redefined recovery tree is thus referred to as the *expanded* recovery tree for distinguishing from the conventional one in [2]. In Section IV, we use the number of the unpunctured variable nodes in the expanded recovery tree as the most important scale metric, and present the puncturing algorithm. We focus on minimizing the scale of the expanded recovery tree for each selected punctured variable node while maintaining non-formation of dead check nodes. In Section V, the performances of the proposed puncturing algorithm are evaluated by simulations for both irregular and regular LDPC mother codes. Simulation results show that the proposed puncturing algorithm can achieve coding gains of about 0.2 dB to 0.5 dB for rates 0.6 to 0.8 over the best puncturing methods proposed in [1], [2], and [7]. Finally, we conclude the paper by Section VI.

II. ANALYSIS OF RECOVERY ERROR PROBABILITY

Consider RC-LDPC codes over the additive white Gaussian noise (AWGN) channel with zero-mean and variance σ^2. Suppose that binary phase-shift keying (BPSK) modulation is used with the mapping $x = 1 - 2v$ between the binary code symbols $v \in \{0, 1\}$ and the binary transmit signals $x \in \{+1, -1\}$. For iterative message-passing decoding at the receiver, the initial log-likelihood ratios (LLRs), $u^{(0)}(v) = \ln \frac{\Pr(x=+1|y)}{\Pr(x=-1|y)}$, are computed from the received symbols y at the unpunctured variable nodes v, while they are set to be zero at the punctured variable nodes.

During decoding iterations, the check nodes receive messages from the neighboring variable nodes, and the variable nodes get messages from the neighboring check nodes. A punctured variable node is said to be *recovered* if it can receive a nonzero LLR message from any neighboring check node, regardless of whether the nonzero LLR value implies the correct bit value or not. However, when the punctured variable node is recovered with a wrong message, a *recovery error* event is said to happen.

For a variable node v, let $C(v)$ be the set of the neighbors to v, and $V(c)$ be the set of the neighbors to $c \in C(v)$. In iterations $k > 0$, let $\Gamma_e^{(k)}(c \to v)$ denote the probability that the check node c conveys a wrong message to the variable node v, and $P_c^{(k)}(v \to c)$ denote the probability that v conveys a correct message to c. The relationship between the probabilistic measures can be expressed as [3]

$$\Gamma_e^{(k)}(c \to v) = 1 - \prod_{v' \in V(c)\backslash v} P_c^{(k-1)}(v' \to c), \tag{1}$$

$$P_c^{(k)}(v \to c) = 1 - \prod_{c' \in C(v)\backslash c} \Gamma_e^{(k)}(c' \to v), \tag{2}$$

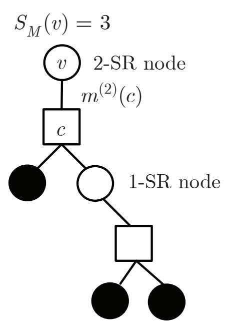


Fig. 1. The recovery tree of a 2-SR node; the squares represent survived check nodes, the black circles represent unpunctured variable nodes, and the empty circles represent punctured variable nodes.

where $P_c^{(0)}(v \to c) = 1/(1 + \exp\{-u^{(0)}(v)\})$, and $V(c)\backslash v$ is a subset of $V(c)$ with the element v excluded.

Suppose that v is a K-SR node for $K \geqslant 0$ as defined in [2]. The recovery error probability of v at iteration K is given by

$$P_e^{(K)}(v) = \prod_{c \in C(v)} \Gamma_e^{(K)}(c \to v). \tag{3}$$

It is known that the initial LLR messages $u^{(0)}(v)$ at the unpunctured variable nodes follow the Gaussian distribution $\mathcal{N}(2/\sigma^2, 4/\sigma^2)$ with mean $m^{(0)}(v) = 2/\sigma^2$. By Gaussian Approximation, for iterations k, the LLR messages updated with SPA at c and v follow $\mathcal{N}\left[m^{(k)}(c), 2m^{(k)}(c)\right]$ and $\mathcal{N}\left[m^{(k)}(v), 2m^{(k)}(v)\right]$, respectively, with means

$$m^{(k)}(c) = \phi^{-1}\left(1 - \left[1 - \phi\left(m^{(k-1)}(v)\right)\right]^{d(c)-1}\right) \tag{4}$$

$$m^{(k)}(v) = m^{(0)}(v) + (d(v) - 1)\, m^{(k)}(c), \tag{5}$$

where $d(v)$ and $d(c)$ denote the degrees of the variable node v and the check node c, respectively; and

$x\uparrow \Rightarrow \phi(x)\downarrow$

$$\phi(x) \triangleq \begin{cases} 1 - \frac{1}{\sqrt{4\pi x}} \int_R \tanh\frac{u}{2} \exp\left\{-\frac{(u-x)^2}{4x}\right\} du, & x > 0 \\ 1, & x = 0. \end{cases} \tag{6}$$

Note that the recovery error of a punctured variable node v, i.e., a K-SR node with $K > 0$, is resulted mainly due to the messages propagated from the neighboring check nodes that contain further punctured variable nodes. When a variable node is punctured, it is usual to specify one of its neighbors as the reserved check node to guarantee its recovery. Let v^* be a specific punctured variable node, and c^* be the correspondingly reserved check node. The recovery tree defined in [2] is spanned by a unique branch from v^* to c^* and extended by the sub-branches from c^* to all the variable nodes in $V(c^*)\backslash v^*$. If there exists one or more punctured variable nodes in $V(c^*)\backslash v^*$, the above spanning process is repeated at each punctured variable node until all the sub-branches are terminated with unpunctured variable nodes.

On the other hand, the propagated messages are essentially based on the information from all the unpunctured variable nodes in the recovery tree of v as shown in Fig. 1. Let $S_M(v)$ denote the number of unpunctured variable nodes in the recovery tree of v. It was shown in [2] that the recovery error probability of v can be approximated to be

$$P_e^{(K)}(v) \approx Q\left(\sqrt{\frac{m^{(K)}(c)}{2}}\right), \tag{7}$$

where

$$m^{(K)}(c) = \phi^{-1}\left(1 - \left[1 - \phi\left(m^{(0)}(v)\right)\right]^{\overset{d(c)-1}{\cancel{S_M(v)}}}\right) \tag{8}$$

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-\frac{t^2}{2}} dt. \tag{9}$$

Furthermore, after a maximum number of $I_{\max}$ iterations, the decoding error probability can be estimated by Gaussian Approximation in terms of bit error rate as

$$P_b(v) = Q\left(\sqrt{\frac{m^{(0)}(v) + d(v)m^{(I_{\max})}(c)}{2}}\right). \tag{10}$$

$d(v)\uparrow \rightarrow P_e^{(k)}(v)\downarrow$ 递减函数 $S_M(v)\uparrow \rightarrow P_e^{(k)}(v)\uparrow$

It is seen that whether the recovery error probability $P_e^{(K)}(v)$ or the decoding error probability $P_b(v)$ is a decreasing function of $d(v)$ by (3) and (10), respectively. More importantly, $P_e^{(K)}(v)$ is also an increasing function of $S_M(v)$ by (8). Notice that the variable nodes with high degrees can be quickly and correctly recovered and then improve the global decoding performance. Thus, we may expect small $S_M(v)$ while maintaining small $d(v)$ for the punctured variable nodes v in order to reduce the recovery error probability and improve the global decoding performance. In next section, we extend the metric to the redefined recovery tree.

III. THE EXPANDED RECOVERY TREE

Following the principle of SPA for iterative decoding, we redefine a recovery tree which is fully spanned by the branches from the root v^* to all the neighbors $c \in C(v^*)$. Then, each branch spanned by c is extended by the sub-branches to all its neighbors except for the root, i.e., $V(c)\backslash v^*$. Likewise, if there exists any punctured variable nodes in a branch, the fully spanning process is repeated until all the sub-branches are terminated with unpunctured variable nodes.

In the sequel, the redefined recovery tree is referred to as the *expanded* recovery tree, whereas the recovery tree defined in [2] is referred to as the *minimum* recovery tree. In the proposed puncturing algorithm below, we also extend the definition of the expanded recovery tree with a check node as root.

The notations and definitions for the codes and the recovery trees are listed in Table 1. In particular, for a punctured variable node v, we use $S_E(v)$ to represent the number of unpunctured variable nodes in the expanded recovery tree, whereas we use $S_M(v)$ to represent the number of unpunctured variable nodes in the minimum recovery tree.

In Fig. 2, we demonstrate the expanded recovery trees, where the empty circles represent the punctured variable nodes, the dark circles represent the unpunctured variable nodes, and the empty squares represent the check nodes. Fig. 2(a) shows the expanded recovery tree for a specific punctured variable node v_1,

Table 1. Notations.

Variable	Definition
V	The set of variable nodes for the mother code, i.e., $V = \{1, \cdots, n\}$.
C	The set of check nodes for the mother code, i.e., $C = \{1, \cdots, m\}$.
V_u	The set of unpunctured variable nodes.
V_p	The set of punctured variable nodes.
C_r	The set of reserved check nodes.
$V(c)$	The set of variable nodes neighboring to $c \in C$. CN旁的 VN set
$V_u(c)$	The set of unpunctured variable nodes in $V(c)$. CN旁的 unpunc VN set
$d(c)$	The degree of check node c, i.e., $d(c) = \lvert V(c)\rvert$.
$d_u(c)$	The number of unpunctured variable nodes in $V(c)$, i.e., $d_u(c) = \lvert V_u(c)\rvert$. CN 旁 unpunc VN 數量
$d_p(c)$	The number of punctured variable nodes in $V(c)$, i.e., $d_p(c) = d(c) - d_u(c)$. CN旁的 punc VN 數量
$C(v)$	The set of check nodes neighboring to $v \in V$. VN旁的 CN set
$d(v)$	The degree of variable node v, i.e., $d(v) = \lvert C(v)\rvert$.
$d_r(v)$	The number of reserved check nodes in $C(v)$, i.e., $d_r(v) = \lvert C(v) \bigcap C_r\rvert$.
$d_u(v)$	The number of unreserved check nodes in $C(v)$, i.e., $d_u(v) = d(v) - d_r(v)$.
$W(c \rightarrow v)$	The number of unpunctured variable nodes in the branch spanned by $c \in C(v)$ from v.
$S(v \rightarrow c)$	The number of unpunctured variable nodes in the branch spanned by $v \in V(c)$ from c.
$W(c)$	The number of unpunctured variable nodes in the recovery tree for the check node c
$S_M(v)$	The number of unpunctured variable nodes in the minimum recovery tree of v, i.e., $S_M(v) = \min_{c\in C(v)} W(c \rightarrow v)$. recovery tree中, degree低的
$S_E(v)$	The number of unpunctured variable nodes in the expanded recovery tree of v, i.e., $S_E(v) = \boxed{\sum_{c\in C(v)} W(c \rightarrow v)}$.

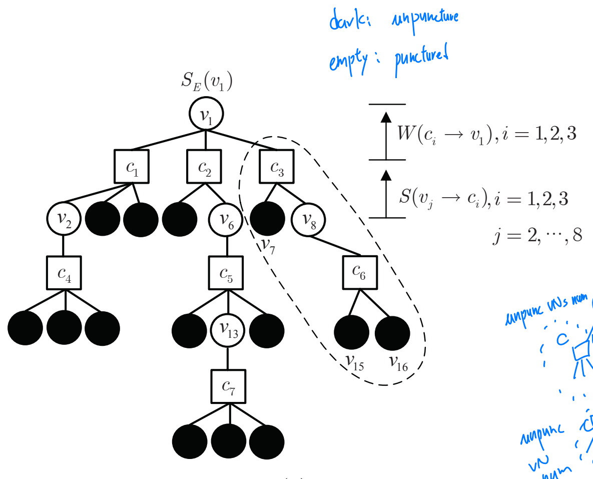
$S_M(v)\uparrow \Rightarrow P_e^{(k)}\uparrow$:代表 1-SR 最好 最 recovery
2-SR ⇒ 更多 unpunc node 也算進來

and Fig. 2(b) shows the expanded recovery tree for the check node c_3. In the figures, c_3 and c_6 are reserved for v_1 and v_8, respectively. Note that in [2], v_1 and v_8 are referred to as a 2-SR node and a 1-SR node, respectively. Clearly, the minimum recovery tree in Fig. 1 just contains only the branch enclosed by the dotted curve in Fig. 2(a). Suppose that the minimum recovery tree were spanned instead by c_1 with the left branch in Fig. 2(a). It would incur a larger metric $S_M(v_1) = 5$. The reservation of c_3 yields the smallest $S_M(v_1)$ for the example 2-SR node v_1.

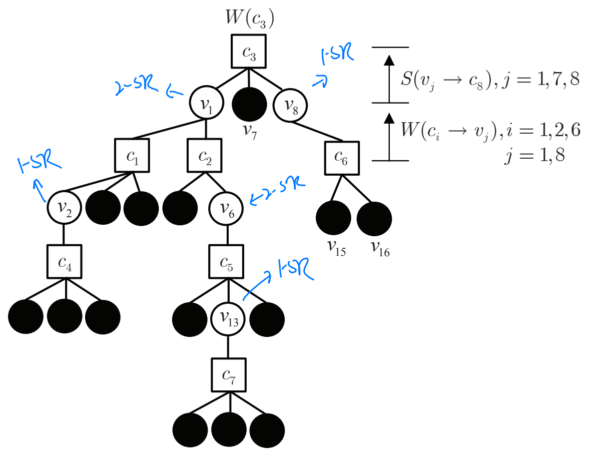
☆As concluded from (5)–(9), the minimum recovery tree represents the most effective routine to recover the root, whereas the expanded recovery tree reflects the essence of iterative decoding based on SPA. Due to the channel noise and the principle of SPA, even if the received message from the smallest branch with the minimal $S_M(v)$ may be unreliable, the messages from other branches may still be reliable to help recovering the root correctly. Consequently, from the point of view of global SPA

W(c): root 為 c 時的 recovery tree 中 unpunc_vn 的數量

dp(c) = c 所連接 punc_vn 的數量



(a)



(b)

Fig. 2. The expanded recovery trees: (a) For the variable node v_1 and (b) for the check node c_3.

```
Algorithm 1 Find the set of prior candidates for puncturing
Input: $H_{m\times n}$, and $N_p$ →要找出的VN數量
Output: The set of prior candidates $\mathcal{P}_\mathcal{V}$, and the set of reserved
        check nodes $\mathcal{P}_\mathcal{C}$
 1: $V_u \leftarrow \{1, \cdots, n\}$, $V_p \leftarrow \emptyset$
 2: $C_u \leftarrow \{1, \cdots, m\}$, $C_r \leftarrow \emptyset$
 3: for all $v \in V_u$ and all $c \in C(v)$ do $S(v \to c) \leftarrow 1$
 4: repeat        計算 recovery tree 中 的 unpunc VN nodes
 5:   for all $c \in C_u$ do
 6:     $W(c \to v) \leftarrow \sum_{v' \in V(c) \backslash v} S(v' \to c)$ for $\forall v \in V(c)$
 7:     $W(c) \leftarrow \sum_{v' \in V(c)} S(v' \to c)$
 8:   end for     → 挑出 check node 連接中 最少 unpunc vn node，且 degree 低 且在從裡面挑 臑沒有連過 punc vn 的 node
 9:   $\Psi_1 \leftarrow \{$all $c \in C_u$ with the minimal $W(c)\}$
10:   $\Psi_2 \leftarrow \{$all $c \in \Psi_1$ with the minimal $d(c)\}$
11:   $\Psi_3 \leftarrow \{$all $c \in \Psi_2$ with $d_p(c) = 0\}$
12:   $\Omega_0 \leftarrow \{$all $v \in \bigcup_{c\in\Psi_3} V_u(c)\}$  cn 旁的 unpunc.vn
13:   $\Omega_1 \leftarrow \{$all $v \in \Omega_0$ with the minimal $d_r(v)\}$  → reserved check node 數量
14:   $\Omega_2 \leftarrow \{$all $v \in \Omega_1$ with the minimal $d(v)\}$  → 選 degree 低的
15:   for all $v \in \Omega_2$ do $S_E(v) \leftarrow \sum_{c\in C(v)} W(c \to v)$  計算 recovery_tree 中的 unpunc vn
16:   $\Omega_3 \leftarrow \{$all $v \in \Omega_2$ with the minimal $S_E(v)\}$  ← recovery tree with v 有最少的 unpuncture variable node
17:   if $\Omega_3$ is nonempty then
18:     select arbitrarily $v^* \in \Omega_3$ and $c^* \in C(v^*) \bigcap \Psi_3$
19:     update $V_p \leftarrow V_p \bigcup \{v^*\}$, $V_u \leftarrow V_u \backslash V(c^*)$  v 所連接到的 c
20:     update $C_r \leftarrow C_r \bigcup \{c^*\}$, $C_u \leftarrow C_u \backslash c^*$  Nei_VN 都去掉
21:     for all $c \in C(v^*)$ do
22:       $S(v^* \to c) \leftarrow \sum_{c' \in C(v^*)\backslash c} W(c' \to v^*)$
23:     end for
24:   end if
25: until $|V_p| = N_p$ or $\Omega_3 = \emptyset$
26: return $\mathcal{P}_\mathcal{V} \leftarrow V_p$, $\mathcal{P}_\mathcal{C} \leftarrow C_r$
```

reserv cn

decoding iterations, we investigate the design of puncturing for RC-LDPC codes based on the expanded recovery tree.

IV. NON-GREEDY PUNCTURING

Consider a mother code C_0 with rate r_0, length n, and the parity-check matrix $\boldsymbol{H} = [h_{ij}]_{m\times n}$, where $h_{ij} \in \{0, 1\}$ for $1 \leq i \leq m$ and $1 \leq j \leq n$. To obtain a punctured code with rate $r > r_0$, the number of required punctured variable nodes is given by

新 code rate

$$N_p = \lceil n\,(r - r_0)\,/r_0 \rceil\,, \tag{11}$$

where $\lceil x \rceil$ denotes the smallest integer equal to or greater than the real number x.

In the puncturing method proposed below, each punctured variable node is allocated with a single reserved check node in order to assure its recoverability, and all the check nodes remain survived for all punctured codes. The puncturing method is composed of three algorithms for finding candidates for prior puncturing, (1) determining the variable nodes for puncturing, and verifying non-formation of dead check nodes, respectively.

In Algorithm 1, the candidates with the highest puncturing priority, noted '*prior candidates*', are iteratively searched for all target code rates. At the beginning, all the variable nodes are initialized as 'unpunctured', and all the check nodes are initialized as 'un-reserved' in lines 1 and 2. During each search round, only one prior candidate v^* may be found while a check node c^* is accordingly reserved. The search criterion is established in an order of priorities shown in lines 9–16 and explained as follows:

1. The minimal $W(c)$ which indicates a minimum span from c in terms of the number of unpunctured variable nodes.
2. The minimal $d(c)$ which indicates a minimum span from c in terms of the number of sub-branches.
3. The condition $d_p(c) = 0$ indicates that a check node can be reserved for only one neighboring prior candidate.
4. The minimal $d_r(v)$ which indicates a minimum number of reserved check nodes and hence previously punctured variable nodes in the expanded recovery tree for the favorable v.
5. The minimal $d(v)$ which indicates a minimum degree of the favorable v.
6. The minimal $S_E(v)$ which indicates a minimum number of unpunctured variable nodes in the expanded recovery tree for the variable node v.

→ v 的 recovery tree 中，unpunc 越少越好

In summary, the found candidate shall have the smallest $S_E(v)$ while minimizing the relative degrees during the current search round. The required metrics are initialized in line 3, and updated in lines 5–8 and 21–23, respectively. Thus, the prior candidate v^*, together with all other neighbors to c^*, is excluded from V_u in line 19, and c^* is excluded from C_u in line 20.

★多了驗證 是否會造成 dead check node

感覺 Alg 1 就可以了 如果 punc vn 不多的話

Algorithm 2 Determine the variable nodes for puncturing.

```
Input: $H_{m\times n}, N_1, \cdots, N_p, \mathcal{P}_\mathcal{V}$, and $\mathcal{P}_\mathcal{C}$
Output: The set of variable nodes for puncturing $V_p$
1:  $V_u \leftarrow \{1, \cdots, n\}, V_p \leftarrow \emptyset$
2:  $C_u \leftarrow \{1, \cdots, m\}, C_r \leftarrow \emptyset$
3:  for all $v \in V_u$ and all $c \in C(v)$ do $S(v \to c) \leftarrow 1$
4:  repeat
5:      for all $c \in C_u$ do
6:          $W(c \to v) \leftarrow \sum_{v' \in V(c)\backslash v} S(v' \to c)$ for $\forall v \in V(c)$
7:          $W(c) \leftarrow \sum_{v' \in V(c)} S(v' \to c)$
8:      end for
9:      comment: 1. Find the set of eligible candidates
10:     $\Psi_1 \leftarrow \{$all $c \in C_u$ with the minimal $W(c)\}$
11:     $\Psi_2 \leftarrow \{$all $c \in \Psi_1$ with the minimal $d(c)\}$   change
12:     $\Psi_3 \leftarrow \{$all $c \in \Psi_2$ with the minimal $d_p(c)\}$
13:     $\Omega_0 \leftarrow \{$all $v \in \bigcup_{c\in\Psi_3} V_u(c)\}$
14:     $\Omega_1 \leftarrow \{$all $v \in \Omega_0$ with the minimal $d_r(v)\}$
15:     $\Omega_2 \leftarrow \{$all $v \in \Omega_1$ with the minimal $d(v)\}$
16:     for all $v \in \Omega_2$ do $S_E(v) \leftarrow \sum_{c\in C(v)} W(c \to v)$
17:     $\Omega_3 \leftarrow \{$all $v \in \Omega_2$ with the minimal $S_E(v)\}$
18:     comment: 2. Find the intersection with prior candidates
19:     $\mathcal{P}_1 \leftarrow \Omega_3 \bigcap \mathcal{P}_\mathcal{V}$   candidate
20:     $\mathcal{P}_2 \leftarrow \{$all $v \in \mathcal{P}_1$ with the maximal $d_u(v)\}$   → unreservation CN 數量
21:     $\mathcal{P}_3 \leftarrow \{$all $v \in \mathcal{P}_2$ with the minimal $d(v)\}$
                                          ↓ degree 低
22:     if $\mathcal{P}_3$ is nonempty then
23:         select arbitrary $v^* \in \mathcal{P}_3$
24:         if $\mathcal{P}_\mathcal{C}(v^*) \in \Psi_3$ then
25:             select $c^* \leftarrow \mathcal{P}_\mathcal{C}(v^*)$
26:         else
27:             select arbitrary $c^* \in C(v^*) \bigcap \Psi_3$
28:         end if
29:         update $\mathcal{P}_\mathcal{V} \leftarrow \mathcal{P}_\mathcal{V} \backslash v^*$
30:     else
31:         select arbitrary $v^* \in \Omega_3$
32:         select arbitrary $c^* \in C(v^*) \bigcap \Psi_3$
33:     end if
34:     update $V_u \leftarrow V_u \backslash v^*$
35:     comment: 3. Verify if dead check nodes are formed
36:     option of calling Algorithm 3 with $(v^*, c^*, V_p)$
37:     if Algorithm 3 succeeds then
38:         update $V_p \leftarrow V_p \bigcup \{v^*\}$
39:         update $C_r \leftarrow C_r \bigcup \{c^*\}$, $C_u \leftarrow C_u \backslash c^*$
40:         for all $c \in C(v^*)$ do
41:             $S(v^* \to c) \leftarrow \sum_{c' \in C(v^*)\backslash c} W(c' \to v^*)$
42:         end for   如果 succeeds，代表不會因為 $v^*$ 造成 Died check node.
43:     end if
44:     if $|V_p| = N_i, i \in \{1, \cdots, p\}$ then
45:         return $V_P$
46:     end if
47: until $|V_p| = N_p$
48: return $V_p$
```

The search process continues until no more prior candidates can be found, or the maximum number N_P of prior candidates is reached, i.e., $|\mathcal{P}_\mathcal{V}| \le N_P$. Clearly, there is an one-to-one correspondence between the prior candidates in $\mathcal{P}_\mathcal{V}$ and the reserved

Algorithm 3 Verify if puncturing v^* forms dead check nodes

```
Input: $v^*$, $c^*$, and $V_p$
Output: The indicator as ‘failure’ or ‘success’
1:  if $d_u(v^*) = d(v^*)$ or $d_u(c^*) = d(c^*)$ then
2:      return ‘success’
3:  else
                            → 假設 punc 掉 $v^*$
4:      $I \leftarrow 0, V'_p \leftarrow V_p \bigcup \{v^*\}$
5:      repeat
6:          for all $v \in V'_p$ do
7:              if $\exists c \in C(v)$ such that $(V(c)\backslash v) \bigcap V'_p = \emptyset$ then
                        ↓ 代表 v 的相鄰 VN 都沒 punc
                          → 有一個 survived CN
8:                  $V'_p \leftarrow V'_p \backslash v$
9:              end if
10:         end for
            $I \leftarrow I + 1$
11:     until $(V'_p = \emptyset)$ or $(I = I_{\max})$
12:     if $V'_p = \emptyset$ then return ‘success’ else return ‘failure’
13: end if
```

check nodes in $\mathcal{P}_\mathcal{C}$, which is denoted as $c^* = \mathcal{P}_\mathcal{C}(v^*)$.

Algorithm 2 is the main routine for iteratively determining the sets V_i of punctured variable nodes for all target rates r, where $V_1 \subset V_2 \subset \cdots \subset V_i$. Similarly, during each search round, a punctured variable node v^* and its reserved check node c^* is determined under a series of priorities as explained above.

$d_p(c) = 0$ $\Downarrow$ min $d_p(c)$

Note that the condition of minimal $d_p(c)$ in line 12 of Algorithm 2 is different from the condition $d_p(c) = 0$ in line 11 of Algorithm 1. Only a few of variable nodes can be selected by the criterion $d_p(c) = 0$. In order to get high code rate RC-LDPC codes, the criterion $d_p(c) > 0$ is necessary. With larger value of $d_p(c)$, c should be a dead check node with more possibility. This means the messages from c to its neighboring variable nodes become more unreliable in the iterative decoding process. In a word, the minimal $d_p(c)$ implies that more than one, but as few as possible, neighboring check nodes of a punctured node are allowed to be reserved for newly punctured nodes.

為了 high coderate 才改

Algorithm 3 is a subroutine of Algorithm 2 for verifying whether any dead check nodes will be formed by puncturing the candidate v^*. Note that more than one dead check node may be formed by puncturing a single variable node. We explain the formation of dead check nodes using the example expanded recovery trees for a specific candidate v_1 in Fig. 2.

In Fig. 3(a), we suppose that v^* is the variable node selected for puncturing in addition to $V_p = \{v_1, v_2, v_3, v_4\}$. During the first iteration of decoding, v_4 is recovered through its reserved check node c_6 which is connected with three unpunctured variable nodes. Similarly, during the second iteration, v_3 is recovered through its reserved check node c_5. To this point, c_1, c_2, and c_4 have appeared as dead since each of them is connected with two punctured variable nodes.

In Fig. 3(b), we suppose that the variable node v_2 is connected with one more check node c_3. During the first iteration of decoding, v_2 is recovered through c_3. This results in the recovery of v_1 and v^* respectively through c_1 and c_4 during the second iteration. Consequently, all the punctured variable nodes can be recovered in two iterations. This verifies that puncturing v^* does not yield dead check nodes.

Clearly, in the special cases $d_u(v^*) = d(v^*)$ or $d_u(c^*) =$

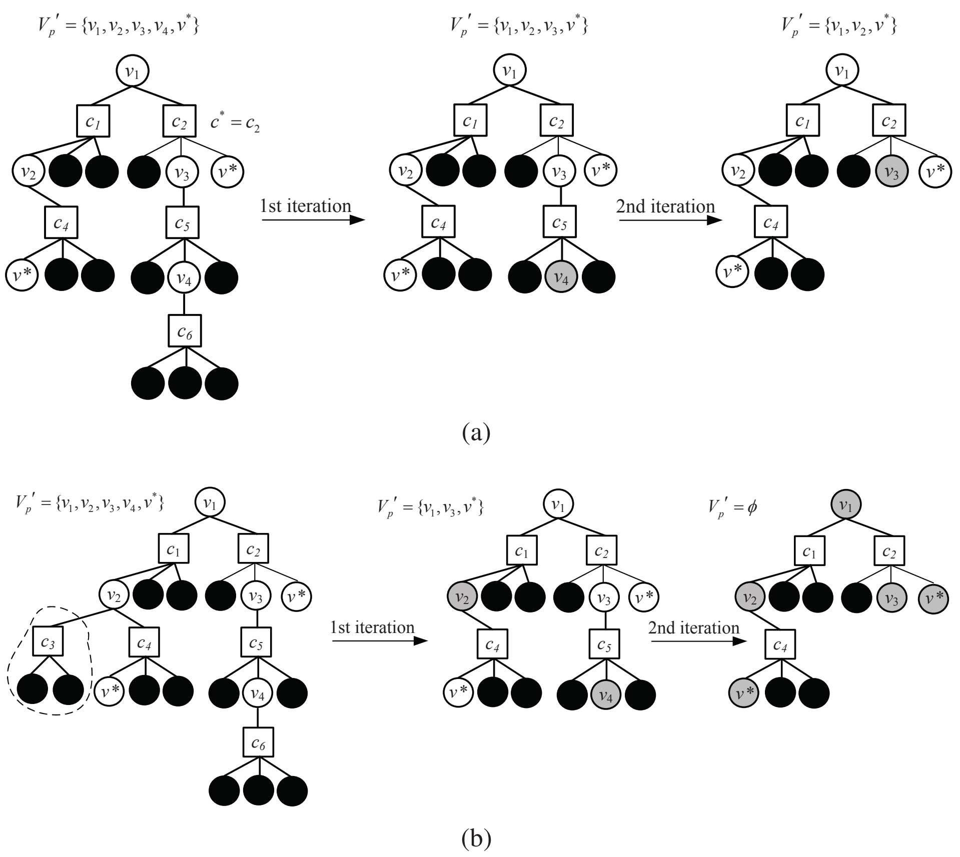

$V_p' = \{v_1, v_2, v_3, v_4, v^*\}$
$c^* = c_2$
1st iteration
$V_p' = \{v_1, v_2, v_3, v^*\}$
2nd iteration
$V_p' = \{v_1, v_2, v^*\}$
(a)
$V_p' = \{v_1, v_2, v_3, v_4, v^*\}$
1st iteration
$V_p' = \{v_1, v_3, v^*\}$
2nd iteration
$V_p' = \phi$
(b)

Fig. 3. (a) Verification of formation of dead check nodes and (b) verification of non-formation of dead check nodes when puncturing a candidate variable node v^*.

$d(c^*)$, the check node c^* must not become dead. By definition, the condition $d_u(v^*) = d(v^*)$ means that none of the neighbors to v^* is reserved, and the condition $d_u(c^*) = d(c^*)$ means that none of the neighbors to c^* is punctured. For any punctured variable node $v \in V_p'$, if there exists a neighboring check node c with all other neighbors in $V(c)\backslash v$ being either unpunctured or recovered unpunctured, i.e., $(V(c)\backslash v) \bigcap V_p' = \emptyset$, then v' must be recoverable. In this case, V_p is updated with excluding v. If V_p' finally becomes empty, all the punctured variable nodes in the original set $V_p \bigcup v^*$ are recoverable, meaning that puncturing v^* does not yield dead check nodes.

V. SIMULATION RESULTS

In this section, we present simulation results on binary RC-LDPC codes obtained by our proposed puncturing algorithm and the puncturing algorithms proposed in [1], [2], and [7]. The AWGN channel is assumed with BPSK modulation. The standard SPA is used for decoding with the maximum number of iterations $I_{\max} = 50$. In the simulations, we employ one irregular LDPC code C_1 and one regular LDPC code C_2 as the mother codes. Both codes have rate $r_0 = 0.5$ and length $n = 1008$ and are constructed by the PEG algorithm [21] using the degree distributions in Table 2, where $\lambda(x)$ and $\rho(x)$ denote the degree distributions of the variable nodes and the check nodes, respectively. The RC-LDPC codes have rates 0.6, 0.7, 0.8, and 0.9. It should be noted that the puncturing algorithm of [2] cannot construct the punctured code with rate 0.9.

Table 2. Parameters of simulated mother codes.

Code	**Degree distributions**
C_1	$\lambda(x) = 0.4762x + 0.2788x^2 + 0.1081x^3 + 0.1012x^4 + 0.0357x^{14}$, $\rho(x) = 0.0174x^6 + 0.9826x^7$
C_2	$\lambda(x) = x^2$, $\rho(x) = x^5$

regular

For the RC-LDPC codes based on the irregular mother code C_1, Table 3 shows the number of punctured variable nodes with different degree values $d(v) \in \{2, 3, 4, 5, 15\}$. Table 4 shows the number of check nodes with different $d_p(c)$ values, i.e., the number of punctured variable nodes neighboring to a check node c. Note that formation of dead check nodes due to puncturing can result in a decreased number of survived check nodes for punctured codes. Therefore, from Table 4, it is obvious that the puncturing algorithm in [1] must yield dead check nodes.

Roughly stated, the algorithm in [7] yields the smallest number of punctured variable nodes with degrees greater than 2 as well as the smallest number of survived check nodes with punctured degrees greater than zero. In this sense, the proposed algorithm appears a little inferior to the algorithm in [7], but superior to the algorithms in [1] and [2].

As analyzed in Section II, a punctured variable node v should have a small $S_E(v)$ value for high recovery probability. In other

Table 3. Number of punctured nodes with different degrees.

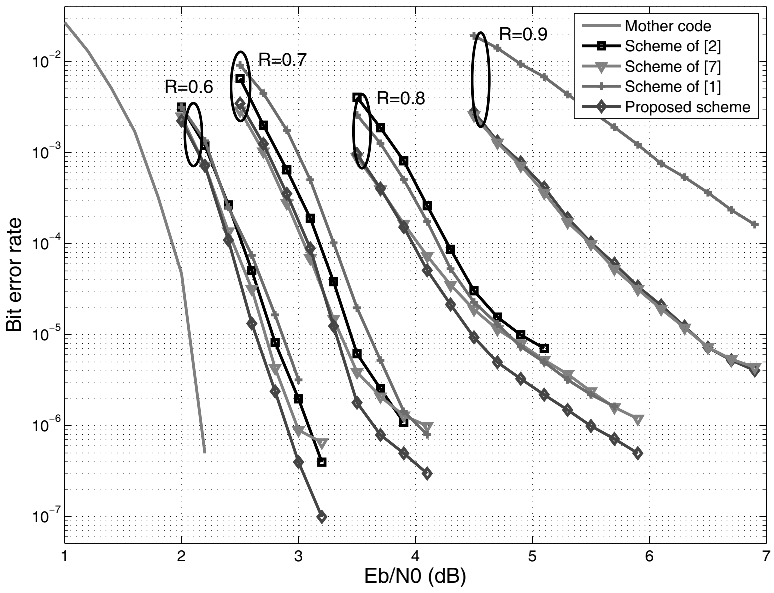
Schemes	**Scheme in [7]**				**Proposed scheme**				**Scheme in [2]**			**Scheme in [1]**			
Rates	0.6	0.7	0.8	0.9	0.6	0.7	0.8	0.9	0.6	0.7	0.8	0.6	0.7	0.8	0.9
$d(v) = 2$	167	284	375	413	166	277	366	397	137	257	310	115	192	278	344
$d(v) = 3$	0	2	2	34	0	8	10	49	25	25	55	42	51	60	69
$d(v) = 4$	0	0	0	0	0	0	0	0	1	1	1	2	6	8	5
$d(v) = 5$	0	1	1	1	1	2	2	2	4	4	12	7	20	19	12
$d(v) = 15$	0	0	0	0	0	0	0	0	0	0	0	1	18	13	18
Total	167	287	378	448	167	287	378	448	167	287	378	167	287	378	448

Table 4. Number of check nodes with different punctured degrees.

Schemes	**Scheme in [7]**				**Proposed scheme**				**Scheme in [2]**			**Scheme in [1]**			
Rates	0.6	0.7	0.8	0.9	0.6	0.7	0.8	0.9	0.6	0.7	0.8	0.6	0.7	0.8	0.9
$d_p(c) = 0$	170	0	0	0	168	0	0	0	143	23	0	94	0	0	0
$d_p(c) = 1$	334	430	248	119	335	418	240	129	349	373	319	410	238	137	105
$d_p(c) = 2$	0	73	255	341	1	86	257	306	12	85	66	0	186	218	175
$d_p(c) = 3$	0	1	1	44	0	0	7	68	0	22	84	0	63	126	162
$d_p(c) \geq 4$	0	0	0	0	0	0	0	1	0	1	35	0	17	23	62
Total	504	504	504	504	504	504	504	504	504	504	504	504	504	504	504

Table 5. Number of punctured nodes within different $S_E(v)$ ranges.

Schemes	**Scheme in [2]**			**Proposed scheme**				**Scheme in [7]**				**Scheme in [1]**			
Rates	0.6	0.7	0.8	0.6	0.7	0.8	0.9	0.6	0.7	0.8	0.9	0.6	0.7	0.8	0.9
$S_E(v) \leq 10$	167	287	310	167	257	257	257	167	252	252	252	167	244	269	162
$11 \leq S_E(v) \leq 20$	0	0	61	0	30	121	171	0	35	126	178	0	41	77	134
$S_E(v) \geq 21$	0	0	5	0	0	0	20	0	0	0	18	0	2	32	52
Total	167	287	378	167	287	378	448	167	287	378	448	167	287	378	448



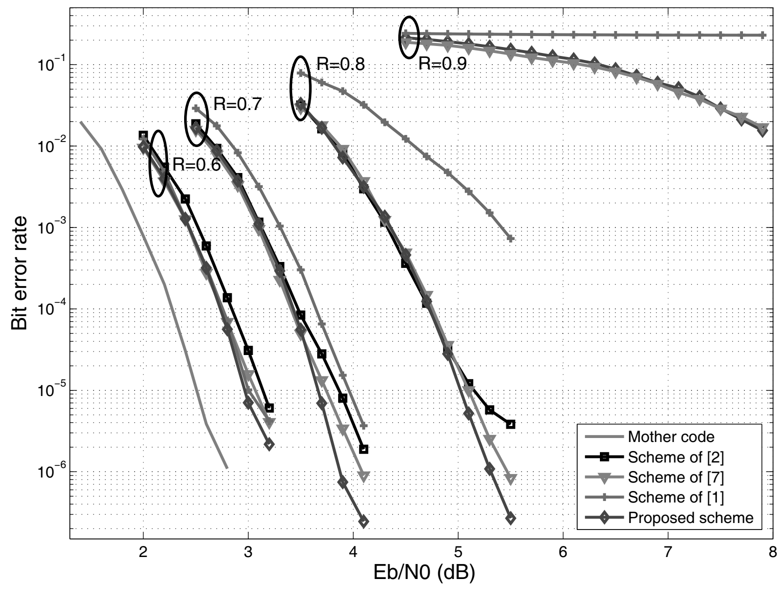
[illegible]

Fig. 4. BER performance comparisons for RC-LDPC codes obtained by various puncturing methods from the irregular mother code C_1.

words, fewer punctured variable nodes with large $S_E(v)$ values can yield better overall decoding performance. In this sense, it is seen from Table 5 that the proposed algorithm is superior to the algorithm in [7].

Fig. 4 compares the bit error rate performances of the RC-LDPC codes obtained from the irregular mother code C_1 using the above puncturing algorithms. Clearly, the proposed algorithm results the best performance at rates 0.6, 0.7, and 0.8 over a wide signal-to-noise ratio (SNR) range. For example, at rates 0.7 and 0.8, the proposed algorithm achieves an increase in coding gain of about 0.4 dB for the BER of 10^{-6}. However, when the code rate is increased to 0.9, the proposed algorithm performs similarly as the algorithm in [7].

Fig. 5 compares the bit error rate performances of the RC-LDPC codes obtained from the regular mother code C_2. While the error performances present a similar tendency to that in



← regular
都比較好一點

Fig. 5. BER performance comparisons for RC-LDPC codes obtained by various puncturing methods from the regular mother code C_2.

Fig. 4, the improvement of the proposed algorithm for the regular mother code is observed mainly in a relative high SNR range.

It may be concluded that the proposed algorithm yields the best error performances although it does not give out the best results on node distributions in Tables 3 and 4. This error performance improvement may be explained as a result of minimizing the metric $S_E(v)$, i.e., the total number of unpunctured variable nodes in the recovery tree, for each punctured variable node v.

VI. CONCLUSION

This paper has investigated the puncturing scheme for constructing binary RC-LDPC codes. Based on the redefinition of the recovery tree, a puncturing algorithm is proposed for dynamically minimizing the number of unpunctured variable nodes in the recovery tree by following an order of priorities. Simulation results show that the proposed puncturing algorithm outperforms several existing best puncturing algorithms. In practical system, only a little of extra storage space is necessary to store the sets of punctured nodes in tables. That will hardly increase the complexity of the communication system.

REFERENCES

[1] 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.

[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, no. 2, pp. 728–738, Feb. 2006.

[3] J. Ha, J. Kim, and S. W. McLaughlin, "Rate-compatible puncturing of low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 50, no. 11, pp. 2824–2836, Nov. 2004.

[4] M. R. Yazdani and A. H. Banihashemi, "On construction of rate-compatible low-density parity-check codes," *IEEE Commun. Lett.*, vol. 8, no. 3, pp. 159–161, Mar. 2004.

[5] D. Bi and L. C. Perez, "Rate-compatible low-density parity-check codes with rate-compatible degree profiles," *Electron. Lett.*, vol. 42, no. 1, pp. 41–43, Jan. 2006.

[6] 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.

[7] 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, no. 11, pp. 3914–3919, Nov. 2007.

[8] H. Chun-Hao and A. Anastasopoulos, "Capacity achieving LDPC codes through puncturing," *IEEE Trans. Inf. Theory*, vol. 54, no. 10, pp. 4698–4706, Oct. 2008.

[9] Q. Diao, Y. Y. Tai, S. Lin, and K. Abdel-Ghaffar, "LDPC codes on partial geometries: Construction, trapping sets structure, and puncturing," *IEEE Trans. Inf. Theory*, vol. 59, no. 12, pp. 7898–7914, Dec. 2013.

[10] 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.

[11] L. Jingjing and R. C. de Lamare, "Finite-length rate-compatible LDPC codes based on extension techniques," in *Proc. IEEE ISWCS*, Nov. 2011, pp. 41–45.

[12] R. Asvadi and A. H. Banihashemi, "A rate-compatible puncturing scheme for finite-length LDPC codes," *IEEE Commun Lett.*, vol. 17, no. 1, pp. 147–150, Jan. 2013.

[13] Z. Hua, D. G. M. Mitchell, N. Goertz, and Jr. D. J. Costello, "Robust rate-compatible punctured LDPC convolutional codes," *IEEE Trans. Commun.*, vol. 61, no. 11, pp. 4428–4439, Nov. 2013.

[14] K. Deka, A. Rajesh, and P. K. Bora, "An improved puncturing method for rate-compatible LDPC codes," in *Proc. IEEE NCC*, Feb. 2014, pp. 1–6.

[15] L. Zhou, B. Bai, and M. Xu, "Design of nonbinary rate-compatible LDPC codes utilizing bit-wise shortening method," *IEEE Commun. Lett.*, vol. 14, no. 10, pp. 963–965, Oct. 2010.

[16] T. Okamura, "A hybrid ARQ scheme based on shortened low-density parity-check codes," in *Proc. IEEE WCNC*, Apr. 2008, pp. 82–87.

[17] T. V. Nuguyen, A. Nosratinia, and D. Divsalar, "The design of rate-compatible protograph LDPC codes," *IEEE Trans. Commun.*, vol. 60, no. 10, pp. 2841–2850, Oct. 2012.

[18] S. Zhongwei, R. Thobaben, and M. Skoglund, "Rate-compatible LDPC convolutional codes achieving the capacity of the BEC," *IEEE Trans. Inf. Theory*, vol. 58, no. 6, pp. 4021–4029, June 2012.

[19] T. V. Nguyen and A. Nosratinia, "Rate-compatible short-length protograph LDPC codes," *IEEE Commun. Lett.*, vol. 17, no. 5, pp. 948–951, May 2013.

[20] H. Jie, Z. Wei, and Z. shengli, "Structured nonbinary rate-compatible low-density parity-check codes," *IEEE Commun. Lett.*, vol. 15, no. 9, pp. 998–1000, Sept. 2011.

[21] X. 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.

Lin Zhou was born in Nanyang, Henan, China in 1982. He received the B.E., M.S., and Ph.D. degrees in Communications Engineering from Xidian University, Xi'an, China in 2004, 2007, and 2011, respectively. He is currently a Lecturer at National Huaqiao University, Xiamen. He is also a Member of the State Key Laboratory of ISN, Xidian University. His major research interests include information theory and coding, LDPC codes, capacity-approaching coded modulation systems, and other related topics.

Weicheng Huang received the B.S. degree in Communication Engineering from Heilongjiang Bayi Agricultural University, Daqing, China in 2013, and the M.S. degree in Communication and Information System from Huaqiao University, Xiamen, China in 2016. His research interests include coding theory and its applications to communication systems.

Rui Zhao received the double bachelor degrees from Harbin Institute of Technology in 2003, and the M.S. and Ph.D. degrees in Electrical Engineering from Southeast University, China in 2006 and 2010, respectively. After graduation, he joined the School of Information Science and Engineering, Huaqiao University, China, where he is currently an Associate Professor. His current research interests include cooperative communications, physical-layer security communications, and MIMO communication systems.

Yucheng He received his Ph.D. degree in Information and Communications Engineering from Xidian University and completed a two-year Postdoctoral Fellowship at Ecole Polytechnique de Montreal, Quebec, Canada. He is currently a Professor and the Director of the Xiamen Key Laboratory of Mobile Multimedia Communications, National Huaqiao University, Xiamen. He is also a Member of the State Key Laboratory of ISN, Xidian University. His research interests include cooperative communications and channel coding.