A Good Puncturing Scheme for Rate Compatible Low-Density Parity-Check Codes

Sunghoon Choi, Sungroh Yoon, Wonjin Sung, Hongkyu Kwon, and Jun Heo

Abstract: **We consider the challenges of finding good puncturing patterns for rate-compatible low-density parity-check code (LDPC) codes over additive white Gaussian noise (AWGN) channels. Puncturing is a scheme to obtain a series of higher rate codes from a lower rate mother code. It is widely used in channel coding but it causes performance is lost compared to non-punctured LDPC codes at the same rate. Previous work, considered the role of survived check nodes in puncturing patterns. Limitations, such as single survived check node assumption and simulation-based verification, were examined. This paper analyzes the performance according to the role of multiple survived check nodes and multiple dead check nodes. Based on these analyses, we propose new algorithm to find a good puncturing pattern for LDPC codes over AWGN channels.**

Index Terms: **block-type LDPC codes (B-LDPC), density evolution (DE), low-density parity-check code (LDPC) codes, puncturing.**

I. INTRODUCTION

Design of good rate-compatible low-density parity-check code (LDPC) codes is a major concern for recent mobile communication standardization efforts. Several schemes to achieve effective rate compatibility have been researched in the literature. Among those schemes, puncturing is known as one of the most effective and convenient ways to change code rates. However, the performance of punctured codes strongly depends on the nodes that are punctured (i.e., puncturing pattern). Therefore, many research studies have sought good puncturing patterns for LDPC codes [1]–[8].

The density evolution (DE) technique has been widely used in the literature to analyze the asymptotic performance of LDPC codes. It recursively tracks the probability density of the extrinsic information between the variable nodes and check nodes of an LDPC code [9]. A simplified version of the DE technique was introduced with a Gaussian approximation in [10]. The Gaussian approximation was used on the basis that extrinsic information can closely approximate a Gaussian random variable as the number of iterations increases.

In [4], a puncturing scheme for LDPC codes with short and moderate block lengths was proposed in a systematic way; the performance of LDPC codes was analyzed by the DE technique. The punctured nodes were categorized by the required number of iterations to be recovered. For instance, the k-SR node is the punctured variable node that is recovered after kth iterations. Based on this categorization, a search algorithm named as *grouping* and *sorting* for punctured nodes was proposed. The code structure, however, was over-simplified, such that each punctured node had only one survived check node that gave messages to recover the punctured node.

Conversely, LDPC codes have some disadvantages compared to other codes, such as an encoding complexity and a high memory requirement. Recently, many people have tried to solve them in some structured forms, such as a lower triangular shape and in new code types such as quasi-cyclic LDPC (QC-LDPC) codes [11]. The block-type LDPC (B-LDPC) code, a kind of QC-LDPC codes, was adopted in the IEEE 802.16e standards [12]. In [6], a block-unit puncturing for block-type LDPC codes was considered. In addition, it was suggested that the number of survived check nodes be maximized to enhance performance. However, the suggestion was not based on any analytical methods. Rather it was based on Monte-Carlo simulation with particular parity check matrices. It is also well-known that the cycles of LDPC codes affect performance, especially in the high signal-to-noise area. In [13], the relationship between puncturing patterns and the cycles of punctured nodes was presented. It was suggested to puncture the variable nodes with large cycles for a good rate compatible LDPC code.

In this paper, we use the DE technique to analyze the positive effect of multiple survived check nodes and multiple dead check nodes. We show that the mean of log likelihood ratio (LLR) passed between variable nodes and check nodes converges to the correct value faster as the number of survived check nodes and number of dead check nodes increases. Based on this analytical result, we propose an algorithm to find good puncturing pattern. We also show that the analysis results hold for block-type LDPC codes as well. Numerical results on both regular LDPC code and irregular LDPC code are in step with the analytical results.

II. BASIC DEFINITIONS AND NOTATIONS

k-step recoverable (k-SR) nodes, *survived check* (SC) nodes and *dead check* (DC) nodes are defined as in [4] to categorize punctured nodes based on their characteristics. The punctured nodes have zero channel LLR values and they spread to the other unpunctured nodes through neighboring check nodes during the

Manuscript received May 07, 2008; approved for publication by Giorgio Taricco, Division I Editor, February 27, 2009.

This research is supported by the Ubiquitous Computing and Network (UCN) Project, Knowledge and Economy Frontier R&D Program of the Ministry of Knowledge Economy (MKE) in Korea as a result of UCN's subproject 09C1-C2-31T and the MKE (Ministry of Knowledge Economy), Korea, under the ITRC(Information Technology Research Center) support program supervised by the IITA (Institute for Information Technology Advancement) (IITA-2009-C1090-0902-0045).

S. Choi is with the LG electronics Inc., Seoul, Korea, email: monodt@lge.com.

W. Sung is with the Department of Electronic Engineering, Sogang University, Seoul, Korea, email: wsung@sogang.ac.kr.

H. Kwon is with the Department of Industrial and Management Engineering, Namseoul University, Cheonan, Korea, email: hongkyuk@nsu.ac.kr.

S. Yoon and J. Heo (corresponding author) are with the School of Electrical Engineering Korea University, Seoul, Korea, email: {sryoon, junheo}@korea.ac.kr.

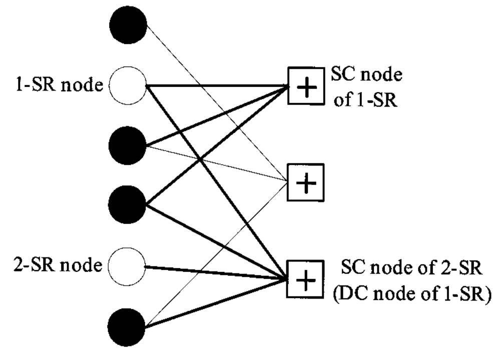


Fig. 1. Description of 1-SR node, 2-SR node, SC node and DC node (filled circle: Punctured node, unfilled circle: Punctured node).

iterative decoding process. This is the main reason for the performance loss of punctured LDPC codes. Therefore, we should choose the punctured nodes that can have non-zero LLR values (it is defined as *recover*) as quickly as possible.

The k-SR node is defined as a punctured variable node that is recovered after kth iterations where k corresponds to the recovery level. If any punctured node has at least one neighboring check node whereby neighbors are all unpunctured nodes, then it is called a 1-SR node. After the first iteration, the 1-SR node can be recovered by unpuntured nodes sharing the same check nodes. The neighboring check node(s) with all unpunctured nodes is(are) defined as SC node(s) of the 1-SR node. The other check nodes, except for the SC node(s), are defined as DC nodes of the 1-SR node. That is the SC node is the check node that helps the k-SR node recover and the DC node is the check node which does not help the k-SR node recover. Therefore, if a punctured node becomes a k-SR node, its SC node(s) of k-SR should neighbor at least one (k-1)-SR node while the others are k'-SR nodes, where $0 \leq k' \leq k-1$. Examples of a 1-SR node and 2-SR node are as described in Fig. 1. The filled circles represent unpunctured nodes; unfilled circles represent punctured nodes. A k-SR node can have a single or multiple SC nodes. A k-SR node may or may not have a DC node(s). A hierarchical structure consisting of various SR nodes, SC nodes, and DC nodes is defined as a *recovery tree* [4].

III. ANALYSIS OF SURVIVED CHECK NODES

A. Density Evolution

The DE technique has been widely used in the literature to analyze the asymptotic performance of LDPC codes. It recursively tracks the probability density of the extrinsic information between the variable nodes and check nodes of an LDPC code [9]. A simplified version of the DE technique was introduced with a Gaussian approximation in [10]. The Gaussian approximation was based on the extrinsic information being approximated to a Gaussian random variable as the number of iterations increases. The extrinsic information is usually expressed as the log-likelihood ratio (LLR) defined as

$$v = \ln \frac{p(y \mid a = 0)}{p(y \mid a = 1)}$$

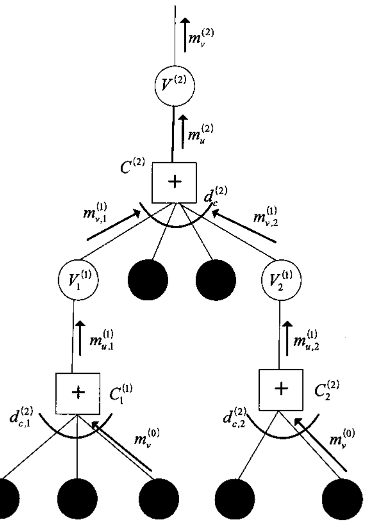


Fig. 2. Recovery tree in the case where each punctured node has a single SC node.

where a denotes a random variable describing the bit value of the transmitted codeword and $p(\cdot)$ denotes the probability density function. The output LLR message from a variable node with d_v edges can be calculated as

$$v = v^{(0)} + \sum_{i=1}^{d_v-1} u_i$$

where $v^{(0)}$ denotes the received LLR value from the channel, which is a function of signal-to-noise ratio, and u_i denotes the LLR message from the ith check node to the variable node. The output LLR message from a check node with d_c edges can be calculated as

$$\tanh \frac{u}{2} = \prod_{j=1}^{d_c-1} \tanh \frac{v_j}{2}$$

where v_j denotes the LLR message from the jth variable node to the check node. The recursive calculation between variable nodes and check nodes is executed for a sufficiently large number of iterations to determine if the LLR message converges to the correct codeword at a specified channel noise level.

B. The Recovery Tree with a Single SC Node

In this section, we assume that each punctured node is connected to a single SC node that gives recovery information to the punctured node. Let $V^{(k)}$ denote a k-SR node and $C^{(k)}$ denote a neighboring check node of the k-SR node. Let u denote the LLR value from a check node to a variable node and v denote the LLR value from a variable node to a check node. $u^{(k)}$ and $m_u^{(k)}$ denote the LLR value and its mean from the SC node of a k-SR node to a k-SR node. $v^{(k)}$ and $m_v^{(k)}$denote the LLR value and its mean from a k-SR node to the SC node of a $(k+1)$-SR node. $v^{(0)}$ and $m_v^{(0)}$ represent the LLR value and its mean from the additive white Gaussian noise (AWGN) channel. Based on DE with Gaussian approximation [10], we present how LLR messages evolve on the recovery tree. For simplicity, only the 1-SR and 2-SR nodes are considered in Fig. 2.

At the survived check node $C_1^{(1)}$, the LLR value is updated by the following equation

$$E\left[\tanh\frac{u_1^{(1)}}{2}\right] = \left\{E\left[\tanh\frac{v^{(0)}}{2}\right]\right\}^{d_{c,1}^{(1)}-1} \tag{1}$$

where $d_{c,1}^{(1)}$ denotes the degree of the check node $C_1^{(1)}$. Assuming Gaussian approximation, we can modify (1) as

$$1-\phi(m_{u,1}^{(1)}) = [1-\phi(m_v^{(0)})]^{d_{c,1}^{(1)}-1} \tag{2}$$

where the function $\phi(\cdot)$ is defined as [10]

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

Then, the mean value $m_{u,1}^{(1)}$ from the check node $C_1^{(1)}$ to the punctured variable node $V_1^{(1)}$ is obtained as

$$m_{u,1}^{(1)} = \phi^{-1}\{1-[1-\phi(m_v^{(0)})]^{d_{c,1}^{(1)}-1}\}.$$

Because of the assumption that a punctured node has only one SC node, $m_{v,1}^{(1)}$ equals to $m_{u,1}^{(1)}$. Similarly, for the other SC node $C_2^{(1)}$, the mean value $m_{u,2}^{(1)}$ from the check node $C_2^{(1)}$ to the punctured variable node $V_2^{(1)}$ is obtained as

$$\begin{aligned} 1-\phi(m_{u,2}^{(1)}) &= [1-\phi(m_v^{(0)})]^{d_{c,2}^{(1)}-1} \\ m_{u,2}^{(1)} &= \phi^{-1}\{1-[1-\phi(m_v^{(0)})]^{d_{c,2}^{(1)}-1}\} \end{aligned} \tag{3}$$

where $d_{c,2}^{(1)}$ denotes the degree of the check node $C_2^{(1)}$. With the assumption that a punctured node has only one SC node, $m_{v,1}^{(2)}$ equals to $m_{u,1}^{(2)}$. Then, the mean value $m_u^{(2)}$ from the check node $C^{(2)}$ to the punctured variable node $V^{(2)}$ is obtained using (2) and (3) as

$$\begin{aligned} &1-\phi(m_u^{(2)}) \\ &=[1-\phi(m_{v,1}^{(1)})][1-\phi(m_{v,2}^{(1)})][1-\phi(m_v^{(0)})]^{d_c^{(2)}-3} \\ &=[1-\phi(m_v^{(0)})]^{d_{c,1}^{(1)}-1}[1-\phi(m_v^{(0)})]^{d_{c,1}^{(1)}-1}[1-\phi(m_v^{(0)})]^{d_c^{(2)}-3} \\ &=[1-\phi(m_v^{(0)})]^{S(C^{(2)})} \end{aligned}$$

where $d_c^{(2)}$ denotes the degree of the check node $C^{(2)}$ and $S(C^{(2)})$ is the total number of unpunctured nodes under the SC node $C^{(2)}$ of 2-SR variable node $V^{(2)}$ as

$$S(C^{(2)}) = (d_{c,1}^{(1)}-1)+(d_{c,2}^{(1)}-1)+(d_c^{(2)}-3).$$

Then, $\mathrm{m}_u^{(2)}$ can be obtained as

$$m_u^{(2)} = \phi^{-1}\{1-[1-\phi(m_v^{(0)})]^{S(C^{(2)})}\}.$$

Generally, for k-SR nodes

$$\begin{aligned} m_u^{(k)} &= \phi^{-1}\{1-[1-\phi(m_v^{(0)})]^{S(C^{(k)})}\} \\ m_v^{(k)} &= m_u^{(k)} \end{aligned} \tag{4}$$

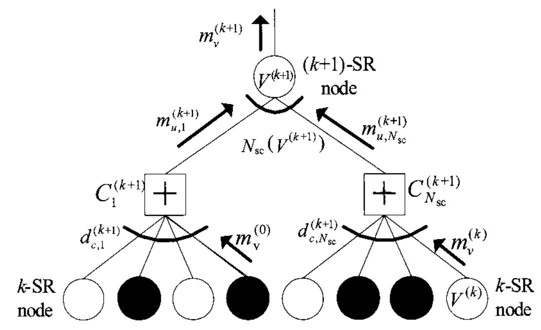


Fig. 3. Recovery tree with multiple survived check (SC) nodes.

where $S(C^{(k)})$ is the total number of unpunctured nodes under the SC node $C^{(k)}$ of k-SR variable node $V^{(k)}$. It is noteworthy that the mean of the LLR value is inversely proportional to $S(C^{(k)})$ because the function $\phi(\cdot)$ is a monotonically decreasing function and $\phi(\cdot) \in (0,1]$. In the AWGN channel, assuming the independent, identical distribution and symmetry condition, the error probability of a node V is given by

$$P_e(V) = Q\left(\frac{m_v}{\sqrt{\sigma_v^2}}\right) = Q\left(\sqrt{\frac{m_v}{2}}\right) \tag{5}$$

where σ_v^2 is the variance of the LLR value from a node V. Based on (4) and (5), and the single SC node assumption, we can conclude that the error probability increases when the number of unpunctured nodes increases.

C. The Recovery Tree with Multiple SC Nodes

The goal of this section is to analyze the effect of multiple SC nodes with LLR message passing. In Fig. 3, a recovery tree with multiple SC nodes of $(k+1)$-SR is shown. Let $N_{sc}(V^{(k+1)})$ denote the number of SC nodes of the $(k+1)$-SR node $V^{(k+1)}$. For simplicity, we assume that there is a single SC node with up to k-SR nodes. Based on the derivation of the previous section, the LLR values from two SC node $C_i^{(k+1)}$ can be represented as

$$m_{u,i}^{(k+1)} = \phi^{-1}\{1-[1-\phi(m_v^{(0)})]^{S(C_i^{(k+1)})}\}. \tag{6}$$

Because the outgoing LLR value from a variable node is the sum of the incoming LLR values from other variable nodes,

$$\begin{aligned} m_v^{(k+1)} &= \sum_{i=1}^{N_{sc}(V^{(k+1)})} m_{u,i}^{(k+1)} \\ &= \sum_{i=1}^{N_{sc}(V^{(k+1)})} \phi^{-1}\{1-[1-\phi(m_v^{(0)})]^{S(C_i^{(k+1)})}\}. \end{aligned} \tag{7}$$

In (7) each term $m_{u,i}^{(k+1)}$ is a nonnegative value with all zero codeword assumption. Therefore, the multiple SC nodes case with $N_{sc}(V^{(k+1)})$ nonnegative terms makes the outgoing LLR value $m_v^{(k+1)}$ evolve faster than that with the single SC node case. As a special case, we assume that all SC nodes $C_i^{(k+1)}$, $(1 \le i \le N_{sc}(V^{(k+1)}))$ have the same number of unpunctured nodes under the SC nodes of $(k+1)$-SR node. Then,

$$\begin{aligned} C_i^{(k+1)} &= C^{(k+1)} \\ m_{u,i}^{(k+1)} &= m_u^{(k+1)} \end{aligned}$$

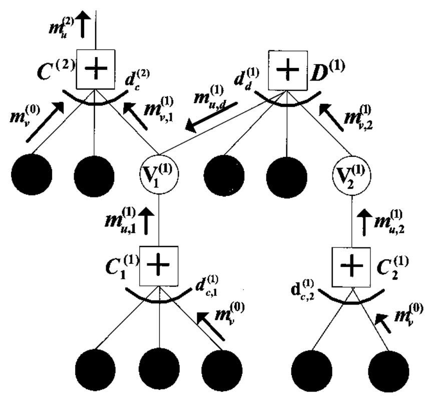


Fig. 4. Recovery tree with multiple dead check (DC) nodes.

where $(1 \leq i \leq N_{sc}(V^{(k+1)}))$. Therefore, the number of survived check nodes $N_{sc}(V^{(k+1)})$ is multiplied to $m_u^{(k+1)}$ as

$$\begin{aligned} m_v^{(k+1)} &= N_{sc}(V^{(k+1)})m_u^{(k+1)} \\ &= N_{sc}(V^{(k+1)})\phi^{-1}\{1-[1-\phi(m_v^{(0)})]^{S(C^{(k+1)})}\}. \end{aligned}$$

It is noted that the total number of unpunctured nodes under the $(k+1)$-SR node $V^{(k+1)}$ increase $N_{sc}(V^{(k+1)})$ times such as $N_{sc}(V^{(k+1)})S(C^{(k+1)})$. However the effective number of unpunctured nodes which gives a negative effect on performance is still $S(C^{(k+1)})$ which is the power of $1-\phi(m_v^{(0)})$ in the previous equation. Therefore the mean of the LLR value increases faster than that of the single survived check node. As a result, the error probability in (5) decreases.

D. The Recovery Tree with Multiple DC Nodes

To prove the effect of DC nodes, we use a recovery tree with a DC node as depicted in Fig. 4. Let $D^{(1)}$ denote the DC node of 1-SR nodes $V_1^{(1)}$ and $V_2^{(1)}$, $m_{u,d}^{(1)}$ denote the mean LLR value from $D^{(1)}$ to $V_1^{(1)}$. Then, $m_{v,1}^{(1)}$ is represented as $m_{v,1}^{(1)} = m_{u,1}^{(1)} + m_{u,d}^{(1)}$. Each $m_{u,1}^{(1)}$ and $m_{u,2}^{(1)}$ can be obtained similarly to equation (6). After first iteration, $m_{v,1}^{(1)}$ is equal to $m_{u,1}^{(1)}$ because there are no evolved LLR values from $D^{(1)}$ due to the zero-LLR value of 1-SR node $V_2^{(1)}$. However, after second iteration, $m_{u,d}^{(1)}$ is a non-zero LLR value because $V_2^{(1)}$ was already recovered after first iteration. Therefore, $m_{u,d}^{(1)}$ can be represented as

$$\begin{aligned} m_{u,d}^{(1)} &= \phi^{-1}\{1-[1-\phi(m_v^{(0)})]^{d_d^{(1)}+S(C_2^{(1)})-2}\} \\ &= \phi^{-1}\{1-[1-\phi(m_v^{(0)})]^{S(D^{(1)})-S(C_2^{(1)})}\} \end{aligned}$$

where $S(D^{(1)})$ is the total number of unpunctured nodes under the DC node $D^{(1)}$. Conclusively, the mean LLR value $m_{v,1}^{(1)}$ after second iteration becomes

$$\begin{aligned} m_{v,1}^{(1)} =& \phi^{-1}\{1-[1-\phi(m_v^{(0)})]^{S(C_1^{(1)})}\} \\ &+ \phi^{-1}\{1-[1-\phi(m_v^{(0)})]^{S(D^{(1)})-S(C_2^{(1)})}\}. \quad (8) \end{aligned}$$

Based on (5) and (8), we can conclude that the DC node gives positive effect (i.e., make $m_{v,1}^{(k)}$ evolve faster) to the k-SR node after all punctured nodes under the DC node are recovered. Also, we can easily know that the more number of DC nodes makes the lower error probability similarly to the multiple SC nodes case.

IV. PROPOSED PUNCTURING ALGORITHM FOR LDPC CODES

In this section, we present a new puncturing algorithm that maximizes the number of SC nodes and DC nodes. The new puncturing algorithm consists of *grouping* and *ordering* algorithms. In the previous section, we showed a higher number of SC nodes and DC nodes results in better performance. To implement this idea, we try to select as many SC nodes as possible from check nodes by a grouping algorithm. Then the selected punctured nodes are ordered according to their number of SC nodes and DC nodes by the ordering algorithm. Further information on the new algorithm is detailed below.

Proposed Puncturing Algorithm < **Definition** >

1. Parity-check matrix of a mother code is given by $\mathbf{H}$, of which size is $m \times n$.
2. Categorize the variable nodes according to their edge distributions into multiple subsets, $V_0, V_1, \cdots, V_{d_{v,\max}}$, where V_0 represents the set of variable nodes which have the lowest number of edges and $V_{d_{v,\max}}$ represents the set of variable nodes which have the highest number of edges.
3. Define the variable node sets $G_0, G_1, \cdots$, such as G_0 is the set of unpunctured nodes,G_1 is the set of 1-SR nodes, and G_k is the set of k-SR nodes.
4. Define the check node sets $C_0, C_1, \cdots$, such as C_0 is the set of unselected check nodes (neither SC nodes nor DC nodes) and C_k is the set of SC nodes of k-SR variable nodes.

< **Initialization** >

5. Set the k-SR variable node set $G_k = \phi$, where $k \geq 1$ and set G_0 as a set of all variable nodes, where ϕ denotes an empty set.
6. Set the k-SR SC node set $C_k = \phi$, where $k \geq 1$ and set C_0 as a set of all check nodes.
7. Set the variable node edge degree level $i = 0$, the number of shared SC nodes $l = 0$, the recovery level $k = 1$, the ordering index $p = 1$, and $P = \phi$.

< **Grouping** >

8. Select a variable node v from $v \in (G_0 \cap V_i)$, which satisfies following three conditions. If there are more than one variable nods satisfying the following three conditions, pick one randomly. If there is no variable node which satisfies following three conditions, go to step 12.
   (a) $|N^v \cap C_k| = l$, where $|\cdot|$ is a cardinality of a set, $N^v = \{c|H_{c,v} = 1, \text{ and } 1 \leq c \leq m\}$(i.e., N^v is a set of the neighboring check nodes of variable node v), and $H_{c,v}$ is the element of c th row and v th column of the parity check matrix $\mathbf{H}$.
   (b) $|N^v \cap C_0| \geq 1$ (i.e., at least one SC node for the variable node v).
   (c) $(N^{v'} \cup C_{k'}) \not\subseteq N^v$, where $1 \leq k' \leq k$ and $\forall v' \in G_{k'}$, where $v' \neq v$ (i.e., at least one SC node for the previously punctured variable nodes).

```
9. $G_0 = G_0\backslash\{v\}, G_k = G_k \cup \{v\}$, and $P = P \cup \{v\}$ (i.e., update $k$-SR node sets and SC node sets ).
10. For check nodes which are the elements of $N^v$
    (a) If $c \in N^v \cap C_k$, then $C_k = C_k\backslash\{c\}$.
    (b) If $c \in N^v \cap C_0$, then $C_0 = C_0\backslash\{c\}$ and $C_k = C_k \cup \{c\}$.
    (c) If $c \in N^v \cap C_{k'}$, where $1 \le k' \le k$ then $C_{k'} = C_{k'}\backslash\{c\}$.
11. Go to step 8 (i.e., repeat the selection of variable node).
12. If $l = 0$, then $i = i + 1$. Else if $l \neq 0$, then $i = i - 1$.
13. If $0 \le i \le d_{v,\max}$, go to step 8.
  < Ordering >
14. Choose the node $v$ in $P$ which has max $|N^v \cap C_k|$(i.e., maximum number of survived checks). If there are more than one variable nodes with the same conditions, pick one which has max $|N^v|$.
15. Assign present order $p$ to selected node $v$.
16. $P = P\backslash\{v\}$ and $p = p + 1$.
17. If $P = \phi$, go to step 20 (i.e., restart grouping).
18. If $P \neq \phi$, go to step 15 (i.e., repeat ordering).
  < Parameter setting >
19. Set $l = l + 1$ and $i = d_{v,\max}$.
20. If $l < \max|N^v|$ for all $v$, go to step 8.
21. If $l = \max|N^v|$, set $k = k+1, l = 0$ and set $C_0$ as a set of all check nodes. Go to step 8.
```

It is assumed that a mother code is given with $m \times n$ parity check matrix $\mathbf{H}$ in step 1. In step 2, we categorize all variable nodes in each V_i according to the degree(i.e., number of connected edges). In step 3 and 4, G_k and C_k are defined as the set of k-SR nodes and the set of SC nodes of k-SR nodes respectively. G_0 is the set of unpunctured nodes and C_0 is the set of check nodes that are unselected for either SC nodes or DC nodes up to the current recovery level. Steps 5 to step 7 initialize parameters. In step 7, i denotes the variable edge degree, k denotes the recovery level, l denotes the number of shared SC nodes by two different variable nodes, and p denotes the puncturing order.

In step 8, we choose one variable node in V_i satisfying three conditions as a k-SR node. In the first condition 8-(a), we choose the node that has the lowest number of SC nodes shared by other variable nodes to maximize the number of SC nodes and to use more check nodes in $\mathbf{H}$ as SC nodes, since more check nodes give a greater chance of more SC nodes. The set N^v denotes the neighboring check nodes of variable node v. The second condition 8-(b) and the third condition 8-(c) are required to guarantee at least one SC node of the currently selected punctured node and previously selected punctured nodes. Otherwise the punctured node would not be recovered at the kth iteration. Steps 9 and 10 update the k-SR node set and SC node sets after choosing one k-SR node v. Selecting a variable node is repeated until there is no variable node in V_i satisfying the three conditions in step 8. If no more nodes of V_i satisfy the condition in Step 8, the value of i is changed according to l. If $l = 0$ (i.e., the first time grouping in the k-SR node group), i increases from 0 to $d_{v,\max}$ to maximize the number of elements in the lower recovery level. Conversely, if $l > 0$, i decreases from $d_{v,\max}$ to 0 in order to maximize the number of SC nodes, since nodes with more edges may have more SC nodes.

Steps 14–18 assign ordering numbers to the selected punctured nodes. The node with the greater number of SC nodes is assigned to a lower ordering number. If two or more variable nodes are selected, we choose the node with more DC nodes. When we select a puncturing pattern for a desired code rate, the punctured nodes with the lower ordering number are selected first. After ordering, the number of shared SC nodes is updated $l = l + 1$, and the selection of a variable node is repeated until there is no variable node in V_i satisfying the three conditions in step 8. If l becomes the maximum variable degree, the grouping of the present k-SR nodes ends. Then, we set $k = k + 1$ and perform grouping and ordering for the (k + 1)-SR node. Whole algorithm ends when no variable node is selected in step 8 for the newly updated (k + 1)-SR node.

$$H = \begin{bmatrix} P_{0,0} & P_{0,1} & P_{0,2} & \cdots & P_{0,n_b-2} & P_{0,n_b-1} \\ P_{1,0} & P_{1,1} & P_{1,2} & \cdots & P_{1,n_b-2} & P_{1,n_b-1} \\ P_{2,0} & P_{2,1} & P_{2,2} & \cdots & P_{2,n_b-2} & P_{2,n_b-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ P_{m_b-1,0} & P_{m_b-1,1} & P_{m_b-1,2} & \cdots & P_{m_b-1,n_b-2} & P_{m_b-1,n_b-1} \end{bmatrix}$$

Fig. 5. Parity-check matrix of QC-LDPC codes.

V. BLOCK-TYPE LDPC CODES

In this section, we apply the new puncturing algorithm to the block-type LDPC codes. It is well known that LDPC codes have complex encoding problem and high memory requirement problem as their disadvantages. The block-type LDPC codes have been successfully developed as a solution of those problems. We will show that the proposed puncturing algorithm is also valid for the block-type LDPC codes without any modification.

A. Parity-Check Matrices of the QC-LDPC Codes

We can describe the parity-check matrix of QC-LDPC codes as shown in Fig. 5. The matrix $\mathbf{H}$ is expanded from a binary base matrix $\mathbf{H}_b$ of size $m_b \times n_b$. In the base matrix $\mathbf{H}_b$, each element $(H_b)_{i,j}$ is represented by 1 or 0. In Fig. 5, $\mathbf{P}_{i,j}$ is the circulant permutation matrix which is a shifted identity matrix or a zero matrix where $0 \le i \le m_b - 1$ and $0 \le j \le n_b - 1$. We define the size of $\mathbf{P}_{i,j}$ as $z \times z$, with an integer $z \ge 1$. Nonzero $\mathbf{P}_{i,j}$ is obtained by $q_{i,j}$ time right shifted identity matrix $(0 \le q_{i,j} < z)$. Let n be the codeword size and m be the parity bit size. Then, $\mathbf{H}$ is a $m \times n$ matrix where $m = m_b z$ and $n = n_b z$. Although the base matrix offers us information about the basic pattern of the expanded parity-check matrix, it does not show the forms of the right-shifted permutation matrices. A model matrix shows not only the information offered by a base matrix, but also the right-shifted value. In the model matrix, the shifted number is shown in the location of a permutation matrix and a negative number or blank is used in a zero matrix location.

B. Block-Type LDPC Codes

B-LDPC codes are kinds of QC-LDPC codes where degree distributions are irregular (i.e., dual diagonal parity part) for fast encoding. We can divide the base matrix of the B-LDPC codes into two parts, where the systematic part $\mathbf{H}_{b1}$ and the parity part

$\mathbf{H}_{b2}$ are such that

$$\mathbf{H}_b = [(\mathbf{H}_{b1})_{m_b \times k_b} | (\mathbf{H}_{b2})_{m_b \times m_b}]. \tag{9}$$

In (9), k_b means the size of the systematic part in the base matrix. $\mathbf{H}_{b2}$ is partitioned into two parts. One is a column vector $\mathbf{h}_b$ where the column weight is 3. The other part is a matrix $\mathbf{H}'_{b2}$ which has a dual diagonal structure. A vector $\mathbf{h}_b$ is inserted to prevent that the right-margin of $\mathbf{H}'_{b2}$ having a column weight 1.

There are many advantages using B-LDPC codes. First, it is possible to do a block-unit operation when performing the encoding and decoding processes. That is, a system needs not know the entire information of the parity-check matrix; it only needs the information of the model matrix. In that case, it is possible to operate with the entire information using shift registers. We can reduce much of the memory by a factor of $1/z$. Second, codewords of various sizes are easily encoded using only one model matrix with various sizes of z. Third, fast and simple encoding is available based on the Richardson-Urbanke encoding method instead of Gaussian elimination to generate a matrix, because of the structure of the parity part $\mathbf{H}_{b2}$ that is a dual diagonal type [11], [14].

C. Puncturing of Block-type LDPC Codes

The analytical results in Section III are good for all kinds of LDPC codes. However, in B-LDPC codes, it is effective to puncture the nodes on a block-unit. Hence, we consider the block-unit puncturing for B-LDPC codes. It can be shown that the results in Section III hold for block-unit puncturing as follows.

Theorem 1: When $\mathbf{H}_b$ and $\mathbf{H}$ denote a binary base matrix and the corresponding expanded parity check matrix respectively, variable nodes in $\mathbf{H}$ corresponding to a k-SR node in $\mathbf{H}_b$ are also k-SR nodes and check nodes in $\mathbf{H}$ corresponding to a SC (DC) node in $\mathbf{H}_b$ are also SC (DC) nodes.

Proof: To prove this theorem, we use a simple tree structure. Fig. 6(a) is the tree structure of k-SR node $V_a^{(k)}$ in a base matrix $\mathbf{H}_b$. It can be easily noticed that the check node $C^{(k)}$ is the SC node of $V_a^{(k)}$ and $D^{(k)}$ is the DC node of $V_a^{(k)}$ due to another k-SR node $V_b^{(k)}$. Assuming that z is two, an expanded tree structure in parity check matrix $\mathbf{H}$ is as shown in Fig. 6(b). In the expanded structure, each node in Fig. 6(a) becomes two corresponding nodes. For example, k-SR node $V_a^{(k)}$ is expanded into $V_{a,1}^{(k)}$ and $V_{a,2}^{(k)}$. Because each block $\mathbf{P}_{i,j}$(Fig. 5) is based on an right shifted identity matrix, which has only one weight at each row and column, a connecting structure between variable nodes and check nodes in a block is one-to-one correspondence (i.e., if $V_a^{(k)}$ is connected to $C^{(k)}$ in $\mathbf{H}_b$, $V_{a,1}^{(k)}$ and $V_{a,2}^{(k)}$ are connected to $C_1^{(k)}$ and $C_2^{(k)}$ in $\mathbf{H}$ respectively). Also, because each $\mathbf{P}_{i,j}$ has different shifting value $q_{i,j}$, an expanded tree structure becomes a tangled form. Therefore, if we untie the connections, we can get two separate tree structures as depicted in Fig. 6(c) and each tree structure has an identical form comparing with the tree structure in Fig. 6(a). This means that variable nodes in $\mathbf{H}$ corresponding to a k-SR node in $\mathbf{H}_b$ are also k-SR nodes and check nodes in $\mathbf{H}$ corresponding to a SC (DC) node in $\mathbf{H}_b$ are also SC (DC) nodes. We can extend this proof for any value of z. □

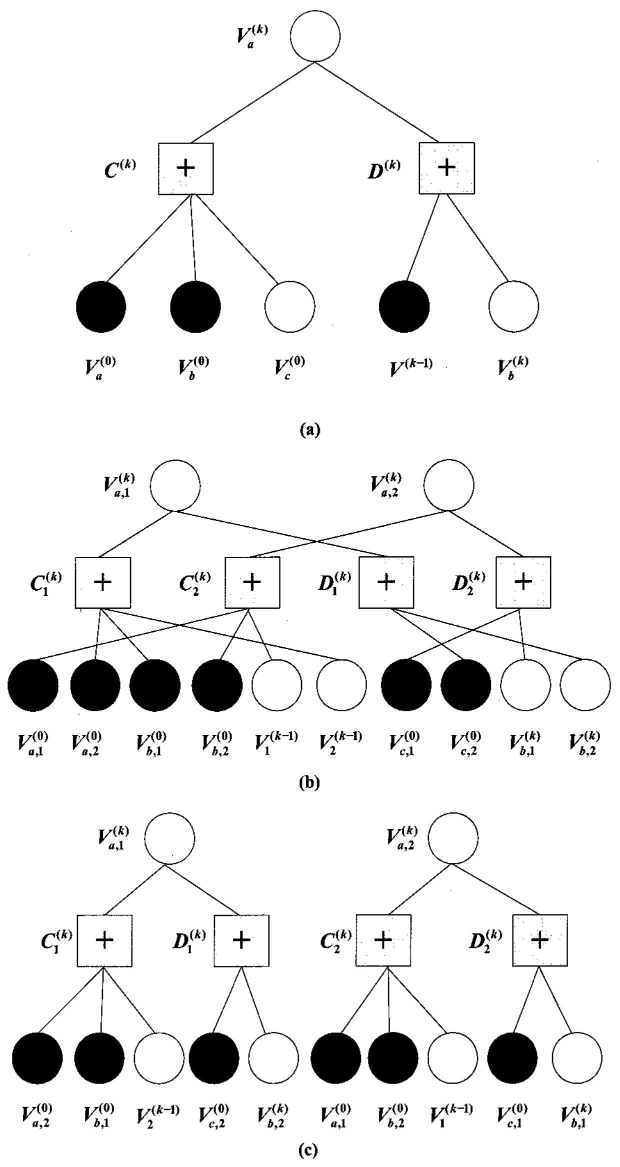

$V_a^{(k)}$
$C^{(k)}$
$D^{(k)}$
$V_a^{(0)}$ $V_b^{(0)}$ $V_c^{(0)}$ $V^{(k-1)}$ $V_b^{(k)}$
(a)
$V_{a,1}^{(k)}$ $V_{a,2}^{(k)}$
$C_1^{(k)}$ $C_2^{(k)}$ $D_1^{(k)}$ $D_2^{(k)}$
$V_{a,1}^{(0)}$ $V_{a,2}^{(0)}$ $V_{b,1}^{(0)}$ $V_{b,2}^{(0)}$ $V_1^{(k-1)}$ $V_2^{(k-1)}$ $V_{c,1}^{(0)}$ $V_{c,2}^{(0)}$ $V_{b,1}^{(k)}$ $V_{b,2}^{(k)}$
(b)
$V_{a,1}^{(k)}$ $V_{a,2}^{(k)}$
$C_1^{(k)}$ $D_1^{(k)}$ $C_2^{(k)}$ $D_2^{(k)}$
$V_{a,2}^{(0)}$ $V_{b,1}^{(0)}$ $V_2^{(k-1)}$ $V_{c,2}^{(0)}$ $V_{b,2}^{(k)}$ $V_{a,1}^{(0)}$ $V_{b,2}^{(0)}$ $V_1^{(k-1)}$ $V_{c,1}^{(0)}$ $V_{b,1}^{(k)}$
(c)

Fig. 6. Example of tree structures of k-SR node and its SC (DC) node. (a) tree structure of k-SR node and its SC (DC) node in the base matrix H_b, (b) example of expanded tree structure of k-SR nodes and its SC (DC) nodes in the expanded matrix **H**, and (c) untied form of (b).

Based on Theorem 1, the problem of finding good puncturing patterns in an expanded parity-check matrix $\mathbf{H}$ is equivalent to the problem of finding good puncturing patterns in a base matrix $\mathbf{H}_b$. Puncturing of one column (variable node) in $\mathbf{H}_b$ means the puncturing of corresponding z-columns (variable nodes) in $\mathbf{H}$. When using block-unit puncturing, we can have the following advantages:

- Simpler block-unit puncturing is available during the encoding and decoding processes.
- The memory for storing puncturing patterns can be reduced by a factor of $1/z$.
- Punctured nodes can be selected regardless of the block length only using one puncturing pattern on a base matrix.

Table 1. Three different puncturing patterns (single SC, grouping, proposed) for the irregular LDPC code. PI denotes the punctured node index and SC denotes the number of corresponding SC nodes.

Single SC		Grouping		Proposed	
PI	SC	PI	SC	PI	SC
14	1	13	2	14	2
15	1	16	2	16	2
17	1	18	1	18	2
18	1	19	1	20	2
20	1	21	1	22	2
21	1	22	1	24	2

VI. SIMULATIONS

In this section, we first compare the performance of LDPC code with different number of SC nodes in the AWGN channel. The irregular B-LDPC code in [12], with the variable node edge degree distribution $\lambda(x) = 0.2895x + 0.3158x^2 + 0.3947x^5$ and the check node edge degree distribution $\rho(x) = 0.6316x^5 + 0.3684x^6$ is considered as mother code. The base matrix size is fixed as 12×24. For B-LDPC codes, we know that block-unit puncturing is effective for both performance and complexity. Hence, we used block-unit puncturing for the numerical result. In this simulation, we used a sum-product belief-propagation (BP) decoding algorithm with a maximum of 50-iterations. The information size is 1080 bits. Based on the $1/2$ mother code, $2/3$ rate codes were obtained by puncturing six blocks. We considered three different puncturing patterns (Table 1). The '*Single SC*' means the puncturing pattern was purposely chosen to have a single SC node at each punctured node and the '*Grouping*' means the puncturing patterns following the algorithm in [4] that selects punctured nodes maintaining the recovery step value as low as possible. A grouping algorithm in [4] makes each punctured node have at least one SC node, but it does not guarantee a maximal number of SC nodes. Meanwhile, the '*Proposed*' is the puncturing pattern maximizing the number of SC nodes. The bit error rate (BER) and frame error rate (FER) performance curves are shown in Fig. 7. The number of SC nodes corresponding to each puncturing pattern are shown in Table 1. The number of survived check nodes for the '*Single SC*' and the '*Grouping*' are six and eight, respectively. Conversely, the number of survived check nodes for the '*Proposed*' is twelve, the maximum possible value on the given code rate and base matrix size. The puncturing pattern of the '*Proposed*' can be easily obtained by uniform puncturing every other parity node. As shown in Fig. 7, the '*Proposed*' shows the best performance and the '*Single SC*' shows the worst performance. It is noted that these numerical results match the analytical results using DE in the previous section. The numerical result according to the number of DC nodes is shown in Fig. 8. The simulation is performed for the same irregular B-LDPC code in [12] based on the sum-product BP decoding algorithm with a maximum of 50-iterations. The information size is 1080 bits. We compare the puncturing patterns that have the same number of SC nodes and different number of DC nodes to concentrate on the DC node effect. It is noted that the performance is superior with more DC nodes with the same number of SC nodes.

Next, we compare the performance of the proposed punc-

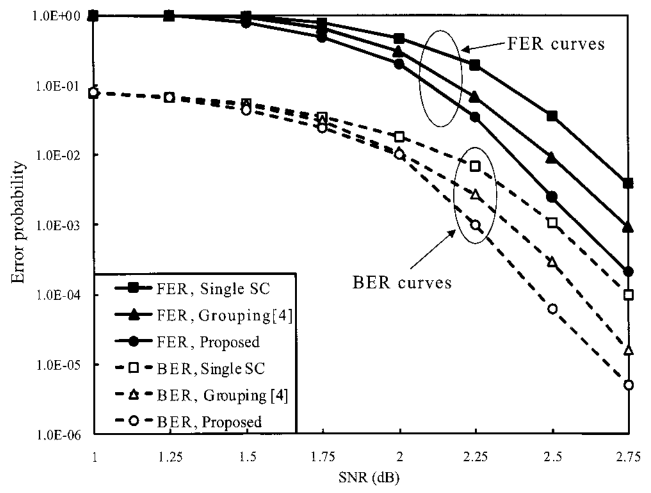


Fig. 7. BER and FER performance curves with different puncturing patterns for the irregular LDPC code with rate 2/3 and information length 1080 bits. The number of SC nodes corresponding to each puncturing pattern are shown in Table 1.

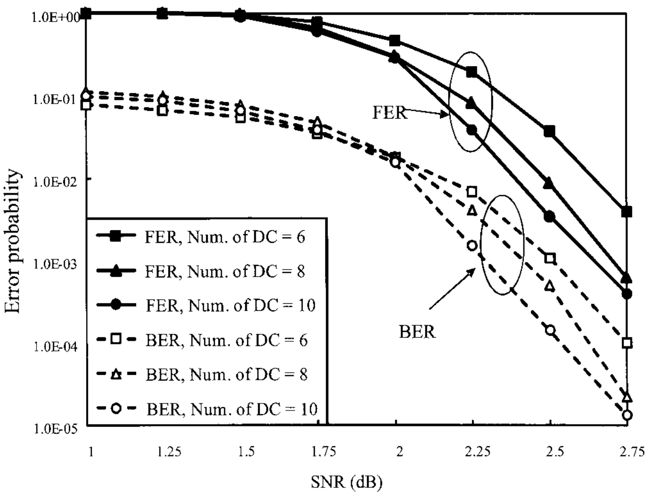


Fig. 8. BER and FER performance curves with the same number of SC nodes and different number of DC nodes for the irregular LDPC code with rate 2/3 and information length 1080 bits. (Num. of DC denotes the number of DC nodes).

turing algorithm with that of previous *grouping* algorithm at various code rates. In this simulation, we use two different LDPC codes. One is the 1/2 rate (3,6)-regular LDPC code, in which the parity check matrix is constructed by the progressive edge-growth (PEG) algorithm [15], and the other is the irregular block-type LDPC code in [12]. We considered 1080 information bit size and code rates of 0.6, $2/3$, 0.7, and 0.75. Both bit puncturing and block-unit puncturing are implemented. Table 2 shows the number of variable nodes at each recovery group for both the proposed algorithm and the previous algorithm. Table 3 shows the number of punctured nodes(bits or blocks) for various code rates, respectively. The number of SC nodes(bits or blocks) is also shown. The proposed algorithm has more SC nodes than the previous *grouping* algorithm. The number of punctured nodes of the proposed algorithm at a lower recovery level is less than that of previous algorithm. That is, the proposed algorithm has more SC nodes, while puncturing vari-

Table 2. Number of variable nodes (bits or blocks) at each recovery group in the irregular LDPC code of 1080 information bits.

Group	Grouping		Proposed	
	Bit	Block	Bit	Block
0-SR	1373	15	1260	14
1-SR	679	8	540	6
2-SR	105	1	270	3
3-SR	3	0	90	1
Total	2160	24	2160	24
Highest rate	0.787	0.8	0.857	0.857

Table 3. Number of punctured nodes and their corresponding SC nodes for desired code rates when using bit (block) puncturing in the irregular LDPC code of 1080 information bits (numbers in parentheses mean the number of blocks when block puncturing).

	Rate=0.6		Rate=2/3	
	Grouping	Proposed	Grouping	Proposed
1-SR	360 (4)	360 (4)	540 (6)	540 (6)
2-SR	0 (0)	0 (0)	0 (0)	0 (0)
No. of SC	633 (7)	720 (8)	765 (8)	1080 (12)
	Rate=0.7		Rate=0.75	
	Grouping	Proposed	Grouping	Proposed
1-SR	630 (7)	540 (6)	679 (8)	540 (6)
2-SR	0 (0)	90 (1)	41 (0)	180 (2)
No. of SC	841 (8)	1080 (12)	944 (8)	1080 (12)

able nodes of the higher recovery level. At lower rates, most of punctured nodes belong to the 1-SR group for both the proposed and previous algorithm. Therefore, the proposed algorithm with more SC nodes performs better at lower rates. At higher rates, the advantage of more SC nodes is compensated by the disadvantage of puncturing higher recovery level nodes. Therefore, the performance of the proposed algorithm is very close to that of previous algorithm. FER performance curves of regular and irregular punctured LDPC codes are shown in Figs. 9 and 10, respectively. In both figures, we can find that the performance of proposed puncturing algorithm is better than that of grouping algorithm at 0.6 and 2/3 rates. In contrast, the performance of the proposed puncturing algorithm is almost the same as that of grouping algorithm at 0.7 and 0.75 rates. In conclusion, the '*Proposed*' algorithm is more effective for the relatively lower code rate with fewer number of punctured nodes. Furthermore, the '*Proposed*' algorithm has another advantage, the highest achievable code rate by puncturing is higher than that of previous '*Grouping*' algorithm. For the irregular LDPC code in [12], the highest achievable code rate by the proposed puncturing was 0.857 and the highest achievable code rate by the grouping algorithm was 0.8. This expands the range of code rate compatibility, one of important factors for a rate compatible code.

VII. CONCLUSION

In this paper, we used the DE technique to analyze the positive effect of multiple survived check nodes and dead check nodes. We showed that the mean of LLR passed between variable nodes and check nodes converges faster as the number of survived check nodes and the number of dead check nodes increases. We also showed that the analysis results are valid for both bit-unit and block-unit puncturing. Based on these analytical results, a new algorithm to choose good puncturing patterns for regular and irregular LDPC codes was proposed. It maximizes the number of SC nodes and DC nodes. Numerical results on the regular (3,6) LDPC code and the irregular LDPC code in IEEE P802.16e/D9 matched the analytical results. It is noted that the proposed algorithm performs better than the previous puncturing algorithm at medium and low rates.

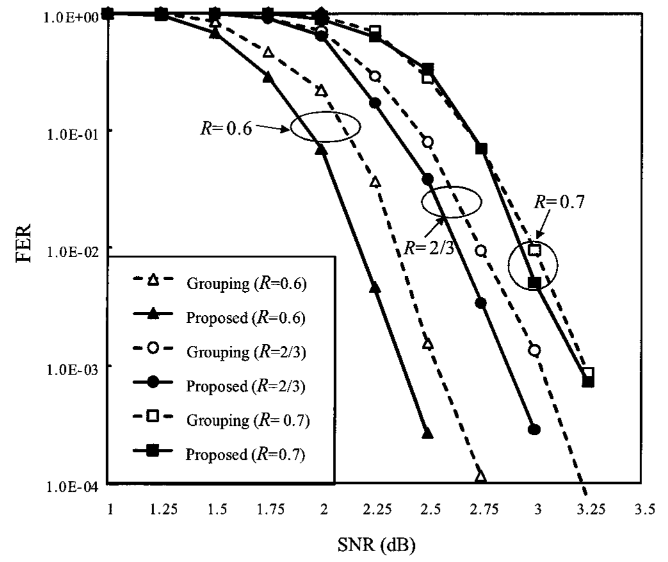


Fig. 9. FER performance curves for the regular (3,6) LDPC code with rates 0.6, 2/3, 0.7 respectively (information length=1080 bits and bit puncturing).

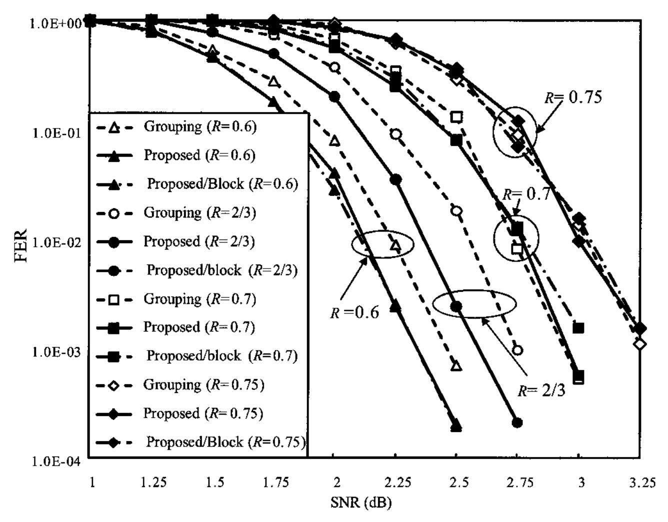


Fig. 10. FER performance curves for the irregular LDPC code with rates 0.6, 2/3, 0.7, 0.75 respectively (information length=1080 bits). *Grouping* means the previous algorithm with bit puncturing, *Proposed* means the new algorithm with bit puncturing, and *Proposed/BLOCK* means the new algorithm with block puncturing.

REFERENCES

[1] J. Hagenauer, "Rate-compatible punctured convolutional codes (rcpc codes) and their applications," *IEEE Trans. Commun.*, vol. 36, pp. 389–400, Apr. 1988.

[2] J. Li and K. Narayanan, "Rate-compatible low-density parity-check codes for capacity-approaching ARQ scheme in packet data communications," in *Proc. CIIT*, Nov. 2002.

[3] T. Tian and C. R. Jones, "Construction of rate-compatible LDPC codes uti-

lizing information shortening and parity puncturing," *EUR. Wireless Commun. Netw.*, vol. 5, pp. 1–7, 2005.

[4] 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.

[5] D. Klinc, J. Ha, J. Kim, and S. W. McLaughlin, "Rate-compatible punctured low-density parity-check codes for ultra wide band systems," in *Proc. IEEE GLOBECOM*, Nov. 2005, pp. 3856–3860.

[6] E. Y. Choi, S. B. Suh, and J. H. Kim, "Rate-compatible puncturing for low-density parity-check codes with dual-diagonal parity structure," in *Proc. IEEE PIMRC*, Sept. 2005, pp. 2642–2646.

[7] S. H. Choi, Y. S. Shin, J. Heo, K. H. Cho, and M. S. Oh, "Effective puncturing schemes for block-type low-density parity-check codes," in *Proc. IEEE VTC-spring*, Apr. 2007, pp. 1841–1845.

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

[9] T. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inf. Theory*, vol. 47, pp. 599–618, Feb. 2001.

[10] S. Y. Chung, T. J. Richardson, and R. L. Urbanke, "Analyisis of sum-product decoding of low-density parity-check codes using a Gaussian approximation," *IEEE Trans. Inf. Theory*, vol. 47, pp. 657–670, Feb. 2001.

[11] S. H. Myung, K. C. Yang, and J. Y. Kim, "Quasi-cyclic ldpc codes for fast encoding," *IEEE Trans. Inf. Theory*, vol. 51, pp. 2894–2901, Aug. 2005.

[12] I. . W. Group, "Part 16: Air interface for fixed and mobile broadband wireless access systems," 2005.

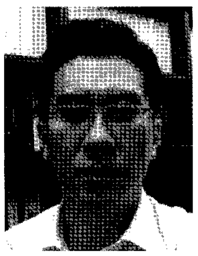
[13] C. Zheng, N. Miyazaki, and T. Suzuki, "Rate compatible low-density parity-check codes based on progressively increased column weights," *IEICE Trans. Fundamentals*, vol. E89, pp. 2493–2500, Oct. 2006.

[14] T. Richardson and R. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 47, pp. 638–656, Feb. 2001.

[15] X. 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.

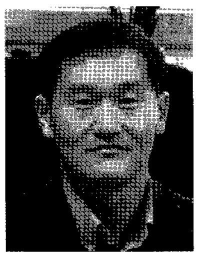
Sunghoon Choi received the B.S. degrees in Electronics Engineering from Konkuk university, Seoul, Korea in 2005. He is presently an M.S. student in electronics engineering from Konkuk university, Seoul, Korea. His research interests include LDPC codes and Turbo codes. He is presently with the LG electronics Inc., Seoul, Korea.

Sungroh Yoon received the B.S. degree in Electrical Engineering form Seoul National University, Seoul, Korea, in 1996 and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, USA in 2002 and 2006, respectively. From 2006 to 2007, he was with Intel Corporation, Santa Clara, USA, where he participated in developing Intel Atom and Core i7 microprocessors. Previously he held research positions at Stanford University and Synopsys Inc., Mountain View, USA. He is currently an Assistant Professor of Electrical Engineering at Korea University, Seoul, Korea.



Wonjin Sung received his B.S. degree from Seoul National University, Korea in 1990, and the M.S. and Ph.D. degrees in Electrical Engineering from University of Michigan, Ann Arbor, MI, in 1992 and 1995, respectively. From January 1996 through August 2000, he worked at Hughes Network Systems, Germantown, MD, USA, where he participated in development projects for cellular systems including the IS-136 base station modems and low-complexity channel decoders. Since September 2000, he has been with the Department of Electronic Engineering at Sogang University, Seoul, Korea, where he is currently an Associate Professor. His research interests are in the areas of mobile wireless transmission, statistical communication theory, and channel coding.

Hongkyu Kwon received the B.S. degree in Industrial Engineering from Dongguk University, Seoul, Korea in 1995 and the M.S. and Ph.D. degrees in industrial system engineering from the University of Southern California, Los Angeles, USA in 1998 and 2002, respectively. During 2002–2005, he was a Senior and Chief Research Engineer at LG Electronics production research center. During the 2005–2007, he was an Adjunct Professor in the Industrial Management Engineering dept., Chungju National University, Chungju, Korea. He is presently an Assistant Professor in the industrial management engineering dept., Namseoul University, Chunan, Korea. His research interests include CAD/CAM, rapid prototyping, and digital manufacturing and production.



Jun Heo received the B.S. and M.S. degrees in Electronics Engineering from Seoul National University, Seoul, Korea in 1989 and 1991, respectively and the Ph.D. degree in Electrical Engineering from the University of Southern California, Los Angeles, USA in 2002. During the 1991–1997, he was a senior research engineer at LG Electronics Co., Inc. During the 2003–2006, he was an Assistant Professor in the Electronics Engineering Department, Konkuk University, Seoul, Korea. He is presently an Associate Professor in the School of Electrical Engineering at Korea University, Seoul, Korea. His research interests include channel coding theory and digital communication systems.