Efficient Puncturing Method for Rate-Compatible Low-Density Parity-Check Codes

Hyo Yol Park, *Student Member, IEEE,* Jae Won Kang, Kwang Soon Kim†, *Senior Member, IEEE,* and Keum Chan Whang

Abstract— **In this paper, we propose an efficient puncturing method for LDPC codes. The proposed algorithm provides the order of variable nodes for puncturing based on the proposed cost function. The proposed cost function tries to maximize the minimum reliability among those provided from all check nodes. Also, it tries to allocate survived check nodes evenly to all punctured variable nodes. Furthermore, the proposed algorithm prevents the formation of a stopping set from the punctured variable nodes even when the amount of puncturing is quite large. Simulation results show that the proposed punctured LDPC codes perform better than existing punctured LDPC codes.**

Index Terms— **Low-density parity-check code, rate-compatible code, puncturing.**

I. Introduction

LOW-density parity-check (LDPC) codes have very powerful and promising advantages in both performance and complexity over turbo codes. Because of such advantages, IEEE 802.16e recently adopted LDPC codes as optional forward error correction (FEC) codes [1]. In existing third-generation cellular system standards, rate-compatible punctured turbo (RCPT) codes have been adopted for adaptive modulation and coding (AMC) and hybrid automatic repeat request (HARQ) techniques [2], which can enhance the system performance. Those RCPT codes can support various code rates from a single mother code by applying appropriate puncturing patterns. However, in LDPC codes, such a puncturing technique has not yet been well established, which has been considered a major drawback of LDPC codes compared to turbo codes. Recently, there have been some efforts to develop solutions for rate-compatible punctured LDPC codes [3]–[7]. In [3], capacity-approaching puncturing distributions were proposed for irregular LDPC codes. However, the puncturing distributions cannot determine the exact locations of punctured bits and are valid only when the codeword length is infinite and there is no cycle in the graph associated with the code. In [4], the structure of a parity check matrix for a rate-compatible punctured LDPC code was proposed. However, it cannot support various code rates and cannot determine the exact puncturing locations, either. In [5], a group-wise puncturing method was proposed by maximizing the number of k-step recoverable (k-SR) nodes at the lowest value of k (denoted as the minimum k-SR nodes in the sequel) and puncturing the group of the same k-SR nodes for practical regular LDPC codes. Here, a k-SR node denotes a punctured variable node that can be recovered at the kth iteration. However, this algorithm cannot provide bit-wise puncturing and does not take the recovery reliability of a k-SR node into account. Later in [6], the number of survived check nodes of a k-SR node was considered based on the fact that a k-SR node with more survived check nodes can be recovered more reliably. However, it can be applied only to an LDPC code with a dual-diagonal block structure and does not take the recovery reliability fully into account. In [7], it was shown that a punctured node with a smaller number of unpunctured variable nodes (denoted as 0-SR nodes in the sequel) in its recovery tree is very likely to have higher recovery reliability. Taking such recovery reliability into account, the algorithm in [7] tries to maximize the number of the minimum k-SR nodes (i.e, to minimize the number of iterations required to recover a punctured node). After that, starting from the minimum k-SR nodes, the order of a bit-wise puncturing among the same k-SR nodes is determined by trying to allocate more survived check nodes to the variable nodes to be punctured earlier. However, trying to maximize the recovery reliability of the punctured nodes cannot directly improve the overall decoding performance because it does not consider the fact that a check node cannot improve the reliability of the neighboring 0-SR nodes until all the neighboring punctured nodes are recovered. In addition, as the amount of puncturing becomes larger, the increasing number of survived check nodes of one k-SR node reduces the number of survived check nodes of other k-SR nodes, which results in poor recovery reliability of the other k-SR nodes.

In this paper, we propose an efficient puncturing algorithm for LDPC codes that does not try to minimize the number of iterations required to recover each punctured node and maximize the recovery reliability of punctured nodes as in [7], but instead tries to maximize the minimum reliability provided among check nodes (the reliability provided from a check node is denoted as the initial reliability of the check node in the sequel). For this purpose, a pruned recovery tree is newly defined and used. In addition, the proposed algorithm can provide similar recovery reliability to all punctured nodes by trying to allocate survived check nodes evenly to all punctured nodes by reserving only one survived check node for each punctured node. Furthermore, the proposed algorithm provides an efficient method for preventing punctured nodes

Manuscript received June 19, 2006; revised January 21, 2007, March 22, 2007, and May 16, 2007; accepted July 25, 2007. The associate editor coordinating the review of this paper and approving it for publication was A. Grant.

The authors are with the Department of Electrical and Electronic Engineering, Yonsei University, 134 Shinchon-dong, Seodaemun-gu, Seoul 120-749, Korea.

† Corresponding author (e-mail: ks.kim@yonsei.ac.kr).

Digital Object Identifier 10.1109/TWC.2007.060386.

from forming a stopping set. The proposed algorithm can be applied to any LDPC code because it does not assume any structure on the parity check matrix. To prove the usefulness of the proposed algorithm, we will obtain puncturing patterns of some LDPC codes and will show that the frame error rate (FER) performance of the proposed punctured LDPC codes is better than that of previously known punctured LDPC codes.

II. THE PROPOSED PUNCTURING ALGORITHM

We define some notations that will be used throughout this paper as follows:

- $\mathbf{V}$ ($\mathbf{C}$): the set of all variable (check) nodes.
- $\mathbf{V}_c$ ($\mathbf{C}_v$): the set of the neighbor variable (check) nodes of a check node c (variable node v).
- N_p: the total number of variable nodes to be punctured.
- $SR(v)$: the number of iterations required to recover the punctured node v, i.e, $SR(v) = k$ if v is a k-SR node.
- $SR(c) \triangleq \max_{v\in\mathbf{V}_c} SR(v)$.
- $T_l(c)$: the l-step pruned recovery tree of a check node c, $0 \le l \le SR(c)$.
- $\mathbf{C}_m(T_l(c))$: the set of all check nodes in the $(2m-1)$th level of $T_l(c)$, $1 \le m \le SR(c)+1$.
- $\mathbf{V}_m(T_l(c))$: the set of all variable nodes in the $(2m)$th level of $T_l(c)$, $1 \le m \le SR(c)+1$.
- $\mathbf{C}(T_l(c)) \triangleq \bigcup_{m=1}^{SR(c)+1} \mathbf{C}_m(T_l(c))$: the set of all check nodes in $T_l(v)$.
- $\mathbf{V}(T_l(c)) \triangleq \bigcup_{m=1}^{SR(c)+1} \mathbf{V}_m(T_l(c))$: the set of all variable nodes in $T_l(c)$.
- $\mathbf{V}_{c_1}(T_l(c))$: the set of all 0-SR nodes in $\mathbf{V}(T_l(c))$ spanned from a check node $c_1 \in \mathbf{C}(T_l(c))$.
- $p(v)$: the puncturing status of a variable node v. $p(v) = 0$ (1) when v is punctured (not punctured).
- $R(c)$: the reservation status of a check node c. $R(c) = 1$ (0) when c is reserved (not reserved).
- $\mathbf{C}_0 \triangleq \{c|R(c) = 0\}$.
- $s(v) \triangleq \min_{\overline{c}\in\mathbf{C}_v} |\mathbf{V}_{\overline{c}}(T_q(c))|$ for a q-SR node $v \in \mathbf{V}(T_0(c))$. Note that $s(v)$=1 for a 0-SR node v.
- $C(c) \triangleq \sum_{v\in\mathbf{V}_c} s(v)$: the cost function of a check node c. As $C(c)$ decreases, the initial reliability of c increases.
- $r(v) \triangleq \sum_{c\in\mathbf{C}_v} R(c) + 1$.
- $b(v) \triangleq |\mathbf{C}_v|$, where $|A|$ denotes the cardinality of a set A.
- $k(v) \triangleq \sum_{c\in\mathbf{C}_v} C(c)$.
- $C(v) \triangleq r(v) + b(v)w^{-1} + k(v)w^{-2}$, where the weight w is $2n^2$: the cost function of a variable node v.

The proposed puncturing algorithm assumes a conventional message-passing decoder using belief propagation. At each time to select the next puncturing node, we first construct the $SR(c)$-step pruned recovery tree, $T_{SR(c)}(c)$, for each unreserved check node $c \in \mathbf{C}$ as follows.

1) Construction of $T_0(c)$: An ordinary tree is constructed from the check node c except for the rule that each branch of the tree stops spanning at a 0-SR node. This tree is denoted as the 0-step pruned recovery tree of c.
2) Pruning Procedure: For a given $T_q(c)$, $0 \le q \le SR(c)$, $T_{q+1}(c)$ is constructed as follows. If there exists any set of check nodes $\mathbf{C}_I = \{c_i, i \in I\}$ such that $\mathbf{C}_I \subset \mathbf{C}_{SR(c)-q+1}(T_q(c))$ and $\mathbf{C}_I \subset \mathbf{C}_{v'}$ for any $v' \in \mathbf{V}_{SR(c)-q}(T_q(v))$, only one check node c_{i^*} is selected where $i^* = \arg\min_{i\in I} |\mathbf{V}_{c_i}(T_q(c))|$ and check nodes and their successors are pruned in $T_q(c)$. This process continues until there is no such a set and the remaining tree is $T_{q+1}(c)$.

An example of the pruning procedure is shown in Fig. 1. Here, $\mathbf{C}_0 = \{c_1, c_2\}$, where $SR(c_1) = SR(c_2) = 2$, and the corresponding 0-step recovery trees $T_0(c_1)$ and $T_0(c_2)$, are constructed, respectively. At the first pruning step, there is nothing to prune because each 1-SR node in level 4 has only one child check node. At the second pruning step, the check nodes c_3 and c_5 in level 3 are pruned with their successors because they have more 0-SR nodes as their successors than check nodes c_4 and c_6, respectively. After obtaining the $SR(c)$-step pruned recovery tree $T_{SR(c)}(c)$ for each $c \in \mathbf{C}_0$, $C(c)$ and $C(v)$, $v \in \mathbf{V}_c$, are calculated in each $SR(c)$-step pruned recovery tree $T_{SR(c)}(c)$. Then, the selection of a pair of a check node (to be a reserved survived check node) and a variable node (to be punctured) is performed as follows. At first, construct $\mathbf{C}^*$ as the set of unreserved check nodes c^* with the minimum cost function $C(c)$ among $\forall c \in \mathbf{C}_0$. Also, construct $\mathbf{V}^*$ as the set of variable nodes v^* with the minimum cost function $C(v)$ among the set $\{v|v \in \mathbf{V}_c$ for $c \in \mathbf{C}^*$, $p(v) = 1\}$. If there is only one element in $\mathbf{V}^*$, the variable node $v^* \in \mathbf{V}^*$ and the connected check node $c^* \in \mathbf{C}^* \cap \mathbf{C}_{v^*}$ are selected as the punctured node and the corresponding reserved check node, respectively. If there exists more than one element in $\mathbf{V}^*$, one element is selected arbitrarily and its neighbor check node in $\mathbf{C}^*$ is selected as its reserved check node. If the selected variable node fails the stopping set check, the next candidate is tried, and the process repeats until the stopping set check is satisfied. Note that the reliability provided to the variable nodes in $\mathbf{V}_c$ from the survived check node c after the $SR(c)$-step recovery increases as $C(c)$ decreases. Therefore, it is expected to enhance the overall decoding performance by making $\max_{c\in\mathbf{C}} C(c)$ as small as possible. Also, note that $C(v)$ consists of three functions. Here, the weight w is introduced to combine them into a single cost function. Because $r(v)$, $b(v)$, and $k(v)$ are strictly less than $2n^2$, the cost function is dominated by $r(v)$. If two nodes have the same value of $r(v)$, then the cost function is dominated by $b(v)$, and so on. At first, the cost function selects the variable node with the lowest $r(v)$. The higher the value of $r(v)$ is, the more important the variable node v is because such a variable node should be used for the recovery of more punctured nodes than others. Thus, such a variable node should be punctured later. This concept of reservation basically reduces the probability that punctured nodes form a stopping set. After that, the cost function selects the variable node with the lowest degree (the smallest value of $b(v)$) among the variable nodes with the same $r(v)$. By puncturing the variable node with the lowest $b(v)$, the number of check nodes whose initial reliability decreases ($C(c)$ increases) due to the puncturing is reduced, which helps the recovery reliability of other variable nodes to be punctured. Finally, the cost function selects the variable node with the lowest value of $k(v)$ among the variable nodes with the same $r(v)$ and $b(v)$. By puncturing the variable node with the lowest value of $k(v)$, the maximum

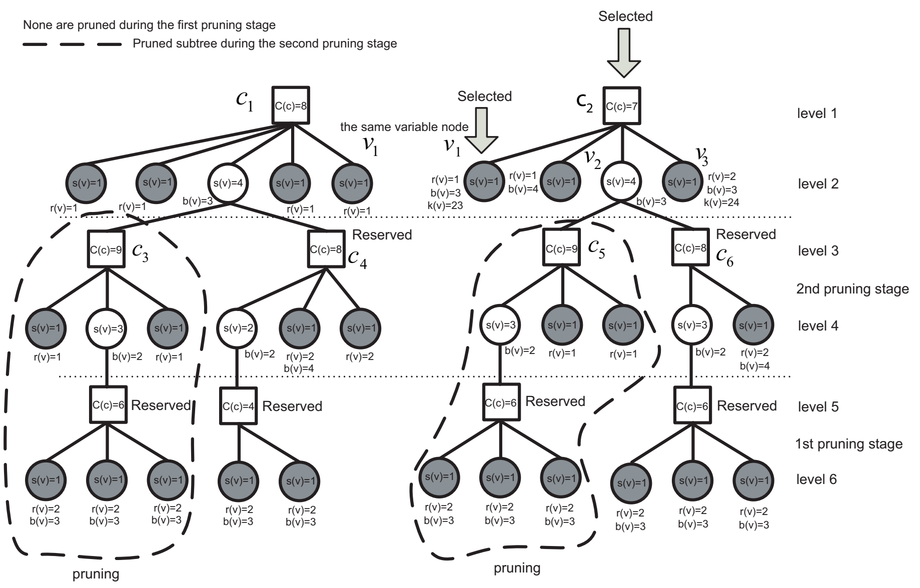

Fig. 1. Examples of the construction and pruning procedure of the pruned recovery tree: the square, the circle, and the gray circle denote a survived check node, an already punctured node, and a 0-SR node, respectively.

value of $C(c)$ tends to be minimized, and $C(c)$'s of all the check nodes tend to be uniform. By using these cost functions, the maximum $C(c)$ among check nodes can be minimized and the survived check nodes can be allocated evenly to the punctured nodes. Also, note that by minimizing $C(c)$, the number of the 0-SR nodes in $T_{SR(c)}(c)$ is very likely to be minimized. In the example shown in Fig. 1, the pair v_1 and c_2 is selected according to the proposed cost functions. Also, it is easily seen that better recovery reliability (fewer 0-SR nodes in the $SR(c)$-step pruned recovery tree) is provided to v_1 by c_2 than by c_1, and the proposed algorithm can determine the best choice.

Although the proposed algorithm is described in a complex sequential manner for easy understanding, the pruning procedure and the calculation of the cost functions in the actual implementation are performed simultaneously among all candidates by updating the cost functions in each time to select the next puncturing node as shown in Table I. In addition, as shown at the bottom of Table I, the stopping set check algorithm examines the existence of a punctured node which cannot be recovered permanently by using simple binary operations, as in [8]. Although the proposed algorithm looks like a refined version of [7], they are conceptually quite different because the main cost functions of the two algorithms are quite different. Whereas the algorithm in [7] tries to minimize the maximum level of recoverability K and the recovery reliability of the punctured nodes, the proposed algorithm tries to minimize $C(c)$, which predicts the overall decoding performance much more accurately than the cost function in [7] (which will be shown in Table III). In addition, both algorithms are basically greedy; the proposed one considers the overall optimization implicitly by using the terms $b(v)$ and $k(v)$ in the proposed cost function $C(v)$.

III. Simulation Result

To demonstrate the usefulness of the proposed algorithm, we will show the performance of the proposed punctured LDPC codes from two irregular mother LDPC codes. The parity check matrices of the mother LDPC codes are designed with the density evolution (DE) technique [9][10] and the progressive edge growth (PEG) algorithm [11] to get a better girth distribution. Also, the mother LDPC codes are all systematic, and only parity bits are candidates for puncturing for better performance. The FER of each code is evaluated by observing 200 erroneous codewords at each E_b/N_0 point using binary phase shift keying modulation in an additive white Gaussian noisy channel. The sum-product decoding algorithm is used, and the maximum number of iteration is set to fifty.

The dedicated LDPC codes used in the simulation are described in Table II. Here, a dedicated LDPC code denotes one designed with optimized degree distribution for a given code rate. We obtained the optimized degree distributions of the dedicated codes at a given maximum degree using the DE technique [9][10].

A. *Puncturing from an Irregular LDPC Code with Rate=0.5, Length=1024*

We will show the performance of the punctured LDPC codes from an irregular LDPC mother code with a block length of 1024 and a rate of 0.5. The degree distribution pair of the mother code is shown in Table II as can be obtained in [9]. The performance of the punctured LDPC codes and the dedicated LDPC codes with the rates of 0.6, 0.7, 0.8 and 0.9 are compared in Figs. 2 and 3. Due to the randomness of the puncturing algorithm, we obtain fifty puncturing patterns for each rate. The simulation results show that the proposed punctured codes have much better and more consistent performance than those obtained from the algorithm in [7]. Also, the proposed codes do not suffer from error floor in all cases due to the reduced randomness of the proposed algorithm compared to that in [7]. Note that the proposed algorithm first punctures 1-SR nodes with a lower degree (degree 2), whereas the algorithm in [7] starts to puncture 1-

TABLE I
THE PROPOSED PUNCTURING ALGORITHM AND STOPPING SET CHECK ALGORITHM

(Puncturing Algorithm)

Step 0.0	[*Initialization*] $s(v) := 1$, $r(v) := 0$, $p(v) := 1\ \forall v \in \mathbf{V}$, $R(c) := 0$ for $\forall c \in \mathbf{C}$, $p := N_p$
Step 1.0	[*Find candidate check nodes with the min. cost function*] Make a subset $\mathbf{C}^*$ of $\mathbf{C}$ such that $\forall c^* \in \mathbf{C}^*$, $R(c^*) = 0$ and $C(c^*) \le C(c)$ for any $c \in \mathbf{C}$.
Step 1.1	[*Find candidate variable nodes with the min. cost function*] For $\forall v \in \mathbf{V}_c$, $c \in \mathbf{C}^*$, make a subset $\mathbf{V}^*$ of $\mathbf{V}$ such that $\forall v^* \in \mathbf{V}^*$, $p(v^*) = 1$ and $C(v^*) \le C(v)$ for any $v \in \mathbf{V}_{c^*}$, $c^* \in \mathbf{C}^*$.
Step 2.0	[*Select a pair*] Select a pair of a variable node v^* and a check node $c^* \in \mathbf{C}^* \cap \mathbf{C}_{v^*}$. If there is more than one pair, select one randomly.
Step 3.0	[*Check the variable node to determine whether it is reserved or not*] If $r(v^*) = 0$, go to Step 4.0.
Step 3.1	[*Check the reserved variable node to determine whether it can be recovered by the other reserved variable nodes or not*] If $r(v^*) > 0$ and for $\forall v \in \mathbf{V}_{c^*}$, $p(v) = 1$, go to Step 4.0.
Step 3.2	[*Run the stopping set check algorithm*] If $r(v^*) > 0$ and there exists $v \in \mathbf{V}_{c^*}$ such that $p(v) = 0$, run the stopping set check algorithm. If the stopping set check is successful, go to Step 4.0.
Step 3.3	[*Select another pair of a check node and a variable node*] $\mathbf{V}^* = \mathbf{V}^* \setminus \{v^*\}$ and $\mathbf{V} = \mathbf{V} \setminus \{v^*\}$. If $\mathbf{V}^*$ is not an empty set, go to Step 2.0. Otherwise, go to Step 1.0.
Step 4.0	[*Puncture one variable node*] Puncture the variable node v^*
Step 5.0	[*Update all parameters*] $p(v^*) := 0$, $R(c^*) := 1$, $p := p-1$, $s(v^*) := \Sigma_{v \in \mathbf{V}_{c^*} \setminus \{v^*\}} s(v)$, $r(v) := r(v) + 1$ for $\forall v \in \mathbf{V}_{c^*} \setminus \{v^*\}$, and update cost function $C(c)$ and $C(v)$ for $\forall v \in \mathbf{V}$ and $\forall c \in \mathbf{C}$.
Step 6.0	[*Check if there remains a variable node to be punctured*] If $p = 0$, then STOP. Otherwise, go to Step 1.0.

(Stopping Set Check Algorithm)

Step 0.0	[*Initialization*] Set $p'(v) := p(v)$ for $\forall v \in \mathbf{V}$, $p'(v^*) := 0$, $iter := max\_iter$.
Step 1.0	[*Run one iteration*] For $\forall v^* \in \mathbf{V}$ such that $p'(v^*) = 0$, if $p'(v) = 1$ for $\forall v \in \mathbf{V}_c \setminus \{v^*\}$, $\exists c \in \mathbf{C}_{v^*}$, set $p'(v^*) := 1$.
Step 2.0	[*Check if there remain any punctured nodes*] If there exists no $v \in \mathbf{V}$ such that $p'(v) = 0$, the check is successful.
Step 3.0	[*Proceed to the next iteration*] $iter := iter - 1$. If $iter = 0$, the check fails. Otherwise, go to step 1.0.

SR nodes with a higher degree (degree 3). [1] Thus, the overall reliability provided from the check nodes of the proposed punctured codes is better than that of the punctured codes in [7], and the performance gap between the proposed codes and the codes in [7] increases as the amount of puncturing become larger. However, at a rate between 0.7 and 0.8, the proposed algorithm starts to puncture variable nodes with degree 3 in this example, which results in the reduction of the performance gap, as shown in Fig. 3 (at the rate of 0.8). In [7], a 0-SR node should not be punctured to guarantee that there exists no stopping set comprised of punctured nodes. Using the algorithm in [7], we observe that the number of 0-SR nodes is about 640, which means that the maximum number of punctured nodes guaranteeing that the punctured nodes do not form a stopping set is about $1024-640 = 384$. However, to make the puncturing pattern for the rate of 0.9, the 0-SR nodes should be punctured and we select 0-SR nodes randomly in the simulation for comparison because there is no consideration on such cases in [7]. On the other hand, the proposed algorithm is able to continue further in the puncturing process and can puncture a reserved variable node without forming a stopping set by using the proposed stopping set check algorithm.

Thus, it is observed from Fig. 3 that the performance gap (either between the best ones or in ensemble-average sense among the fifty trials) increases up to about 0.5dB at the FER of 10^{-3} as the amount of puncturing increases.[2][3] Also, as the amount of puncturing increases up to the rate of 0.8, the performance gaps between the proposed punctured codes

[1] With the grouping algorithm in [7], the group of 1-SR nodes contains variable nodes with degrees 2 and 3 in this example. At the sorting algorithm in [7], the node with the largest number of survived check nodes is selected at step 1.0. At the beginning of the sorting algorithm [7], a degree-3 node is selected because the number of survived check nodes of a variable node is the same to the degree of it. Thus, the sorting step starts with degree-3 nodes. In addition, a degree-3 node is selected earlier than a degree-2 node until one of its check nodes is allocated as a survived check node of other variable node.

TABLE II
THE DEDICATED LDPC CODES USED IN THE SIMULATIONS.

R	N	Degree Distributions & Girth
0.5	1024	$\lambda(x)=0.27253x+0.23755x^2+0.07038x^3+0.41953x^9$ $\rho(x)=0.7x^6+0.3x^7$, Girth=6, (Mother code)
0.6	854	$\lambda(x)=0.22222x+0.36976x^2+0.23501x^8+0.17301x^9$ $\rho(x)=x^8$, Girth=6
0.7	732	$\lambda(x)=0.182019x+0.315058x^2+0.502923x^9$ $\rho(x)=0.449130x^{12}+0.550870x^{13}$, Girth=6
0.8	640	$\lambda(x)=0.10000x+0.47143x^2+0.42857x^9$ $\rho(x)=x^{20}$, Girth=4
0.9	569	$\lambda(x)=0.14809x+0.313269x^2+0.538639x^9$ $\rho(x)=0.95739x^{43}+0.04261x^{44}$, Girth=4
0.3	1920	$\lambda(x)=0.241453x+0.122945x^2+0.635602x^{14}$ $\rho(x)=x^6$, Girth=6, (Mother code)

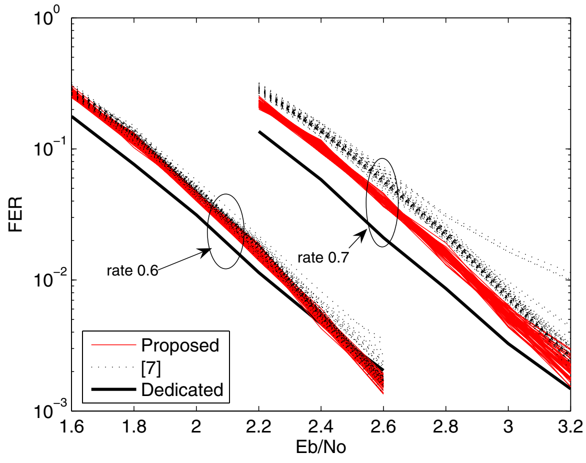

Fig. 2. FER performance of the proposed punctured codes (R=0.6 and 0.7, N=854 and 732, N_p=170 and 292, respectively).

[2] There may be another method for selecting 0-SR nodes for the conventional algorithm better than the random selection used in this paper. In this case, the performance gap may be slightly reduced.

[3] The frame error considered in this simulation includes the undetected frame error (i.e., the event when the decoded codeword satisfies all parity-check constraint but is not the transmitted one), which is very undesirable in practical situations. However, it is observed that a code with better FER performance does not guarantee better undetected error performance. Thus, it would be helpful to consider the undetected error performance as well as the FER performance.

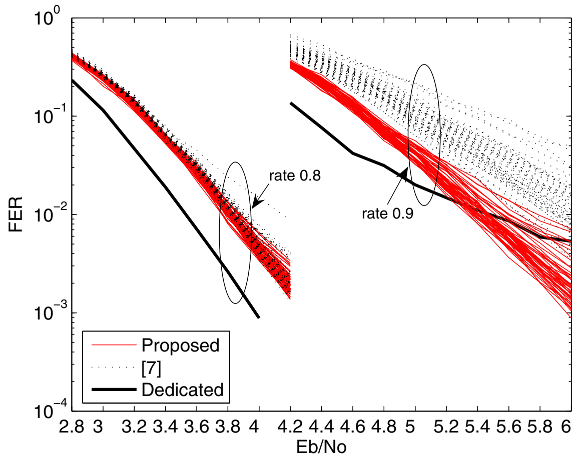

Fig. 3. FER performance of the proposed punctured codes (R=0.8 and 0.9, N=640 and 569, N_p=384 and 455, respectively).

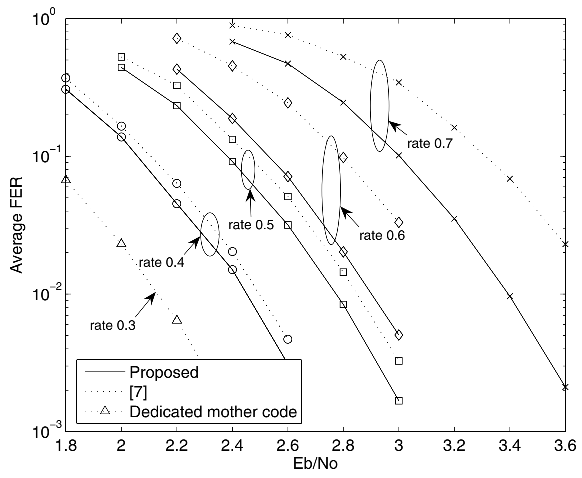

Fig. 4. Averaged FER performance of the proposed punctured codes (R=0.4, 0.5, 0.6 and 0.7, N_p=480, 768, 960, and 1098, respectively).

and the dedicated codes increase because a large amount of puncturing perturbs the mother code's structure. However, at the rate of 0.9, the proposed punctured codes show good performance while the dedicated code suffers from error floor. This is mainly due to the fact that no special care, such as in [12], is applied for lowering the error-floor in the dedicated code. However, it is worthwhile to mention that the mother codes used in this paper are also designed without any special care for low error-floor and the proposed punctured codes do not suffer from error-floor even at a very high code rate due to the longer block length of the mother code. (Note that this was also mentioned in [5][7].) In addition, it is observed from Figs. 2 and 3 that the best code among the fifty trials using the proposed algorithm shows comparable performance to the dedicated codes over a wide range in the amount of puncturing.

B. Puncturing from an Irregular LDPC Code with Rate=0.3, Length=1920

Here, we use another irregular LDPC mother code with a block length of 1920 and a rate of 0.3. The degree distribution pair of the mother code is obtained in [9] and shown in Table II. The performances of the proposed punctured LDPC codes with the rates of 0.4, 0.5, 0.6 and 0.7 are compared in Fig. 4. Here, we obtained 20 puncturing patterns for each rate, and the results show the ensemble averaged FER performance obtained from the 20 puncturing patterns. The results show that the proposed algorithm is better than the algorithm in [7] in all cases. As the amount of puncturing increases, the performance gap between the proposed algorithm and the algorithm in [7] becomes wider. When we use the algorithm in [7], we observe that the number of 0-SR nodes is about 992, which means that the maximum number of punctured nodes without forming a stopping set is about $1920 - 992 = 928$. Thus, at a rate equal to or higher than 0.6 (N_p=960), the performance gap becomes extremely large as the number of the punctured nodes increases. In addition, the performance gap reduction observed in Fig. 3 does not appear in this example because the proposed algorithm begins to puncture variable nodes with degree 3 from the rate of 0.6, at which the performance gap has already become sufficiently large.

TABLE III
$\max K$ AND $\max C(c)$ OF THE PROPOSED PUNCTURED CODES AND THE CODES IN [7].

Punctured rate	A	B	A-B	C	D	C-D
mother code (n,k)=(1024,512)/ R=0.5						
0.6 Proposed	14	14	0	1	1	0
0.6 [7]	19	26	7	1	1	0
0.7 Proposed	22	28	6	2	2	0
0.7 [7]	29	36	7	1	1	0
0.8 Proposed	56	114	58	7	4	3
0.8 [7]	67	166	99	6	3	3
0.9 Proposed	206	580	374	15	7	8
0.9 [7]	458	1730	1272	23	9	14
mother code (n,k)=(1920,576)/R=0.3						
0.4 Proposed	12	12	0	1	1	0
0.4 [7]	22	28	6	1	1	0
0.5 Proposed	26	32	6	3	2	1
0.5 [7]	27	42	15	2	1	1
0.6 Proposed	54	87	33	8	4	4
0.6 [7]	102	142	40	7	4	3
0.7 Proposed	106	234	128	10	6	4
0.7 [7]	210	352	142	10	8	3
A: min max $C(c)$ B: max max $C(c)$						
C: min max K D: max max K						

In Table III, we compared the conventional puncturing [7] with the proposed one in terms of both $C(c)$ (the proposed cost function) and the maximum level of recoverability K (the cost function in [7]). Here, $\max C(c)$ denotes the maximum value of $C(c)$ among all check nodes in a punctured code and min (max)$\{\max C(c)\}$ denotes the minimum (maximum) value of $\{\max C(c)\}$ among the fifty puncturing patterns. From the table, one can easily see that $C(c)$ estimates the actual performance much more accurately than K does. The difference between the maximum and the minimum is related with the performance deviation of the fifty (twenty) FER curves. Although not shown explicitly, we can observe a tendency in the relationship between the actual performance of a code and its $\max C(c)$ among the fifty trials. Thus, without performing a simulation, we can greatly reduce the number of candidates to obtain the best code. For example, among the fifty trials with the proposed algorithm for the rate of 0.9, we can find fairly good codes whose performance gaps (to the best code among them) are within 0.1dB at the FER of 0.01 by selecting the best ten codes in terms of $\max C(c)$.

IV. Conclusion

This paper proposed an efficient puncturing method for rate-compatible LDPC codes based on the proposed cost functions. The proposed algorithm improves the performance of the punctured LDPC code by i) maximizing the minimum initial reliability among all check nodes, ii) allocating survived check nodes evenly to punctured nodes, and iii) preventing the formation of a stopping set from punctured nodes to increase the maximum number of puncturing variable nodes without serious performance degradation. Simulation results showed that the performance of the proposed punctured LDPC codes is much better than those of existing punctured LDPC codes. To the best knowledge of the authors, the proposed puncturing algorithm provides the best ensemble average performance and the smallest performance deviation of any puncturing algorithm. In addition, using the proposed algorithm, we can obtain sufficiently good rate-compatible LDPC codes with rates up to 5/6 or 7/8 from a single mother code, which is the highest code rate used in commercial standards [1][2]. Thus, the proposed punctured LDPC codes can be applied to practical mobile communication systems, such as IEEE 802.16e, to reduce the complexity of saving various LDPC codes or to provide various types of HARQ techniques for better performance.

V. Acknowledgement

This work was supported by the MIC (Ministry of Information and Communication), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Advancement) in part by (IITA-2007-(C1090-0701-0003)) and in part by (IITA-2006-(C1090-0603-0011)).

References

[1] IEEE P802.16e, IEEE Standard for Local and Metropolitan area networks Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access System, Feb. 2006.

[2] 3GPP, TR 25.848: Physical layer aspects of UTRA High Speed Downlink Packet Access, v4.0.0, March 2001.

[3] J. Ha, J. Kim, and S. W. McLaughlin, "Optimal puncturing distribution for rate compatible low density parity check code," *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, March 2004.

[5] J. Ha, J. Kim, and S. W. McLaughlin, "Puncturing for finite length low-density parity-check codes," in *Proc. Inter. Symp. Inform. Theory (ISIT)*, pp. 151, June 2004.

[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 Symp. Person. Indoor Mobile Radio Commun.*, vol. 4, pp. 2642–2646, Sept. 2005.

[7] 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.

[8] S. H. Lee, K. S. Kim, Y. H. Kim, J. Y. Ahn, "A cycle search algorithm based on a message-passing for the design of good LDPC codes," *IEICE Trans. Fundamentals*, vol. E88-A, no. 6, pp. 1599–1604, June 2005.

[9] http://lthcwww.epfl.ch/research/ldpcopt/

[10] S. Y. Chung, T. J. Richardson, and R. 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.

[11] X. Hu, E. Eleftheriou, and D. M. Arnold, "Progressive edge-growth Tanner graphs," in *Proc. IEEE Global Commun. Confer. (GLOBECOM)*, pp. 995–1001, Nov. 2001.

[12] S. H. Lee, K S Kim, J. K. Kwon, Y. H. Kim, and J. Y. Ahn, "Design of an LDPC code with low error floor," in *Proc. Inter. Symp. Inform. Theory (ISIT)*, pp. 990–994, Sept. 2005.