# Design of an LDPC Code with Low Error Floor

*Sang Hyun Lee, †Kwang Soon Kim, *Jae Kyun Kwon, ‡Yun Hee Kim, and *Jae Young Ahn

* Telecomm. Research Lab., Electronics and Telecommunications Research Institute, Daejeon, Korea
E-mail: {sh-lee, jack, jyahn}@etri.re.kr

† Department of Electrical and Electronic Engineering, Yonsei University, Seoul, Korea
E-mail: ks.kim@yonsei.ac.kr

‡ School of Electronics and Information, Kyung Hee University, Korea
E-mail: yheekim@khu.ac.kr

***Abstract*— A search algorithm for stopping sets in a Tanner graph is proposed for designing good low-density parity-check (LDPC) codes. By applying the belief-propagation algorithm with messages containing the information of originated variable nodes and their connected edges, the stopping sets can be detected. Furthermore, a code design method using the algorithm is presented and the performances of the designed code over several channels are shown.**

## I. Introduction

An LDPC code has attracted much consideration due to its performance very close to the Shannon capacity limit under the assumption of asymptotically long codeword length. However, it has been reported that a moderate-length code suffers from an error floor in a high SNR region [1]. Also, it has been shown that, under message-passing decoding, small stopping sets in the graph dominantly give rise to high error floors [2]. Although a conditioning method for implicit alleviation of error floors has been proposed to make degrees of nodes associated with small cycles high [3], the method does not guarantee actual detection of small stopping sets. This paper addresses an algorithm that detects stopping sets up to pre-defined size. The proposed scheme performs a belief-propagation algorithm with messages carrying the information of originated variable nodes and their connected edges. Furthermore, a code design method using the algorithm is presented and the performances of the designed code over several channels are shown.

## II. Definition of Message and Update Processing

The basic idea of the proposed algorithm is to run a belief-propagation algorithm on a Tanner graph with particular messages. A message in the proposed algorithm contains information on the variable nodes which the message has passed so far and the edges where the message leaves those nodes. To deal with the messages, two different kinds of node update equations are defined: i) for constraint nodes, an input message entering through one of the connected edge is distributed as the output messages to the other edges and ii) for variable nodes, additional information of the current variable node and the constraint node which the message propagates to is added to the aggregate of the input messages. Then, we can examine the existence of a stopping set in the following way: i) send a particular message to all edges connected to the variable node under examination, which is referred to as the starting variable node in the sequel, ii) update the messages with the node update equations at each node in the graph, iii) construct a set of input messages entering the constraint nodes accepting more than two messages or the variable nodes accepting simultaneous messages through all connected edges, iv) check whether a subset of nodes contained in the constructed message set satisfies a particular condition assuring the presence of a stopping set, and v) reconstruct a new subset of valid nodes forming a stopping set out of nodes contained in the constructed message sets. Here, we define two different types of messages as follows.

The first type message is a binary digit indicating whether the second type message carries a nontrivial message or not. The message '1' indicates that the second type message through the edge carries useful information for stopping set detection. Note that we can examine cycles in a Tanner graph with the first type messages [4] and it will be denoted as a cycle detection (CD) message. The CD message is updated for each edge connected to variable or constraint node by

$$\mu^{O}_{(n_i,e_j)} = \left( \bigoplus_{e'_j \in E(n_i)} \mu^{I}_{(n_i,e'_j)} \right) \bigotimes \mu^{I}_{(n_i,e_j)}, \qquad (1)$$

where $\mu^{I}_{(n_i,e_j)}$ is the input message of the $j$th edge, $e_j$, entering the node $n_i$, $\mu^{O}_{(n_i,e_j)}$ is the output message of the $j$th edge, $e_j$, leaving the node $n_i$, and the operations $\oplus$ and $\otimes$ denote the logical OR and XOR operations, respectively. Also, $E(n_i)$ denotes the set of edges connected to the node $n_i$. For the starting variable node, output messages of all connected edges are '1'. Then, the output messages of the neighboring nodes are calculated by (1). Note that the message '1' cannot pass an edge more than once [4].

The second type message carries the information of the variable and the constraint nodes which form the shortest paths from the starting node. Since this type of message is directly used for stopping set detection, it will be denoted as stopping set detection (SD) message. Note that the SD message of an edge with a null CD message is an empty set and that a nontrivial SD message can exist only once for each edge. For a complete definition, a single SD message is defined as the set of ordered three-tuples containing a variable node and a constraint node, which uniquely define the information of the edge connecting them, and a flag set. At any class of node, the update of an SD message is performed as follows: the initial output SD message of the starting variable node, $v^*$, to its

neighboring constraint node $c$, $\xi(v^*, c)$, is defined as

$$\xi(v^*, c) = \{(v^*, c, \varnothing)\}, \tag{2}$$

where the third entry is reserved for a flag set, which will be discussed later. Now, we define some notations used throughout this paper as follows:

- $(n_1, n_2)$: the edge connecting the two nodes $n_1$ and $n_2$.
- $V_I(c)$ and $E_I(c)$ ($V_O(c)$ and $E_O(c)$): the sets of the variable nodes and the corresponding edges with nonzero output (input) CD messages to (from) the constraint node $c$, respectively.
- $C_I(v)$ and $E_I(v)$ ($C_O(v)$ and $E_O(v)$): the sets of the constraint nodes and the corresponding edges with nonzero output (input) CD messages to (from) the variable node $v$, respectively.
- $(\xi)_{k,i}$: the $i$th entry of the $k$th three-tuples in $\xi$.
- $|A|$: the cardinality of a set $A$.
- hitting node: a variable node accepts nonempty SD messages through its all connecting edges or a constraint node accepts nonempty SD messages through at least two connecting edges.
- $\mathcal{C}_i$ ($\mathcal{V}_i$): the set of all hitting constraint (variable) nodes at the $i$th iteration.
- $V(\xi)$, $C(\xi)$, $E(\xi)$, and $F(\xi)$: the set of the first entries (variable nodes), the second entries (constraint nodes), the set of edges connecting the first and the second entries, and the collection of the nonempty third entries of all elements in a message $\xi$, respectively.
- $V(\Omega)$, $C(\Omega)$, $E(\Omega)$, and $F(\Omega)$: the unions of $V(\xi)$, $C(\xi)$, $E(\xi)$, and $F(\xi)$ for all $\xi \in \Omega$, respectively.
- $E_n(\Omega)$: the set of all edges that are both included in $E(\Omega)$ and connected to a node $n \in V(\Omega) \cup C(\Omega)$.
- $V(A)$ and $C(A)$: the sets of all variable and constraint nodes in a set $A$ composed of nodes.

Note that $V_I(c)$ and $V_O(c)$ ($C_I(v)$ and $C_O(v)$) are mutually exclusive due to (1). Then, the update processing of an SD message is given as follows:

*1) At constraint nodes:* The output message leaving a constraint node $c$ is given by

$$\xi(c, v) = \begin{cases} \bigcup_{v' \in V_I(c)} \check{\xi}(v', c), & \text{if } v \in V_O(c) \\ \varnothing, & \text{otherwise} \end{cases} \tag{3}$$

where $\check{\xi}(v, c) = \xi(v, c)$ except that, for $k = 1, \cdots, |\xi(v, c)|$,

$$\left(\check{\xi}(v, c)\right)_{k,3} = \begin{cases} (\xi(v, c))_{k,3} & \text{if } |V_I(c)| = 1 \\ (\xi(v, c))_{k,3} \cup \{v\} & \text{if } |V_I(c)| \geq 2. \end{cases} \tag{4}$$

Thus, for messages arriving at a constraint node with at least two nonzero CD messages, the third entry of each constituent element is updated by adding the index of the variable node where the message comes from.

*2) At variable nodes:* The output message through an edge $(v, c)$ with a nonzero CD message is defined as the union of the two sets: the set of input messages entering the variable node $v$ in the same way as in a constraint node and the set of one three-tuple, $(v, c, \varnothing)$. Then, the output message leaving $v$ through the connected edge $(v, c)$ is updated by

$$\xi(v, c) = \begin{cases} \left(\bigcup_{c' \in C_I(v)} \xi(c', v)\right) \cup \{(v, c, \varnothing)\} & \text{if } c \in C_O(v) \\ \varnothing & \text{otherwise.} \end{cases} \tag{5}$$

Thus, one can see that the size of an SD message grows as it traverses in the graph.

## III. Stopping Set Detection

Now, consider the stopping set detection method. Here, we assume a node-by-node construction of the parity-check matrix. Once new edges, as many as the variable node degree, are added to the graph being constructed, a search for stopping sets caused by the new edges is performed. Initially, each output CD message from the starting variable node $v^*$ to the neighboring constraint node $c$, $c \in C_O(v^*)$, are set to '1'. Also, the corresponding output SD messages are set as in (2). During the belief-propagation processing, each node accepts messages through the edges connected to itself. A nonzero input CD message to a node means that the current node and the edge which the message passed through are included in a path that may be contained in a stopping set. Also, the corresponding input SD message carries the information of the variable and constraint nodes contained in the path. We refer to a single constraint node update processing and a single variable node update processing together as a single iteration. The stopping set detection can be performed twice in each iteration at the end of each update processing. At the $i$th iteration, the aggregate message at each hitting constraint (variable) node $c \in \mathcal{C}_i$ ($v \in \mathcal{V}_i$), $\bar{\xi}_i^C(c)$ ($\bar{\xi}_i^V(v)$), and the set of all hitting nodes, $\Phi_i^C$ ($\Phi_i^V$), are obtained as the union of the messages of the hitting node after the constraint (or variable) node update processing as follows:

$$\bar{\xi}_i^C(c) = \bigcup_{(v,c) \in E_I(c)} \xi(v, c), \tag{6}$$

$$\bar{\xi}_i^V(v) = \bigcup_{(c,v) \in E_I(v)} \xi(c, v), \tag{7}$$

$$\Phi_i^C = C_i \cup V_{i-1}, \tag{8}$$

$$\Phi_i^V = C_i \cup V_i, \tag{9}$$

where $C_i$ and $V_i$ denote $\bigcup_{j \leq i} \mathcal{C}_j$ and $\bigcup_{j \leq i} \mathcal{V}_j$, respectively. Since (6) and (7) ((8) and (9)) are actually identical operations, we will omit the superscripts $C$ and $V$ in the sequel unless it causes any confusion. Finally, the collection of the aggregate messages at the $i$th iteration, $\Omega_i$, is obtained as

$$\Omega_i = \{\bar{\xi}_i(n) | n \in \Phi_i\}. \tag{10}$$

Note that $\Omega_0$ is an empty set. Recall in (4) that, for all the three-tuples contained in the input SD messages of a hitting constraint node $c$, the adjacent variable node which sent a nonzero SD message is collected into their last entries as a flag set. The new element of this flag indicates the variable node that makes the corresponding three-tuple be collected into $\bar{\xi}_i(c)$ by sending the nonzero SD message to $c$. If an SD

message passes more than one hitting constraint node, the third entry of its constituent three-tuples has as many elements as the number of hitting constraint nodes it has passed. This flag will facilitate the determination of the nodes to be removed for the stopping set search.

Let $\mathcal{N}_i$ be $V(\Omega_i) \cup C(\Omega_i) \cup \Phi_i$ and $\mathcal{M}_i$ be the subset of $\mathcal{N}_i$ comprised of the variable nodes in $V(\Omega_i) \cup V(\Phi_i)$ and the constraint nodes in $C(\Omega_i) \cup C(\Phi_i)$, satisfying the following conditions: i) all neighboring constraint nodes of each variable node in $\mathcal{M}_i$ are contained in $\mathcal{M}_i$ and ii) at least two neighboring variable nodes of each constraint node in $\mathcal{M}_i$ are contained in $\mathcal{M}_i$. Then, one can see from the definition of a stopping set that $\mathcal{M}_i$ forms a stopping set. Thus, we can find a stopping set if there exists such a nonempty $\mathcal{M}_i$ in a Tanner graph. To find $\mathcal{M}_i$ out of nodes in $\mathcal{N}_i$, the following scheme proceeds. Firstly, $V(\Omega_i)$ and $E(\Omega_i)$ are constructed from $\Omega_i$. Then, for each variable node $v$ in $V(\Omega_i)$, $|E_v(\Omega_i)|$ is compared with $|E_O(v)|$. If the two values are the same, it indicates that all edges of $v$ are contained in $E(\Omega_i)$. Thus, $v$ can be included in a stopping set. We will refer to this condition as the *cardinality condition*. If the cardinality condition is satisfied for every node in $V(\Omega_i)$, $\mathcal{N}_i$ forms a stopping set. However, there may be some variable nodes in $\mathcal{N}_i$ not satisfying the cardinality condition. For $n \in \Phi_i$, let $\bar{\xi}_i^f(n) \subset \bar{\xi}_i(n)$ be the collection of all the three-tuples in $\bar{\xi}_i(n)$, whose first entries fail to satisfy the cardinality condition. Also, let $\Omega_i^f = \cup_{n\in\Phi_i} \bar{\xi}_i^f(n)$ be the collection of all $\bar{\xi}_i^f(n)$, $n \in \Phi_i$. Then, each variable node in $V(\Omega_i^f)$ has at least one 'open' edge in the subgraph constructed by $\mathcal{N}_i$. Note that a node connected to such an 'open' edge cannot be included in a stopping set. Thus, if $V(\Omega_i^f)$ is nonempty, we should remove some nodes from V($\Omega_i$) to examine the existence of a stopping subset $\mathcal{M}_i$ of $\mathcal{N}_i$ according to the following removal procedure.

**Procedure 1** *The removal procedure is as follows:*

i) *For each $n \in \Phi_i$, $\bar{\xi}_i(n)$ is erased in $\Omega_i$ and $n$ is erased in $\Phi_i$ if there exists any constituent three-tuple of $\bar{\xi}_i(n)$ whose first entry is included in $V(\Omega_i^f)$ and third entry is an empty set.*

ii) *For each $n \in \Phi_i$, each remaining three-tuple in $\bar{\xi}_i(n)$ is erased if its first entry is included in $V(\Omega_i^f)$ or its third entry is included in $F(\Omega_i^f)$.*

iii) *Let $\Omega_i'$ and $\Phi_i'$ denote the reduced sets of $\Omega_i$ and $\Phi_i$ according to the steps 1 and 2, respectively. Then, for each $c \in C(\Phi_i')$, $\bar{\xi}_i(c)$ is erased in $\Omega_i'$ and $c$ is erased in $\Phi_i'$ if $|E_c(\Omega')| < 2$. Also, for each $v \in V(\Phi_i')$, $\bar{\xi}_i(v)$ is erased in $\Omega_i'$ and $v$ is erased in $\Phi_i'$ if $|E_v(\Omega')| < |E_O(v)|$.*

Let $\Omega_i''$ and $\Phi_i''$ denote the resulting reduced sets according to Procedure 1. If the cardinality condition is satisfied for every node in $V(\Omega_i'')$, $\mathcal{M}_i = V(\Omega_i'') \cup C(\Omega_i'') \cup \Phi_i''$ forms a stopping set. Whether a stopping set is found or not from $\Omega_i''$, we can repeat Procedure 1 to further check the existence of smaller stopping sets. However, it is not necessary since the smaller stopping sets should be already found in earlier iterations if they exist. Thus, the removal procedure is performed only once in each iteration.

## IV. An Example of Stopping Set Detection

Fig. 1 illustrates a simple example of the proposed algorithm. Here, for notational simplicity, all variable nodes and constraint nodes are enumerated in Fig. 1. Let $\nu_i^C$ and $\nu_i^V$ denote the set of constraint and variable nodes accepting nonzero CD messages at the $i$th iteration, respectively. Suppose that the variable node $v_1$ and its four edges are newly added to the graph under construction ($\nu_0^V = \{v_1\}$). Initially, the CD messages through all connected edges of node $v_1$ are '1' and the corresponding SD messages to the neighboring constraint nodes $\nu_1^C = \{c_1, c_2, c_3, c_{12}\}$ are defined as

$$\begin{aligned} &\xi(v_1, c_1) = \{(v_1, c_1, \varnothing)\}, \quad \xi(v_1, c_2) = \{(v_1, c_2, \varnothing)\} \\ &\xi(v_1, c_3) = \{(v_1, c_3, \varnothing)\}, \quad \xi(v_1, c_{12}) = \{(v_1, c_{12}, \varnothing)\}, \end{aligned} \tag{11}$$

respectively. At the constraint node update of the first iteration, the input SD messages entering each constraint node are transparently distributed according to (1). Note that the set of variable nodes for nonempty output SD messages is

$$\nu_1^V = \{v_2, v_3, v_4, v_5, v_6, v_{14}, v_{15}\}. \tag{12}$$

Since $\Omega_1^C$ is empty, the belief propagation proceeds. At the variable node update of the first iteration, the variable node $v_{15}$ accepts nonempty SD messages through its all connected edges. Thus, $v_{15}$ is collected in $\mathcal{V}_1$, the messages entering $v_{15}$ form the aggregate message, $\bar{\xi}_1(v_{15})$, and $\bar{\xi}_1(v_{15})$ is collected in $\Omega_1^V$. Then, we can see that $V(\Omega_1^V) = \{v_1\}$. Since two edges (the second and the third edges of $v_1$) are missing in $E_{v_1}(\Omega_1^V)$, $v_1$ fails to satisfy the cardinality condition. After the removal procedure, we see that $(\Omega_1^V)'$ is empty. Thus, no stopping set is found and the belief propagation continues. Each variable node in $\nu_1^V$ makes its output CD and SD messages according to (1) and (3), respectively. Then, $\nu_2^C$ is given as

$$\nu_2^C = \{c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11}\}. \tag{13}$$

At the constraint node update of the second iteration, the constraint nodes $c_5$,$c_8$ and $c_9$ are hitting nodes. Thus they are included in $\mathcal{C}_2$ and their aggregate messages are included in $\Omega_2^C$ together with $\bar{\xi}_1(v_{15})$ in $\Omega_1^V$. For every three-tuple contained in the aggregate messages collected at the constraint node $c_5$, $c_8$, and $c_9$, the flag set is updated by including the variable node which sent a nonempty SD message at the previous iteration. For instance, the aggregate message of $c_5$ can be expressed as

$$\begin{aligned} \bar{\xi}_2^C(c_5) \quad = \quad &\{(v_1, c_1, \{v_2\}), (v_2, c_2, \{v_2\}), (v_1, c_1, \{v_3\}), \\ &(v_3, c_1, \{v_3\}), (v_1, c_2, \{v_3\})\}. \end{aligned} \tag{14}$$

The cardinality condition check on $V(\Omega_2^C)$ shows that some connected edges of variable nodes in $V((\Omega_2^C)^f) = \{v_2, v_3, v_4, v_6\}$ are missing. Also, we obtain

$$\begin{aligned} (\Omega_2^C)^f \quad = \quad &\{\{(v_2, c_5, \{v_2\}), (v_3, c_5, \{v_3\})\}, \{(v_4, c_8, \{v_4\})\}, \\ &\{(v_6, c_9, \{v_6\})\}\} \end{aligned} \tag{15}$$

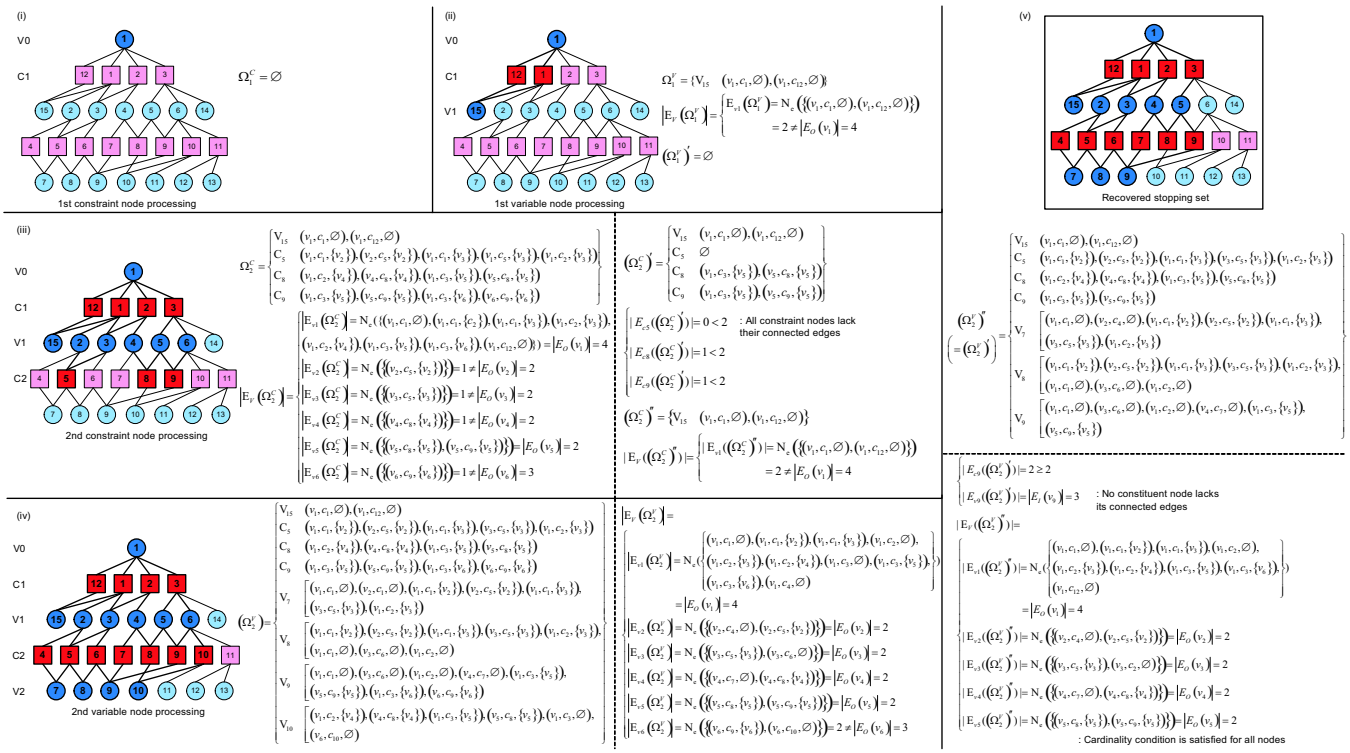


Fig. 1. An example of stopping set detection method.

and $F((\Omega_2^C)^f) = \{\{v_2\}, \{v_3\}, \{v_4\}, \{v_6\}\}$. Here, we see that no element in $V((\Omega_2^C)^f)$ is contained in a three-tuple with an empty flag set. Thus, $(\Omega_2^C)'$ is obtained from $\Omega_2^C$ by removing all three-tuples whose first element is in $V((\Omega_2^C)^f)$ or third element is in $F((\Omega_2^C)^f)$ according the second step of Procedure 1. Now, we see that $(\Phi_2^C)' = \{v_{15}, c_5, c_8, c_9\}$ and that $|E_{c_5}((\Omega_2^C)')| = 0 < 2$, $|E_{c_8}((\Omega_2^C)')| = 1 < 2$ and $|E_{c_9}((\Omega_2^C)')| = 1 < 2$. Thus, from the third step of Procedure 1, $(\Phi_2^C)''$ are obtained by erasing $c_5$, $c_8$ and $c_9$ in $(\Phi_2^C)'$ and $(\Omega_2^C)''$ is obtained by erasing the corresponding aggregate messages in $(\Omega_2^C)'$ as

$$(\Omega_2^C)'' = \{\{(v_1, c_1, \varnothing), (v_1, c_{12}, \varnothing)\}\} \tag{16}$$

One can see that the cardinality condition check on $(\Omega_2^C)''$ is failed since $|E_{v_1}((\Omega_2^C)'')| = 2 < E_O(v_1) = 4$. At the variable node update of the second iteration, the aggregate messages at variable nodes $v_7, v_8, v_9$ and $v_{10}$ are newly added to update $\Omega_2^V$. Since only $v_6$ fails to satisfy the cardinality condition, the removal procedure is performed as follows: according to the step i), $v_{10}$ and the corresponding aggregate message are removed from $\Phi_2^V$ and $\Omega_2^V$, respectively. Then, according to the step ii), the three tuples $(v_1, c_3, \{v_6\})$ and $(v_6, c_9, \{v_6\})$ are erased. In the step iii), it is easily seen that no node in $(\Phi_2^V)'$ should be erased. Thus, $(\Omega_2^V)'$ and $(\Omega_2^V)''$ are the same. Finally, one can easily see that $V((\Omega_2^V)'') = \{v_1, v_2, v_3, v_4, v_5\}$ and all the variable nodes in $V((\Omega_2^V)'')$ satisfy the cardinality condition. Therefore, the variable node set of a stopping set $\mathcal{S}$ is determined as

$$V(\mathcal{S}) = \{v_1, v_2, v_3, v_4, v_5, v_7, v_8, v_9, v_{15}\} \tag{17}$$

Note that the proposed algorithm does not guarantee that the detected set is the minimal stopping set. Here, the variable node set of the minimal stopping set is $V(\mathcal{S}^{min}) = \{v_1, v_2, v_3, v_4, v_5, v_7, v_9, v_{15}\}$. However, since the stopping set is a set of nodes, the same set is detected when the starting variable node is set to different variable node in the stopping set. Then, the minimal stopping set can be very often found by choosing the smallest set out of the similar stopping sets detected with different starting variable nodes.

## V. The Outlines of LDPC Code Design Algorithm and Code Design Example

Let $r_{\mathcal{S}}$ denotes the radius of the stopping set $\mathcal{S}$ which is defined as the number of the node update processing until the stopping set is detected. This quantity is used for an LDPC code design as well as the number of variable nodes in the stopping set $\mathcal{S}$. Since it is likely that a stopping set with larger radius has more nodes contained in it, the large radius is beneficial for the good code design. Also, the code designed only by the stopping set detection often shows the degraded bit error performance in low SNR region because a large number of connected small cycles constituting a large stopping set degrades a message-passing decoding performance. Therefore, a small cycle conditioning is essentially applied together with the stopping set detection.

In Fig. 2, the outlines of the code design algorithm using the proposed detection method are summarized. Here, $N_v(n)$ and $N_c(m)$ are the node degrees for the $n$th variable node and the $m$th constraint node, respectively. For a new variable node, a

Input : $N, K, L, L_{cycle}, N_v(n), n = 0, \ldots, N-1, N_c(m), m = 0, \ldots, N-K-1$

1: set $n \leftarrow 0$, $ind \leftarrow 0$, $R_c \leftarrow \{0, \ldots, N-K-1\}$
2: do
3: $l \leftarrow 0$
4: while ($l < N_{try}$)
5: choose randomly $N_v(n)$ elements among the elements $R_c'$ with nonzero remaining degree $N_c(m)$
6: form $N_v(n)$-tuple $\mathbf{e}$ with randomly chosen $N_v(n)$ elements
7: evaluate the cycle distribution $f_c(\mathbf{e})$ up to $2L$ caused by edge configuration $\mathbf{e}$
8: if all cycles associated with $f_c(\mathbf{e})$ is larger than $L_{cycle}$
9: execute the stopping set detection
10: evaluate the metric $f_s(\mathbf{e})$ with the number of variable nodes $|V(\mathbf{e})|$ and the radius $r_s(\mathbf{e})$
11: if $f_s(\mathbf{e})$ is better than $f_s(\mathbf{e}^*)$
12: update $f_s(\mathbf{e}^*) \leftarrow f_s(\mathbf{e})$, $\mathbf{e}^* \leftarrow \mathbf{e}$
13: endif
14: endif
15: $l \leftarrow l+1$
16: endwhile
17: for ($k = 0; k < N_v(n); k++$)
18: $VIndex(ind+k) \leftarrow n$, $CIndex(ind+k) \leftarrow \mathbf{e}^*(k)$, $N_c(\mathbf{e}^*(k)) \leftarrow N_c(\mathbf{e}^*(k)) - 1$
19: endfor
20: $ind \leftarrow ind + N_v(n)$, $n \leftarrow n+1$
21: while ($n < N$)

Output : $VIndex, CIndex$

Fig. 2. The outline of the proposed code design algorithm

random configuration of edges connecting that variable node to $N_v(n)$ constraint nodes is placed in the Tanner graph being constructed. A sufficiently large number of random edge configuration is examined for a short-length cycle conditioning. We used the algorithm proposed in [4], which is very similar to the equation (1), to evaluate the distribution of cycles with relatively short lengths. If all cycles caused by addition of new $N_v(n)$ edges are larger than pre-defined cycle length, stopping set detection is performed for such edge configurations. The outputs of stopping set detection are the number of constituent variable nodes and the set radius. With these two values, a metric function for a stopping set is evaluated. We used a weighted sum of two values for the metric function. If new metric function is better than the currently best metric function, the best metric function is updated and its edge configuration is stored. After sufficiently large number of random edge configurations, the best edge configuration is chosen for the position of new $N_v(n)$ edges. The same edge placement processing is repeated until $n < N$.

We used the proposed algorithm to construct a simple irregular codes of $(N, K) = (800, 300)$ and $(800, 500)$. The codewords were sent through binary erasure channel (BEC) and additive white Gaussian noise (AWGN) channel. Three kinds of LDPC codes are designed for comparison. A girth-conditioned code is designed by only checking whether the girth is larger than pre-defined length. The ACE [3] is a metric for implicit stopping set conditioning method. The corresponding code is designed by placing edges which maximize the sum of the variable node degrees contained in cycles. All designed codes are guaranteed to remove short cycles of length at least 6. Fig. 3 depicts the performance over BEC. The performance over BEC distinguishes the codes designed by the proposed method from other codes designed by existing methods because it strictly depends on the designed code structure. The performance over AWGN channel is plotted in Fig. 4. The sum-product algorithm is performed with iteration number of 50 for decoding. While the performance in the

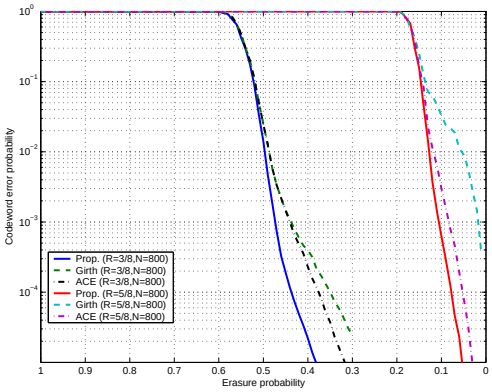


Fig. 3. Simulation result for designed codes over binary erasure channel.

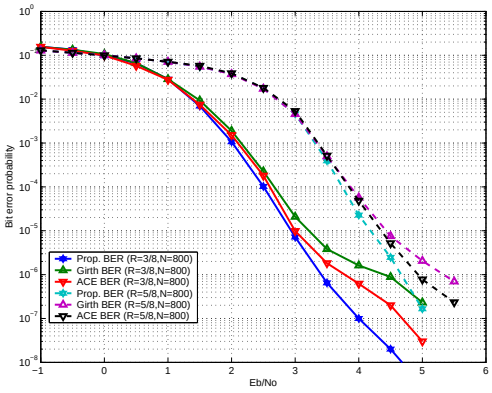


Fig. 4. Simulation result for designed codes over AWGN channel.

low SNR region is similar because relatively short cycles are conditioned for all codes, the code designed by the proposed algorithm has a performance improvement in the error floor region.

## VI. Conclusion

In this paper, a stopping set detection scheme using a belief-propagation algorithm was proposed for designing an LDPC code. The proposed scheme can detect the existence of stopping sets by using two-step belief-propagation algorithm. The constituent nodes of the detected stopping set are recovered from the messages and their number is used for the code design. The simulation results showed that the proposed LDPC codes outperforms conventional LDPC codes over BEC and AWGN channels, especially in the error floor region.

## References


[1] D. J. C. MacKay, "Good error correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, vol. 45, pp. 399-431, Mar. 1999.

[2] C. Di, D. Proietti, E. Telatar, T. Richardson, and R. Urbanke, "Finite length analysis of low-density parity-check codes on the binary erasure channel," *IEEE Trans. Inform. Theory*, vol. 48, pp. 1570-1579, June 2002.

[3] T. Tian, C. Jones, J. D. Villasenor, and R. D. Wesel, "Construction of irregular LDPC codes with low error floors," *Proc. IEEE Int. Conf. Comm.*, vol. 5, pp. 3125-3129, Anchorage, AK, U.S.A., May 2003.

[4] S. H. Lee, K. S. Kim, Y. H. Kim, and J. Y. Ahn, " A cycle search algorithm for an LDPC code design", *Proc. Int. Symp. Inform. Theory and Applications*, Wed 2-2-5, Parma, Italy, Oct. 2004.