Error Correction Capability of Column-Weight-Three LDPC Codes Under the Gallager A Algorithm—Part II

Shashi Kiran Chilappagari, *Member, IEEE*, Dung Viet Nguyen, *Student Member, IEEE*, Bane Vasic, *Senior Member, IEEE*, and Michael W. Marcellin, *Fellow, IEEE*

***Abstract*—The relation between the girth and the error correction capability of column-weight-three LDPC codes under the Gallager A algorithm is investigated. It is shown that a column-weight-three LDPC code with Tanner graph of girth $g \geq 10$ can correct all error patterns with up to $(g/2-1)$ errors in at most $g/2$ iterations of the Gallager A algorithm. For codes with Tanner graphs of girth $g \leq 8$, it is shown that girth alone cannot guarantee correction of all error patterns with up to $(g/2-1)$ errors under the Gallager A algorithm. Sufficient conditions to correct $(g/2-1)$ errors are then established by studying trapping sets.**

***Index Terms*—Error floor, Gallager A algorithm, girth, low-density parity-check (LDPC) codes, trapping sets.**

I. Introduction

ITERATIVE message passing algorithms for decoding low-density parity-check (LDPC) codes [1] operate by passing messages along the edges of a graphical representation of the code known as the Tanner graph [2]. These algorithms are optimal when the underlying graph is a tree (see [3] and [4] for general theory of LDPC codes), but in the presence of cycles, the decoding becomes sub-optimal and there exist low-weight patterns known as near codewords [5] or trapping sets [6] uncorrectable by the decoder. It is now well established that the trapping sets lead to error floor in the high signal-to-noise (SNR) region (see [7] for a list of references). While it is generally known that high girth codes have better performance in the error floor region, the exact relation between the girth and the slope of the frame error rate (FER) curve in the error floor region is unknown. In this paper, we consider the error correction capability of column-weight-three LDPC codes under the Gallager A decoding algorithm as a function of the girth of the underlying Tanner graph of the code.

Manuscript received July 22, 2008; revised May 08, 2009. Current version published May 19, 2010. This work was supported in part by the NSF under Grant CCF-0634969, IHCS-0725405, ITR-0325979, and in part by the INSIC-EHDR program. The work of S. K. Chilappagari was performed when he was with the Department of Electrical and Computer Engineering, University of Arizona, Tucson. The material in this paper was presented in part at the IEEE Information Theory Workshop (ITW), Porto, Portugal, May 5–9, 2008.

S. K. Chilappagari is with Marvell Semiconductor, Inc., Santa Clara, CA 95054 USA (e-mail: shashickiran@gmail.com).

D. V. Nguyen, B. Vasic, and M. W. Marcellin are with the Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ 85721 USA (e-mail: nguyendv@ece.arizona.edu; vasic@ece.arizona.edu; marcellin@ece.arizona.edu).

Communicated by I. Sason, Associate Editor for Coding Theory.

Digital Object Identifier 10.1109/TIT.2010.2046203

Gallager [1] showed that the minimum distance of ensembles of (d_v, d_c)-regular LDPC codes with $d_c > d_v \geq 3$, grows linearly with the code length. While this implies that there exist codes in these ensembles capable of correcting a linear number of errors in the code length under maximum-likelihood (ML) decoding, it does not necessarily imply the same for sub-optimal decoding algorithms. Zyablov and Pinsker [8] initiated the study of guaranteed error correction capability of LDPC codes. They showed that almost all the codes in the regular code ensembles with $d_v \geq 5$ are capable of correcting a linear number of errors in the code length under the parallel bit flipping algorithm. Sipser and Spielman [9] established similar results using expander graph based arguments. Burshtein and Miller [10] applied expansion arguments to irregular code ensembles to show that message passing algorithms are also capable of correcting a linear number of errors in the code length when the degree of each variable node is at least six. Recently, Burshtein [11] showed that regular code ensembles with $d_v = 4$ are also capable of correcting a linear number of errors in code length under a variant of the parallel bit flipping algorithm. Expansion arguments have also been successfully applied in the case of linear programming (LP) decoding [12]. The LP decoding algorithm first proposed by Feldman *et al.* [12] is another sub-optimal decoding algorithm which has higher complexity compared to the message passing algorithms, but is attractive from an analysis standpoint. Feldman *et al.* [13] established that the LP decoding algorithm can correct a linear number of errors in the code length when the underlying Tanner graph is a good expander. The arguments in [13] are applicable to codes with $d_v \geq 4$. The results of [13] were refined in [14], in which the authors also initiated the probabilistic analysis of LP decoding.

Another approach to the study of error correction capability of LDPC codes is based on the girth of the underlying Tanner graph. Tanner in [2] studied codes based on bipartite graphs and derived lower bounds on the minimum distance of graph based codes as a function of the left degree and girth of the Tanner graph. For $d_v \geq 3$, the minimum distance was shown to increase exponentially with the girth, which implies that for ML decoding, the error correction capability increases exponentially with the girth. The minimum pseudo-codeword weight on the binary symmetric channel (BSC) for LP decoding was also shown to increase exponentially with the girth for $d_v \geq 3$, thereby guaranteeing that the error correction capability grows exponentially with the girth of the Tanner graph. In [15], it was shown that the size of variable node sets that have the expan-

sion required by the expansion based arguments also increases exponentially with girth for $d_v \geq 5$, leading to the result that the error correction capability under the bit flipping algorithms grows exponentially with the girth. It is worth noting here that the minimum stopping set [16] size also grows exponentially with the girth for $d_v \geq 3$ [17].

One observes that the above discussion does not address the error correction capability of LDPC codes with $d_v = 3$ (henceforth referred to as column-weight-three LDPC codes) under the Gallager A algorithm as well as the bit flipping algorithms. This is due to the fact that the Tanner graphs of column-weight-three LDPC codes cannot achieve the expansion required by the expansion based arguments. Also, girth based arguments have not been investigated in sufficient detail for this case.

Burshtein [11] was the first to observe that it cannot be proved that almost all the codes in the ensemble of column-weight-three LDPC codes can correct a linear number of errors as a non-negligible fraction of codes have parallel edges in their Tanner graphs and such codes cannot correct a single worst case error. A stronger statement was proved in [18], where it was shown that at sufficiently large code length, no code in the ensemble of column-weight-three codes can correct a linear number of errors in the code length. The result was extended to the bit flipping algorithms also. The result was established by showing that a necessary condition for a column-weight-three code to correct all error patterns with up to $t \geq 5$ errors under the Gallager A algorithm is to avoid cycles of length up to $2t$ in its Tanner graph. Since the girth of the Tanner graph grows only logarithmically with the code length, it follows that correction of a linear number of errors is not possible.

The results derived in this paper are a continuation of the results presented in [18] and complete the investigation of error correction capability of column-weight-three codes under the Gallager A algorithm. We show that a column-weight-three LDPC code with Tanner graph of girth $g \geq 10$ can correct all error patterns with up to $(g/2-1)$ errors in at most $g/2$ iterations of the Gallager A algorithm. For codes with Tanner graphs of girth $g = 4$ or $g = 6$, we show that avoiding codewords of weight $2(g/2-1)$ is sufficient to guarantee correction of all error patterns with up to $(g/2-1)$ errors under the Gallager A algorithm. For codes with Tanner graph of girth $g = 8$, we show that there exist certain trapping sets which should be avoided to guarantee correction of all error patterns with up to three errors (these results first appeared in [19]). From the results presented in this paper, it follows that the error correction capability of column-weight-three codes under the Gallager A algorithm grows linearly with the girth of the Tanner graph and consequently logarithmically with the code length. Apart from the obvious theoretical significance of this problem, guaranteed error correction capability of column-weight-three codes is interesting from a practical point of view. Applications that demand high speed decoders (with transmission rates in the range of 40–80 Gbps) need to employ low-complexity decoding algorithms such as the Gallager A algorithm and in this respect column-weight-three codes are superior to higher column weight codes.

The rest of the paper is organized as follows. In Section II, we establish the notation and provide some background material. A detailed analysis of the Gallager A algorithm is presented in Section III. In Section IV, we consider the case of codes with Tanner graphs of girth $g \geq 10$. We investigate codes with Tanner graphs of girth $g \leq 8$ in Section V and conclude in Section VI with a few remarks.

II. Preliminaries

A. Notation

The Tanner graph G of an LDPC code of length n consists of two sets of nodes: the set of variable nodes V with $|V| = n$ and the set of check nodes C. The check nodes (variable nodes) connected to a variable node (check node) are referred to as its neighbors. The degree of a node is the number of its neighbors. A (d_v, d_c) regular LDPC code has a Tanner graph in which the degree of each variable node (column weight) is d_v and the degree of each check node (row weight) is d_c. Note that in the literature, the column weight and the row weight are also referred to as left degree and right degree respectively. An edge e is an unordered pair $\{v, c\}$ of a variable node v and a check node c and is said to be incident on v and c. A directed edge $\vec{e}$ is an ordered pair (v, c) or (c, v) corresponding to the edge $e = \{v, c\}$. With a moderate abuse of notation, we denote directed edges by simple letters (without arrows) but specify the direction. Hence, $\{v, c\}$ denotes an undirected edge while (v, c) or (c, v) denotes a directed edge. The girth g is the length of the shortest cycle in G. For a given node u, the neighborhood of depth d, denoted by $\mathcal{N}_u^d$, is the induced subgraph consisting of all nodes reached and edges traversed by paths of length at most d starting from u (including u). The directed neighborhood of depth d of a directed edge $e = (v, c)$ denoted by $\mathcal{N}_e^d$, is defined as the induced subgraph containing all edges and nodes on all paths $e_1, \ldots, e_d$ starting from v such that $e_1 \neq e$ (see [3] for definitions and notation). In a Tanner graph with girth g, we note that $\mathcal{N}_u^d$ is a tree when $d \leq (g/2-1)$. Also, if $e_1 = (v, c)$ and $e_2 = (c, v)$, then $\mathcal{N}_{e_1}^{d_1} \cap \mathcal{N}_{e_2}^{d_2} = \phi$ for $d_1 + d_2 < g-1$. Let ℓ denote the number of iterations for which the incoming messages to the nodes are statistically independent, as defined by Gallager in [1]. In a Tanner graph of girth g, the number of independent iterations satisfies the relation $g/4 - 1 \leq \ell < g/4$.

We consider binary LDPC codes and binary message passing decoding algorithms. The original value of a variable node is its value in the transmitted codeword. We say a variable node is good if its received value is equal to its original value and bad otherwise. A message is said to be correct if it is equal to the original value of the corresponding variable node and incorrect otherwise. In this paper, $\circ$ denotes a good variable node, $\bullet$ denotes a bad variable node and $\square$ denotes a check node. When the channel and the decoder satisfy certain symmetry conditions (see [3] for details), we can assume, without loss of generality, that the all zero codeword is transmitted. We make this assumption throughout the paper. Hence, a bad variable node has received value 1 and an incorrect message has a value of 1. A configuration of bad variable nodes is a subgraph in which the location of bad variable nodes relative to each other is specified. A valid configuration $\mathcal{C}_g$ is a configuration of at most $(g/2-1)$ bad variable nodes free of cycles of length less than g. The set of

bad variable nodes in $\mathcal{N}_e^d$ is denoted by $\mathcal{B}(\mathcal{N}_e^d)$ and $|\mathcal{B}(\mathcal{N}_e^d)|$ is denoted by $B(\mathcal{N}_e^d)$. The number of bad variable nodes at depth d in $\mathcal{N}_e^d$ is denoted by b_e^d.

B. Binary Message Passing Algorithms

Gallager in [1] proposed two simple binary message passing algorithms for decoding over the BSC; Gallager A and Gallager B. See [20] for a detailed description of the Gallager B algorithm. For column-weight-three codes, which are the focus of this paper, these two algorithms are the same. Every round of message passing (iteration) starts with sending messages from variable nodes to check nodes (first half of the iteration) and ends by sending messages from check nodes to variable nodes (second half of the iteration). Let $\mathbf{r} = (r_1, \ldots, r_n)$, an n-tuple be the input to the decoder. Let $\omega_j(v, c)$ denote the message passed by a variable node v to its neighboring check node c in the jth iteration and $\varpi_j(c, v)$ denote the message passed by a check node c to its neighboring variable node v. Additionally, let $\omega_j(v, :)$ denote the set of all messages from v, $\omega_j(v, : \backslash c)$ denote the set of messages from v to all its neighbors except to c and $\omega_j(:, c)$ denote the set of all messages to c. Let $|\varpi(:, v) = m|$ denote the number of incoming messages to v which are equal to $m \in \{0, 1\}$. Further, if all the messages in the set $\varpi_{j-1}(: \backslash c, v)$ are equal to $m \in \{0, 1\}$, we say $\varpi_{j-1}(: \backslash c, v) = \{m\}$. The terms $\omega_j(: \backslash v, c), \varpi_j(c, :), \varpi_j(c, : \backslash v), \varpi_j(:, v)$ and $\varpi_j(: \backslash c, v)$ are defined similarly. The Gallager A algorithm can then be defined as follows:

$$\omega_1(v, c) = r_v$$

$$\omega_j(v, c) = \begin{cases} 1, & \text{if } \varpi_{j-1}(: \backslash c, v) = \{1\} \\ 0, & \text{if } \varpi_{j-1}(: \backslash c, v) = \{0\} \\ r_v, & \text{otherwise} \end{cases}$$

$$\varpi_j(c, v) = \left(\sum_{u \in \mathcal{N}_c^1 \backslash v} \omega_j(u, c) \right) \bmod 2.$$

At the end of each iteration, an estimate of each variable node is made based on the incoming messages and possibly the received value. The decoded word at the end of the jth iteration is denoted as $\mathbf{r}^{(j)}$. The decoder is run until a valid codeword is found or a maximum number of iterations M is reached, whichever is earlier. The output of the decoder is either a codeword or $\mathbf{r}^{(M)}$. We say that the Gallager A algorithm is successful on a configuration of bad variable nodes in j iterations if the decoded word at the end of the jth iteration is equal to the transmitted codeword. Otherwise, we say that the Gallager A algorithm is not successful on the configuration.

Different rules to estimate a variable node after each iteration are possible and it is likely that changing the rule after certain number of iterations may be beneficial. However, the analysis of various scenarios is beyond the scope of this paper. For column-weight-three codes, only two rules are possible.

- Decision rule A: if all incoming messages to a variable node from its neighboring check nodes are equal, set the variable node to that value; else set it to received value.
- Decision rule B: set the value of a variable node to the majority of the incoming messages; majority always exists since the column weight is three.

In this paper, we assume Decision rule B for codes with Tanner graphs of girth $g \geq 10$ and Decision rule A for codes with Tanner graphs of girth $g \leq 8$. A detailed discussion of this is provided in Section V.

C. Trapping Sets and Inducing Sets

We now provide a brief overview of the characterization of failures of the Gallager A algorithm. We adopt the definitions of the terms eventually correct variable nodes, trapping sets and failure sets from [6] and the definition of inducing set from [18]. We also adopt the definition of *critical number* of a trapping set from [21]. The support of a vector $\mathbf{r} = (r_1, r_2, \ldots, r_n)$, denoted by $\mathrm{supp}(\mathbf{r})$ is defined as the set of all variable nodes v for which $r_v \neq 0$. Since the transmitted codeword is assumed to be the all-zero-codeword, the support of the input to the decoder is simply the set of variable nodes flipped by the channel (or, in other words, the set of variable nodes initially in error).

Let $\mathbf{r}$ denote the input to the Gallager A decoder. A variable node v is said to be *eventually correct* if there exists a positive integer j_c such that for all l with $j_c \leq l \leq M$, $r_v^l = 0$. For an input $\mathbf{r}$, the *failure set* $\mathbf{T}(\mathbf{r})$ is defined as the set of variable nodes that are not eventually correct. The decoding on the input $\mathbf{r}$ is successful if and only if $\mathbf{T}(\mathbf{r}) = \emptyset$. If $\mathbf{T}(\mathbf{r}) \neq \emptyset$, then we say that $\mathbf{T}(\mathbf{r})$ is a *trapping set* and $\mathrm{supp}(\mathbf{r})$ is an *inducing set*. The size of an inducing set is its cardinality. Since the failure sets of two different input vectors can be the same trapping set, we denote a trapping set simply by $\mathcal{T}$. A trapping set $\mathcal{T}$ is said to be an (a, b) trapping set if it has a variable nodes and b odd-degree check nodes in the sub-graph induced by $\mathcal{T}$. Note that in order to show that a given set of variable nodes $\mathcal{T}$ is a trapping set, we should exhibit a vector $\mathbf{y}$ for which $\mathbf{T}(\mathbf{y}) = \mathcal{T}$.

Definition 1 ([21]): (Critical number for Gallager A algorithm) Let $\mathcal{T}$ be a trapping set for the Gallager A algorithm and let $\mathbf{r} \in \mathrm{GF}(2)^n$. Let $\mathbf{Y}(\mathcal{T}) = \{\mathbf{r} | \mathbf{T}(\mathbf{r}) = \mathcal{T}\}$. The critical number $m(\mathcal{T})$ of trapping set $\mathcal{T}$ for the Gallager A algorithm is the minimum number of variable nodes that have to be initially in error for the decoder to end up in the trapping set $\mathcal{T}$, i.e.,

$$m(\mathcal{T}) = \min_{\mathbf{Y}(\mathcal{T})} |\mathrm{supp}(\mathbf{r})|.$$

From the above discussion, it follows that it is necessary to avoid all trapping sets with critical number t to guarantee correction of all error patterns with up to t errors.

III. First ℓ Iterations

In this section, we outline a method to find, for different values of ℓ, all the possible configurations of at most $(2\ell + 1)$ bad variable nodes in the neighborhood of a variable node such that the variable node sends an incorrect message in the $(\ell+1)$th iteration. We begin with the following lemma.

Lemma 1: The messages passed by a variable node v and a check node c in the Gallager A algorithm operating on a column-weight-three LDPC code can be described in the following manner.

i) If v is a bad variable node, then we have $\omega_1(v, :) = \{1\}$ and:

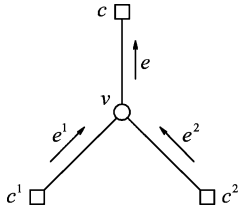

Fig. 1. Illustration of neighborhood up to depth 1 of a good variable node.

- $\omega_j(v,:) = \{1\}$ if $|\varpi_{j-1}(:,v) = 1| \geq 2$, i.e., v sends incorrect messages to all its neighbors if it receives two or more incorrect messages from its neighboring check nodes in the previous iteration;
- $\omega_j(v,: \setminus c) = \{1\}$ and $\omega_j(v,c) = 0$ if $\varpi_{j-1}(: \setminus c, v) = \{0\}$ and $\varpi_{j-1}(c,v) = 1$, i.e., v sends one correct message and two incorrect messages if it receives one incorrect message from its neighboring check nodes in the previous iteration; the correct message is sent along the edge on which the incorrect message is received;
- $\omega_j(v,:) = \{0\}$ if $\varpi_{j-1}(:,v) = \{0\}$, i.e., v sends all correct messages if it receives all correct messages from its neighboring check nodes in the previous iteration.

ii) If v is a good variable node, then we have $\omega_1(v,:) = \{0\}$ and:
- $\omega_j(v,:) = \{0\}$ if $|\varpi_{j-1}(:,v) = 0| \geq 2$, i.e., v sends all correct messages if it receives two or more correct messages from its neighboring check nodes in the previous iteration;
- $\omega_j(v,: \setminus c) = \{0\}$ and $\omega_j(v,c) = 1$ if $\varpi_{j-1}(: \setminus c, v) = \{1\}$ and $\varpi_{j-1}(c,v) = 0$, i.e., v sends one incorrect message and two correct messages if it receives two incorrect messages from its neighboring check nodes in the previous iteration; the incorrect message is sent along the edge on which the correct message is received;
- $\omega_j(v,:) = \{1\}$, if $\varpi_{j-1}(:,v) = \{1\}$, i.e., v sends all incorrect messages if it receives all incorrect messages from its neighboring check nodes in the previous iteration.

iii) For a check node c, we have:
- $\varpi_j(c,v) = \omega_j(v,c) \oplus 1$, if $|\omega_j(:,c) = 1|$ is odd, i.e., c sends incorrect messages along the edges on which it received correct messages and correct messages along the edges on which it received incorrect messages, if the total number of incoming incorrect messages from its neighboring variable nodes is odd;
- $\varpi_j(c,v) = \omega_j(v,c)$, if $|\omega_j(:,c) = 1|$ is even, i.e., c sends incorrect messages along the edges on which it received incorrect messages and correct messages along the edges on which it received correct messages, if the total number of incoming incorrect messages from its neighboring variable nodes is even.

iv) A variable node is estimated incorrectly at the end of an iteration if it receives at least two incorrect messages.

Proof: Follows from the description of the Gallager A algorithm. ■

Let v be a variable node which sends an incorrect message along the edge $e = (v,c)$ in the $(j+1)$th iteration for some j. Let the other two neighboring check nodes of v be denoted by c^1 and c^2. Additionally let $e^1 = (c^1, v)$ and $e^2 = (c^2, v)$ denote the directed edges from c^1 and c^2, respectively, to v (see Fig. 1 for an illustration when v is a good variable node). From Lemma 1, we see that only the following three cases are possible.

1) v is a good variable node and $\mathcal{N}_{e^1}^1$ and $\mathcal{N}_{e^2}^1$ contain an odd number of variable nodes which send an incorrect message in the jth iteration to c^1 and c^2.
2) v is a bad variable node and $\mathcal{N}_{e^1}^1$ and $\mathcal{N}_{e^2}^1$ contain an odd number of variable nodes which send an incorrect message in the jth iteration to c^1 and c^2.
3) v is a bad variable node and one of $\mathcal{N}_{e^1}^1$ and $\mathcal{N}_{e^2}^1$ contains an odd number of variable nodes while the other contains an even number of variable nodes which send an incorrect message in the jth iteration to c^1 and c^2.

Now, consider a variable node v which sends an incorrect message along the edge $e = (v,c)$ in the $(\ell+1)$th iteration. The message along e depends only on the variable nodes and check nodes in $\mathcal{N}_e^{2\ell}$. Since $\mathcal{N}_e^{2\ell}$ is a tree, all the variable nodes and check nodes in $\mathcal{N}_e^{2\ell}$ are guaranteed to be distinct. This fact coupled with the above observations provides a recursive method to determine all the possible configurations of at most $2\ell+1$ bad variable nodes in $\mathcal{N}_e^{2\ell}$.

For a variable node v to send an incorrect message along e in the $(\ell+1)$th iteration, $\mathcal{N}_e^2$ must contain variable node/nodes which sends/send an incorrect message in the ℓth iteration. Let v' be the generic label used to denote a variable node that sends an incorrect message along edge e' (note that $e' = (v', c^1)$ or $e' = (v', c^2)$) in the ℓth iteration. The total number of bad variable nodes in $\mathcal{N}_e^{2\ell}$ is the sum (over all possible e') of all bad variable nodes in $\mathcal{N}_{e'}^{2\ell-2}$. It is easy to see that if we derive a configuration of bad variable nodes in $\mathcal{N}_e^{2\ell}$ from the configuration of bad variable nodes in $\mathcal{N}_{e'}^{e\ell-2}$ by applying observations (1)–(3), the number of bad variable nodes increases by at least one. Hence, all the configurations of at most $(2\ell+1)$ bad variable nodes in $\mathcal{N}_e^{2\ell}$ can be constructed from the configurations of at most 2ℓ bad variable nodes in $\mathcal{N}_{e'}^{2\ell-2}$ by applying observations (1)–(3). We illustrate this with two examples.

Example 1: Let us begin by considering the case $\ell = 1$ in which we require configurations of at most three bad variable nodes in $\mathcal{N}_e^2$ for a variable node v to send an incorrect message along $e = (v,c)$ in the second iteration. The only variable nodes that send incorrect messages in the first iteration are the bad variable nodes. It can be seen that there are only three configurations of at most three bad variable nodes which are described below.

1) v is a bad variable node and $\mathcal{N}_{e^1}^1$ contains another bad variable node.
2) v is a bad variable node and $\mathcal{N}_{e^1}^1$ and $\mathcal{N}_{e^2}^1$ each contain one bad variable node.
3) v is a good variable node and $\mathcal{N}_{e^1}^1$ and $\mathcal{N}_{e^2}^1$ each contain one bad variable node.

The three cases are illustrated in Fig. 2(a). It can be seen that if $\mathcal{N}_{e^1}^1$ and $\mathcal{N}_{e^2}^1$ each contain one bad variable node, v sends an incorrect message irrespective of whether it is good or bad. Hence, to avoid drawing too many figures, we illustrate cases (2) and (3) in one figure in which the variable node v is represented

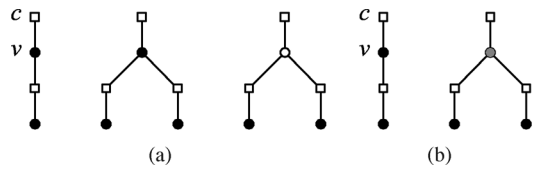

Fig. 2. (a) All configurations of at most three bad variable nodes in $\mathcal{N}_e^2$ in which v sends an incorrect message to c in the second iteration. (b) Configurations illustrated using neutral variable node.

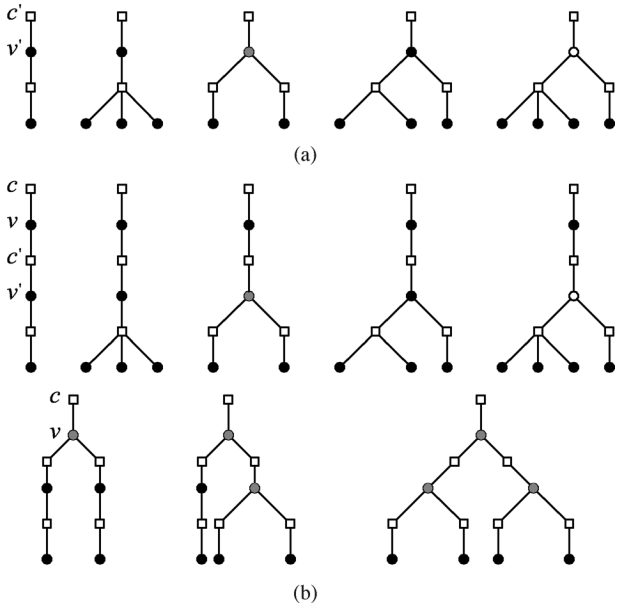

Fig. 3. (a) All configurations of at most four bad variable nodes in $\mathcal{N}_{e'}^2$ in which v' sends an incorrect message to c' in the second iteration. (b) All configurations of at most five bad variable nodes in $\mathcal{N}_e^4$ in which v sends an incorrect message to c in the third iteration.

by the gray circle and refer to the variable node as a neutral variable node. This is illustrated in Fig. 2(b).

Example 2: For $\ell = 2$, we require configurations of at most five bad variable nodes in $\mathcal{N}_e^4$ for a variable node v to send an incorrect message along $e = (v, c)$ in the third iteration. We first determine all the configurations of at most four bad variable nodes in $\mathcal{N}_{e'}^2$ for a variable node v' to send an incorrect message along $e' = (v', c')$ in the second iteration. The configurations are shown in Fig. 3(a). Then, applying observations (1)–(3), we derive all possible configurations for $\ell = 2$. Fig. 3(b) illustrates the configurations for $\ell = 2$ from which all the possible configurations can be derived in the following way. For each configuration illustrated in Fig. 3(b) that contains neutral variable nodes, convert some of the neutral variable nodes to bad variable nodes such that the total number of bad variable nodes does not exceed five. Carrying this out in all the possible ways generates all the possible configurations.

Proceeding along similar lines, we can obtain all possible configurations with at most seven bad variable nodes such that v sends an incorrect message along edge e in the fourth iteration under the assumption that $\mathcal{N}_e^6$ is a tree. Fig. 4 illustrates the configurations for $\ell = 3$ from which all the possible configurations can be derived.

We now prove bounds on $B(\mathcal{N}_e^{2\ell})$ in the following lemma.

Lemma 2: Let v be a variable node that sends an incorrect message along the edge $e = (v, c)$ in the $(\ell+1)$th iteration and assume that $\mathcal{N}_e^{2\ell}$ is a tree.

i) If v is a bad variable node, then $B(\mathcal{N}_e^{2\ell}) \geq \ell + 1$. If $B(\mathcal{N}_e^{2\ell-2}) = 1$, i.e., $b_e^d = 0$ for $d = 2, 4, \ldots, 2(\ell-1)$, then $B(\mathcal{N}_e^{2\ell}) \geq 2^{(\ell-1)} + 1$. If $B(\mathcal{N}_e^{2\ell-2}) = 2$, then $B(\mathcal{N}_e^{2\ell}) \geq 2^{(\ell-2)} + 2$.

ii) If v is a good variable node, then $B(\mathcal{N}_e^{2\ell}) \geq 2\ell$. If $B(\mathcal{N}_e^{2\ell-2}) = 0$, then $B(\mathcal{N}_e^{2\ell}) \geq 2^{\ell}$. If $B(\mathcal{N}_e^{2\ell-2}) = 1$, then $B(\mathcal{N}_e^{2\ell}) \geq 2^{(\ell-1)} + 2^{(\ell-2)} + 1$. If $B(\mathcal{N}_e^{2\ell-2}) = 2$, then $B(\mathcal{N}_e^{2\ell}) \geq 2^{(\ell-1)} + 2$.

Proof: The proof is by induction on ℓ. It is easy to verify the bounds for $\ell = 2$. Let the bounds be true for some $\ell \geq 2$, and let v_0 be a bad variable node sending an incorrect message on $e = (v_0, c)$ in the $(\ell+2)$th iteration. Further, assume that $\mathcal{N}_e^{2\ell+2}$ is a tree. Then, $\mathcal{N}_e^1$ has at least one check node c_1 which sends an incorrect message along the edge $e_1 = (c_1, v_0)$ in the $(\ell+1)$th iteration. This implies that $\mathcal{N}_e^2$ has at least one variable node v_2 sending an incorrect message in the $(\ell+1)$th iteration along the edge $e_2 = (v_2, c_1)$. Since a path of length 2 exists between v_0 and v_1, $\mathcal{N}_{e_2}^d \subset \mathcal{N}_e^{d+2}$.

If v_2 is a bad variable node, then $B(\mathcal{N}_{e_2}^{2\ell}) \geq \ell+1$ and consequently $B(\mathcal{N}_e^{2\ell+2}) \geq \ell + 2$. If v_2 is a good variable node, then $B(\mathcal{N}_{e_2}^{2\ell}) \geq 2\ell$ and consequently $B(\mathcal{N}_e^{2\ell+2}) \geq 2\ell + 1 > \ell + 1$.

If $b_e^d = 0$ for $d = 2, 4, \ldots, 2\ell$, then v_2 is a good variable node such that $B(\mathcal{N}_{e_2}^{2\ell-2}) = 0$ which implies that $B(\mathcal{N}_{e_2}^{2\ell}) \geq 2^{\ell}$ by the induction hypothesis. Hence, $B(\mathcal{N}_e^{2\ell+2}) \geq 2^{\ell} + 1$.

If $B(\mathcal{N}_e^{2\ell}) = 2$ then either (a) v_2 is a bad variable node with $b_{e_2}^d = 0$ for $d = 2, 4, \ldots, 2(\ell-1)$ which implies that $B(\mathcal{N}_{e_2}^{2\ell}) \geq 2^{(\ell-1)} + 1$ by the induction hypothesis. Hence, $B(\mathcal{N}_e^{2\ell+2}) \geq 2^{(\ell-1)} + 2$, or (b) v_2 is a good variable node with $B(\mathcal{N}_{e_2}^{2\ell-2}) = 1$ which implies that $B(\mathcal{N}_{e_2}^{2\ell}) \geq 2^{(\ell-1)} + 2^{(\ell-2)} + 1$ by the induction hypothesis. Hence, $B(\mathcal{N}_e^{2\ell+2}) \geq 2^{(\ell-1)} + 2^{(\ell-2)} + 2 > 2^{(\ell-1)} + 2$.

By the principle of mathematical induction, the bounds are true for all ℓ when v_0 is a bad variable node. The proofs are similar for the case when v_0 is a good variable node. ■

IV. Main Theorem

In this section, we prove that a column-weight-three code with Tanner graph of girth $g \geq 10$ can correct all error patterns with up to $(g/2-1)$ errors in at most $g/2$ iterations of the Gallager A algorithm. The proof proceeds by finding, for a particular choice of ℓ, all configurations of $(g/2-1)$ or less bad variable nodes on which the Gallager A algorithm is not successful in $\ell+1$ iterations and then prove that the Gallager A algorithm is successful on these configurations in subsequent iterations. When $g/2$ is even, we use $\ell = g/4 - 1$ (or $(g/2-1) = 2\ell+1$) and when $g/2$ is odd, we use $\ell = (g-2)/4$ (or $(g/2-1) = 2\ell$). We deal with these cases separately.

A. $g/2$ Is Even

Let v_0 be a variable node which receives two incorrect messages along the edges $e_1 = (c_1^1, v_0)$ and $e_2 = (c_1^2, v_0)$ at the end of the $(\ell+1)$th iteration (v_0 can be either good or bad). This implies that $\mathcal{N}_{e_1}^1$ and $\mathcal{N}_{e_2}^1$ each has a variable node, v_2^1 and v_2^2 respectively, that sends an incorrect message in the $(\ell+1)$th iteration to check nodes c_1^1 and c_1^2, respectively. Let

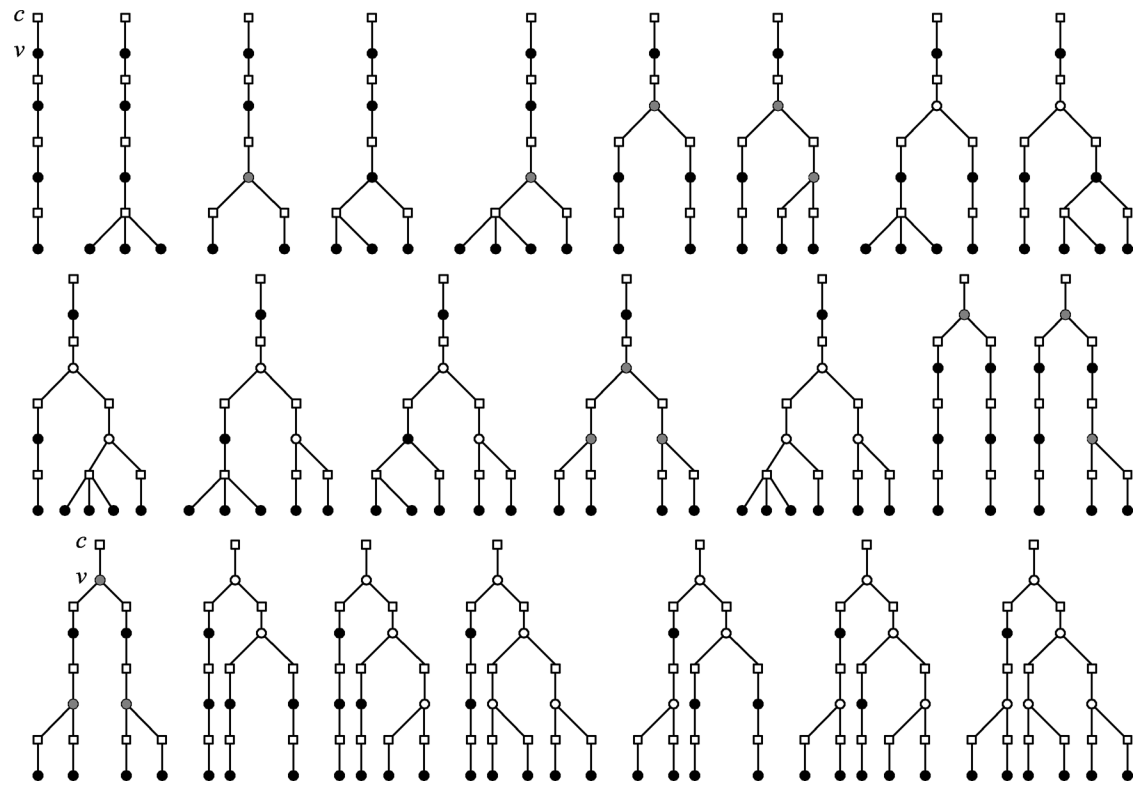

Fig. 4. All configurations of at most seven bad variable nodes in $\mathcal{N}_e^6$ in which v sends an incorrect message to c in the fourth iteration.

$e_3 = (v_2^1, c_1^1)$, $e_4 = (v_2^2, c_1^2)$, $e_5 = (c_1^1, v_2^1)$, and $e_6 = (c_1^2, v_2^2)$ [see Fig. 5(a) for an illustration]. All possible configurations of bad variable nodes in $\mathcal{N}_{e_3}^{2\ell}$ and $\mathcal{N}_{e_4}^{2\ell}$ can be determined using the method outlined in Section III. Since there exists a path of length 3 between v_2^2 and c_1^1, we have $\mathcal{N}_{e_4}^{d} \subset \mathcal{N}_{e_5}^{d+3}$. Also, $\mathcal{N}_{e_3}^{d_1} \cap \mathcal{N}_{e_5}^{d_2} = \phi$ for $d_1 + d_2 < g - 1 = 4\ell + 3$. Therefore, $\mathcal{N}_{e_3}^{d_1} \cap \mathcal{N}_{e_4}^{d_2} = \phi$ for $d_1 + d_2 < 4\ell$. This implies that $\mathcal{N}_{e_3}^{2\ell}$ and $\mathcal{N}_{e_4}^{2\ell}$ can have a common node only at depth 2ℓ. The total number of bad variable nodes in $\mathcal{N}_{e_3}^{2\ell} \cup \mathcal{N}_{e_4}^{2\ell}$, $B(\mathcal{N}_{e_3}^{2\ell} \cup \mathcal{N}_{e_4}^{2\ell})$, in any configuration is, therefore, lower bounded by $B(\mathcal{N}_{e_3}^{2\ell-2}) + B(\mathcal{N}_{e_4}^{2\ell-2}) + \max(b_{e_3}^{2\ell}, b_{e_4}^{2\ell})$ or equivalently $\max(B(\mathcal{N}_{e_3}^{2\ell-2}) + B(\mathcal{N}_{e_4}^{2\ell}), B(\mathcal{N}_{e_3}^{2\ell}) + B(\mathcal{N}_{e_4}^{2\ell-2}))$. Hence, we have

$$B\left(\mathcal{N}_{e_3}^{2\ell} \cup \mathcal{N}_{e_4}^{2\ell}\right) \geq \max\left(B\left(\mathcal{N}_{e_3}^{2\ell-2}\right) + B\left(\mathcal{N}_{e_4}^{2\ell}\right) \right. \\ \left. B\left(\mathcal{N}_{e_3}^{2\ell}\right) + B\left(\mathcal{N}_{e_4}^{2\ell-2}\right)\right). \quad (1)$$

We are interested only in the valid configurations, i.e., at most $(g/2 - 1)$ bad variable nodes, free from cycles of length less than g. We divide the discussion into three parts: 1) we find all the possible valid configurations for the case when $g = 16$; 2) we then proceed iteratively for $g \geq 20$; 3) we consider the case $g = 12$ separately as the arguments for $g \geq 16$ do not hold for this case.

1) $g = 16$: Let $v_0, v_2^1, v_2^2, e_1, e_2, e_3, e_4$ be defined as above with $\ell = 3$. Since v_2^1 and v_2^2 are variable nodes that send incorrect messages along e_3 and e_4, respectively, in the fourth iteration, the configuration of bad variable nodes in $\mathcal{N}_{e_3}^{2\ell}$ and $\mathcal{N}_{e_4}^{2\ell}$ must resemble one of the configurations illustrated in Fig. 4. In order to find all valid configurations on which the Gallager A algorithm is not successful at the end of 4 iterations, we consider each possibility in Fig. 4 for $\mathcal{N}_{e_3}^{2\ell}$ and $\mathcal{N}_{e_4}^{2\ell}$. From (1) and the constraint $B(\mathcal{N}_{e_3}^{2\ell} \cup \mathcal{N}_{e_4}^{2\ell}) \leq 7$, we see that there are only three possible configurations which are shown in Fig. 5.

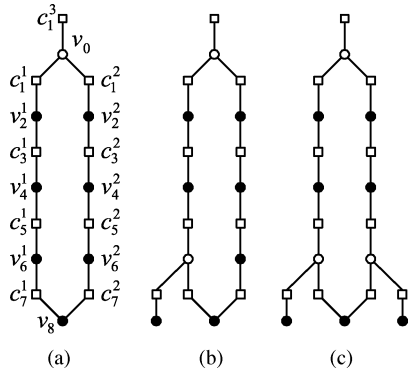

Fig. 5. Configurations of at most seven bad variable nodes, free of cycles of length less than 16, on which the Gallager A algorithm is not successful in four iterations.

Remark: Since, all the configurations contain seven bad variable nodes excluding the variable node v_0, we note that for a valid configuration, v_0 has to be a good variable node. Hence, there do not arise cases where a bad variable node receives two incorrect message at the end of fourth iteration.

2) $g \geq 20$: Let $\mathcal{C}_g$ be a valid configuration in which there exists a variable node v_0 which receives two incorrect messages along the edges $e_1 = (c_1^1, v_0)$ and $e_2 = (c_1^2, v_0)$ at the end of the $(\ell + 1)$th iteration. This implies that $\mathcal{N}_{e_1}^1$ and $\mathcal{N}_{e_2}^1$ each has a variable node, v_2^1 and v_2^2, respectively, that sends an incorrect message in the $(\ell+1)$th iteration. We have the following lemma.

Lemma 3: v_2^1 and v_2^2 are bad variable nodes.

Proof: The proof is by contradiction. We know that the lower bound on the total number of bad variable nodes in any configuration is given by (1) and that $\ell > 3$. We have two cases.

a) v_2^1 and v_2^2 are both good variable nodes: We first note that in any valid configuration, $B(\mathcal{N}_{e_3}^{2\ell-2}) \geq 2$. Otherwise, we have the following two possibilities.
   i) $B(\mathcal{N}_{e_3}^{2\ell-2}) = 1$. In this case, from Lemma 2, we see that $B(\mathcal{N}_{e_3}^{2\ell}) \geq 2^{(\ell-1)} + 2^{(\ell-2)} + 1 > 2\ell + 1$, since $\ell \geq 4$.
   ii) $B(\mathcal{N}_{e_3}^{2\ell-2}) = 0$. In this case, $B(\mathcal{N}_{e_3}^{2\ell}) \geq 2^{\ell} + 2 > 2\ell + 1$, since $\ell \geq 4$.

   Both cases are a contradiction as we have at most $2\ell + 1$ bad variable nodes. Hence, $B(\mathcal{N}_{e_3}^{2\ell-2}) \geq 2$.
   Now, $B(\mathcal{N}_{e_4}^{2\ell}) \geq 2\ell$, and, hence, we have $B(\mathcal{N}_{e_3}^{2\ell} \cup \mathcal{N}_{e_4}^{2\ell}) \geq B(\mathcal{N}_{e_4}^{2\ell}) + B(\mathcal{N}_{e_3}^{2\ell-2}) \geq 2\ell + 2$, which is a contradiction.

b) v_2^1 is a bad variable node and v_2^2 is a good variable node. The opposite case is identical.

First, we claim that in any valid configuration, $B(\mathcal{N}_{e_3}^{2\ell-2}) \geq 2$. Since v_2^1 is a bad variable node, $B(\mathcal{N}_{e_3}^{2\ell-2}) \neq 0$. Assume that $B(\mathcal{N}_{e_3}^{2\ell-2}) = 1$. Then $B(\mathcal{N}_{e_3}^{2\ell}) \geq 2^{(\ell-1)} + 1$. Again, $B(\mathcal{N}_{e_4}^{2\ell-2}) \geq 2$ implies that $B(\mathcal{N}_{e_3}^{2\ell} \cup \mathcal{N}_{e_4}^{2\ell}) \geq 2^{(\ell-1)} + 3 > 2\ell + 1$ (as $\ell > 3$), which is a contradiction. Hence, $B(\mathcal{N}_{e_3}^{2\ell-2}) \geq 2$.

Now, $B(\mathcal{N}_{e_3}^{2\ell-2}) \geq 2$ and $B(\mathcal{N}_{e_4}^{2\ell}) \geq 2\ell$ imply that $B(\mathcal{N}_{e_3}^{2\ell} \cup \mathcal{N}_{e_4}^{2\ell}) \geq 2\ell + 2$ which is a contradiction.

Hence, v_2^1 and v_2^2 are both bad variable nodes. ∎

We now have the following theorem.

Theorem 1: If $\mathcal{C}_g$ is a valid configuration on which the Gallager A algorithm is not successful in $(\ell + 1)$ iterations, then there exists a valid configuration $\mathcal{C}_{g-4}$ on which the Gallager A algorithm is not successful in ℓ iterations.

Proof: In the configuration $\mathcal{C}_g$, v_2^1 and v_2^2 are bad variable nodes which send incorrect messages to check nodes c_1^1 and c_1^2, respectively, in the $(\ell+1)$th iteration. This implies that $\mathcal{N}_{e_3}^1$ and $\mathcal{N}_{e_4}^1$ each has a check node, c_3^1 and c_3^2, respectively, that sends an incorrect message in the ℓth iteration to v_2^1 and v_2^2, respectively. Now consider a configuration $\mathcal{C}_{g-4}$ constructed from $\mathcal{C}_g$ by removing the nodes $v_2^1, v_2^2, c_1^1, c_1^2$ and the edges connecting them to their neighbors and introducing the edges (v_0, c_3^1) and (v_0, c_3^2) (see Fig. 6(a)). If $\mathcal{C}_g$ has at most $2\ell + 1$ bad variable nodes and no cycles of length less than g, then $\mathcal{C}_{g-4}$ has at most $2(\ell-1)+1$ bad variable nodes and no cycles of length less than $g-4$. In $\mathcal{C}_{g-4}$, variable node v_0 receives two incorrect messages at the end of ℓ iterations, and, hence, $\mathcal{C}_{g-4}$ is a valid configuration on which the Gallager A algorithm is not successful in ℓ iterations. ∎

Theorem 1 gives a method to construct valid configurations of bad variable nodes for girth g from valid configurations for girth $g + 4$. Also, if $\mathcal{C}_g^1$ and $\mathcal{C}_g^2$ are two distinct valid configurations, then the configurations $\mathcal{C}_{g-4}^1$ and $\mathcal{C}_{g-4}^2$ constructed from $\mathcal{C}_g^1$ and $\mathcal{C}_g^2$, respectively, are distinct. Hence, the number of valid configurations for girth g is greater than or equal to the number of valid configurations for girth $g + 4$. Note that the converse of Theorem 1 is not true in general. However, for $g \geq 16$, we will show in Theorem 2 that any configuration for girth g can be extended to a configuration for girth $g + 4$.

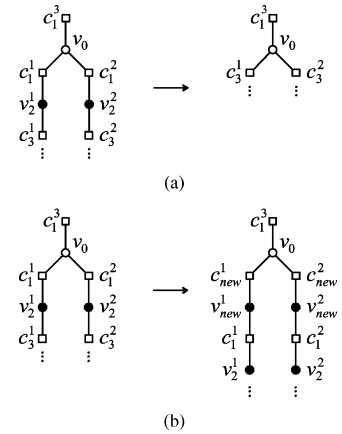

Fig. 6. (a) Construction of $\mathcal{C}_{g-4}$ from $\mathcal{C}_g$. (b) Construction of $\mathcal{C}_{g+4}$ from $\mathcal{C}_g$.

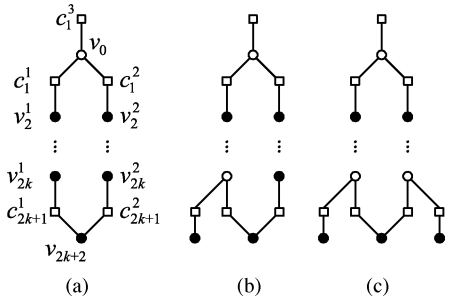

Fig. 7. Configurations of at most $2\ell + 1$ bad variable nodes free of cycles of length less than $4\ell + 4$ on which the Gallager A algorithm is not successful in $(\ell + 1)$ iterations.

Theorem 2: For $g/2$ even and $\ell \geq 3$, there are only three valid configurations on which the Gallager A algorithm is not successful in $(\ell + 1)$ iterations.

Proof: For $\ell = 3$, we have $g = 16$ and there are only three valid configurations as given in Fig. 5. So, for $g \geq 16$ and $g/2$ even, there can be at most three valid configurations. Each valid configuration for $g = 16$, can be extended to a configuration $\mathcal{C}_{20}$ for $g = 20$ by the addition of two bad variable nodes v_{new}^1 and v_{new}^2 in the following way. Remove the edges (v_0, c_1^1) and (v_0, c_1^2). Add bad variable nodes v_{new}^1 and v_{new}^2 and check nodes c_{new}^1 and c_{new}^2. Introduce the edges (v_0, c_{new}^1), (v_0, c_{new}^2), $(v_{\text{new}}^1, c_{\text{new}}^1)$, $(v_{\text{new}}^2, c_{\text{new}}^2)$, $(v_{\text{new}}^1, c_1^1)$ and $(v_{\text{new}}^2, c_1^2)$ (see Fig. 6(b) for an illustration). It can be seen that $\mathcal{C}_{20}$ is a valid configuration for girth $g = 20$. In general, the configurations constructed using the above method from the valid configurations for $g \geq 16$ are valid configurations for $g + 4$. Fig. 7 illustrates the three configurations for all $\ell \geq 3$. ∎

Remark: In all valid configurations $\mathcal{C}_g$ with $g \geq 16$, no bad variable node receives two incorrect messages at the end of the $(\ell + 1)$th iteration.

Theorem 3: The Gallager A algorithm is successful on all valid configurations $\mathcal{C}_g$ in $g/2$ iterations.

Proof: We prove the theorem for one configuration for $g = 16$ only. The proof is similar for other configurations. At the end of fourth iteration, let v_0 receive two incorrect messages [see Fig. 5(a)]. It can be seen that there cannot exist another variable node (either good or bad) which receives two incorrect messages without violating the $g = 16$ constraint. Also, v_8 receives all

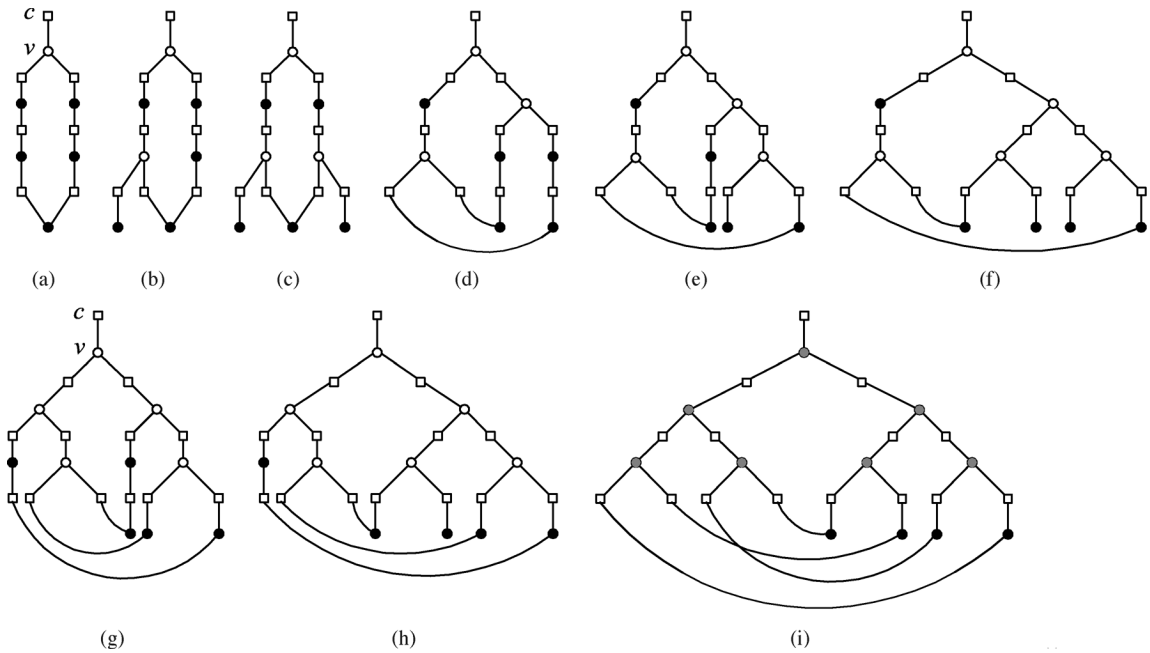

Fig. 8. Configurations of at most five bad variable nodes free of cycles of length less than 12 on which the Gallager A algorithm is not successful in three iterations.

correct messages and $v_2^1, v_2^2, v_4^1, v_4^2, v_6^1, v_6^2$ receive one incorrect message each from $c_3^1, c_3^2, c_5^1, c_5^2, c_7^1, c_7^2$, respectively. In the fifth iteration, we have

$$\begin{aligned}\omega_5\left(v_0, c_1^3\right) &= 1\\ \omega_5(v, : \backslash c) &= \{1\}, \quad (v,c) \in \left\{\left(v_2^1, c_3^1\right), \left(v_2^2, c_3^2\right)\right.\\ &\qquad \left.\left(v_4^1, c_5^1\right), \left(v_4^2, c_5^2\right), \left(v_6^1, c_7^1\right), \left(v_6^2, c_7^2\right)\right\}\\ \omega_5(v,c) &= 0, \quad \text{otherwise.}\end{aligned}$$

In the sixth iteration, we have

$$\begin{aligned}\omega_6\left(v_0, c_1^3\right) &= 1\\ \omega_6(v, : \backslash c) &= \{1\}, \quad (v,c) \in \left\{\left(v_2^1, c_3^1\right), \left(v_2^3, c_3^2\right), \left(v_4^1, c_5^1\right)\right.\\ &\qquad \left.\left(v_4^2, c_5^2\right)\right\}\\ \omega_6(v,c) &= 0, \quad \text{otherwise.}\end{aligned}$$

In the seventh iteration, we have

$$\begin{aligned}\omega_7\left(v_0, c_1^3\right) &= 1\\ \omega_7(v, : \backslash c) &= \{1\}, \quad (v,c) \in \left\{\left(v_2^1, c_3^1\right), \left(v_2^2, c_3^2\right)\right\}\\ \omega_7(v,c) &= 0, \quad \text{otherwise.}\end{aligned}$$

Finally in the eighth iteration, we have

$$\begin{aligned}\omega_8\left(v_0, c_1^3\right) &= 1\\ \omega_8(v,c) &= 0, \quad \text{otherwise.}\end{aligned}$$

At the end of eighth iteration, no variable node receives two incorrect messages, and, hence, the decoder is successful. ■

3) $g = 12$: In this case, $\ell = 2$ and the incoming messages at the end of the second iteration are independent. We need to prove that any code with Tanner graph with $g = 12$, can correct all error patterns of weight less than six. Let v be a variable node which sends an incorrect message in the third iteration along edge $e = (v, c)$ given that there are at most 5 bad variable nodes and $\mathcal{N}_e^5$ is a tree. Fig. 4 illustrates different configurations of bad variable nodes in $\mathcal{N}_e^4$.

Fig. 8 shows all possible configurations of five or less bad variable nodes on which the Gallager A algorithm is not successful at the end of three iterations. However, the Gallager A algorithm is successful on all the configurations in six iterations. The proofs for configurations in Fig. 8(a)–(h) are similar to the proof for configuration in Fig. 5(a) and are omitted. Since, configuration (i) has only four bad variable nodes, a complete proof for convergence requires considering all possible locations of the fifth bad variable node, but other than that the structure of the proof is identical to that of the proof for the configuration in Fig. 5(a). It is worth noting that in this case, there exist configurations in which a bad variable node receives two incorrect messages at the end of the third iteration. However, the Gallager A algorithm is successful on all the configurations eventually.

B. $g/2$ Is Odd

In this case, we have $\ell = (g-2)/4$ and we need to prove that the code is capable of correcting all error patterns of weight $(g/2-1) = 2\ell$ or less. The methodology of the proof is similar to the proof in the case when $g/2$ is even. In this case, we have $\mathcal{N}_{e_3}^{d_1} \cap \mathcal{N}_{e_4}^{d_2} = \phi$ for $d_1 + d_2 < 4\ell - 2$. This implies that $\mathcal{N}_{e_3}^{2\ell}$ and $\mathcal{N}_{e_4}^{2\ell}$ can have a common node at depth $2\ell - 1$. Therefore, in any configuration, $B(\mathcal{N}_{e_3}^{2\ell} \cup \mathcal{N}_{e_4}^{2\ell})$ is lower bounded by $\max\left(B(\mathcal{N}_{e_3}^{2\ell-4}) + B(\mathcal{N}_{e_4}^{2\ell}), B(\mathcal{N}_{e_3}^{2\ell}) + B(\mathcal{N}_{e_4}^{2\ell-4})\right)$. The valid configurations in this case are the ones which satisfy $B(\mathcal{N}_{e_3}^{2\ell} \cup \mathcal{N}_{e_4}^{2\ell}) \leq 2\ell$. We again deal with $g \geq 14$ and $g = 10$ separately.

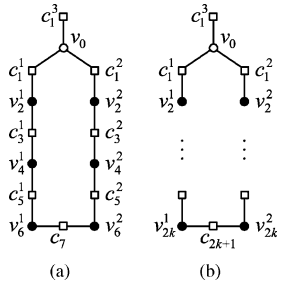

Fig. 9. (a) Configuration of at most six bad variable nodes free of cycles of length less than 14 on which the Gallager A algorithm is not successful in four iterations. (b) Configuration of at most 2ℓ bad variable nodes free of cycles of length less than $4\ell + 2$ on which the Gallager A algorithm is not successful in $\ell + 1$ iterations.

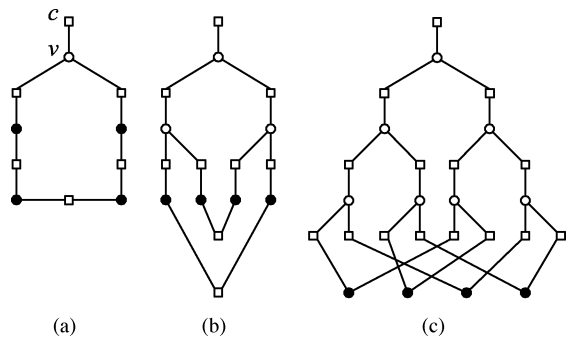

Fig. 10. Configurations of at most four variable nodes free of cycles of length less than ten on which the Gallager A algorithm is not successful in three iterations.

1) $g \geq 14$:

Lemma 4: For $g = 14$, there is only one configuration of six bad variable nodes on which the Gallager A algorithm is not successful in four iterations.

Proof: Using arguments outlined above and the configurations in Fig. 3(b) along with the constraint that $g \geq 14$, we conclude that there is only one configuration which is not successful in four iterations, which is shown in Fig. 9(a). ■

Lemma 5: If $\mathcal{C}_g$ with $g \geq 14$ is a valid configuration on which the Gallager A algorithm is not successful in $\ell + 1$ iterations, then v_1^1 and v_1^2 are bad variable nodes.

Proof: Similar to the proof of Lemma 3. ■

Theorem 4: If $\mathcal{C}_g$ is a valid configuration on which the Gallager A algorithm is not successful in $\ell+1$ iterations, then there exists a valid configuration $\mathcal{C}_{g-4}$ on which the Gallager A is not successful in ℓ iterations.

Proof: Similar to the proof of Theorem 1. ■

Theorem 5: For $\ell \geq 3$, there is only one valid configuration on which the Gallager A algorithm is not successful in $\ell + 1$ iterations.

Proof: For $\ell = 3$, we have $g = 14$ and there is only one configuration. For $\ell = 4$, the number of valid configurations cannot be more than one. The valid configuration for $g = 14$, can be extended to a configuration for $g = 18$ (in the same manner as in Theorem 2). In general, the valid configuration for girth g can be extended to a valid configuration for girth $g + 4$. Fig. 9(b) shows $\mathcal{C}_g$ for all $g \geq 14$. ■

Theorem 6: The Gallager A algorithm is successful on the configuration $\mathcal{C}_g$ in $g/2$ iterations.

Proof: Similar to the proof of Theorem 3. ■

2) $g = 10$: In this case, $\ell = 2$ and there are three configurations on which the Gallager A algorithm is not successful at the end of the third iteration. Fig. 10 shows the three configurations. It can be shown that the Gallager A algorithm is successful on these configurations in five iterations.

V. Case of Girth $g \leq 8$

In Section IV, we established that the Gallager A algorithm for a column-weight-three LDPC code with Tanner graph of girth $g \geq 10$ can correct $(g/2 - 1)$ errors in $g/2$ iterations. In [18], it was shown that the Gallager A algorithm for a column-weight-three code with $g \geq 10$ cannot correct all error patterns with up to $g/2$ errors. Hence, the bounds established in this paper are tight for codes with Tanner graphs of girth $g \geq 10$ and cannot be improved. In this section, we discuss the case of codes with Tanner graphs of girth $g \leq 8$ and the conditions on the Tanner graph to correct all error patterns with up to 1, 2, and 3 errors.

Before we proceed, we give an important remark about the decision rule. As mentioned in Section II-B, we adopt Decision rule A for codes with Tanner graphs of girth $g \leq 8$. This is due to the fact that when the number of errors is small (as is the case), adopting Decision rule A leads to faster convergence to a codeword than adopting Decision rule B. In Decision rule B, any variable node which receives two incorrect messages is decoded wrongly. For Tanner graphs of girth $g \leq 8$, due to the dense connectedness, it is possible that many good variable nodes receive two incorrect messages thereby leading to non-convergence. In Decision rule A, a good variable node is decoded wrongly only if it receives three incorrect messages and if the number of errors is small, it is less likely for this scenario to arise. While bad variable nodes need only one incorrect message to be decoded wrongly, when the number of errors is small, such variable nodes receive all correct messages as the iterations progress. It is worth noting that the results established for codes with Tanner graphs of girth $g \geq 10$ hold for Decision rule A also. But we adopted Decision rule B, as the proof in this case is more elegant. In practice, however, it is beneficial to adopt Decision rule A due to the following reasons. The bit error rate (BER) performance when the number of errors introduced by the channel is small is better for Decision rule A compared to Decision rule B. Moreover, the Tanner graphs associated with small to moderate length codes of high rates have low girth and adopting Decision rule A would lead to faster convergence.

For a column-weight-three code with Tanner graph of girth $g \leq 8$, it cannot be proved that the Gallager A algorithm can correct all error patterns with up to $(g/2-1)$ errors. This is due to the fact that such Tanner graphs can contain codewords of weight less than or equal to $2(g/2-1)$ and in such cases even the maximum likelihood decoder is not guaranteed to correct $(g/2-1)$ errors. Specifically, a code with Tanner graph of girth four can contain a codeword of weight two, a code with Tanner

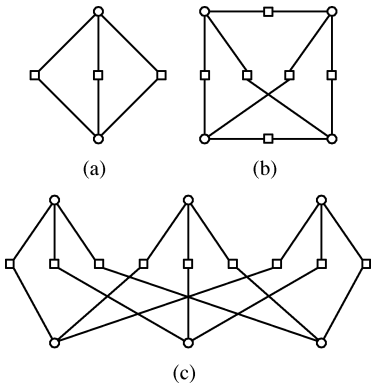

Fig. 11. (a) Codeword of weight two in a Tanner graph of girth $g = 4$. (b) Codeword of weight four in a Tanner graph of girth $g = 6$. (c) Codeword of weight six in a Tanner graph of girth $g = 8$.

graph of girth six can contain a codeword of weight four and a code with Tanner graph of girth eight can contain a codeword of weight six [see Fig. 11(a)–(c) for illustration[1]]. Hence, it is necessary to avoid all codewords of weight less than or equal to $2(g/2-1)$ to guarantee correction of $(g/2-1)$ errors.

For codes with Tanner graphs of girth four, it is easy to show that the Gallager A algorithm can correct a single error in one iteration if there are no codewords of weight two. For codes with Tanner graphs of girth six, we have the following theorem.

Theorem 7: A column-weight-three LDPC code with Tanner graph G of girth $g = 6$ can correct all error patterns with up to two errors in at most two iterations of the Gallager A algorithm if G does not contain any codewords of weight four.

A. Case of $g = 8$

Codes with Tanner graphs of girth $g = 8$ require a separate discussion. In this case, avoiding codewords up to weight six alone cannot guarantee correction of all error patterns with up to three errors under the Gallager A algorithm.

We first exhibit two trapping sets with induced subgraphs of girth $g = 8$, which have critical number three under certain conditions. We then show that avoiding these trapping sets in a Tanner graph of girth $g = 8$ is sufficient to guarantee the correction of three errors under the Gallager A algorithm.

Lemma 6: The Tanner graph of a column-weight-three code with girth $g = 8$ can contain trapping sets with critical number three. Specifically, there exist (5,3) and (8,0) trapping sets with critical number three.

Proof: We prove the lemma for the case of the (5,3) trapping set shown in Fig. 12(a). The proof for the (8,0) trapping set shown in Fig. 12(b) is similar and is omitted.

Consider the (5, 3) trapping set shown in Fig. 13. Let $V_0 := \{v_0^1, v_0^2, v_0^3\}$ be the set of variable nodes which are initially in error. Let $C_1 := \{c_1^1, c_1^2, \ldots, c_1^9\}$ and $V_2 := \{v_2^1, v_2^2\}$. Also,

[1]We note here that in Figs. 11, 12, and 15, ∘ just indicates a variable node with no value (good or bad) attached.

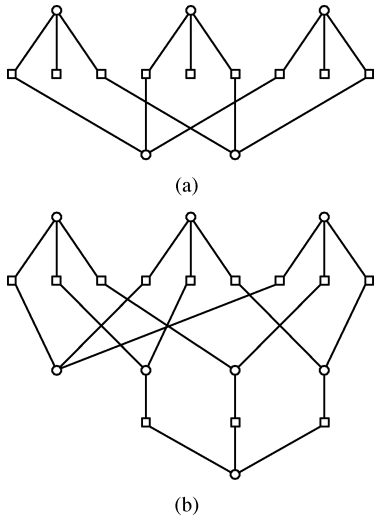

Fig. 12. Trapping sets with induced subgraphs of girth $g = 8$ and critical number three. (a) A (5,3) trapping set. (b) An (8,0) trapping set.

assume that no variable node in $V \backslash (V_0 \cup V_2)$, has two or more neighbors in C_1. In the first iteration, we have

$$\omega_1(v,c) = \begin{cases} 1, & \text{if } v \in V_0 \\ 0, & \text{otherwise} \end{cases}$$

and

$$\varpi_1(c,v) = \begin{cases} 1, & \text{if } c \in C_1, v \notin V_0 \\ 0, & \text{otherwise.} \end{cases}$$

All variable nodes in V_2 receive three incorrect messages and are decoded incorrectly at the end of the first iteration. In the second iteration

$$\omega_2(v,c) = \begin{cases} 1, & \text{if } v \in V_2 \\ 0, & \text{otherwise} \end{cases}$$

and

$$\varpi_2(c,v) = \begin{cases} 1, & \text{if } c \in C_1 \backslash \left\{c_1^2, c_1^5, c_1^8\right\}, v \notin V_2 \\ 0, & \text{otherwise} \end{cases}$$

and all variable nodes in V_0 are decoded incorrectly. Continuing in this fashion, $\omega_3(v,c) = \omega_1(v,c)$ and $\varpi_3(c,v) = \varpi(c,v)$. That is, the messages being passed in the Tanner graph repeat after every two iterations. Hence, three variable nodes in error initially can lead to a decoder failure, and, therefore, this (5, 3) trapping set has critical number equal to three. ■

Theorem 8: If the Tanner graph of a column-weight-three LDPC code has girth eight and does not contain a subgraph isomorphic to a (5,3) trapping set of Fig. 12(a) or a subgraph isomorphic to an (8,0) trapping set of Fig. 12(b), then the code can correct all error patterns with up to three errors in at most three iterations of the Gallager A algorithm.

Proof: Let $V_0 := \{v_0^1, v_0^2, v_0^3\}$ be the set of three variable nodes initially in error and C_1 be the set of the check nodes connected to the variable nodes in V_0. In a column-weight-three code free of cycles up to length six, the variable nodes in V_0 can induce only one of the four subgraphs given in Fig. 14. In each case, $\omega_1(v,c) = 1$ if $v \in V_0$ and is 0 otherwise. The proof proceeds by examining these subgraphs one at a time and

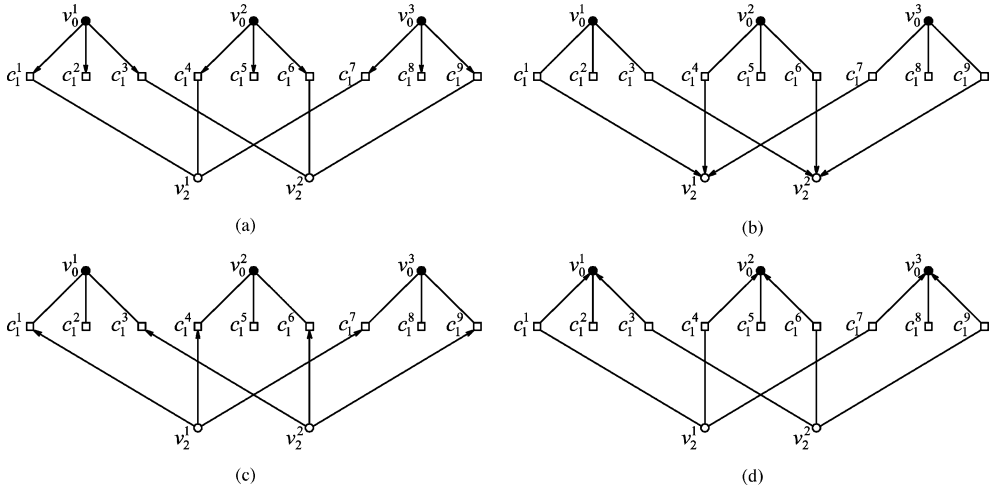

Fig. 13. Illustration of message passing for a (5,3) trapping set: (a) variable node to check node messages in the first iteration, (b) check node to variable node messages in the first iteration, (c) variable node to check node messages in the second iteration, and (d) check node to variable node messages in the second iteration. Arrow-heads indicate the messages with value 1.

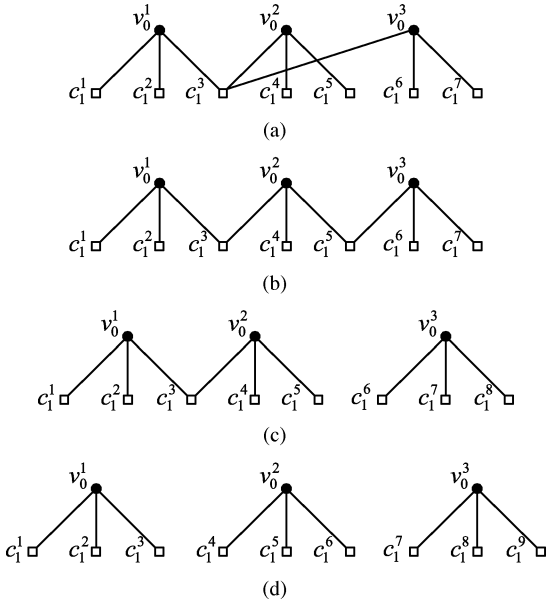

Fig. 14. All the possible subgraphs that can be induced by three variable nodes in a column-weight-three code with Tanner graph of girth $g = 8$.

proving the correction of the three errors in each case. Below we provide the complete proof for the case depicted in Fig. 14(b) and relegate the proofs for other cases to Appendix A.

Subgraph 2: The variable nodes in V_0 induce the subgraph shown in Fig. 14(b). At the end of the first iteration

$$\varpi_1(c,v)=\begin{cases}1, & \text{if } c\in C_1\backslash\left\{c_1^3,c_1^5\right\}, v\notin V_0\\ 1, & \text{if } c\in\left\{c_1^3,c_1^5\right\}, v\in V_0\\ 0, & \text{otherwise.}\end{cases} \tag{2}$$

For no $v\notin V_0$, $\varpi_1(:,v)=\{1\}$ as this would introduce a four-cycle or a six-cycle in the graph. For any $v\notin V_0$, $\omega_2(v,c)=1$ only if $\varpi_1(:\backslash c,v)=\{1\}$. This implies that v must have two check node neighbors in $C_1\backslash\{c_1^3,c_1^5\}$. Let V_2 be the set of such variable nodes. We have the following lemma:

Lemma 7: There can be at most one variable node in V_2.

Proof: Suppose $|V_2|=2$. Specifically, assume $V_2=\{v_2^1,v_2^2\}$. The proof is similar for $|V_2|>2$. First note that for any $v\in V_2$, v cannot be connected to c_1^4 as it would introduce a six-cycle. Next, let $C_1^1:=\{c_1^1,c_1^2\}$ and $C_1^2:=\{c_1^6,c_1^7\}$. Then, any $v\in V_2$ cannot have both check nodes in either C_1^1 or C_1^2 as this would introduce a four cycle. Hence, any $v\in V_2$ has one check node neighbor in C_1^1 and one check node neighbor in C_1^2. Assume without loss of generality that $v_2^1\in V_2$ is connected to c_1^1 and c_1^6. Then, v_2^2 cannot be connected to c_1^1 and c_1^7 as this would introduce a six-cycle. Also, v_2^2 cannot be connected to c_1^2 and c_1^7 as this would introduce a (5, 3) trapping set. Hence, $|V_2|<2$. ∎

Now, let $v_2^1\in V^2$ be connected to c_1^1, c_1^6 and an additional check c_3^1. In the second iteration

$$\omega_2(v,c)=\begin{cases}1, & \text{if } v\in\left\{v_0^1,v_0^3\right\}, c\notin\left\{c_1^3,c_1^5\right\}\\ 1, & \text{if } v=v_0^2\\ 1, & \text{if } v=v_2^1, c=c_3^1\\ 0, & \text{otherwise}\end{cases}$$

and

$$\varpi_2(c,v)=\begin{cases}1, & \text{if } c\in C^1\backslash\left\{c_1^4\right\}, \quad v\notin V_0\\ 1, & \text{if } c\in\left\{c_1^3,c_1^4,c_1^5\right\}, \quad v\neq v_0^2\\ 1, & \text{if } c=c_3^1, v\neq v_2^1\\ 0, & \text{otherwise.}\end{cases}$$

We have the following lemma.

Lemma 8: There cannot exist any variable node $v\notin V_0\bigcup V_2$ that receives two or more incorrect messages at the end of the second iteration.

Proof: Suppose there exists a variable node v that receives two incorrect messages in the second iteration. Then, it would be connected to two check nodes in the set $C_1\bigcup\{c_3^1\}$. This is not possible as it would introduce a four-cycle or a six-cycle or a (5, 3) trapping set (e.g., if v is connected to c_1^4 and c_3^1, then $\{v_0^1,v_0^2,v_0^3,v_2^1,v\}$ would form a (5, 3) trapping set). ∎

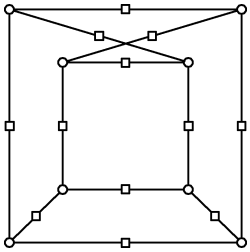

Fig. 15. Codeword of weight eight with critical number four.

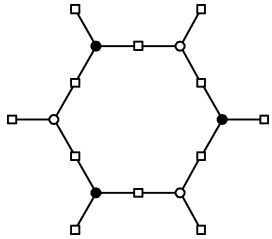

Fig. 16. Illustration of a cycle of length 12.

Thus, in the third iteration

$$\omega_3(v,c)=\begin{cases}1, & \text{if } v\in\left\{v_0^1,v_0^3\right\}, c\notin\left\{c_1^3,c_1^5\right\}\\ 1, & \text{if } v=v_2^1, c=c_3^1\\ 0, & \text{otherwise}\end{cases}$$

and

$$\varpi_3(c,v)=\begin{cases}1 & \text{if } c\in\left\{c_1^1,c_1^2,c_1^6,c_1^7\right\}, v\notin\left\{v_0^1,v_0^3\right\}\\ 1 & \text{if } c=c_3^1, v\neq v_2^1\\ 0, & \text{otherwise.}\end{cases}$$

At the end of the third iteration, $\varpi(:,v)=\{0\}$ for all $v\in V_0$. Also, we have the following lemma.

Lemma 9: There exists no $v\notin V_0$ such that $\varpi_3(:,v)=\{1\}$.

Proof: Suppose there exists v such that $\varpi_3(:,v)=\{1\}$. Then, v is connected to three check nodes in the set $\{c_1^1,c_1^2,c_1^6,c_1^7,c_3^1\}$. This implies that $\varpi_2(:,v)=\{1\}$. However, from Lemma 8 it is evident that no such v exists. ∎

Hence, if a decision is made after the third iteration, a valid codeword is found and the decoder is successful.

Remark: It is interesting to note that there is another (8,0) trapping set which is shown in Fig. 15. But the critical number of this (8,0) trapping set is four.

VI. Discussion

Gallager in [1] showed that the girth of the Tanner graph of any code in the ensemble of (n,d_v,d_c) regular codes is bounded by

$$g\leq 4\left(\frac{\log n}{\log\left((d_v-1)(d_c-1)\right)}+1\right).$$

Additionally, an explicit construction of codes whose Tanner graphs satisfy

$$g>4\left(\frac{\log n+\log\frac{d_vd_c-d_v-d_c}{2d_c}}{2\log(d_v-1)(d_c-1)}-1\right)$$

was proposed. Codes with Tanner graphs of girth growing logarithmically with the code length were also proposed by Margulis [22] and further investigated by Rosenthal and Vontobel [23].

The results presented in this paper, combined with the results of [18], establish that the guaranteed error correction capability of column-weight-three LDPC codes under the Gallager A algorithm grows logarithmically with the code length. This is an interesting result due to the fact that while the minimum distance of codes in this ensemble increases linearly with the code length, the guaranteed error correction capability increases only logarithmically with the code length. This is in contrast to the results obtained so far for higher column weight codes for which correction of a linear number of errors in code length was established. Considering error correction capability as a function of girth also provides interesting results. The minimum distance in this scenario increases exponentially with girth while the guaranteed error correction capability grows only linearly with the girth. It is worth also noting that for LP decoding on the BSC, the error correction capability for column-weight-three codes grows exponentially with the girth.

The results derived in this paper do not apply to the parallel bit flipping algorithm. It can be easily shown that when $g/2$ is even, $g/4$ variable nodes in error can lead to oscillations in the bit flipping algorithm. Fig. 16 shows a set of six variable nodes that form a cycle of length 12. If the bad variable nodes are located as shown in Fig. 16, it can be seen that the variable nodes in error at the end of a decoding round in the parallel bit flipping algorithm oscillate, thereby resulting in a decoding failure. This observation leads to the conclusion that the Gallager A algorithm can correct (approximately) twice the number of errors compared to the parallel bit flipping algorithm.

We conclude with a few open problems and suggestions for future work that are of considerable interest:

- Is there a decoding algorithm, perhaps one that employs more bits in the message alphabet, under which column-weight-three LDPC codes can correct a linear number of errors in the code length?
- Can it be proved that column-weight-three LDPC codes can correct a linear number of errors in the code length under the LP decoder?
- It would be interesting to see if the method outlined in this paper can be applied to higher column weight codes to identify trapping sets as well as derive tighter bounds on the guaranteed error correction capability.
- Algorithms that utilize soft information from the channel and/or employ more bits in the message alphabet have the potential to correct more errors. Future work includes the extension of the results derived in this paper to quantized belief propagation algorithms over the BSC as well as the additive white Gaussian noise (AWGN) channel.

Appendix A
Proof of Theorem 8

Subgraph 1: The variable nodes in V_0 induce the subgraph shown in Fig. 14(a). At the end of the first iteration

$$\varpi_1(c,v)=\begin{cases}1, & \text{if } c\in C_1, \quad v\notin V_0\\ 0, & \text{otherwise.}\end{cases}$$

There cannot exist a variable node $v \notin V_0$ which is connected to two or more check nodes in the set C_1 without introducing either a six-cycle or a subgraph isomorphic to (5,3) trapping set. At the end of first iteration, $\varpi_1(:,v) = \{0\}$ for all $v \in V_0$. Furthermore, there exists no $v \notin V_0$ for which $\varpi_1(:,v) = \{1\}$. Hence, if a decision is made after the first iteration, a valid codeword is found and the decoder is successful.

Subgraph 3: The variable nodes in V_0 induce the subgraph shown in Fig. 14(c). At the end of the first iteration

$$\varpi_1(c,v) = \begin{cases} 1, & \text{if } c \in C_1 \backslash \{c_1^3\}, \quad v \notin V_0 \\ 1, & \text{if } c = c_1^3, v \in \{v_0^1, v_0^2\} \\ 0, & \text{otherwise.} \end{cases}$$

For no $v \in V \backslash V_0, \varpi_1(:,v) = \{1\}$. For any $v \in V \backslash V_0$, $\omega_2(v,c) = 1$ only if $\varpi_1(: \backslash c, v) = \{1\}$. We have the following lemma:

Lemma 10: Let V_2 be the set of all variable nodes in $V \backslash V_0$ that receive two incorrect messages at the end of the first iteration. Then, (i) V_2 has at most four variable nodes and, (ii) no two variable nodes in V_2 can share a check node in $C \backslash C_1$.

Proof: There exists no variable node which is connected to two check nodes from the set $\{c_1^1, c_1^2, c_1^3, c_1^4, c_1^5\}$ as it would introduce a four-cycle or a six-cycle. However, a variable node can be connected to one check node from $\{c_1^1, c_1^2, c_1^4, c_1^5\}$ and to one check node from $\{c_1^6, c_1^7, c_1^8\}$. Note that a variable node connected to c_1^3 does not receive an incorrect message at the end of the first iteration. It is easy to see that there can be at most four variable nodes which receive two incorrect messages without introducing a four cycle or a six cycle. Also, these four variable nodes cannot share check nodes outside the set C_1 without introducing a four cycle or a six cycle. ∎

Let these four variable nodes be labeled $v_2^1, v_2^2, v_2^3, v_2^4$ and their third check node neighbors be $c_3^1, c_3^2, c_3^3, c_3^4$, respectively. Let $C_3 := \{c_3^1, c_3^2, c_3^3, c_3^4\}$. Hence, in the second iteration

$$\omega_2(v,c) = \begin{cases} 1, & \text{if } v \in \{v_0^1, v_0^2\}, \quad c \neq c_1^3 \\ 1, & \text{if } v \in V_2, \quad c \in C_3 \\ 0, & \text{otherwise} \end{cases}$$

and

$$\varpi_2(c,v) = \begin{cases} 1, & \text{if } c \in \{c_1^1, c_1^2, c_1^4, c_1^5\}, v \notin \{v_0^1, v_0^2\} \\ 1, & \text{if } c \in C_3, v \notin V_2 \\ 0, & \text{otherwise.} \end{cases}$$

At the end of the second iteration, $\varpi_2(:,v) = \{0\}$ for all $v \in V_0$. Moreover, for no $v \notin V_0$, $\varpi_2(:,v) = \{1\}$. So, if a decision is made after the second iteration, a valid codeword is reached and the decoder is successful.

Subgraph 4: The variable nodes in V_0 induce the subgraph shown in Fig. 14(d). At the end of the first iteration

$$\varpi_1(c,v) = \begin{cases} 1, & \text{if } c \in C_1, \quad v \notin V_0 \\ 0, & \text{otherwise.} \end{cases}$$

If there exists no variable node $v \in V \backslash V_0$ such that $\varpi(:,v) = \{1\}$, a valid codeword is reached after the first iteration. Suppose this is not the case. Then, we have the following lemma.

Lemma 11: Let V_2 be the set of variable nodes which receive two or more incorrect messages at the end of the first iteration. Then, (i) there exists at most one variable node $v_2^1 \in V_2$ such that $\varpi_1(:,v_2^1) = \{1\}$, and (ii) there exist at most three variable nodes $\{v_2^2, v_2^3, v_2^4\} \in V_2$ which receive two incorrect messages at the end of the first iteration. Furthermore, no two variable nodes in $\{v_2^2, v_2^3, v_2^4\}$ share a check node in $C \backslash C_1$.

Let the third check nodes connected to $\{v_2^2, v_2^3, v_2^4\}$ be c_3^2, c_3^3 and c_3^4, respectively, and let $C_2 := \{c_3^2, c_3^3, c_3^4\}$. In the second iteration

$$\omega_2(v,c) = \begin{cases} 1, & \text{if } v = v_2^1 \\ 1, & \text{if } v \in V_2 \backslash \{v_2^1\}, \quad c \in C_2 \\ 0, & \text{otherwise} \end{cases}$$

and

$$\varpi_2(c,v) = \begin{cases} 1, & \text{if } c \in \{c_1^1, c_1^3, c_1^7\}, \quad v \neq v_2^1 \\ 1, & \text{if } c \in C_3, v \notin V_2 \\ 0, & \text{otherwise.} \end{cases}$$

It can be shown that there cannot exist a variable node which is connected to one check node from C_3 and to one check node from $\{c_1^1, c_1^3, c_1^7\}$. It can also be shown that there cannot be a variable node which is connected to all three check nodes in the set C^2 as this would introduce a graph isomorphic to the (8,0) trapping set. However, there can be at most two variable nodes which receive two incorrect messages from the check nodes in C_3, say v_4^1 and v_4^2. Let the third check nodes connected to them be c_5^1 and c_5^2, respectively. Let $V_4 := \{v_4^1, v_4^2\}$ and $C_5 := \{c_5^1, c_5^2\}$. At the end of the second iteration, variable nodes v_0^1, v_0^2 and v_0^3 receive one incorrect message each. Variables in the set V_4 receive two incorrect messages each. Therefore, in the third iteration, we have

$$\omega_3(v,c) = \begin{cases} 1, & \text{if } v \in V_0, \quad c \notin \{c_1^1, c_1^4, c_1^7\} \\ 1, & \text{if } v \in V_4, c \in C_3 \\ 0, & \text{otherwise} \end{cases}$$

and

$$\varpi_3(c,v) = \begin{cases} 1, & \text{if } c \in C_1 \backslash \{c_1^1, c_1^4, c_1^7\}, \quad v \notin V_0 \\ 1, & \text{if } c \in C_5, v \notin V_4. \end{cases}$$

At the end of the third iteration, $\varpi_3(:,v) = \{0\}$ for all $v \in V_0$. Furthermore, for no $v \notin V_0$, $\varpi_3(:,v) = \{1\}$. So, if a decision is made after the third iteration, a valid codeword is reached and the decoder is successful. ∎

REFERENCES

[1] R. G. Gallager, *Low Density Parity Check Codes*. Cambridge, MA: MIT Press, 1963.

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

[3] T. J. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 599–618, Feb. 2001.

[4] T. J. Richardson, M. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 638–656, Feb. 2001.

[5] D. J. C. MacKay and M. J. Postol, "Weaknesses of Margulis and Ramanujan–Margulis low-density parity-check codes," in *Proc. MFCSIT*, 2003, vol. 74 [Online]. Available: http://www.inference.phy.cam.ac.uk/mackay/abstracts/margulis.html, Elsevier, Galway, ser. Electronic Notes in Theoretical Computer Science

[6] T. J. Richardson, "Error floors of LDPC codes," in *Proc. 41st Annu. Allerton Conf. Communications, Control and Computing*, 2003, pp. 1426–1435 [Online]. Available: http://www.hpl.hp.com/personal/Pascal_Vontobel/pseudocodewords/papers

[7] M. Ivkovic, S. K. Chilappagari, and B. Vasic, "Eliminating trapping sets in low-density parity-check codes by using Tanner graph covers," *IEEE Trans. Inf. Theory*, vol. 54, no. 8, pp. 3763–3768, Aug. 2008.
[8] V. V. Zyablov and M. S. Pinsker, "Estimation of the error-correction complexity for Gallager low-density codes," *Probl. Inf. Transmission*, vol. 11, no. 1, pp. 18–28, 1976.
[9] M. Sipser and D. Spielman, "Expander codes," *IEEE Trans. Inf. Theory*, vol. 42, no. 6, pp. 1710–1722, Nov. 1996.
[10] D. Burshtein and G. Miller, "Expander graph arguments for message-passing algorithms," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 782–790, Mar. 2001.
[11] D. Burshtein, "On the error correction of regular LDPC codes using the flipping algorithm," *IEEE Trans. Inf. Theory*, vol. 54, no. 2, pp. 517–530, Feb. 2008.
[12] J. Feldman, M. Wainwright, and D. Karger, "Using linear programming to decode binary linear codes," *IEEE Trans. Inf. Theory*, vol. 51, no. 3, pp. 954–972, Mar. 2005.
[13] J. Feldman, T. Malkin, R. A. Servedio, C. Stein, and M. J. Wainwright, "LP decoding corrects a constant fraction of errors," *IEEE Trans. Inf. Theory*, vol. 53, no. 1, pp. 82–89, Jan. 2007.
[14] C. Daskalakis, A. G. Dimakis, R. M. Karp, and M. J. Wainwright, "Probabilistic analysis of linear programming decoding," *IEEE Trans. Inf. Theory*, vol. 54, no. 8, pp. 3565–3578, Aug. 2008.
[15] S. K. Chilappagari, D. V. Nguyen, B. Vasic, and M. W. Marcellin, "On trapping sets and guaranteed error correction capability of LDPC codes and GLDPC codes," *IEEE Trans. Inf. Theory* vol. 56, no. 4, pp. 1600–1611, Apr. 2010.
[16] C. Di, D. Proietti, T. Richardson, E. Telatar, and R. Urbanke, "Finite length analysis of low-density parity-check codes on the binary erasure channel," *IEEE Trans. Inf. Theory*, vol. 48, pp. 1570–1579, 2002.
[17] A. Orlitsky, R. Urbanke, K. Viswanathan, and J. Zhang, "Stopping sets and the girth of Tanner graphs," in *Proc. IEEE Int. Symp. Information Theory*, 2002, p. 2.
[18] S. K. Chilappagari and B. Vasic, "Error correction capability of column-weight-three LDPC codes," *IEEE Trans. Inf. Theory*, vol. 55, no. 5, pp. 2055–2061, May 2009.
[19] S. K. Chilappagari, A. R. Krishnan, and B. Vasic, "LDPC codes which can correct three errors under iterative decoding," in *Proc. IEEE Inf. Theory Workshop*, May 2008, pp. 406–410.
[20] A. Shokrollahi, "An introduction to low-density parity-check codes," in *Theoretical Aspects of Computer Science: Advanced Lectures*. New York: Springer-Verlag, 2002, pp. 175–197.
[21] S. K. Chilappagari, M. Chertkov, M. G. Stepanov, and B. Vasic, "Instanton-based techniques for the analysis and reduction of error floors of LDPC codes," presented at the IEEE JSAC on Capacity Approaching Codes, 2009 [Online]. Available: http://arxiv.org/abs/0903.1624
[22] G. A. Margulis, "Explicit constructions of graphs without short cycles and low density codes," *Combinatorica*, vol. 2, no. 1, pp. 71–78, 1982.
[23] J. Rosenthal and P. O. Vontobel, "Constructions of LDPC codes using Ramanujan graphs and ideas from Margulis," in *Proc. 38th Annu. Allerton Conf. Communication, Control, and Computing*, 2000, pp. 248–257.

Shashi Kiran Chilappagari (S'05–M'09) received the B.Tech. and M.Tech. degrees in electrical engineering from the Indian Institute of Technology, Madras, India, in 2004, and the Ph.D. degree in electrical engineering from the University of Arizona, Tucson, in 2008.

He was a Research Engineer in the Department of Electrical and Computer Engineering at the University of Arizona from January 2009 to December 2009. He is currently with Marvell Semiconductor, Inc., Santa Clara, CA. His research interests include error control coding and information theory with focus on the analysis of failures of various suboptimal decoding algorithms for LDPC codes.

Dung Viet Nguyen (S'07) received the B.S. degree in electrical engineering from the University of Arizona, Tucson, in 2007, where he is currently pursuing the Ph.D. degree.

His research interests include digital communications and information theory.

Bane Vasic (S'92–M'93–SM'02) received the B.Sc., M.Sc., and Ph.D. degrees in electrical engineering from the University of Nis, Nis, Yugoslavia (now Serbia), in 1989, 1991, and 1994, respectively.

From 1996 to 1997, he was a Visiting Scientist at the Rochester Institute of Technology and Kodak Research, Rochester, NY, where he was involved in research in optical storage channels. From 1998 to 2000, he was with Lucent Technologies, Bell Laboratories (Bell-Labs). He was involved in research in iterative decoding and low-density parity-check codes, as well as development of codes and detectors implemented in Bell-Labs chips. Presently, he is a Professor in the Electrical and Computer Engineering Department, University of Arizona, Tucson. His research interests include coding theory, information theory, communication theory, and digital communications and recording.

Dr. Vasic is a Member of the Editorial Board for the IEEE TRANSACTIONS ON MAGNETICS. He served as Technical Program Chair of the IEEE Communication Theory Workshop in 2003 and as Co-organizer of the Center for Discrete Mathematics and Theoretical Computer Science (DIMACS) Workshops on Optical/Magnetic Recording and Optical Transmission, and Theoretical Advances in Information Recording in 2004. He was Co-organizer of the Los Alamos Workshop on Applications of Statistical Physics to Coding Theory in 2004, the Communication Theory Symposium within the IEEE International Conference on Communications (ICC 2006), and IEEE Communication Theory Workshop (CTW 2007).

Michael W. Marcellin (S'81–M'87–SM'93–F'02) was born in Bishop, CA, on July 1, 1959. He graduated summa cum laude with the B.S. degree in electrical engineering from San Diego State University, San Diego, CA, in 1983, where he was named the most outstanding student in the College of Engineering. He received the M.S. and Ph.D. degrees in electrical engineering from Texas A&M University, College Station, in 1985 and 1987, respectively.

Since 1988, he has been with the University of Arizona, Tucson, where he holds the title of Regents' Professor of Electrical and Computer Engineering, and of Optical Sciences. His research interests include digital communication and data storage systems, data compression, and signal processing. He has authored or coauthored more than two hundred publications in these areas. He is a major contributor to JPEG2000, the emerging second-generation standard for image compression. Throughout the standardization process, he chaired the JPEG2000 Verification Model Ad Hoc Group, which was responsible for the software implementation and documentation of the JPEG2000 algorithm. He is coauthor of the book *JPEG2000: Image Compression Fundamentals, Standards and Practice* (Kluwer, 2002). This book is intended to serve as a graduate level textbook on image compression fundamentals, as well as the definitive reference on JPEG2000. He served as a consultant to Digital Cinema Initiatives (DCI), a consortium of Hollywood studios, on the development of the JPEG2000 profiles for digital cinema.

Prof. Marcellin is a member of Tau Beta Pi, Eta Kappa Nu, and Phi Kappa Phi. He is a 1992 recipient of the National Science Foundation Young Investigator Award and a corecipient of the 1993 IEEE Signal Processing Society Senior (Best Paper) Award. He has received teaching awards from NTU (1990, 2001), IEEE/Eta Kappa Nu student sections (1997), and the University of Arizona College of Engineering (2000). In 2003, he was named the San Diego State University Distinguished Engineering Alumnus. He is the recipient of the 2006 University of Arizona Technology Innovation Award. From 2001 to 2006, he was the Litton Industries John M. Leonis Professor of Engineering. He is currently the International Foundation for Telemetering Professor of Electrical and Computer Engineering at the University of Arizona.