Low-Density Parity-Check Codes Which Can Correct Three Errors Under Iterative Decoding

Graduate Students: Anantha Raman Krishnan and Shashi Kiran Chilappagari
Advisors: Bane Vasic and Michael W. Marcellin
Department of Electrical and Computer Engineering, University of Arizona, Tucson

Abstract

In this paper, we give necessary and sufficient conditions for low-density parity-check (LDPC) codes with column-weight four to correct three errors when decoded using hard-decision message-passing decoding. We then give a construction technique which results in codes satisfying these conditions. We also provide numerical assessment of code performance via simulation results.

I. INTRODUCTION

First introduced by Gallager [1], LDPC codes have been the focus of intense research in the past decade and many of their properties are now well understood. The iterative decoding algorithms for LDPC codes have been analyzed in detail, and asymptotic performance results have been derived [2]. However, estimation of frame-error-rate (FER) for iterative decoding of finite-length LDPC codes is still an unsolved problem. A special case of interest is the performance of iterative decoding at high signal-to-noise ratio (SNR). At high SNRs, a sudden degradation in the performance of iterative decoders has been observed [3], [4]. This abrupt change manifested in the FER curve is termed as an “error-floor.”

The error-floor problem is well understood for iterative decoding over the binary erasure channel (BEC) [5]. Combinatorial structures called “stopping sets” were used to characterize the FER for iterative decoding of LDPC codes over the BEC. It was established that decoding failure occurs whenever all the variables belonging to stopping sets are erased. Tian et al. [6] used this fact to construct irregular LDPC codes that avoid small stopping sets thus improving the guaranteed erasure recovery capability of codes under iterative decoding, and hence improving the error-floors. As in the case of BEC, a strong connection has been found between the existence of low-weight uncorrectable error patterns and error-floors for additive white Gaussian noise (AWGN) channels and binary symmetric channels (BSC) (see [4] and [7]). Hence, studying the guaranteed error correction capability of codes under iterative decoding is important in the context of characterization and improvement of the performance of iterative decoding strategies.

In the past, guaranteed error correction has been approached from the perspective of the decoding algorithm as well as from the perspective of code construction. Sipser and Spielman [8] used expansion arguments to derive sufficient conditions for the parallel bit-flipping algorithm to correct a fraction of errors in codes with column-weight greater than four. Burshtein [9] proved that for large enough lengths, almost all codes with column-weights greater than or equal to four can correct a certain fraction of errors under the bit-flipping algorithm. Burshtein and Miller [10] derived the sufficient conditions for message-passing decoding to correct a fraction of errors for codes of column-weight greater than five. However, these proofs were not constructive, i.e., no explicit code construction which satisfied the sufficient conditions was provided. Moreover, the code-lengths required to guarantee the correction of a small number of errors (say 3) is very high. Also, these arguments cannot be extended for message-passing decoding of codes with column-weight three or four.

E-mail: {ananthak, shashic, vasic, marcellin}@ece.arizona.edu
In order to construct codes with good error correcting properties under iterative decoding, progressive edge growth (PEG) [11] and constructions based on finite geometries [12] have been used. However, codes constructed from finite geometries typically have very high column-weight. Although, it has been proved that minimum distance grows at least linearly for codes constructed using PEG, no results proving guaranteed error correction under iterative decoding exist for these codes.

In this work, we derive the necessary and sufficient conditions for the correction of three errors in four iterations of message-passing decoding of column-weight-four codes. We provide a modified PEG construction which yields codes with such error-correction capability.

The remainder of the paper is organized as follows: We establish the preliminaries of the work in Section II. The necessary and sufficient conditions for the correction of three errors in column-weight-four codes are derived in Section III. In Section IV, we describe a technique to construct codes satisfying the conditions of the theorem and provide numerical results. We conclude with a few remarks in Section V.

II. PRELIMINARIES

In this section, we first describe the Tanner graph representation of LDPC codes. Then, we establish the notation that will be used throughout this paper. Finally, we describe the hard-decision message-passing algorithm that will be used for decoding.

A. Notation

The Tanner graph of an LDPC code, \( \mathcal{G}(V, C) \), is a bipartite graph with two sets of nodes: \( V \), the variable (bit) nodes and \( C \), the check (constraint) nodes. Every edge \( e \) in the bipartite graph is associated with a variable node \( v \) and a check node \( c \). The check nodes (variable nodes, respectively) connected to a variable node (check node, respectively) are referred to as its neighbors. The degree of a node is the number of its neighbors. In a \((\gamma, \rho)\)-regular LDPC code, each variable node has degree \( \gamma \) and each check node has degree \( \rho \). The girth \( g \) is the length of the shortest cycle in \( \mathcal{G} \). Let \( S \subset V \) such that \(|S| = y \). If for all choices of \( S \), there are at least \( z \) neighbors of \( S \) in \( C \), then we say that the \( y \rightarrow z \) condition is satisfied. In this paper, \( \bullet \) represents a variable node, \( \square \) represents an even-degree check node and \( \blacksquare \) represents an odd-degree check node.

B. Hard-Decision Decoding Algorithm

Let \( r = [r(1), r(2), \ldots, r(n)] \), a binary \( n \)-tuple, be the input to the message-passing decoder. Let \( v \in V \) be a variable node with \( r(v) \) as its corresponding bit and \( c \in C \) be a check node neighboring \( v \). Let \( \omega_j(v, c) \) denote the message that \( v \) sends to \( c \) in the first half of the \( j^{th} \) iteration and \( \varpi_j(c, v) \) denote the message that \( c \) sends to \( v \) in the second half of the \( j^{th} \) iteration.

Additionally, let \( \omega_j(v, :) \) be the set of all messages from a variable \( v \) to all its neighboring checks in the first half of the \( j^{th} \) iteration. Let \( \omega_j(v, : \setminus c) \) be the set of all messages that a variable \( v \) sends to all its neighboring checks except \( c \) in the first half of the \( j^{th} \) iteration. Let \( \varpi_j(:, v) \) be the set of all messages received by \( v \) from all its neighboring in the second half of the \( j^{th} \) iteration. Let \( \varpi_j(: \setminus c, v) \) be the set of all messages received by \( v \) from all its neighboring check nodes except \( c \) in the second half of the \( j^{th} \) iteration. \( \varpi_j(c, :) \), \( \varpi_j(c, : \setminus v) \), \( \omega_j(:, c) \) and \( \varpi_j(:, \setminus v, c) \) are defined similarly.

The Gallager algorithms [1] can be defined as follows: The forward messages, \( \omega_j(v, c) \) (from variables to checks), are defined as

\[
\omega_j(v, c) = \begin{cases} 
    r(v), & \text{if } j = 1 \\
    m, & \text{if } |\{c': c' \neq c, \varpi_{j-1}(c', v) = m\}| \geq b_{v, j} \\
    r(v), & \text{otherwise}
\end{cases}
\]  

(1)
where \(|\{c' : c' \neq c, \omega_{j-1}(c', v) = m\}|\) refers to the total number of messages which are of the value \(m \in \{0, 1\}\). The backward messages, \(\omega_j(c, v)\) (from checks to variables), are defined as

\[
\omega_j(c, v) = \left(\sum_{m_j \in \omega_j(c, \setminus v)} m_j\right) \mod 2.
\]  

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 decoder is run until a valid codeword is found or until a maximum number of iterations, say \(D\), is reached, whichever is earlier.

In Eqn. (1), \(b_{v,j}\) is a threshold which is generally a function of the iteration number, \(j\), and the degree of the variable \(v\). In this paper, we use \(b_{v,j} = 3\) for all \(v\) when \(1 \leq j \leq 3\) and \(b_{v,j} = 2\) for all \(v\) when \(j \geq 4\).

**Remark:** We note that Eqns. 1 and 2 then correspond to the Gallager-B algorithm [1]. For the Gallager-A algorithm [1], \(b_{v,j} = \gamma_v - 1\), for all \(j\), where \(\gamma_v\) is degree of variable node \(v\).

**A Note on the Decision Rule:** Different rules to estimate a variable node after each iteration are available, and it is likely that changing the rule after a certain number of iterations may be beneficial. However, the analysis of such scenarios is beyond the scope of this paper. Throughout the paper, we use the following decision rule: if all incoming messages to a variable node from neighboring checks are equal, set the variable node to that value; else set it to its received value.

### III. Main Result

The main result of this paper is summarized as follows:

**Theorem 1:** An LDPC code with column-weight four and girth six can correct three errors in four iterations of message-passing decoding if and only if the conditions, \(4 \to 11, 5 \to 12, 6 \to 14, 7 \to 16\) and \(8 \to 18\) are satisfied.

**Remark:** It is worth noting that if a graph of girth six satisfies the \(4 \to 11\) condition, then it satisfies the \(5 \to 12\) condition as well. However, the addition of this extra constraint aids in the proof of the theorem.

**Proof:** First, we prove the sufficiency of the conditions of Theorem 1.

Let \(V^1 := \{v_1, v_2, v_3\}\) be the three erroneous variables. Let \(C^1\) be the set of checks that are connected to the variables in \(V^1\). The variables in \(V^1\) can induce only one of the five subgraphs shown in Fig. 1. We prove that in each case, the decoding algorithm converges to the correct codeword in four iterations. For the sake of compactness, we give the proof only for subgraphs 1 and 2, and omit the proof for the other subgraphs.

**Subgraph 1:** The variables in \(V^1\) induce the subgraph shown in Fig. 1(a). At the end of the first iteration, \(\omega_1(\cdot, v) = \{0\}\) for all \(v \in V^1\). Moreover, no variable receives four incorrect messages after the first iteration as the existence of such a variable node would create a four-cycle. If a decision is made after the first iteration, the decoder is successful.

**Subgraph 2:** The variables in \(V^1\) induce the subgraph shown in Fig. 1(b). At the end of the first iteration:

\[
\omega_1(c, v) = \begin{cases} 
1 & \text{if } c \in C^1 \setminus \{c_1\}, \ v \notin V^1 \\
1 & \text{if } c = c_1, \ v \in \{v_1, v_2\} \\
0 & \text{otherwise}.
\end{cases}
\]  

(3)

For no \(v \in V \setminus V^1, \omega_1(\cdot, v) = \{1\}\) as it would introduce a four-cycle. For any \(v \notin V^1, \omega_2(v, c) = 1\) only if \(\omega_j(\cdot, v) = \{1\}\). This implies that \(v\) is connected to three checks in \(C^1 \setminus \{c_1\}\). Let \(V^2\) denote the set of such variables. We have the following lemma:
**Lemma 1:** There can be at most three variables in $V^2$. Furthermore, no two variable nodes in $V^2$ share any check in the set $C \setminus C^1$.

**Proof:** Let $V^2 = \{v^1_1, v^2_1, v^3_1, v^4_1\}$. Then the set of variable nodes $V^1 \cup V^2$ has at most 15 neighboring checks. This violates the $7 \rightarrow 16$ condition. Hence, $V^2$ can have at most three variables. Next, let $v^1_2, v^2_2 \in V^2$. Suppose they share a fourth check $c$. Since $v^1_3$ can share at most two checks with $v^1_1$ and $v^1_2$, assume that $c_{10}$ and $c_{11}$ are not neighbors of $v^1_2, v^2_2$. The neighbors of the variable nodes in the set $\{v^1_1, v^1_2, v^3_1, v^4_2\}$ all belong to the set $\{c^1_1, \ldots, c^1_{11} \} \cup \{c^2_2\}$ which has cardinality 10, thus violating the $4 \rightarrow 11$ condition.

Let the fourth neighboring checks of $v^1_2, v^2_2$ and $v^3_2$ be $c^1_2, c^2_2$ and $c^3_2$, respectively. Let $C^2 = \{c^1_2, c^2_2, c^3_2\}$. In the second iteration:

$$\omega_2(v, c) = \begin{cases} 1 & \text{if } v \in \{v^1_1, v^2_2\}, c \neq c^1_1 \\ 1 & \text{if } v \in V^2, c \in C^2 \\ 0 & \text{otherwise} \end{cases}$$

$$\overline{\omega}_2(c, v) = \begin{cases} 1 & \text{if } c \in \{c^1_1, c^1_2, c^1_3, c^1_4, c^1_5, c^1_6, c^1_7\}, v \in V^1 \\ 1 & \text{if } c \in C^2, v \notin V^2 \\ 0 & \text{otherwise}. \end{cases}$$

For all $v \in V^1$, $\overline{\omega}_2(:, v) = \{0\}$. For no $v \in V^2$, $\overline{\omega}_2(:, v) = \{1\}$. We now have the following lemma:

**Lemma 2:** There exists no variable $v \notin V^1 \cup V^2$ such that $\overline{\omega}_2(:, v) = \{1\}$.

**Proof:** The proof is by contradiction. Let $v \notin V^1 \cup V^2$ such that $\overline{\omega}_2(:, v) = \{1\}$. Then, $v$ is connected to four checks in $\{c^1_1, c^1_2, c^1_3, c^1_4, c^1_5, c^1_6, c^1_7\} \cup C^2$. Note that only two neighbors of $v$ can belong to $\{c^1_1, c^1_2, c^1_3, c^1_4, c^1_5, c^1_6, c^1_7\}$ without introducing a four-cycle. This combined with the fact that there are at most three variable nodes in $V^2$ implies that there are only two cases:

(a) $v$ has two neighbors in $\{c^1_1, c^1_2, c^1_3, c^1_4, c^1_5, c^1_6, c^1_7\}$ and two neighbors in $C^2$, say $c^1_2$ and $c^2_2$. In this case, the set of variable nodes $V^1 \cup \{v^1_1, v^2_2, v\}$ has 13 check nodes, violating the $6 \rightarrow 14$ condition.
(b) \( v \) has one neighbor in \( \{c_1^1, c_2^1, c_3^1, c_4^1, c_5^1\} \) and three neighbors in \( C^2 \). In this case, the set of variable nodes \( V^1 \cup V^2 \cup \{v\} \) has 14 check nodes, violating the \( 7 \to 16 \) condition.

Hence, if a decision is made after the second iteration, the decoder is successful.

The proof of correct decoding for subgraphs 3-5 is similar to that of subgraphs 1 and 2.

Next, we prove the necessity of the conditions of the theorem. We prove this by giving subgraphs which violate one condition and are not successfully decoded in four iterations. Since the validity of these claims can be checked easily, a detailed proof is omitted.

**Necessity of the \( 4 \to 11 \) condition**

Consider the subgraph shown in Fig. 2. In this case, the \( 4 \to 11 \) condition is not satisfied and the errors are not corrected at the end of the fourth iteration. Hence, in order to guarantee the correction of three errors in four iterations, the \( 4 \to 11 \) condition must be satisfied.

**Necessity of the \( 5 \to 12 \) condition**

There exists no graph of girth six which satisfies the \( 4 \to 11 \) condition but does not satisfy the \( 5 \to 12 \) condition.

**Necessity of the \( 6 \to 14 \) condition**

Consider the graph shown in Fig. 3. The graph shown satisfies the \( 4 \to 11 \) and the \( 5 \to 12 \) conditions but not the \( 6 \to 14 \) condition. The errors are not corrected in four iterations. Hence, in order to guarantee the correction of three errors in four iterations, the \( 6 \to 14 \) condition must be satisfied.
Necessity of the $7 \rightarrow 16$ condition

Consider the graph shown in Fig. 4. The graph shown satisfies the $4 \rightarrow 11$, $5 \rightarrow 12$ and the $6 \rightarrow 14$ conditions, but not the $7 \rightarrow 16$ condition. The errors are not corrected at the end of the fourth iteration. Hence, in order to guarantee the correction three errors in four iterations, the $7 \rightarrow 16$ condition must be satisfied.

![Fig. 4. A $7 \rightarrow 15$ subgraph.](image)

Necessity of the $8 \rightarrow 18$ condition

Consider the graph shown in Fig. 5. The graph shown satisfies the $4 \rightarrow 11$, $5 \rightarrow 12$, $6 \rightarrow 14$ and the $7 \rightarrow 16$ conditions, but not the $8 \rightarrow 18$ condition. The errors are not corrected at the end of the fourth iteration. Hence, in order to guarantee the correction of three errors in four iterations, the $8 \rightarrow 18$ condition must be satisfied.

![Fig. 5. A $8 \rightarrow 17$ subgraph.](image)

In this section, we proved necessary and sufficient conditions to guarantee the correction of three errors in column-weight-four codes using an iterative decoding algorithm. By analyzing the messages being passed in subsequent iterations, it may be possible to get smaller bounds on the number of check nodes required in the “small” subgraphs. However, we hypothesize that the size of subgraphs to be avoided would be larger.
IV. NUMERICAL RESULTS

In this section, we describe a technique to construct codes with column-weight four that can correct three errors. Codes capable of correcting a fixed number of errors show superior performance on the BSC at low values of transition probability $\alpha$. This is because the slope of the FER curve is related to the minimum critical number [14]. A code which can correct $i$ errors has minimum critical number at least $i + 1$ and the slope of the FER curve is $i + 1$. We restate the arguments from [14] to make this connection clear.

Let $\alpha$ be the transition probability of a BSC and $c_k$ be the number of configurations of received bits for which $k$ channel errors lead to codeword (frame) error. The frame error rate (FER) is given by:

$$\text{FER}(\alpha) = \sum_{k=i}^{n} c_k \alpha^k (1 - \alpha)^{n-k}$$

where $i$ is the minimal number of channel errors that can lead to a decoding error and $n$ is length of the code.

On a semi-log scale the FER is given by

$$\log(\text{FER}(\alpha)) = \log\left(\sum_{k=i}^{n} c_k \alpha^k (1 - \alpha)^{n-k}\right)$$

$$= \log(c_i) + i \log(\alpha) + \log((1 - \alpha)^{n-i})$$

$$+ \log\left(1 + \frac{c_{i+1}}{c_i} \alpha(1 - \alpha)^{-1} + \ldots + \frac{c_n}{c_i} \alpha^{n-i}(1 - \alpha)^{-i}\right).$$

For small $\alpha$, the expression above is dominated by the first two terms. That is,

$$\log(\text{FER}(\alpha)) \approx \log(c_i) + i \log(\alpha).$$

The $\log(\text{FER})$ vs. $\log(\alpha)$ graph is close to a straight line with slope equal to $i$, the minimal critical number. If two codes $C_1$ and $C_2$ have minimum critical numbers $i_1$ and $i_2$, such that $i_1 > i_2$, then the code $C_1$ will perform better than $C_2$ for small enough $\alpha$, independent of the number of trapping sets.

**Code Construction**

The construction of column-weight-four codes with a guaranteed error correction capability involves ensuring expansion on subsets of variable nodes as discussed above. This can be done only in time which grows exponentially with the length of the code. Hence, we consider the $4 \rightarrow 12$ condition rather than the necessary and sufficient conditions discussed in Section III. It can be shown that the $4 \rightarrow 12$ condition is sufficient for the $4 \rightarrow 11$, $5 \rightarrow 12$, $6 \rightarrow 14$, $7 \rightarrow 16$ and the $8 \rightarrow 18$ conditions. There are only two graphs of girth 6 with 4 variable nodes and 11 check nodes. Fig. 6 shows these two graphs. Avoiding these two subgraphs will ensure a code which can correct three errors. In order to construct codes that avoid the subgraphs shown in Fig. 6, we use progressive edge-growth (PEG) which is a modification of the PEG construction used by Hu et al. [11]. The algorithm is outlined below.
Algorithm 1: ConstructCode

Data: The set of \( n \) variable nodes \((V)\) and \( m \) check nodes \((C)\). The column weight of the code \((\gamma)\)

Result: Code with column weight \(\gamma\)

for \( j = 1 \) to \( n \)
  for \( k = 1 \) to \( \gamma \)
    if \( k = 1 \) then
      Connect the \(k^{th}\) edge of variable node \( j \) to the check node with the smallest positive degree.
    else
      Expand the tree rooted at node \( j \) to a depth of 6.
      Assimilate all check nodes which do not appear in the tree into \( C_{j,T} \), the set of candidates for connecting variable node \( j \) to.
      while \( k^{th} \) edge is not found do
        Find the check node \( c_i \) in \( C_{j,T} \) with the lowest degree. If connecting \( c_i \) to variable node \( j \) does not create a \((4, 12)\) subgraph, set this as the \( k^{th} \) edge. If it does, remove \( c_i \) from \( C_{j,T} \).
      end
    end
  end
end

This algorithm was used to generate a code of length 816, girth 6 and rate 0.5. The code constructed has a slight irregularity in that three check nodes have degree nine and three have degree seven.

Remark: For the code parameters given above, it was possible to generate a code which satisfied the \(4 \rightarrow 12\) condition. However, it might not be possible to satisfy this condition for codes with higher rate and/or shorter lengths. Should such a scenario arise, the set of subgraphs to be avoided should be changed (e.g., to those specified in the necessary and sufficient conditions). However, the code construction time will be larger. Hence, at the cost of code-construction time and complexity, it is possible to achieve shorter lengths and/or higher rates.

![Graphs with girth 6](image)

(a) ![Graphs with girth 6](image)

(b)

Fig. 6. Graphs with girth 6 which have 4 variable nodes and 11 check nodes. Subgraphs with 4 variable nodes and fewer than 11 check nodes do not exist.

Fig. 7 shows the performance of the code under message-passing decoding. The curve on the left corresponds to four iterations of message-passing. The curve in the right corresponds to 25 iterations of message-passing. After only four iterations, all errors of weight three were corrected. Errors of weight four and above were encountered which were not corrected by the message-passing decoder. However, after 25 iterations, the smallest weight error pattern still remaining had a weight of 7. We note that the average slope of the FER curve is 8 which is the weight of the dominant error event at these probabilities
of error. This suggests that analysis over a higher number of iterations and on “larger” subgraph search will yield a stronger result. However, this is beyond the scope of this paper. Also, it is worth noting that the conditions of Theorem 1 avoid codewords of length 4 through 8 which improves the minimum distance of the code.

![Performance of the example column-weight-four code for different numbers of iterations of message-passing.](image)

Fig. 7. Performance of the example column-weight-four code for different numbers of iterations of message-passing.

V. CONCLUSION

In this paper, we provided a method to derive conditions that guarantee the correction of a finite number of errors by hard-decision decoding. Although more involved than the expander arguments used in previous works, it results in better bounds. Moreover, in contrast to previous expansion arguments, our results give rise to code-construction techniques that yield codes with guaranteed error-correction ability under message-massing decoding at practically feasible lengths. This method can be applied (a) to provide conditions for guaranteed correction of a larger number of errors, and (b) to yield similar results for higher column-weights and/or higher girths. However, such applications would be more involved than the analysis here.

REFERENCES