LDPC Codes Which Can Correct Three Errors Under Iterative Decoding

Shashi Kiran Chilappagari
Dept. of Electrical and Computer Eng.
University of Arizona
Tucson, AZ 85721, USA
Email: shashic@ece.arizona.edu

Anantha Raman Krishnan
Dept. of Electrical and Computer Eng.
University of Arizona
Tucson, AZ 85721, USA
Email: ananthak@ece.arizona.edu

Bane Vasić
Dept. of ECE and Dept. of Mathematics
University of Arizona
Tucson, AZ 85721, USA
Email: vasic@ece.arizona.edu

***Abstract*— In this paper, we provide necessary and sufficient conditions for a column-weight-three LDPC code to correct three errors when decoded using Gallager A algorithm. We then provide a construction technique which results in a code satisfying the above conditions. We also provide numerical assessment of code performance via simulation results.**

I. Introduction

Iterative message passing algorithms for decoding low-density parity-check (LDPC) codes have been the focus of research over the past decade and most of their properties are well understood [1],[2]. These algorithms operate by passing messages along the edges of a graphical representation of the code known as the Tanner graph and are optimal when the underlying graph is a tree. Message passing decoders perform remarkably well which can be attributed to their ability to correct errors beyond the traditional bounded distance decoding capability. However, in contrast to bounded distance decoders (BDDs), iterative decoders cannot guarantee correction of a fixed number of errors at relatively short code lengths. This is due to the fact that the associated Tanner graphs for short length codes have cycles and the decoding becomes suboptimal and there exist a few low-weight patterns (termed as near codewords [3] or trapping sets [4]) uncorrectable by the decoder. It is now well established that the trapping sets lead to the phenomenon of error floor. Roughly, error floor is an abrupt change in the frame error rate (FER) performance of an iterative decoder in the high signal-to-noise ratio (SNR) region.

The error floor problem is well understood for iterative decoding over binary erasure channel (BEC) [5]. The decoder fails when the received vector contains erasures in locations corresponding to a stopping set. For the AWGN channel, Richardson in [4] presented a numerical method to estimate error floors of LDPC codes. He established a relation between trapping sets and the FER performance of the code in the error floor region (the necessary definitions will be given in the next section). The approach from [4] was further refined by Stepanov *et al* in [6]. Vontobel and Koetter [7] established a theoretical framework for finite length analysis of message passing iterative decoding based on graph covers. This approach was used by Smarandache *et al* in [8] to analyze performance of LDPC codes from projective and for LDPC convolutional codes [9]. For the binary symmetric channel (BSC), error floor estimation based on trapping sets was proposed in [10] and we adopt the notation from [10].

In this paper, we make the following two fundamental contributions: (a) give necessary and sufficient conditions for a column-weight-three LDPC code to correct three errors, and (b) propose a construction method which results in a code satisfying the above conditions.

We consider hard decision decoding for transmission over BSC. The BSC is a simple yet useful channel model used extensively in areas where decoding speed is a major factor. Note that the problem of recovering from a fixed number of erasures is solved for the BEC. If the Tanner graph of a code does not contain any stopping sets up to size t (the size of minimum stopping set is $t+1$), then the decoder is guaranteed to recover from any t erasures. An analogous result for the BSC is still unknown. The problem of guaranteed error correction capability is known to be difficult and in this paper, we present a first step toward such result. Previously, expansion arguments were used to show that message passing can correct a fixed fraction of errors [11]. However, the code length needed to guarantee such correction capability is generally very large and to correct three errors, the length would be in the order of a few hundred thousand. Also, these arguments cannot be used for column-weight-three codes. Column-weight-three codes are of special importance as their decoders have very low complexity and are used in a wide range of applications.

We also show that the slope of the frame error rate (FER) is dependent on the critical number of the most relevant trapping sets and hence the slope can be improved by avoiding such trapping sets. We provide a technique to construct codes which outperform empirically best known codes of the same length. Our method can be seen as a modification of the progressive edge growth (PEG) technique proposed in [12].

The rest of the paper is organized as follows. In Section II we establish the notation, describe the Gallager A algorithm and define trapping sets. In Section III we present the main theorem which gives the necessary and sufficient conditions to correct three errors. 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. Decoding Algorithms and Trapping Sets

In this section, we establish the notation and describe a hard decision decoding algorithm known as Gallager A algorithm. We then characterize the failures of the Gallager A decoder with the help of fixed points. We also introduce the notions of trapping sets and critical number.

A. Graphical Representations of LDPC Codes

The Tanner graph of an LDPC code, $\mathcal{G}$, is a bipartite graph with two sets of nodes: variable (bit) nodes and check (constraint) nodes. Every edge e in the bipartite graph is associated with a variable node v and check node 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. In a (γ, ρ) regular LDPC code, each variable node has degree of γ and each check node has degree ρ. The girth g is the length of the shortest cycle in $\mathcal{G}$. 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 Algorithms

Gallager in [13] proposed two simple binary message passing algorithms for decoding over the BSC; Gallager A and Gallager B. See [14] for a detailed description of Gallager B algorithm. For column-weight-three codes, which are the main focus of this paper, these two algorithms are the same. Every round of message passing (iteration) starts with sending messages from variable nodes (first half of the iteration) and ends by sending messages from check nodes to variable nodes (second half of the iteration). Initially, the variable nodes send their received values to the neighboring checks. In the k^{th} iteration $(k = 2, 3, \ldots)$, a variable node, v sends the following message, $\overrightarrow{\mathrm{m_i}}\ (e)$, along edge e to its neighboring check node c ; if all incoming messages to v other than the message from c are equal to a certain value, it sends that value; else, it sends the received value. A check node c sends to a variable node v, the modulo two sum of all incoming messages except the message from v. 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 for a maximum number of iterations is reached, whichever is earlier.

A Note on the Decision Rule: Different rules to estimate a variable node after each iteration are possible and it is likely that changing the rule after certain 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 neighboring checks 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

We adopt Decision rule A throughout this paper.

C. Trapping Sets of Gallager A Algorithm

We now characterize failures of the Gallager A decoder using fixed points and trapping sets. Consider an LDPC code of length n and let $\underline{x}$ be the binary vector which is the input to the Gallager A decoder. Let $\mathcal{S}(\underline{x})$ be the support of $\underline{x}$. The support of $\underline{x}$ is defined as the set of all positions i where $x_i \neq 0$.

Definition 1: A decoder failure is said to have occurred if the output of the decoder is not equal to the transmitted codeword.

Definition 2: $\underline{x}$ is called a *fixed point* if for every edge e and its associated variable node v

$$\overrightarrow{\mathrm{m_k}}\ (e) \quad = \quad \underline{x}(v), \forall k$$

That is, the message passed from variable nodes to check nodes along the edges are the same in every iteration. Since the outgoing messages from variable nodes are same in every iteration, it follows that the incoming messages from check nodes to variable nodes are also same in every iteration and so is the estimate of a variable after each iteration. In fact, the estimate after each iteration coincides with the received value. It is clear from above definition that if the input to the decoder is a fixed point, then the output of the decoder is the same fixed point.

Note: From above discussion it is clear that if the received vector is a fixed point other than the sent codeword, then it leads to a decoder failure. It should be noted that all codewords are fixed points. Also, if $\underline{x}$ is a fixed point and $\underline{c}$ is a codeword, then $\underline{c} + \underline{x}$ is a also a fixed point, where addition is in $GF(2)$. Without loss of generality, we assume that the all zero codeword is sent over BSC and the input to the decoder is the error vector. So, a fixed point with small weight means that few errors lead to decoder failure. A received vector other than a fixed point can also lead to a decoder failure. These configurations are closely related to fixed points and might lead to oscillations, but the analysis of such cases is beyond scope of this paper. A detailed discussion about different kinds of decoder failures is given in [15]. Here, we restrict our attention to just the definitions needed to establish our main theorem.

Definition 3: [10] The support of a fixed point is known as a trapping set. A (V, C) trapping set $\mathcal{T}$ is a set of V variable nodes whose induced subgraph has C odd degree checks.

Our definition of a trapping set gives necessary and sufficient conditions for a set of variable nodes to form a trapping set. We state the following theorem which is a consequence of Fact 3 from [4] without proof.

Theorem 1: Let $\mathcal{T}$ be a set consisting of v variable nodes with induced subgraph $\mathcal{I}$. Let the checks in $\mathcal{I}$ be partitioned into two disjoint subsets; $\mathcal{O}$ consisting of checks with odd degree and $\mathcal{E}$ consisting of checks with even degree. Let $|\mathcal{O}| = c$ and $|\mathcal{E}| = s$. $\mathcal{T}$ is a trapping set if : (a) Every variable node in $\mathcal{I}$ is connected to at least two checks in $\mathcal{E}$ and at most one checks in $\mathcal{O}$ and (b) No two checks of $\mathcal{O}$ are connected to a variable node outside $\mathcal{I}$.

If the variable nodes corresponding to a trapping set are in error, then a decoder failure occurs. However, not all variable nodes corresponding to trapping set need to be in error for a decoder failure to occur.

Definition 4: [10] The minimal number of variable nodes that have to be initially in error for the decoder to end up in the trapping set $\mathcal{T}$ will be referred to as *critical number* m for that trapping set.

Remark: To "end up" in a trapping set $\mathcal{T}$ means that, after a possible finite number of iterations, the decoder will be in error, on at least one variable node from $\mathcal{T}$, at every iteration. It is clear that the performance of a code is dependent on the trapping sets with the least critical number.

III. Necessary and Sufficient Conditions to Correct Three Errors

In this section, we establish the necessary and sufficient conditions for a column-weight-three code to correct three errors. We first illustrate three trapping sets and show that the critical number of these trapping sets is three thereby providing necessary condition to correct three errors. We then prove that avoiding structures isomorphic to these trapping sets in the Tanner graph is sufficient to guarantee correction of three errors.

Fig. 1 shows three subgraphs induced by different number of variable nodes. Let us assume that in all these induced graphs, no two checks are connected to a variable node outside the graph. By the conditions of Theorem 1, all these induced subgraphs are trapping sets. Fig. 1(a) is a $(3,3)$ trapping set, Fig. 1(b) is a $(5,3)$ trapping set and Fig. 1(c) is a $(8,0)$ trapping set. Note that a $(3,3)$ is isomorphic to a six cycle. and the $(8,0)$ trapping set is a codeword of weight eight.

Before proceeding to the main theorem, it is instructive to see what are the types of messages passed by variable nodes and check nodes in an iteration.

A variable node which is in error initially sends incorrect messages in the first iteration and in subsequent iterations it sends

Case 1.a all incorrect messages if it receives two or more incorrect messages from neighboring checks in the previous iteration

Case 1.b all correct messages if it receives all correct messages from neighboring checks in the previous iteration and

Case 1.c one correct message and two incorrect messages if it receives one incorrect message from neighboring checks in the previous iteration. The correct message is sent along the edge on which the incorrect message is received.

A variable node which was not in error initially send correct messages in the first iteration and in subsequent iterations it sends

Case 2.a all incorrect messages if it receives all incorrect messages from neighboring checks in the previous iteration

Case 2.b all correct messages if it receives two or more correct messages from neighboring checks in the previous iteration and

Case 2.c one incorrect message and two correct messages if it receives two incorrect messages from neighboring checks in the previous iteration. The incorrect message is sent along the edge on which the incorrect message is received.

A check node sends

Case 3.a incorrect messages along edges which send incorrect messages and correct messages along edges which send correct messages, if the total number of incoming incorrect messages from neighboring variable nodes is even and

Case 3.b incorrect messages along edges which send correct messages and correct messages along edges which send incorrect messages, if the total number of incoming incorrect messages from neighboring variable nodes is odd.

When an estimate of a variable node is made after every iteration

Case 4.a a variable node initially in error is estimated incorrectly, if it receives at least one incorrect message

Case 4.b a variable node initially not in error is estimated incorrectly if it receives all incorrect messages

With this understanding of message passing rules, we can proceed to prove the following lemma.

Lemma 1: The critical number for $(3,3)$, $(5,3)$ and $(8,0)$ trapping sets is three.

Proof: The proof for the $(3,3)$ trapping set is trivial. We prove for the $(5,3)$ trapping set and omit the proof for $(8,0)$ trapping set due to space considerations. Consider the $(5,3)$ trapping set from Fig. 1(b) and assume that the variable nodes labeled as $1,2$ and 3 are initially in error. Since we assumed that no variable node outside the trapping set is connected to two checks in the trapping set, any variable outside the trapping set receives at most one incorrect message. So, the errors in trapping set do not affect the messages passed by outside variables. Hence in Fig. 2 we show only the messages passed within the trapping set and for the sake of compactness, we highlight incorrect messages by using arrows. In the first half of first iteration of message passing, variable nodes $1,2$ and 3 send incorrect messages to their neighboring checks. Fig. 2(a) illustrates message passing for the first half of first iteration. In the second half of first iteration, check nodes a,b,c,d,e,f,g,h and i send incorrect messages to their neighboring variable nodes except to variable nodes $1,2$ and 3 (case 3.b). This is shown in Fig. 2(b). Variable nodes $1,2$ and 3 receive correct messages, but variable nodes 4 and 5 receive all incorrect messages. So in the first half of second iteration, variable nodes $1,2$ and 3 send correct messages (case 1.b) and variable nodes 4 and 5 send incorrect messages (case 2.a) which is shown in Fig. 2(c). In the second half of second iteration, check nodes a,c,d,f,g and i send incorrect messages to their neighbors except to variable nodes 4 and 5 (case 3.b). So, variable nodes $1,2$ and 3 receive two incorrect messages each whereas variable nodes 4 and 5 receive all correct messages. This is illustrated in Fig. 2(d). In the first half of third iteration, again variable nodes $1,2$ and 3 send all incorrect messages (case 1.a) and the situation is same as in iteration one. So, if variable nodes $1,2$ and 3 are initially in error, then it leads to oscillation in the decoder with period two. At the end of every odd iteration variable nodes 4 and 5 are in error and at the end of every even iteration variable nodes $1,2$ and 3 are in error. Hence, three variable nodes in error initially can lead to a decoder failure and therefore, the $(5,3)$ trapping set has critical number three. ■

Theorem 2: To correct three errors in a column-weight-three LDPC code by Gallager A algorithm, it is necessary to avoid $(3,3)$, $(5,3)$ and $(8,0)$ trapping sets in its Tanner graph.

Proof: Follows from the above discussion. ■

We now state and prove the main theorem.

Theorem 3: If the Tanner graph of a column-weight-three LDPC codes has girth eight and does not contain a graph isomorphic to $(5,3)$ trapping set and a graph isomorphic to $(8,0)$ trapping sets as subgraphs, then any three errors can be corrected using Gallager A algorithm.

Sketch of proof: In a column-weight-three code three variable nodes can induce only one of the five subgraphs given in Fig. 3 and the proof proceeds by examining these subgraphs one at a time. The complete proof involves many arguments and here we just illustrate the methodology of the proof by considering two possible subgraphs. The proof for the remaining subgraphs is given in the Appendix.

Subgraph 1: Since the girth of the code is eight, it has no six cycles and hence the configuration in Fig. 3(a) is not possible.

Subgraph 2: The three variable nodes in error induce a subgraph as shown in Fig. 3(b). In this case, in first half of first iteration variable nodes $1, 2$ and 3 send incorrect messages. In the second half of first iteration, check nodes a, b, c, d, e, f, g, h and i send incorrect messages to their neighboring variable nodes except to nodes $1, 2$ and 3. There cannot be a variable node which is connected to two or more checks in the set $\{a, b, c, d, e, f, g, h, i\}$ without introducing either a six cycle or a subgraph isomorphic to $(5,3)$ trapping set. At the end of first iteration, variable nodes $1, 2$ and 3 receive all correct messages and no variable receives more than one incorrect message. If a decision is made after first iteration, a valid codeword is found and the decoder is successful.

Remark: It is worth noting that the complete proof is more involved than the proofs which use expansion arguments. However, the result is also more precise and holds for codes of small lengths.

IV. Numerical Results

In this section, we describe a technique to construct codes which can correct three errors. Codes capable of correcting a fixed number of errors show superior performance on the BSC at low values of probability of transition α. This is because the slope of the FER curve is related to the minimum critical number [16]. A code which can correct i errors has minimum critical number $i+1$ and the slope of FER curve is $i+1$. We restate the arguments from [16] to make this connection clear.

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

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

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

On a semilog scale the FER is given by the expression

$$\begin{aligned} \log(FER(\alpha)) &= \log\big(\sum_{k=i}^{n} c_k \alpha^k (1-\alpha)^{n-k}\big) \\ &= \log(c_i) + i\log(\alpha) + \log((1-\alpha)^{n-i}) \\ &+ \log\left(1 + \frac{c_{i+1}}{c_i}\alpha(1-\alpha)^{-1} + \ldots + \frac{c_n}{c_i}\alpha^{n-i}(1-\alpha)^{-i}\right) \end{aligned}$$

In the limit $\alpha \to 0$ we note that

$$\lim_{\alpha\to 0}\Big[\log((1-\alpha)^{n-i})\Big] = 0$$

and

$$\lim_{\alpha\to 0}\Big[\log\Big(1+\frac{c_{i+1}}{c_i}\alpha(1-\alpha)^{-1}\ldots+\frac{c_n}{c_i}\alpha^{n-i}(1-\alpha)^{i-n}\Big)\Big] = 0$$

So, the behavior of the FER curve for small α is dominated by

$$\log(FER(\alpha)) \approx \log(c_i) + i\log(\alpha)$$

The $\log(FER)$ 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_2 will perform better than C_1 for small enough α, independent of the number of trapping sets.

From the discussion in Section III and Section IV, it is clear that for a code to have a FER curve with slope at least 4, the corresponding Tanner graph should not contain the trapping sets shown in Fig. 1 as subgraphs. We now describe a method to construct such codes. The method can be seen as a modification of the PEG construction technique used by Hu *et al.* [12]. The algorithm is as follows:

```
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 $n$ do
    for $k = 1$ to $\gamma$ do
        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 $C_{j,\overline{T}}$, 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 $C_{j,\overline{T}}$ with the
                lowest degree. If connecting $c_i$ to
                variable node $j$ does not create a $(5,3)$
                trapping set, set this as the $k^{th}$ edge. If it
                does, remove $c_i$ from $C_{j,\overline{T}}$.
            end
        end
    end
end
```

Note that checking for a graph isomorphic to $(8, 0)$ trapping set is computationally complex. Since, the PEG construction empirically gives good codes, it is unlikely that it introduces a weight-eight codeword. However, once the graph is grown fully, it can be checked for the presence of weight-eight codewords and these can be removed by swapping few edges.

Using the ablove algorithm, a column-weight-three code with 504 variable nodes and 252 check nodes was constructed. The code has slight irregularity in check degree. There is one check node degree five and one check node with degree seven, but the majority of them have degree six. The code has rate 0.5. In the algorithm, we restrict maximum check degree to seven. The performance of the code on BSC is compared with the PEG code of same length. The PEG code is empirically the best known code at that length on AWGN channel [17]. However, it has fourteen $(5, 3)$ trapping sets. Fig. shows the performance comparision of the two codes. As can be seen, the new code performs better than the original PEG code at small values of α.

V. Conclusion

In this paper, we have given conditions for a column-weight-three code to correct three errors. Since, the check degree does not play any part in the proof, it follows that the result is independent of code rate. A direction for future work is extending the analysis to more number of errors and higher column weight codes. Preliminary investigation shows a lot of promise. The complexity of the proof, even in the case of three errors, suggests that solving the problem for an arbitrary number of errors will be a challenge. On the code construction front, we have shown that avoiding trapping sets with minimum critical number is the criterion to suppress error floor. However, the conditions for correcting more errors could be more complicated thereby increasing the complexity of code construction. Deriving bounds on lengths and minimum distance of codes which avoid certain structures also need to be investigated.

Appendix
The Complete Proof

In this Appendix, we consider the remaining possible subgraphs and show that under the conditions of the theorem, the Gallager A algorithm can correct three errors.

Subgraph 3: The three variable nodes in error induce a subgraph as shown in Fig. 3(c). In first half of first iteration $1, 2$ and 3 send incorrect messages. In the second half of first iteration, a, b, d, f and g send incorrect messages to neighboring variables except to $1, 2$ and 3. c sends incorrect messages to 1 and 2. e sends incorrect messages to 2 and 3. There cannot be a variable node which is connected to one check from a and b and to check d. This would introduce a six cycle. Similarly there cannot be a variable node which is connected to one check from f and g and to check d. A variable node can be connected to one check from a and b and to one check from f and g. Suppose a variable node 4 is connected to checks a and f. Then variable nodes $1, 2, 3$ and 4 form an eight cycle and hence there cannot be any other variable node which is connected to one check from a and b and to one check from f and g without introducing a six cycle or a graph isomorphic to $(5, 3)$ trapping set. In the first half of second iteration, 1 sends incorrect messages to a and b, 2 sends incorrect messages to c, d and e, 3 sends incorrect messages to f and g, 4 (if it exists) sends incorrect message to i(say). In the second half of second iteration, ca and b send incorrect messages to neighboring variable nodes except to 1, f and g send incorrect messages to neighboring variable nodes except to 3, check c sends incorrect message to neighboring nodes except 2, e sends incorrect messages to neighboring nodes except 2. So, 1 and 3 receive an incorrect message each from c and e respectively. 4 receives two incorrect messages. There cannot be any other variable node which receives two incorrect messages. In the first half of third iteration, 1 and 3 send incorrect messages to a, b and f, g respectively and correct messages to c and e respectively. 2 sends all correct messages. 4 sends incorrect message to i. All the remaining nodes send correct messages. In the second half of third iteration, a, b and f, g send incorrect messages to neighbors except to 1 and 3 respectively. i sends incorrect messages to its neighbors except to 4. At the end of third iteration, $1, 2$ and 3 receive all correct messages, 4 receives two incorrect messages and all other variable nodes receive at most one incorrect messages. So, if a decision is made after third iteration, a valid codeword is found and the decoder is successful.

Subgraph 4: The three variable nodes in error induce a subgraph as shown in Fig. 3(d). In first half of first iteration $1, 2$ and 3 send incorrect messages. In the second half of first iteration, a, b, d, e, f, g and h send incorrect messages to neighboring variables except to $1, 2$ and 3. c sends incorrect messages to 1 and 2. No variable node can be connected to two checks from the set $\{a, b, c, d, e\}$ as it would introduce a four cycle or six cycle. Hence, there cannot be a variable node which receives three incorrect message. However, a variable node can be connected to one check from $\{a, b, d, e\}$ and to one check from $\{f, g, h\}$. There can be at most four such variable nodes. Also, these four variable nodes cannot share checks outside the set $\{a, b, d, e, f, g, h\}$. Let these four variable nodes be labeled $4, 5, 6$ and 7 and their other checks $\{i, j, k, l\}$. Since, c does not send incorrect messages to variables other than 1 and 2, we need not consider its connections. In the first half of second iteration, 1 and 2 send incorrect messages to a, b and d, e respectively, $4, 5, 6$ and 7 send incorrect messages to i, j, k and l respectively. In the second half of second iteration, a, b send incorrect messages to neighbors other than 1, d, e send incorrect messages to neighbors other than 2, i, j, k and l send incorrect messages to neighbors other than $4, 5, 6$ and 7. It can be shown that there cannot be a variable node which receives three incorrect messages at the end of second iteration. $1, 2$ and 3 receive all correct messages and no other variable receives more than two incorrect messages. So, if a decision is made, a valid codeword is reached and the decoder is successful.

Subgraph 5: The three variable nodes in error induce a subgraph as shown in Fig. 3(e). In first half of first iteration $1, 2$ and 3 send incorrect messages. In the second half of first iteration, a, b, c, d, e, f, g, h and i send incorrect messages to

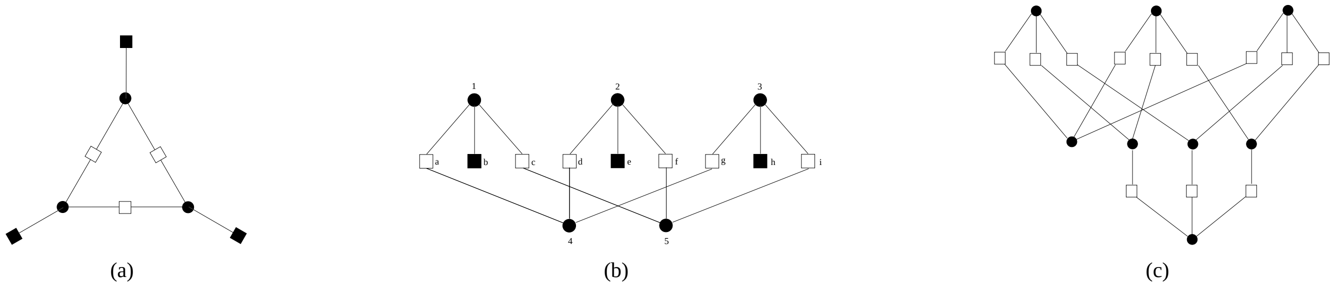

Fig. 1. Examples of trapping sets with critical number three (a) a $(3,3)$ trapping set (b) a $(5,3)$ trapping set and (c) an $(8,0)$ trapping set

neighboring variables except to $1,2$ and 3. If there is no variable node which receives three incorrect messages, a valid codeword is reached after first iteration. On the contrary, assume there exists a variable node, say 4, which receives three incorrect messages (w.l.o.g. we can assume that 4 is connected to a,d and g). Also, there cannot be two such variable nodes as that would introduce a six cycle or a graph isomorphic to $(5,3)$ trapping set. Also, there can be at most three variable nodes which receive two incorrect messages, say, $5,6$ and 7. Let the other checks connected to these variables be j,k and l respectively. In the first half of second iteration, $1,2$ and 3 send all correct messages, 4 sends all incorrect messages, $5,6,7$ send incorrect messages to j,k and l respectively. In second half of second iteration, a,d,g send incorrect messages to their neighbors except to 4. j,k and l send incorrect messages to neighboring variables except to $5,6$ and 7. There cannot be a variable node which is connected to one check from $\{j,k,l\}$ and to one check from $\{a,d,g\}$. Also, there cannot be a variable node which is connected to all the three checks j,k and l as this would introduce a graph isomorphic to $(8,0)$ trapping set. However, there can be at most two variable nodes which receive two incorrect messages from the checks j,k and l, say 8 and 9. Let the other checks connected to 8 and 9 be m and p. At the end of second iteration, $1,2$ and 3 receive one incorrect message, 8 and 9 receive two incorrect messages. In the first half of third iteration, $1,2$ and 3 send two incorrect messages each, 8 and 9 send one incorrect message each. In the second half of third iteration, b,c,e,f,h and i send incorrect messages to their neighbors except to $1,2$ and 3. m and p send incorrect messages to their neighbors except to 8 and 9. It can be shown that there cannot exist a variable node which receives three incorrect messages. At the end of third iteration, $1,2$ and 3 receive all correct messages and no variable node receives all incorrect messages. So, if a decision is made, a valid codeword is reached and decoder is successful.

ACKNOWLEDGMENT

This work is funded by NSF under Grant CCF-0634969 and INSIC-EHDR program.

REFERENCES

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

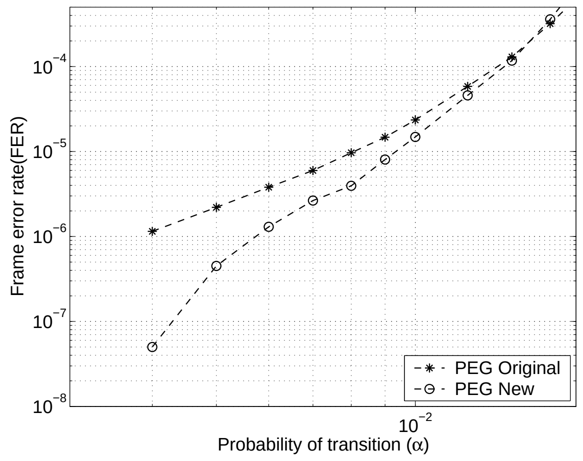

Fig. 4. Performance comparision of original PEG and the new PEG code

[2] T. J. Richardson, M. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 638–656, Feb. 2001.
[3] D. J. C. MacKay and M. J. Postol, "Weaknesses of Margulis and Ramanujan–Margulis low-density parity-check codes," in *Proceedings of MFCSIT2002, Galway*, ser. Electronic Notes in Theoretical Computer Science, vol. 74. Elsevier, 2003. [Online]. Available: http://www.inference.phy.cam.ac.uk/mackay/abstracts/margulis.html
[4] T. J. Richardson, "Error floors of LDPC codes," in *41st Annual Allerton Conf. on Communications, Control and Computing*, 2003, pp. 1426–1435.
[5] C. Di, D. Proietti, T. Richardson, E. Telatar, and R. Urbanke, "Finite length analysis of low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 48, pp. 1570–1579, June 2002.
[6] M. G. Stepanov, V. Chernyak, M. Chertkov, and B. Vasic, "Diagnosis of weaknesses in modern error correction codes: A physics approach," *Phys. Rev. Lett.*, vol. 95, p. 228701, Nov. 2005.
[7] P. O. Vontobel and R. Koetter, "Graph-cover decoding and finite-length analysis of message-passing iterative decoding of LDPC codes," 2005. [Online]. Available: http://www.citebase.org/abstract?id=oai:arXiv.org:cs/0512078
[8] R. Smarandache and P. O. Vontobel, "Pseudo-codeword analysis of Tanner graphs from projective and euclidean planes," *IEEE Trans. Inform. Theory*, vol. 53, no. 7, pp. 2376–2393, July 2007.
[9] R. Smarandache, A. E. Pusane, P. O. Vontobel, and J. C. D.J., "Pseudo-codewords in LDPC convolutional codes," July 2006, pp. 1364–1368.
[10] S. K. Chilappagari, S. Sankaranarayanan, and B. Vasic, "Error floors of LDPC codes on the binary symmetric channel," in *International Conference on Communications*, vol. 3, June 11-15 2006, pp. 1089–1094.
[11] D. Burshtein and G. Miller, "Expander graph arguments for message-passing algorithms," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 782–790, Feb. 2001.
[12] X.-Y. Hu, E. Eleftheriou, and D. M. Arnold, "Regular and irregular progressive edge-growth Tanner graphs," *IEEE Trans. Inform. Theory*, vol. 51, no. 1, pp. 386–398, Jan. 2005.
[13] R. G. Gallager, *Low Density Parity Check Codes*. Cambridge, MA: M.I.T. Press, 1963.

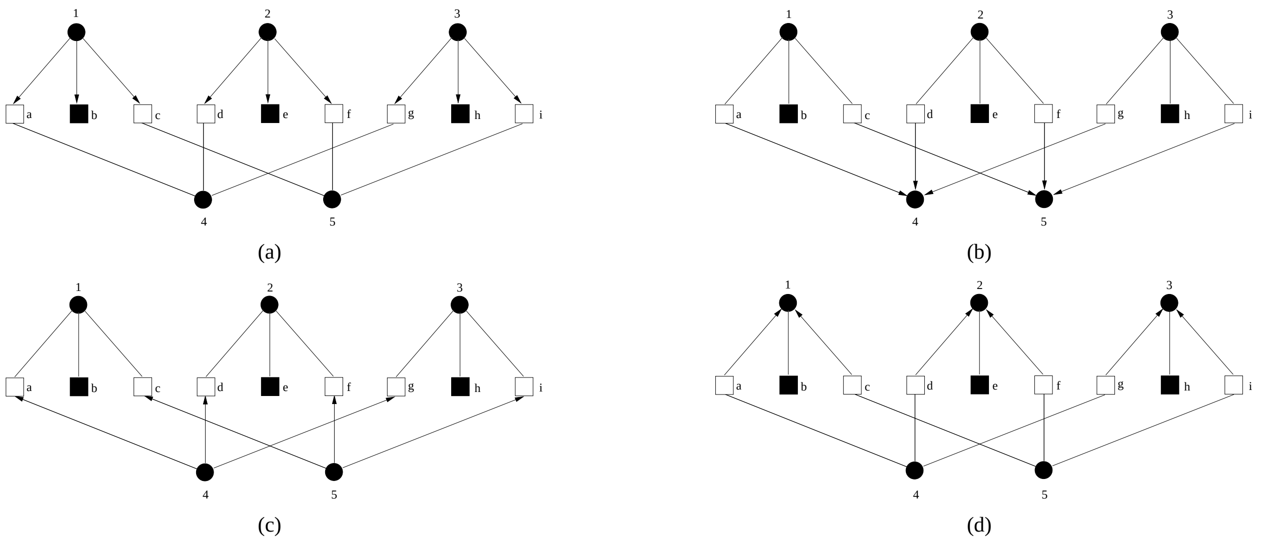

Fig. 2. Illustration of message passing for a $(5,3)$ trapping set (a) variable to check messages in round one (b) check to variable messages in round one (c) variable to check messages in round two (d) check to variable messages in round two

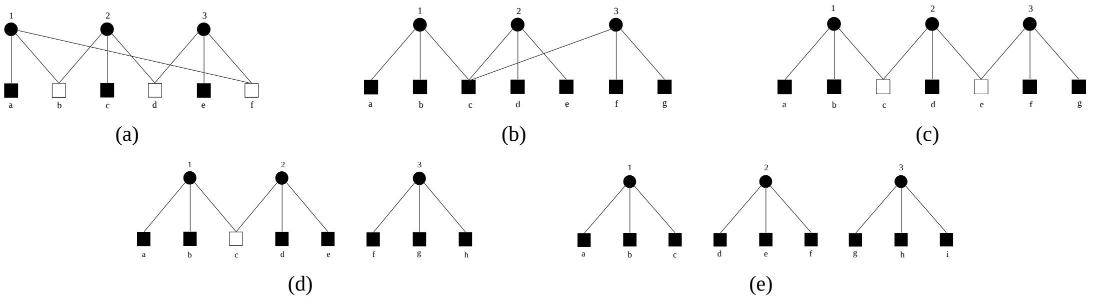

Fig. 3. All the possible subgraphs that can be induced by three variable nodes in a cloumn-weight-three code

[14] A. Shokrollahi, “An introduction to low-density parity-check codes,” in *Theoretical aspects of computer science: advanced lectures*. New York, NY, USA: Springer-Verlag New York, Inc., 2002, pp. 175–197.
[15] S. Sankaranarayanan, S. K. Chilappagari, R. Radhakrishnan, and B. Vasic, “Failures of the Gallager B decoder: Analysis and applications,” in *UCSD Center for Information Theory and its Applications Inaugural Workshop*, Feb 6-9 2006. [Online]. Available: htpp//ita.5i.net/papers/160.pdf
[16] M. Ivkovic, S. K. Chilappagari, and B. Vasic, “Eliminating trapping sets in low-density parity check codes using Tanner graph lifting,” June 24-29 2007, pp. 2266–2270.
[17] D. J. C. MacKay, “Encyclopedia of sparse graph codes.” [Online]. Available: http://www.inference.phy.cam.ac.uk/mackay/codes/data.html