On the Guaranteed Error-Correction of Decimation-Enhanced Iterative Decoders

Shiva Kumar Planjery, David Declercq, Madiagne Diouf
ETIS
ENSEA/Univ. Cergy-Pontoise/CNRS UMR 8051
95014 Cergy-Pontoise, France
Email: {shiva.planjery,declercq,madiagne.diouf}@ensea.fr

Bane Vasic
Dept. of Electrical and Computer Eng.
University of Arizona
Tucson, AZ 85721, U.S.A.
Email: vasic@ece.arizona.edu

***Abstract*—Finite alphabet iterative decoders (FAIDs) proposed for LDPC codes on the binary symmetric channel are capable of surpassing the belief propagation (BP) decoder in the error floor region with lower complexity and precision, but these decoders are difficult to analyze for finite-length codes. Recently, decimation-enhanced FAIDs (DFAIDs) were proposed for column-weight-three codes. Decimation involves fixing the bit values of certain variable nodes during message-passing based on the messages they receive after some number of iterations. In this paper, we address the problem of proving the guaranteed error-correction capability of DFAIDs for column-weight-three LDPC codes. We present the methodology of the proof to derive sufficient conditions on the Tanner graph that guarantee the correction of a given error pattern in a finite number of iterations. These sufficient conditions are described as a list of forbidden graphs that must not be contained in the Tanner graph of the code. As a test case, we consider the problem of guaranteeing the correction of four errors. We illustrate the analysis for a specific 4-error pattern and provide the sufficient conditions for its correction. We also present results on the design of codes satisfying those sufficient conditions and their impact on the achievable code rate.**

I. Introduction

Finite-length analysis of LDPC codes under message-passing (MP) decoding has attracted significant attention over the past decade with some of the notable works being computation trees by Wiberg [1], stopping sets by Di *et al.* [2], and graph-cover decoding by Vontobel and Koetter [3]. In spite of these important contributions, the problem of analyzing a particular MP algorithm for a fixed number of iterations still remains a challenge, and the guaranteed error-correction capability of finite-length LDPC codes under MP decoding remains largely unknown. This is because the dynamics of MP becomes too complex beyond a certain number of iterations as there is exponential growth in the number of nodes with the number of iterations in the computation trees of the decoder on the code. The guaranteed error-correction capability assumes greater importance for the binary symmetric channel (BSC) as it determines the slope of the error floor in the error-rate performance of the decoder [4].

The only known results on the guaranteed error-correction capability of finite-length LDPC codes are for the Gallager-A decoder which were derived by Chilappagari *et al.* [5]. They proved that for a column-weight-three code of girth $g \geq 10$, the Gallager-A decoder corrects $(g/2-1)$ errors in $g/2$ iterations, and for girth $g = 8$, the code required some additional constraints on the Tanner graph to guarantee correction of $(g/2-1) = 3$ errors. Such type of results are not known for any other MP decoder especially when the decoder utilizes soft messages due to the complexity in the analysis.

Recently, a new class of finite alphabet iterative decoders (FAIDs) [6] were proposed that have lower complexity than the BP decoder but are capable of surpassing the BP in the error floor region. While numerical results were provided showing the superior error-rate performance of 3-bit precision FAIDs over BP (floating-point), analyzing these decoders still proved to be difficult. More recently, decimation-enhanced FAIDs (DFAIDs) [7] were proposed, wherein the technique of decimation is incorporated into the FAIDs in a novel manner that allows the decoder to be more amenable to analysis while maintaining their good performance. Decimation involves fixing the bit values of certain variable nodes (see [7] for references). In [7], the proposed DFAID was able to match the good performance of the original FAID while being analyzable at the same time. However, no results on guaranteed error-correction were provided in [7].

In this paper, we address the problem of proving the guaranteed error-correction of 7-level DFAIDs for column-weight-three codes. Using the same 7-level DFAID proposed in [7], we will provide a methodology to analyze the DFAID algorithm on a given error pattern of weight-t and prove its guaranteed correction in a finite number of iterations by enforcing constraints on its neighborhood. These constraints which are regarded as sufficient conditions are described in the form of a list of subgraphs which the Tanner graph of the code must not contain. As a test case, we consider the guaranteed correction of error patterns of weight $t = 4$. While the analysis and proof is limited to one specific 4-error pattern, the methodology is applicable to prove the guaranteed correction of any given error pattern. We also provide some results measuring the impact of each constraint on the achievable code rate. Complete proofs are omitted due to page limitations.

II. Decimation-enhanced FAIDs

Let G denote the Tanner graph of an (N,K) binary LDPC code $\mathcal{C}$ consisting of the set of variable nodes $V =$

$\{v_1, \cdots, v_N\}$ and set of check nodes $C = \{c_1, \cdots, c_M\}$. The degree of a node is its number of neighbors. For column-weight-three codes, every variable node has a degree $d_v = 3$. $\mathcal{N}(u)$ denotes the set of neighbors of a node u and $\mathcal{N}(U)$ denotes the set of neighbors of all $u \in U$. Let $\hat{x}_i^{(k)}$ denote the bit value of a variable node $v_i \in V$ decided at the end of the k^{th} iteration. Let $\mathbf{r} = (r_1, r_2 \ldots, r_N)$ denote the output vector received from the BSC. Let $\mathcal{T}_i^k(G)$ denote a depth-k computation tree of graph G corresponding to a decoder enumerated for k iterations with node v_i as its root.

An N_s-level DFAID [7] denoted by $\mathscr{F}^D$ is defined as a 4-tuple given by $\mathscr{F}^D = (\mathcal{M}, \mathcal{Y}, \Phi_v^D, \Phi_c)$, where $\mathcal{M} = \{-L_s, \ldots, -L_1, 0, L_1, \ldots, L_s\}$ is the alphabet which the messages belong to, and consists of $N_s = 2s + 1$ levels with L_s being the highest level. The set $\mathcal{Y}$ denotes the set of possible *channel values* defined as $\mathcal{Y} = \{\pm\mathrm{C}\}$, and for each node $v_i \in G$, the channel value $y_i \in \mathcal{Y}$ is determined by $y_i = (-1)^{r_i}\mathrm{C}$. Φ_v^D and Φ_c are the variable and check node update functions that will be defined shortly.

Definition 1: A variable node v_i is said to be *decimated* at the end of l^{th} iteration if $\hat{x}_i^{(k)}$ is set to $\hat{x}_i^* \; \forall k > l$. Then, from the l^{th} iteration, v_i sends $(-1)^{\hat{x}_i^*} L_s$ as its outgoing message on all its edges irrespective of its incoming messages.

A *decimation rule* $\beta : \mathcal{Y} \times \mathcal{M}^{d_v} \to \{-1, 0, 1\}$ is a function used at the end of some l^{th} iteration by the decoder to decide whether a variable node should be decimated and what value it should be decimated to based on the incoming messages and the channel value in the l^{th} iteration. Let γ_i denote the output of a decimation rule applied to a node v_i. If $\gamma_i = 0$, then the node is not decimated. If $\gamma_i = 1$, then the node is decimated with $\hat{x}_i^* = 0$, and if $\gamma_i = -1$, then the node is decimated with $\hat{x}_i^* = 1$. Each instance of applying the decimation rule on all variable nodes is a *decimation round*. For this work, the rule β must also satisfy the below properties:

1) $\beta(\mathrm{C}, m_1, m_2, m_3) = -\beta(-\mathrm{C}, -m_1, -m_2, -m_3)$ $\forall m_1, m_2, m_3 \in \mathcal{M}$
2) $\beta(\mathrm{C}, m_1, m_2, m_3) \neq -1$ and $\beta(-\mathrm{C}, m_1, m_2, m_3) \neq 1$ $\forall m_1, m_2, m_3 \in \mathcal{M}$
3) Given $m_1, m_2, m_3 \in \mathcal{M}$, if $\beta(\mathrm{C}, m_1, m_2, m_3) = 1$, then $\beta(\mathrm{C}, m_1', m_2', m_3') = 1$ $\forall m_1', m_2', m_3' \in \mathcal{M}$ such that $m_1' \geq m_1$, $m_2' \geq m_2$, and $m_3' \geq m_3$.

Property 2 implies that a variable node v_i can be decimated only to its received value r_i.

There are two key aspects to note regarding the application of a decimation rule.

1) The decimation rule is applied after messages are passed iteratively for some l iterations.
2) After each decimation round, all messages are cleared to zero (note: the decimated nodes remain decimated).

β is defined using a set Ξ that consists of all unordered triples $(m_1, m_2, m_3) \in \mathcal{M}^3$ such that $\beta(\mathrm{C}, m_1, m_2, m_3) = 1$. Due to property 1, Ξ is sufficient to completely specify β.

Let m_1 and m_2 denote the extrinsic incoming messages to a node $v_i \in V$ with $d_v = 3$. The map $\Phi_v^D : \mathcal{Y} \times \mathcal{M}^{d_v - 1} \times \{-1, 0, 1\} \to \mathcal{M}$ which uses the output of β as one of its arguments is defined as

$$\Phi_v^D(y_i, m_1, m_2, \gamma_i) = \begin{cases} \Phi_v(y_i, m_1, m_2), & \gamma_i = 0 \\ \gamma_i L_s, & \gamma_i = \pm 1 \end{cases}$$

The update function $\Phi_c : \mathcal{M}^{d_c - 1} \to \mathcal{M}$ is used at a check node with degree d_c, and is defined as

$$\Phi_c(m_1, \ldots, m_{d_c - 1}) = \left(\prod_{j=1}^{d_c - 1} \mathrm{sgn}(m_j) \right) \min_{j \in \{1, \ldots, d_c - 1\}} (|m_j|)$$

where sgn denotes the standard signum function.

The subgraph induced by the set of non-decimated nodes is referred to as a *residual graph*. For the analysis in this paper, we shall use the 7-level DFAID defined with the algorithm defined below, where Φ_v is defined by Table I and the set Ξ is defined by Table II.

Algorithm 1 7-level Decimation-enhanced FAID algorithm [7]

```
1) Initialize $\gamma_i = 0 \; \forall v_i \in V$.
2) Run the decoder for three iterations using update maps
   $\Phi_v$ and $\Phi_c$ defined for the 7-level FAID.
3) Perform decimation using the rule $\beta$ for every $v_i \in V$,
   which constitutes the first decimation round.
4) Restart the decoder by resetting all the messages to zero
   and pass messages for one iteration. This implies that a
   decimated node $v_i$ will send $\gamma_i L_3$ and a non-decimated
   nodes $v_j$ will send $\Phi_v(y_j, 0, 0)$.
5) Repeat step 3) only for nodes $v_i \in V$ whose $\gamma_i = 0$,
   followed by 4), until the maximum possible number of
   nodes have been decimated.
6) Run the decoder for the remainder of iterations using
   maps $\Phi_v^D$ and $\Phi_c$.
```

At the end of the k^{th} iteration, the bit value $\hat{x}_i^{(k)}$ for a non-decimated node v_i is determined by assigning $L_i = i$ for $i \in \{1, 2, 3\}$ and calculating the sign of the sum $m_1 + m_2 + m_3 + y_i$.

TABLE I
Φ_v OF 7-LEVEL DFAID DEFINED FOR $y_i = +\mathrm{C}$

$m_1 \backslash m_2$	$-L_3$	$-L_2$	$-L_1$	0	L_1	L_2	L_3
$-L_3$	$-L_3$	$-L_3$	$-L_2$	$-L_1$	$-L_1$	$-L_1$	L_1
$-L_2$	$-L_3$	$-L_1$	$-L_1$	0	L_1	L_1	L_3
$-L_1$	$-L_2$	$-L_1$	0	0	L_1	L_2	L_3
0	$-L_1$	0	0	L_1	L_2	L_3	L_3
L_1	$-L_1$	L_1	L_1	L_2	L_2	L_3	L_3
L_2	$-L_1$	L_1	L_2	L_3	L_3	L_3	L_3
L_3	L_1	L_3	L_3	L_3	L_3	L_3	L_3

TABLE II
SET Ξ DEFINING $\beta(\mathrm{C}, m_1, m_2, m_3) = 1$

m_1	m_2	m_3
L_3	L_3	L_3
L_3	L_3	L_2
L_3	L_3	L_1
L_3	L_3	0
L_3	L_3	$-L_1$
L_3	L_2	L_2
L_3	L_2	L_1
L_3	L_2	0
L_3	L_2	$-L_1$
L_3	L_1	L_1
L_3	L_1	0
L_3	L_1	$-L_1$
L_3	0	0
L_2	L_2	L_2
L_2	L_2	L_1

Note that since $\Phi_v(\mathrm{C}, 0, 0) = L_1$, $\Phi_v(\mathrm{C}, L_1, L_1) = L_2$, and $\Phi_v(\mathrm{C}, L_2, L_2) = L_3$, at least three iterations are required for a variable node to send $\pm L_3$. Also note that the smallest message level a correct node can send is 0 in the second iteration since $\Phi_v(\mathrm{C}, -L_1, -L_1) = 0$, and $-L_1$ in the third iteration since $\Phi_v(\mathrm{C}, -L_2, -L_2) = -L_1$.

III. Analysis: guaranteed error-correction

Given an error pattern on graph G, let variable nodes initially wrong be referred to as *error nodes*, and variable nodes initially correct be referred to as *correct nodes*. Let I denote the maximum number of iterations allowed for MP on the residual graph (step 6) by the 7-level DFAID.

Firstly, note that due to Property 2 of β, an error node can only be decimated to a wrong value, and a correct node can only be decimated to the correct value. This provides a necessary condition for successful correction which is that no error node must be decimated. Therefore, the analysis for proving the guaranteed error correction of a given error pattern can be categorized into three main steps: 1) analyzing the decimation of error nodes and deriving conditions on their neighborhood such that no error node gets decimated, 2) analyzing the decimation of correct nodes in the neighborhood of the error nodes, and examining what the subsequent residual graph is for the given error pattern, and finally 3) examining whether the error nodes get corrected. This must be done for every error pattern of weight t.

We now consider the goal of proving the guaranteed correction of all error patterns of weight $t = 4$ in $I = 3$ iterations. We start out by assuming that the graph G has girth 8 and later add more constraints as needed. Fig. 1 depicts the subgraphs of all possible 4-error patterns assuming girth 8. ● denotes the error node. ■ and □ denote an odd-degree and even-degree check node in the subgraph respectively. Also note that all-zero codeword is assumed throughout the analysis.

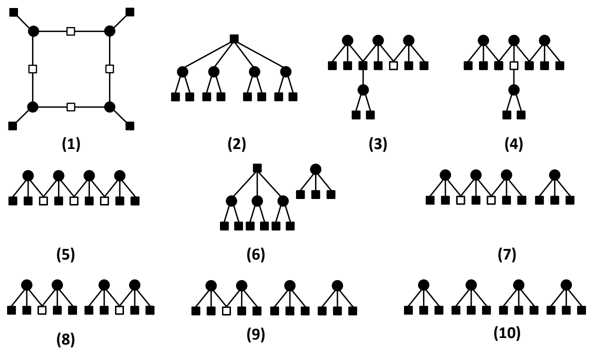

Fig. 1. Subgraphs of all possible 4-error patterns with girth 8

A. Analyzing Decimation of Error Nodes

First, note the following lemma.

Lemma 1: If $\beta(\mathrm{C}, L_1, L_1, L_1) = 0$ and no error node gets decimated at the end of first decimation round, then no error node will get decimated in any subsequent decimation round.

Due to the above lemma, for a given 4-error pattern, it now suffices to show that none of the error nodes will get decimated at the end of first decimation round, i.e., at the end of the third iteration. To examine this, we need to determine the triples of messages that node v_i can receive at the end of the third iteration for different possible neighborhoods (that are allowed) in graph G, and verify that it does not get decimated with rule β. But instead of determining all possible such triples, we can exploit the monotonicity property 3 of β, and just determine the worst-case triple defined below.

Definition 2: The *worst-case triple* of an error node v_i corresponding to a given error pattern is defined as the triple $(m_1, m_2, m_3) \in \mathcal{M}^3$ that root node v_i receives on the computation tree $\mathcal{T}_i^3(G)$ for a particular allowed neighborhood in G, such that any other possible triples $(m_1', m_2', m_3') \in \mathcal{M}^3$ the root v_i satisfies $(m_1', m_2', m_3') \geq (m_1, m_2, m_3)$.

If the worst-case triple for error node v_i is (m_1, m_2, m_3) and $\beta(-\mathrm{C}, m_1, m_2, m_3) = 0$, then error node v_i will not get decimated. However, if $\beta(-\mathrm{C}, m_1, m_2, m_3) = -1$ signifying that it gets decimated, then we have either the choice of modifying β or enforcing some constraints on the neighborhood by forbidding the presence of certain connections between nodes such that (m_1, m_2, m_3) can never be received by node v_i. each message in the triple is referred to as the *worst-case message*.

In order to determine the worst-case triple received by error node v_i, we analyze the different possible messages passed in the computation tree $\mathcal{T}_i^3(G)$ based on its different possible neighborhoods. For convenience, let us use the following conventions regarding the locations of nodes in $\mathcal{T}_i^3(G)$. The zeroth level of the tree $\mathcal{T}_i^k(G)$ is where the root is, the first level of the tree contains the variable nodes that send outgoing messages towards the root, and the k^{th} level is the base of the tree containing all the leaf nodes. Also we introduce the definition of *computation subtree*.

Definition 3: A *computation subtree* is a subset of nodes in the computation tree $\mathcal{T}_i^k(G)$ that forms a tree with a variable node in $\mathcal{T}_i^k(G)$ as its root.

Let H denote the subgraph induced by a given error pattern. In order to determine the worst-case triple for an error node v_i in the given error-pattern on the computation tree $\mathcal{T}_i^3(G)$, the following steps need to be performed.

1) Start out by drawing $\mathcal{T}_i^3(G)$ with error node v_i being the root as if we are drawing $\mathcal{T}_i^3(H)$, i.e., by treating H as an isolated graph. If H contains a degree-one check, then on the tree $\mathcal{T}_i^3(G)$, such a check will have only a correct node as its child. For a check node in H that has a degree greater than one, ignore the correct nodes connected to it on $\mathcal{T}_i^3(G)$ because the worst-case message is determined by the error node(s).
2) At the first level, examine the correct nodes on each branch in this level. Consider all possible connections for the remaining two edges emanating from this correct node: 1)either it is connected to two checks in H, 2) or to just one check in H and to another outside H, 3) or to two checks outside H. For each possibility, we need to verify that such a connection is topologically possible. For instance, if we assume the girth of G is 8, we need to ensure that such a connection does not

introduce a six-cycle. Each valid possibility leads to a subtree of depth 2, which shall be referred to as *first-level candidate subtree*.

3) For each choice of first-level candidate subtree, expand only from check nodes in H to reach the second level. Repeat the above step for the correct nodes in the second level and each possibility is referred to as *second-level candidate subtree*. Once this is done, this automatically determines the variable nodes in the third level.
4) Choose the first-level subtrees and second-level subtrees such that the root receives the worst-case triple.

As an example, let us consider the 4-error pattern on the 8-cycle which corresponds to pattern (1) in Fig. 1. Fig. 2 depicts the analysis done for determining the worst-case triple. Note that the error nodes are depicted by ●, and the correct nodes are depicted by ○.

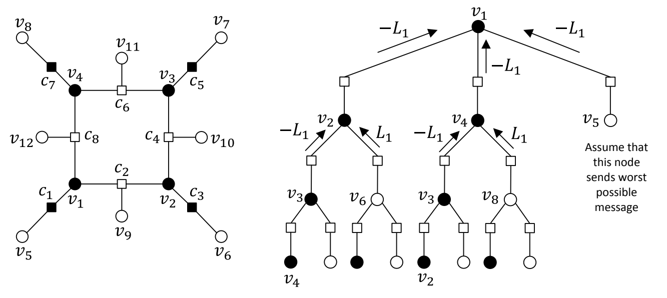

Fig. 2. Subgraph induced by the 4-error pattern and its corresponding minimal worst-case computation tree $\mathcal{S}_1^3$

In the first level, we assume that v_5 sends the worst possible messages (the smallest message) which is $-L_1$, by virtue of Φ_v. Also, in the second level, assuming that the girth is 8, note that v_6 and v_8 can be connected to at most two degree-one checks in H or it would create a 6-cycle. Therefore, the subtrees chosen at the second level do send the worst-case messages. The worst-case triple of node v_1 is $(-L_1, -L_1, -L_1)$. Since $\beta(\mathrm{C}, L_1, L_1, L_1) = 0$, this means $\beta(-\mathrm{C}, -L_1, -L_1, -L_1) = 0$, and therefore v_1 will not be decimated.

By using the symmetry of H, this implies that no error node in H gets decimated. However, note that in general such symmetry may not exist for other error patterns. For such a case, we must also determine the worst-case triples of multiple error nodes in the error pattern in order to ensure that no error node is decimated. For certain error patterns, the analysis may be less involved. For example, for pattern (10) in Fig. 1, a correct node will send in the worst-case $-L_1$ in the third iteration, and since $\beta(-\mathrm{C}, -L_1, -L_1, -L_1) = 0$, no error node gets decimated. In this manner, we obtain the following lemma regarding all 4-error patterns.

Lemma 2: If the graph G of a column-weight-three code has girth-8, then for any 4-error pattern, no error node will get decimated by the 7-level DFAID.

It is evident from the above procedure that this analysis can be done for any given error pattern of weight t. Having proved the no error nodes get decimated, the next step of the analysis is to prove the correction of the error pattern.

B. Analyzing Decimation of Correct Nodes and Sketch of Proof for Guaranteed Correction

Given an error pattern with H being the induced subgraph of the error nodes, let V^0 denote the set of error nodes in H. Let $C^0 = \mathcal{N}(V^0)$ denote the set of all the neighboring checks of the error nodes in V^0. Let $V^1 = \mathcal{N}(C^0) \setminus V^0$, $C^1 = \mathcal{N}(V^1) \setminus C^0$. In general, let $V^l = \mathcal{N}(C^{l-1}) \setminus V^{l-1}$ and $C^l = \mathcal{N}(V^{l-1}) \setminus C^{l-1}$ for $l > 0$. In order to prove the correction of the error pattern for which no error nodes are decimated, we rely on analyzing the decimation of correct nodes in the neighborhood of H using the following lemma which is based on Algorithm 1.

Lemma 3: If a correct node v_j is connected to at most one odd-degree check in H, and at the end of the p^{th} decimation round, it has at least one neighboring check $c_j \in \mathcal{N}(v_j)$ such that every node $v_i \in \mathcal{N}(c_j) \setminus v_j$ is a decimated correct node, then node v_j will be decimated in $(p+1)^{th}$ decimation round by virtue of $\beta(\mathrm{C}, L_3, L_1, -L_1) = 1$.

By the above lemma, if all nodes in V^1 are decimated, this implies all remaining correct nodes will also be decimated in some decimation round, and the residual graph is H. We can then easily verify if the error nodes in H are corrected in I iterations. Therefore, we begin the analysis by examining whether nodes in V^1 get decimated at the end of the first decimation round, which requires some assumptions on the neighborhood of H such as the number of checks in H that a node in V^1 can be connected to, or the number of checks it shares with other nodes in V^1. We then analyze whether a correct node $v_i \in V^1$ is decimated in the first round in a manner similar to the analysis for decimation of error nodes.

If node v_i does not get decimated in the first round, we can then either enforce a constraint on the neighborhood such that it does get decimated, or analyze the decimation of the correct nodes connected to its neighboring check nodes, which could belong to V^1 or V^2, so that Lemma 3 can be invoked to prove its decimation. In this manner, we continue to expand the analysis to nodes in V^2, V^3 and so forth, until either we obtain a node that is decimated in the first round, or conclude that we have reached a possible residual graph. During the analysis, each constraint enforced on the neighborhood is a sufficient condition in the form of a forbidden subgraph.

Let us revert back to the example of the 4-error pattern on the 8-cycle shown in Fig. 2. Let C_1^0 denote the set of degree-1 check nodes in H and C_2^0 denote the set of degree-2 check nodes of H. Also let $\mathcal{G}(N, M)$ denote a forbidden graph with N variable nodes and M check nodes. Regarding the nodes v_9, v_{10}, v_{11}, and v_{12}, we can prove the following lemma

Lemma 4: If a graph G of a column-weight-three code has girth-8 and does not contain $\mathcal{G}(6, 10)$, $\mathcal{G}(6, 11)\{1\}$, and $\mathcal{G}(6, 11)\{2\}$, then for an error pattern whose induced subgraph H is an 8-cycle, any correct node $v_i \in \mathcal{N}(C_2^0)$ which does

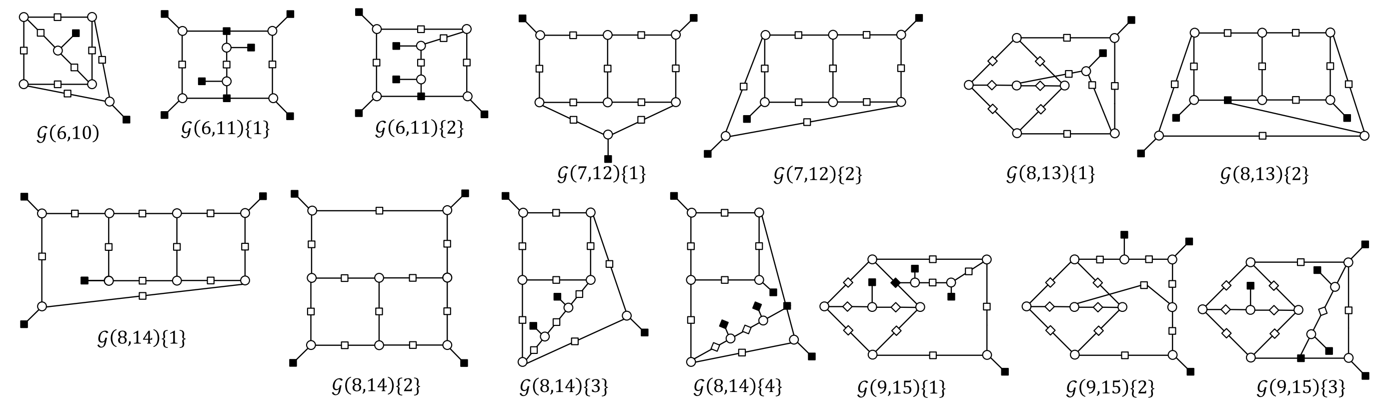

Fig. 3. Forbidden subgraphs belonging to set $\mathscr{G}$ for guaranteeing correction of the 4-error pattern in $I = 3$ iterations

not share a check with $v_j \in \mathcal{N}(C_1^0)$ is decimated in the first decimation round.

The structures of $\mathcal{G}(6,10)$, $\mathcal{G}(6,11)\{1\}$, and $\mathcal{G}(6,11)\{2\}$ are shown in Fig. 3. Note that $\{\}$ is used to distinguish between two non-isomorphic graphs with the same number of variable nodes and check nodes. Let $\mathscr{G}$ denote the set of all forbidden graphs depicted in Fig. 3. Using the above method of analysis, we can prove the following theorem.

Theorem 1: If the graph G of a column-weight-three code has girth-8, and does not contain any subgraphs belonging to $\mathscr{G}$, then the 7-level DFAID requires at most two decimations, and $I = 3$ iterations to correct the 4-error pattern whose induced subgraph forms an 8-cycle.

While the theorem is stated for a specific 4-error pattern, we do not expect the set $\mathscr{G}$ to change for the correction of all the remaining 4-error patterns. The complete proof for the theorem of guaranteeing correction of $t = 4$ errors in $I = 3$ iterations will be provided in the journal version of this paper.

C. Code Design and Achievable Code Rates

In order to verify the existence of codes satisfying the above constraints as well as to study the impact each constraint has on the achievable code rate, we started with a long quasi-cyclic code with degree profile $(d_v, d_c) = (3, 23)$, which was the maximum possible rate we were able to design with circulant size of $L = 179$ ensuring girth 8. We then removed blocks from this code in order to remove these forbidden graphs. Table III lists the statistics of the structures to remove, and the impact each forbidden graph has on the code rate upon removal. Upon removal of all forbidden graphs, the maximum code rate we were able to achieve for the family of quasi-cyclic codes with $d_v = 3$ and $L = 179$ was 0.5714. These statistics can be used as guide to choose which sufficient conditions to enforce or relax in order to obtain the desired code rate.

Acknowledgment

This work was supported by the Seventh Framework Program of the European Union, under Grant Agreement number 309129 (i-RISC project), and by NSF grants CCF-0963726 and CCF-1314147.

TABLE III
Statistics of the structures to avoid in a $(d_v, d_c) = (3, 23)$, $L = 179$ quasi-cyclic code

Forbidden graphs	Number	Max. Blocks	Max. Rate
$\mathcal{G}(6,10)$	37	19	0.8421
$\mathcal{G}(6,11)\{1\}$	8680	9	0.6667
$\mathcal{G}(6,11)\{2\}$	44981	7	0.5714
$\mathcal{G}(7,12)\{1\}$	1683	12	0.7500
$\mathcal{G}(7,12)\{2\}$	3319	12	0.7500
$\mathcal{G}(8,13)\{1\}$	576	15	0.8000
$\mathcal{G}(8,13)\{2\}$	965	19	0.8421
$\mathcal{G}(8,14)\{1\}$	160454	8	0.6250
$\mathcal{G}(8,14)\{2\}$	138285	8	0.6250
$\mathcal{G}(8,14)\{3\}$	177284	8	0.6250
$\mathcal{G}(8,14)\{4\}$	0	23	0.8696
$\mathcal{G}(9,15)\{1\}$	11809	13	0.7692
$\mathcal{G}(9,15)\{2\}$	3868	13	0.7692
$\mathcal{G}(9,15)\{3\}$	0	23	0.8696

References

[1] N. Wiberg, *Codes and Decoding on General Graphs*. PhD thesis, Linkoping University, Sweden, 1996.

[2] C. Di, D. Proietti, I. E. Telatar, T. J. Richardson, and R. L. Urbanke, Finite-length analysis of low-density parity-check codes on the binary erasure channel, *IEEE Trans. on Inform. Theory*, vol. IT48, no. 6, pp. 1570-1579, 2002.

[3] P. Vontobel and R. Koetter, "Graph-Cover Decoding and Finite-Length Analysis of Message-Passing Iterative Decoding of LDPC Codes," http://arxiv.org/abs/cs/0512078.

[4] M. Ivkovic, S. K. Chilappagari, B. Vasic, "Eliminating trapping sets in low-density parity-check codes by using Tanner graph covers," *IEEE Trans. on Inf. Theory*, vol. 54, no. 8, pp. 3763–3768, 2008.

[5] S. K. Chilappagari, D. V. Nguyen, B. Vasic, and M. W. Marcellin, "Error correction capability of column-weight-three LDPC codes under the Gallager A algorithm-Part II," *IEEE Trans. Inf. Theory.*, vol. 56, no. 6, pp. 2626-2639, Jun. 2010

[6] S. K. Planjery, D. Declercq, L. Danjean, and B. Vasic,"Finite alphabet iterative decoders, Part 1: Decoding beyond belief propagation on the binary symmetric channel," *IEEE Trans. Commun.*, vol.61, no.10, Oct. 2013.

[7] S. K. Planjery,B. Vasic,D. Declercq, "Decimation-enhanced finite alphabet iterative decoders for LDPC codes on the BSC," *Proc. Int. Symp. Inf. Theory (ISIT'2011)*, pp. 2383–2387, St. Petersburg, Russia, Jul. 2011.