Message-Aggregation Enhanced Iterative Hard-Decision Decoders

Srdan Brkic, Predrag Ivanis
School of Electrical Engineering, University of Belgrade
Emails: srdjan.brkic@ic.etf.rs, predrag.ivanis@etf.rs

Bane Vasić
Department of ECE, University of Arizona
Email: vasic@ece.arizona.edu

David Declercq
ETIS Lab, ENSEA
University Cergy-Pontoise
Email: declercq@ensea.fr

Abstract—We present an iterative decoding algorithm for annihilating trapping sets in low-density parity-check codes. In addition to classic messages, subsets of variable nodes communicate directly. We show that by allowing variable nodes to collect information from a larger part of a graph, significant improvement can be achieved in the error-floor region, compared to the classic hard decision decoders. We also propose new hybrid hard-decision decoding algorithms which employ described strategy and the Gallager B decoders as its components. Our decoder outperforms all known hard-decision decoders of same or higher complexity.

I. INTRODUCTION

Low-density parity-check (LDPC) codes under iterative decoding have been extensively investigated over the past decades. It is well-known that the message-passing sum-product algorithm (SPA) [1] provides reasonably good performance on Binary Symmetric Channels (BSCs). However, high complexity of SPA makes it unfit for a number of important applications as flash memories, fiber and free-space optical communications. A number of quantized message-passing decoders have been designed with a goal to speed up the decoding, and preserve the error correction capability of the SPA decoder [2], [3]. The effects of message quantization are mostly notable on column-weight-three codes and cause high error-floors, as in the min-sum decoders. Other finite-alphabet iterative decoders (FAIDs) proposed by Planjery et al. [4], [5] perform beyond belief-propagation for a number of practically important column-weight-three codes. However, complexity of FAIDs is still much higher compared to hard-decision decoders.

On the other hand hard-decision Gallager A/B and bit-flipping (BF) decoders are fairly simple and their performance can be evaluated for finite-length codes. Sipser and Spielman [6] showed that BF decoders can correct a constant fraction of errors if a underlying Tanner graph is a good expander, while Burshtein [7] proved that for almost all column-weight-four codes guaranteed correction capability increases linearly with code length. Chilappagari et al. [8]–[10] expressed the correction capability of hard-decision decoders through girth of Tanner graph. They showed that correction capability of BF decoders on column-weight-four codes increases exponentially with girth of Tanner graph [8]. The correction capability of column-weight-three codes is modest and for a given girth $g$, BF and Gallager A/B decoders correct $g/4 - 1$ and $g/2 - 1$, ($g > 8$), errors, respectively, while for the case when $g = 8$ the Gallager A/B decoder can not correct all weight-three error patterns. As girth of Tanner graph grows only logarithmically with the code length, the correction of higher number of errors can be achieved only for large codes, while for shorter codes hard-decision decoders are impractical regardless of their low complexity.

To fill the performance gap between simple hard-decision and FAID decoders, recently a number of bit-flipping and message passing decoders on BSC have been proposed. Nguyen and Vasic [11] proposed a class of two-bit bit-flipping (TBBF) algorithms in which messages passed between nodes in Tanner graph are reinforced with additional bit, which increases the guaranteed error correction capability of column-weight-three codes. In addition, they developed a framework for collective error correction where complementary TBBF decoders are run in parallel, which leads to the correction of all weight-four error patterns with high probability. Wadayama et al. [12] exploited the non-linear optimization of the flipping function and proposed the gradient-descent bit-flipping (GDBF) decoder. Motivated by random perturbations caused by unreliability of logic gates built in new VLSI technologies, Al Raseed et al. [13] developed probabilistic GDBF (PGDBF) decoder in which bits that meet flipping constrains are not flipped automatically, but with some probability. Randomizing the flipping decisions helps optimization process to escape from local minima and converge to a correct codeword. Recently, Ivanis et al. [14] improve the PGDBF decoder using multiple decoding attempts and random re-initializations (MUDRI) of decoders. There are also other popular bit-flipping decoders [15]–[18] which use soft channel information and are unsuitable for application on the BSC channel.

Mobini et al. [19] proposed a message-passing algorithm which updates a soft information, initialized from the channel, differentially at each iteration based on the binary messages send on edges of Tanner graph. Sassatelli et al. [20] developed the two-bit message passing decoder capable of correcting all weight-three error patterns on column-weight-four codes. In our previous work [21] we have shown how randomness caused by gate failures can be exploited to our advantage and lead to an improved performance of the Gallager B decoder.

In this paper we propose a simple deterministic bit-flipping decoder in which in addition to messages passed on edges of
Tanner graph, subsets of variable nodes communicate directly. In our approach variable nodes collect information from a larger part of a graph during a single iteration, unlike the classic iterative decoders performed on Tanner graphs in which messages propagate in multiple iterations. Allowing a variable node to be aware of its surroundings helps escaping from a trapping set and reduces error-floors. Furthermore, we show that the complexity of our decoder is comparable with the complexity of simple Gallager A/B decoder. In addition, we design new hybrid hard-decision decoding algorithm which employs described strategy and the Gallager B decoders as its components. Our approach is simpler than the collective error correction based on TBBF decoders where different component decoders need to be implemented, or MUDRI strategy where only large number of iterations lead to performance improvement. The complexity of our decoder is roughly two times higher than the complexity of Gallager B decoder, improvement. The complexity of our decoder is roughly two times higher than the complexity of Gallager B decoder, improvement.

The rest of the paper is organized as follows. In Section II the preliminaries on codes and decoding algorithms on graphs are discussed. In Section III we introduce new decoding approach. Section IV is dedicated to the hybrid decoder description, which includes the guaranteed error correction analysis. Numerical results are given in Section V, followed by a note on the complexity in Section VI and short discussion given in Section VII.

II. PRELIMINARIES

Consider a $(\gamma, \rho)$-regular binary LDPC code, denoted by $(n, k)$, with code rate $R = k/n \geq 1 - \gamma/\rho$ and parity check matrix $H$. The parity check matrix is the bi-adjacency matrix of a bipartite (Tanner) graph $G = (V \cup C, E)$, where $V$ represents the set of $n$ variable nodes, $C$ is the set of $n\gamma/\rho$ check nodes, and $E$ is the set of $n\gamma$ edges. The length of shortest cycle in $G$ is called girth and denoted by $g$. Each matrix element $H_{ji} = 1$ indicates that there is an edge $e = (v_i, c_j)$ between nodes $c_j \in C$ and $v_j \in V$, which are referred as neighbors. Let $\mathcal{N}(u)$ be a set of neighbors of a node $u$, and similarly let $\mathcal{E}(u)$ denote a set of edges connected to the node $u$. Then, $|\mathcal{N}(v_i)| = \gamma, \forall v_j \in V$ and $|\mathcal{N}(c_j)| = \rho, \forall c_j \in C$, where $\cdot$ denotes cardinality. Let $T_i$ denote a subgraph of $G$ corresponding to a depth-one computation tree of a variable node $v_i$. The sets of variable and check nodes of $T_i$ are $V_{T_i}$ and $C_{T_i}$, respectively.

We define an iterative hard-decision decoder by an 4-tuple $D = (B, C, \Phi, \Psi)$. A set $B = \{0, 1\}$ defines the binary alphabet of messages passed over edges of the Tanner graph. Similarly, a set of possible values received from the channel is also binary, i.e., $Y = \{0, 1\}$. Let a sequence of bits received from the channel be $y = (y_1, y_2, \ldots, y_n)$, $y_i \in B, 1 \leq i \leq n$. In addition, let $x = (x_1, x_2, \ldots, x_n)$ denote a codeword of an LDPC code the input of the binary symmetric channel with probability of error $\alpha$.

The decoder operate by sending binary messages over edges of the graph. Let $\mu_{e}^{(t)}$ be a message passed on edge $e = (v_i, c_j)$ from variable node $v_i$ to the check node $c_j$ in $t$-th iteration. Similarly, $\nu_{e}^{(t)}$ denotes a message passed from the check node $c_j$ to the variable node $v_j$ in the $t$-th iteration. The value of $\mu_{e}^{(t)}$ is obtained by mapping $\Phi^{(t)} : \{0, 1\}^{t+1} \rightarrow \{0, 1\}$, i.e., $\mu_{e}^{(t)} = \Phi^{(t)}(\nu_{e}^{(t)}, y_i)$, where $\nu_{e}^{(t)} = (\mu_{e}^{(t)})_{e \in E(v_i)}$, while $\Psi^{(t)} : \{0, 1\}^{p} \rightarrow \{0, 1\}$ is used to calculate $\nu_{e}^{(t)}$ as $\nu_{e}^{(t)} = \Psi^{(t)}(\mu_{e}^{(t-1)})$, where $\mu_{e}^{(t-1)} = (\mu_{e}^{(t-1)})_{e \in E(c_j)}$. Also, in each iteration a check node evaluate its parity check equation, and we have $c_j = 0$ if the $j$-th equation is satisfied and $c_j = 1$ otherwise.

III. NEW ALGORITHM FOR BREAKING TRAPPING SETS

A. Algorithm Description

The hard-decision decoders are simple but perform poorly in the error-floor region due to existence of certain subgraph structures called trapping sets. Basically, the decoder does not have enough information to make correct decisions, and became stuck in a trapping set. A problem lies in the fact that a variable node does not know the topology of its surroundings, which leads to a wrong decision. We will show that incorporating even a partial knowledge of the neighboring variable nodes in the bit decision rule improves the performance.

Let $U(v_i)$ be the number of unsatisfied checks in the neighborhood of the variable node $v_i$, and let $V_C \subseteq V_{U}$ denote the set of variable nodes whose all checks are satisfied/unsatisfied, i.e., $V_C = \{v_i \in V | U(v_i) = 0\}$ and $V_C = \{v_i \in V | U(v_i) = \gamma\}$. As an evidence of probable correctness of a variable node, we use the function $\psi(v_i) = 1 - I_{V_C}$, where $I$ is the indicator function.

The value $\psi(v_i) = 0$ indicates a based on neighboring check nodes – the variable node $v_i$ is “probably correct.” Similarly, the check node $c_m$ surrounded by the probably correct variable nodes is “probably verified”. The set of these check nodes is denoted by $C_C = \{c_m \in C | \sum_{v_k \in N(c_m)} \psi(v_k) = 0\}$.

Each iteration consists of three steps. In the first step, using the indicator $\psi$, we identify all probably correct variables, thus isolating the remaining “potentially incorrect” variables, $V_I$. The goal of the algorithm is to reduce the cardinality of the set of potentially incorrect variables until only truly corrupt variables remain. Similarly, the variables in $V_C$ are corrupt with high probability as they are connected to only unsatisfied checks. In this step we flip all such variables.

It the second step, to further reduce the set of potentially incorrect variables $V_I$, which after the second step contains variables $v_i$ with $0 < U(v_i) < \gamma$ unsatisfied checks, the algorithm operates on the computation trees of such variables, and aggregates the messages from its computation tree and incorporates it into the flipping rule. Values of $\psi(v_i)$ are recalculated and passed directly to all variable nodes in $T_i$. If a variable node receives zeros from all variables with whom it shares a satisfied parity check, it leaves the set of potentially incorrect variables. However, in the third step we flip a potentially incorrect variable $v_i$ if and only if it is connected to an unsatisfied check $c_j$ and all other variables connected to $c_j$ are not potentially incorrect.
Note that the flipping condition is restrictive and allows flipping of corrupt variables with high probability, while there is negligible probability that a correct variable is flipped. We can describe the above method in a formal way as follows.

Consider the computation tree \( \mathcal{T} \). Let \( \mathcal{T}_{\mathcal{V}} \) denote a subgraph which excludes the subgraphs induced by a particular check \( c_j \), i.e.,

\[
V_{\mathcal{T}_{\mathcal{V}}} = V_{\mathcal{T}} \setminus \mathcal{N}(c_j),
\]

\[
C_{\mathcal{T}_{\mathcal{V}}} = C_{\mathcal{T}} \setminus c_j.
\]

Note that the node \( v_i \) is also excluded from the subgraph \( \mathcal{T}_{\mathcal{V}} \). To each subgraph we associate the criterion function \( \Upsilon : \{0, 1\}^{p-1} \rightarrow \{0, 1\} \) defined as follows

\[
\Upsilon(\mathcal{T}_{\mathcal{V}}) = \begin{cases} 
0, & \text{if } \exists e_m \in C_{\mathcal{T}_{\mathcal{V}}} \cap C_c \\
1, & \text{otherwise.}
\end{cases}
\]

The value \( \Upsilon(\mathcal{T}_{\mathcal{V}}) = 0 \) indicates that \( v_i \) should be excluded from \( V_f \), while otherwise remains in \( V_f \). However, whether a variable \( v_i \) from \( V_f \) will be actually flipped depends on the other variables connected to \( c_j \), as described above. The algorithm for breaking trapping sets can be formally expressed as follows.

**Breaking Trapping Sets Algorithm**

1) Flip all bits with \( \gamma \) unsatisfied checks.

2) Calculate \( \psi(v_i) \) and \( \Upsilon(\mathcal{T}_{\mathcal{V}}) \), \( \forall v_i \in V \) and \( \forall c_j \in \mathcal{N}(v_i) \).

3) Flip each bit \( v_i \) if \( \exists c_j \in \mathcal{N}(v_i) \) such that \( c_j = 1, \Upsilon(\mathcal{T}_{\mathcal{V}}) = 1 \) and \( \forall e_m \in \mathcal{N}(c_j) \setminus v_i, \Upsilon(\mathcal{T}_{e_m}) = 0 \).

4) Repeat first three steps until there is no unsatisfied parity check equation.

We next explain the proposed algorithm on the error pattern of a (3,5)-regular code given in Fig. 1, where black and white circles denote corrupt and correct variable bits, respectively, while black and white squares denote, respectively, odd-degree and even-degree check nodes. A subgraph on interest \( G' = (V' \cup C', E') \) contains a set of variable nodes \( V' = \{v_1, v_2, \ldots, v_5\} \) and a set of check nodes \( C' = \{c_1, c_2, \ldots, c_6\} \). Let us assume that the subgraph is isolated in a sense there is no variable node in computation trees of \( \mathcal{N}(C') \setminus V' \) that share a check with a node from \( V' \). This assumption will be formalized later. Consider the check \( c_4 = 1 \), connected to the corrupt variable \( v_4 \). It can be seen that \( \Upsilon(\mathcal{T}_{c_4}) = 1 \), since both \( c_4 \) and \( c_5 \) are connected to nodes with unsatisfied checks. However, it can be observed that \( \Upsilon(\mathcal{T}_{c_4}) = 0 \) and \( \Upsilon(\mathcal{T}_{c_4}) = 0 \) since \( c_1 \) and \( c_{11} \) are connected to nodes which are involved with only satisfied checks. Similar argument holds for all other variables connected to \( c_8 \), which means that the variable \( v_4 \) needs to be flipped. Applying the same reasoning to \( c_2 \) and \( c_6 \) we can conclude that only \( v_1 \) and \( v_5 \), respectively, can not provide the proof of their correctness, and need to be flipped.

Note that in above example, successful decoding in only one iteration was possible because the trapping set was assumed to be isolated. However, it can also be shown that the error pattern would be corrected if only the check \( c_8 \) is isolated.

We later define a less loose isolation condition, which is an important ingredient of our error-correction analysis.

**B. Hard-Decision Decoder Implementation**

In this subsection we show how our decoding algorithm can be easily represented in a standard parallel implementation, which assumes that during a decoding iteration, variable and check nodes exchange one-bit messages. This means that criterion function computation is performed locally within variable nodes, and no global operations are needed. Consequently, the number of iterations increases since flipping decisions are prolonged until the information required for the criterion function computation reaches all variable nodes. This means that messages exchanged by nodes in Tanner graph different are calculated differently from iteration to iteration. Note that the first step of our algorithm can be perform in one iteration, while we need three additional iterations for the second and the third step – one to calculate \( \psi(v_i) \), other to obtain \( \Upsilon(\mathcal{T}_{\mathcal{V}}) \) for each pair \( (v_i, c_j) \), and final one to propagate \( \Upsilon(\mathcal{T}_{\mathcal{V}}) \) to all variable nodes. However, we were able to represent all operations by two Boolean functions performed on binary tuples \( p = (p_1, \ldots, p_A) \) and \( q = (q_1, \ldots, q_B) \) defined as follows

\[
\begin{align*}
F(p, q) &= \prod_{i=1}^{A} p_i \bigoplus_{i=1}^{B} q_i, \\
G(p, q) &= \prod_{i=1}^{A} p_i \times \bigoplus_{i=1}^{B} q_i,
\end{align*}
\]

where \( \times \) denotes Boolean AND. Messages passed from the variable node \( v_i \) to the check node \( c_j \) on edge \( e \) are calculated as \( \mu_e(\ell) = \Phi(\ell)(\nu_e(\ell)) = F(p_e^{(\ell)}, q_e^{(\ell)}) \), where \( p_e^{(\ell)} = \nu_e^{(\ell)} \), and

\[
q_e^{(\ell)} = \begin{cases} 
\{\hat{x}_i\}, & \ell = 0 \mod 4, \\
\emptyset, & \ell = 1 \mod 4, \\
\{\nu_e^{(1)}, 1\}, & \ell = 2 \mod 4, \\
\{\hat{x}_i, 1\}, & \text{otherwise}
\end{cases}
\]

Recall that \( \hat{x}_i \) represents the current estimate of the \( i \)-th code bit. Similarly, the messages passed on the same edge \( e \) in the opposite direction (from the check node \( c_j \) to the variable node
Algorithm 1 Hard-Decision Decoder Implementation

Input: \( y = (y_1, y_2, \ldots, y_n) \)

\( \ell \leftarrow 1 \)

\( \hat{x} \leftarrow y, \forall v_i \in V : \mu_{c}^{(0)} \leftarrow y_i, \forall e \in E(v_i) \)

\( s \leftarrow \hat{x}H^T(\forall c_j \in C : s_j \leftarrow F(\emptyset, \hat{p}_e^{(0)})) \)

while \( s \neq 0 \) and \( \ell < L \)

\( \forall c_j \in C : \nu_{c}^{(\ell)} \leftarrow \Psi^{(\ell)}(\mu_{c}^{(\ell-1)}), \forall e \in E(c_j) \)

\( \forall v_i \in V : \mu_{e}^{(\ell)} \leftarrow \Phi^{(\ell)}(\nu_{c}^{(\ell)}), \forall e \in E(v_i) \)

if \( \ell \equiv 0 \mod 4 \) then

\( \hat{x}_i \leftarrow F(\hat{p}_e^{(\ell)}, \{\hat{x}_i\}), 1 \leq i \leq n \)

else if \( \ell \equiv 3 \mod 4 \) then

\( \hat{x}_i \leftarrow F(\hat{p}_e^{(\ell)}, \{\hat{x}_i, 1\}), 1 \leq i \leq n \)

end if

\( s \leftarrow \hat{x}H^T(\forall c_j \in C : s_j \leftarrow F(\emptyset, \hat{p}_e^{(\ell)})) \)

\( \ell \leftarrow \ell + 1 \)

end while

Output: \( \hat{x} = (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n) \)

IV. MESSAGE-AGGREGATION ENHANCED DECODER AND ERROR CORRECTION ANALYSIS

It can be observed that correction of a corrupt variable node in the decoding strategy proposed in Section III depends on the correctness of the neighboring variable nodes, i.e. variable nodes with whom it shares checks. If the number of unsatisfied check nodes is the Tanner graph is low, the error pattern can be corrected, but the proposed algorithm fails to correct error patterns with too many unsatisfied checks. This means that the strategy performs well in the error-floor region. Now we show that its vulnerability in the waterfall region can be compensated by employing some other iterative decoder.

For that purpose we propose the hybrid message-aggregation enhanced decoder \( D_1 \) composed of two component decoders \( D' \) and \( D'' \), which operate in parallel as illustrated in Fig. 2.

- \( D' \) decoder: The received word is first decoded by the Gallager B decoder. If the decoding is not successful, the obtained code bit estimates are used as inputs for the

Algorithm 1 decoder. Additional Gallager B decoder is used to correct any residual errors.

- \( D'' \) decoder: The decoding starts from the Algorithm 1 and all uncorrectable error patterns are passed to the Gallager B decoder.

Fig. 2. The block diagram of the hybrid message-aggregation enhanced decoder \( D_1 \).

The decoding in \( D' \) relies on a fact that during each decoding step the number of errors reduces. However, there exist harmful structures that cause Gallager B decoder to oscillate and even increase the number of erroneous bits. For that reason, we add \( D'' \) in which decoding starts from Algorithm 1. It should be noted that in practice only one Gallager B and Algorithm 1 blocks needs to be implemented, since spatial diversity can be replaced by time diversity. We choose the Gallager B decoder due to its simplicity and known trapping sets profiles.

We next investigate the error correction capability of the hybrid decoder, which relies on the isolation condition defined as follows.

**Definition 1.** Consider a subgraph \( G' = (V' \cup C', E') \), \( G' \subset G \) and a set \( C'' \subset C' \), where \( C'' = \{c_j \in C' | c_j = 1\} \). \( G' \) is said to be isolated if \( \exists c_j \in C'' \) such that \( \forall v_i \in \{N(c_j) \setminus V'\} \exists c_m \in \{N(v_i) \setminus c_j\} \subset C_c \). Our decoding strategy applied to a specific subgraph which contains corrupt variables \( G'' \) requires that nodes outside of \( G'' \) would be also involved in the decoding of \( V' \). The isolation condition guarantees that those nodes will not be affected by corrupt variables involved in the subgraph. Note that the isolation condition do not assume the complete isolation of a trapping set, but only forbids specific subgraph structures. In the following proposition we investigate error-correction capabilities of the hybrid decoder \( D'' \).

**Proposition 1.** If the graph \( G \) of a column weight-three code has girth-8 the proposed decoder \( D'' \) can correct all weight-three error patterns which satisfy the isolation condition.

**Sketch of the proof:** In Fig. 3 all four possible weight-three error patterns in a Tanner graph with girth \( g = 8 \) are illustrated. We address each case separately, similarly as in [20], [22]. In Fig. 4 we illustrate the worst case situation related to Case 1. Erroneous variables \( v_1, v_2 \) and \( v_3 \) do not share a check, which means that all of them will be flipped during Step 1 of our strategy. However, correct variables \( v_5 \) and \( v_6 \) will be also flipped. According to the isolation condition we assumed that no neighbour of \( c_1 \) is connected to other unsatisfied checks, and that \( v_4 \) remains correct after the first decoding step, leaving
Based on the above discussion, Proposition 1 can be reformulated as follows: the decoder $D''$ can correct all weight-three error patterns on a code which Tanner graph does not contain $(16,5)$ trapping sets. Note that the Gallager B decoder can correct all weight-three error patterns only if Tanner graph does not contain $(5,3)$ trapping sets, which are more common. Applying Algorithm 1 the number of subgraphs that need to be omitted from the code structure reduces. We are currently investigated the subgraph structures that prevent correction of weight-four patterns, which will enable us to create trapping set profile for the $D''$ decoder.

V. NUMERICAL RESULTS

In this section we numerically express frame error rate (FER) performance of the decoder $D_1$ introduced in Section IV on a number of column weight-three codes. We run Gallager A/B segments for 30 iterations, and Algorithm 1 segments for 16 iterations, which gives 122 iterations in total. The min-sum, the PGDBF and sum-product algorithms are run for the 100 iterations. The two-bit bit-flipping decoder TBBF-$D_1$ is operates for 30 iterations. As TBBF-$D_1$ decoder represents parallel concatenation of four TBBF decoders optimized for Tanner quasi-cyclic code, it is run for $4 \times 30 = 120$ iterations.

In Fig. 5 we compare FER performance of different decoders on the popular Tanner $(155,64)$ code [24]. It can be observed that if targeted FER is lower than $10^{-7}$ the decoder $D_1$ outperforms all considered decoders except SPA. For example, $D_1$ decoder reaches error rate of $10^{-6}$ when $\alpha = 0.007$, while TBBF-$D_1$ decoder, which can correct all weight-three error patterns, for the same error level needs $\alpha < 0.003$. Note also that $D_1$ superiority does not come with computational cost since it complexity is much lower than all other considered decoders except the Gallager A/B decoder.

![Fig. 5: Frame error rate performance on Tanner (155,64) code.](image-url)
It is known that the Gallager A/B decoder performs poorly on Tanner quasi-cyclic (155,64) code, due to existence of (5,3) trapping sets. The decoder $D_1$ breaks the majority of critical structures and enables correction of low-weight error patterns uncorrectable by the Gallager A/B decoder. However, benefits of employing $D_1$ are not related only to the correction of (5,3) trapping sets, but to other harmful structures as well. This fact is illustrated in Fig. 6, where performance of $D_1$ and Gallager A/B decoders are compared on the code based on Latin Squares - LS(155,64). This code has the same length, row and column weights, girth and minimal distance as Tanner code, but it is free from (5,3) trapping sets. This means that the Gallager A/B decoder applied on LS(155,64) code corrects all weight-three error patterns. Even on this optimized code $D_1$ outperforms Gallager A/B by an order of magnitude in the error-floor region. For example, when $\alpha = 0.006$ Gallager A/B achieves FER approximately $10^{-6}$, while for $D_1$ error rate is $4 \times 10^{-8}$.

We also measure the improvement achievable by $D_1$ on longer codes, which is presented in Fig. 7. Three column-weight-three codes are considered: iRISC(1296,972) [25] code of length 1296, row-weight-12, and code rate 0.75; LS(2388,1793) [23] code of length 2388, row-weight-12, and code rate 0.75; Margulis(2640,1320) [26] code of length 2640, row-weight-12 and code rate 0.5. For all three codes significant improvement by an order of magnitude is noticed in the error-floor region. The most superior performance are observed on Margulis(2640,1320) code, where, for example when $\alpha = 0.02$, $D_1$ achieves error rate lower than $10^{-6}$, while the FER of the Gallager A/B decoder is less than $10^{-3}$.

VI. A NOTE ON DECODER COMPLEXITY

We next analyze complexity of the decoder given by Algorithm 1, which we define as the number of 2-input Boolean functions used in the decoder implementation per one code bit.

Note that we here only investigate computational complexity and neglect hardware overhead required for the information storage and other auxiliary operations.

In each variable node the function $\Phi_1(\{\nu \in E(v_i)\}, \emptyset)$ needs to be calculated. This function can be implemented as $\gamma$-input AND logic gate, which can be decomposed to $\gamma - 1$ 2-input AND gates. In addition, 2 XOR operations are used when $\ell = 3$ (mod 4), while $2\gamma$ XOR gates correspond to the case when $\ell = 2$ (mod 4). Thus, variable nodes contribute with $n(3\gamma + 1)$ 2-input logic gate to the overall decoder complexity.

Similarly, in each check node $\rho$-input XOR gate is used when $\ell = 0$ (mod 4). When $\ell = 1$ (mod 4) $\rho$-input AND gate and $2\rho$ XOR gates are used, while other iterations require $\rho$ $(\rho - 1)$-input AND gates and $2\rho$ XOR gates. All $n\gamma/\rho$ check nodes contribute with $\gamma \rho + 4\gamma - 2\gamma/\rho$ logic gates. Thus, the complexity of the decoder given by Algorithm 1 is

$$C_A = \gamma \rho + 7\gamma - 2\gamma/\rho + 1.$$  

We compare the complexity of our decoder with complexity of the Gallager-B decoder which can be expressed by

$$C_{GB} = \gamma(\rho + M_k - 1 - 1/\rho) + M_\gamma,$$

where $k = \gamma - (1 + (-1)^\gamma)/2$, and $M_m$ represents the complexity of the $m$-input majority logic gate and can be
calculated as [27]

\[ M_m = \left( \frac{m}{[m/2]} \right) - 1 + \sum_{i=0}^{[m/2]-2} \left( \frac{m - i}{[m/2] - i} \right). \]

Note that we include in \( C_{GB} \) the complexity of the final bit decision logic and syndrome checker. We compare \( C_{GB} \) and \( C_A \) in Fig. 8. Note that for column weight-three codes the decoder given by Algorithm 1 is slightly more complex than Gallager-B and it can be observed that for all \( \rho > 5 \) \( |C_A - C_{GB}|/C_{GB} \leq 20\% \). However, complexity of the Gallager-B decoder increases rapidly with \( \gamma \) and, for example, when \( \gamma = 5 \) and \( \rho \leq 12 \) its complexity is more than twice higher than the complexity of our strategy.

We have shown that our message-aggregation enhanced decoder, with computational complexity roughly two times higher than the complexity of the Gallager A/B decoder outperforms a number of state-of-the-art hard and soft decision decoders. Our decoder is not designed specifically for particular code or trapping set profile and can be applied for a number of different codes, as illustrated by our numerical results. Note that we employ the Gallager A/B decoder in our hybrid decoder due to its low complexity. However, using the proposed strategy, a more powerful decoders can potentially be improved. We are currently investigating the hybrid decoders that use finite-alphabet iterative decoders.

ACKNOWLEDGEMENT

This work was supported by the Seventh Framework Program of the European Union, under Grant Agreement number 309129 (i-RISC project), and in part by the NSF under Grants CCF-0963726, CCF-1314147 and ECCS-1500170.

REFERENCES