# 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 a new hybrid hard-decision decoding algorithm which employs 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 design 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 underlaying 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 $\lceil g/4 \rceil - 1$ 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 increase 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 cause 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 then the complexity of Gallager B decoder, but it outperform other hard-decision decoders on practically important column-weight-three codes.

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 $\mathbf{H}$. 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 $\mathbf{H}_{j,i} = 1$ indicates that there is an edge $e = (v_i, c_j)$ between nodes $c_i \in C$ 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 $\mathcal{T}_i$ denotes 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 $\mathcal{T}_i$ are $V_{\mathcal{T}_i}$ and $C_{\mathcal{T}_i}$, respectively.

We define an iterative hard-decision decoder by an 4-tuple $D = (\mathcal{B}, \mathcal{Y}, \Phi^{(\ell)}, \Psi^{(\ell)})$. A set $\mathcal{B} = \{0, 1\}$ 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., $\mathcal{Y} = \{0, 1\}$. Let a sequence of bits received from the channel be $\mathbf{y} = (y_1, y_2, \ldots, y_n)$, $y_i \in \mathcal{B}, 1 \leq i \leq n$. In addition, let $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ 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 $\mu_e^{(\ell)}$ be a message passed on edge $e = (v_i, c_j)$ from variable node $v_i$ to the check node $c_j$ in $\ell$-th iteration. Similarly, $\nu_e^{(\ell)}$ denotes a message passed from the check node $c_j$ to the variable node $v_j$ in the $\ell$-th iteration. The value of $\mu_e^{(\ell)}$ is obtained by mapping $\Phi^{(\ell)} : \{0,1\}^{\gamma+1} \to \{0,1\}$, i.e., $\mu_e^{(\ell)} = \Phi^{(\ell)}(\boldsymbol{\nu}_i^{(\ell)}, y_i)$, where $\boldsymbol{\nu}_i^{(\ell)} = (\nu_e^{(\ell)})_{e \in \mathcal{E}(v_i)}$, while $\Psi^{(\ell)} : \{0,1\}^{\rho} \to \{0,1\}$ is used to calculate $\nu_e^{(\ell)}$ as $\nu_e^{(\ell)} = \Psi^{(\ell)}(\boldsymbol{\mu}_i^{(\ell-1)})$, where $\boldsymbol{\mu}_i^{(\ell-1)} = (\mu_e^{(\ell-1)})_{e \in \mathcal{E}(c_j)}$. 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 $V_C$/$V_{\bar{C}}$ 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

$$V_{\bar{C}} = \{v_i \in V | U(v_i) = \gamma\}.$$

As an evidence of probable correctness of a variable node, we use the function

$$\psi(v_i) = 1 - \mathbb{1}_{V_C},$$

where $\mathbb{1}$ is the indicator function.

The value $\psi(v_i) = 0$ indicates that – 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

$$\mathcal{C}_C = \{c_m \in C | \sum_{v_k \in \mathcal{N}(c_m)} \psi(v_k) = 0\}.$$

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 $V_{\bar{C}}$ 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 $\mathcal{T}_i$. 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 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 $\mathcal{T}_i$. Let $\mathcal{T}_{i\setminus j}$ denote a subgraph which excludes the subgraphs induced by a particular check $c_j$, i.e.,

$$V_{\mathcal{T}_{i\setminus j}} = V_{\mathcal{T}_i} \setminus \mathcal{N}(c_j),$$
$$C_{\mathcal{T}_{i\setminus j}} = C_{\mathcal{T}_i} \setminus c_j.$$

Note that the node $v_i$ is also excluded from the subgraph $\mathcal{T}_{i\setminus j}$. To each subgraph we associate the criterion function $\Upsilon : \{0,1\}^{\rho-1} \to \{0,1\}$ defined as follows

$$\Upsilon(\mathcal{T}_{i\setminus j}) = \begin{cases} 0, & \text{if } \exists c_m \in C_{\mathcal{T}_{i\setminus j}} \cap \mathcal{C}_C \\ 1, & \text{otherwise .} \end{cases}$$

The value $\Upsilon(\mathcal{T}_{i\setminus j}) = 0$ indicates that $v_i$ should be excluded from $V_I$, while otherwise remains in $V_I$. However, whether a variable $v_i$ from $V_I$ 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 $\Upsilon(\mathcal{T}_{i\setminus j})$, $\forall v_i \in V$ 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 $c_j = 1$, $\Upsilon(\mathcal{T}_{i\setminus j}) = 1$ and $\forall v_m \in \{\mathcal{N}(c_j) \setminus v_i\}$ $\Upsilon(\mathcal{T}_{m\setminus j}) = 0$.
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 $C' = \{c_1, c_2, \ldots, c_9\}$. 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 $c_8 = 1$, connected to the corrupt variable $v_4$. It can be seen that $\Upsilon(\mathcal{T}_{4\setminus 8}) = 1$, since both $c_4$ and $c_5$ are connected to nodes with unsatisfied checks. However, it can be observed that $\Upsilon(\mathcal{T}_{6\setminus 8}) = 0$ and $\Upsilon(\mathcal{T}_{7\setminus 8}) = 0$ since $c_{10}$ and $c_{11}$ are connected

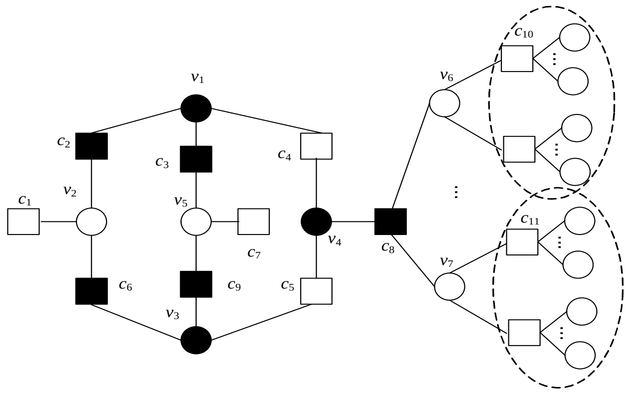

Fig. 1: Weight-three error pattern used in the example.

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 $v_3$, respectively, can not provide the proof of they 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 be also shown that the error pattern would be corrected if only the check $c_8$ is isolated. We later define a less looser 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 performed 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 $\Upsilon(\mathcal{T}_{i\setminus j})$ for each pair $(v_i, c_j)$, and final one to propagate $\Upsilon(\mathcal{T}_{i\setminus j})$ to all variable nodes. However, we were able to represent all operations by two Boolean functions performed on binary tuples $\mathbf{p} = (p_1, \ldots, p_A)$ and $\mathbf{q} = (q_1, \ldots, q_B)$ defined as follows

$$F(\mathbf{p}, \mathbf{q}) = \prod_{i=1}^{A} p_i \oplus \bigoplus_{i=1}^{B} q_i,$$
$$G(\mathbf{p}, \mathbf{q}) = \prod_{i=1}^{A} p_i \times \bigoplus_{i=1}^{B} q_i,$$

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 $\mu_e^{(\ell)} = \Phi^{(\ell)}(\boldsymbol{\nu}_i^{(\ell)}) = F(\mathbf{p}_e^{(\ell)}, \mathbf{q}_e^{(\ell)})$, where $\mathbf{p}_e^{(\ell)} = \boldsymbol{\nu}_i^{(\ell)}$, and

$$\mathbf{q}_e^{(\ell)} = \begin{cases} \{\hat{x}_i\}, & \ell = 0 \bmod 4, \\ \emptyset, & \ell = 1 \bmod 4, \\ \{\nu_e^{(\ell)}, 1\}, & \ell = 2 \bmod 4, \\ \{\hat{x}_i, 1\}, & \text{otherwise} \end{cases}$$

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 $v_i$) are calculated by

$$\nu_e^{(\ell)} = \Psi^{(\ell)}(\boldsymbol{\mu}_i^{(\ell-1)}) = \begin{cases} F(\emptyset, \tilde{\mathbf{p}}_e^{(\ell)}), & \ell = 0 \bmod 4, \\ F(\tilde{\mathbf{p}}_e^{(\ell)}, \tilde{\mathbf{q}}_e^{(\ell)}), & \ell = 1 \bmod 4, \\ G(\{\tilde{\mathbf{p}}_e^{(\ell)} \setminus e\}, \tilde{\mathbf{q}}_e^{(\ell)}), & \text{otherwise,} \end{cases}$$

where $\tilde{\mathbf{p}}_e^{(\ell)} = \boldsymbol{\mu}_i^{(\ell-1)}$ and $\tilde{\mathbf{q}}_e^{(\ell)} = \{\mu_e^{(\ell-1)}, 1\}$. It can be observed that four successive iterations form a cycle, and that bits are only flipped at the first and the last iteration of the cycle. Between them nodes exchange messages related only to the number of satisfied parity check equations. After each cycle, the decoding is restarted and estimates of the code bits $\hat{x}_i$, $1 \leq i \leq n$, obtained at the end of the cycle represents only information passed from one to the next cycle. This means that the decoder is symmetric and its operations do not depend on the transmitted codewords.

For completeness we also give the formal description of our hard-decision decoder, as illustrated in Algorithm 1.

**Algorithm 1 Hard-Decision Decoder Implementation**

```
Input: $\mathbf{y} = (y_1, y_2, \ldots, y_n)$
$\ell \leftarrow 1$
$\hat{\mathbf{x}} \leftarrow \mathbf{y}$, $\forall v_i \in V : \mu_e^{(0)} \leftarrow y_i, \forall e \in \mathcal{E}(v_i)$
$\mathbf{s} \leftarrow \hat{\mathbf{x}}\mathbf{H}^T (\forall c_j \in C : s_j \leftarrow F(\emptyset, \tilde{\mathbf{p}}_e^{(0)}))$
while $\mathbf{s} \neq \mathbf{0}$ and $\ell < L$ do
    $\forall c_j \in C : \nu_e^{(\ell)} \leftarrow \Psi^{(\ell)}(\boldsymbol{\mu}_i^{(\ell-1)}), \forall e \in \mathcal{E}(c_j)$
    $\forall v_i \in V : \mu_e^{(\ell)} \leftarrow \Phi^{(\ell)}(\boldsymbol{\nu}_i^{(\ell)}), \forall e \in \mathcal{E}(v_i)$
    if $\ell = 0 \bmod 4$ then
        $\hat{x}_i \leftarrow F(\mathbf{p}_e^{(\ell)}, \{\hat{x}_i\}), 1 \leq i \leq n$
    else if $\ell = 3 \bmod 4$ then
        $\hat{x}_i \leftarrow F(\mathbf{p}_e^{(\ell)}, \{\hat{x}_i, 1\}), 1 \leq i \leq n$
    end if
    $\mathbf{s} \leftarrow \hat{\mathbf{x}}\mathbf{H}^T (\forall c_j \in C : s_j \leftarrow F(\emptyset, \tilde{\mathbf{p}}_e^{(\ell)}))$
    $\ell \leftarrow \ell + 1$
end while
Output: $\hat{\mathbf{x}} = (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n)$
```

## IV. Message-Aggregation Enhanced Decoder and Error Correction Analysis

It can be observed that correction of a corrupt variable node in the decoding algorithm 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 algorithm 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 Algorithm 1 and all uncorrectable error patterns are passed to the Gallager B decoder.

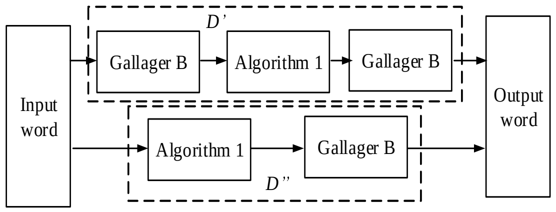

Fig. 2: The block diagram of the hybrid message-aggregation enhanced decoder $D_1$.

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 $\forall v_i \in \{\mathcal{N}(c_j) \setminus V'\}$ $\exists c_m \in \{\mathcal{N}(v_i) \setminus c_j\} \subset \mathcal{C}_C$.*

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

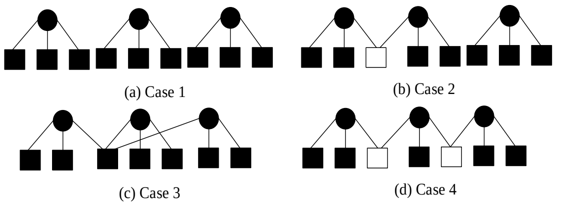

Fig. 3: All possible subgraphs subtended by three erroneous variable nodes in a graph with girth $g = 8$.

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 decoder. 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 only unsatisfied checks $c_2$, $c_3$, $c_5$, $c_6$, $c_8$ and $c_9$. It can be easily shown that these checks do not share neighbours in girth-8 graphs and that $v_5$ and $v_6$ will be corrected during Steps 2 and 3. Note also that if the subgraph were not isolated in sense of Definition 1 ($v_4$ is connected to $c_1$ not $c_{10}$) the error pattern can be corrected using the Gallager B decoder in one iteration.

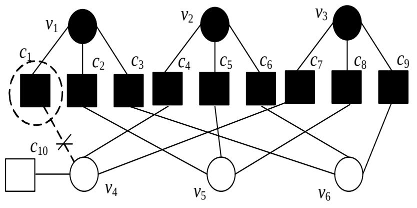

Fig. 4: Subgraph used in the explanation of Case 1 from Fig. 3.

In Case 2 there are two corrupt variables that share a check and there is no other node in girth-8 Tanner graph that is connected to all three unsatisfied checks. This means that only one node will be flipped in Step 1. If the trapping set is isolated, there exist a check node $c_j = 1$ and the variable node $v_i$ such that $\Upsilon(\mathcal{T}_{i\backslash j}) = 1$. This leads to correction of a corrupt variable connected to $c_j$. The remaining corrupt variable will be corrected in the next iteration. Note also that if the trapping set were not isolated the two corrupt variables can not be corrected by the strategy. However it is know that Gallager B in girth-8 Tanner graph can correct all weight-two error patterns.

In Case 3 all three corrupt variable share a check, which means that there is no other variable node in girth-8 Tanner graph, connected to three unsatisfied checks. According to our decoder in Step 1 only corrupt variables will be flipped and the error pattern will be corrected.

Case 4 is presented in Fig. 1. It is known (5,3){2} trapping set [23] uncorrectable by the Gallager B decoder. It can be observed that applying Algorithm 1 will lead to correction of at least one error if the subgraph were isolated. The remaining errors will be corrected by the subsequent Gallager B decoder. If the graph were not isolated the correction can not be guaranteed. However, it can be observed that there exist only one subgraph structure that corresponds to that case. It involves 16 variable nodes and 5 odd-degree check nodes and represents (16,5) trapping set for our decoder. ■

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 wight-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-$D_1$ operates for 30 iterations. As TBBF-$D_3$ 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. This is illustrated in the next section.

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) [23]. 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)

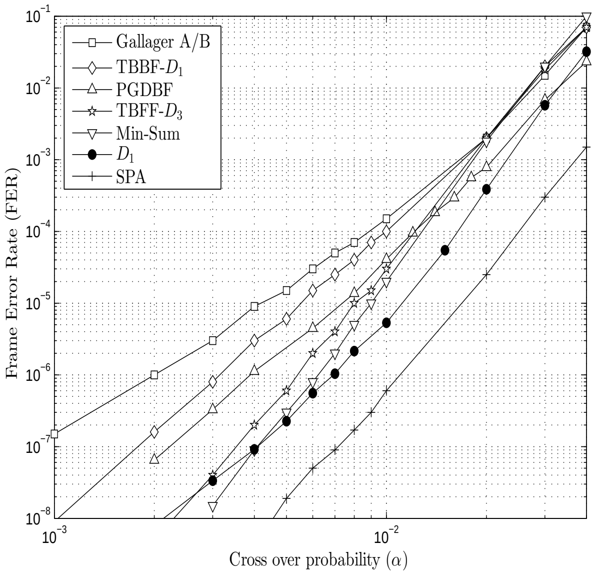

Fig. 5: Frame error rate performance on Tanner (155,64) code.

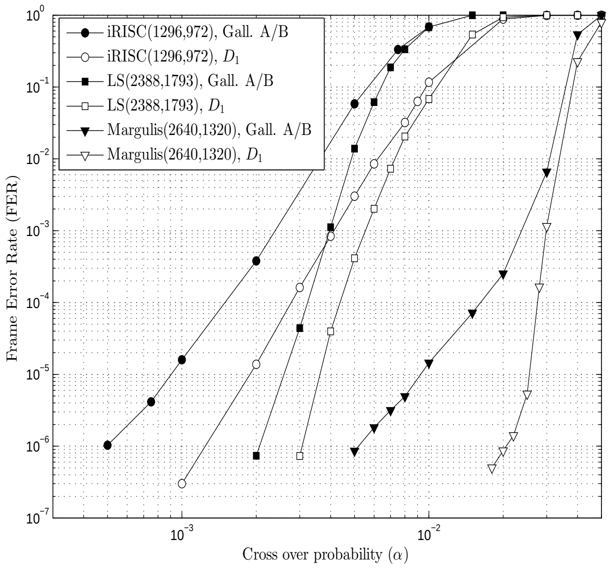

Fig. 7: Frame error rate comparison of $D_1$ and Gallager A/B decoders on different column weight-3 codes.

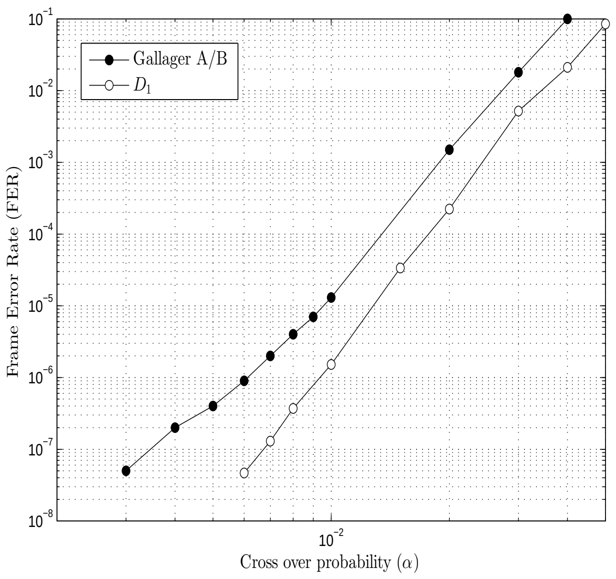

Fig. 6: Frame error rate comparison of $D_1$ and Gallager A/B decoders on Latin Square (155,64) code.

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 high 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 $10^{-4}$.

## 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 $F(\{\nu_e^{(\ell)} | e \in \mathcal{E}(v_i)\}, \emptyset)$ 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 $\ell = 3 \pmod 4$, while $2\gamma$ XOR gates correspond to the case when $\ell = 2 \pmod 4$. Thus, variable nodes contribute with $n(3\gamma+1)$ 2-input logic gates to the overall decoder complexity.

Similarly, in each check node $\rho$-input XOR gate is used when $\ell = 0 \pmod 4$. When $\ell = 1 \pmod 4$ $\rho$-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 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]

$$M_m = \binom{m}{\lceil m/2 \rceil} - 1 + \sum_{i=0}^{\lceil m/2 \rceil - 2} \binom{m-i}{\lceil m/2 \rceil - i}.$$

Note that we include in $\mathcal{C}_{GB}$ 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 decoder.

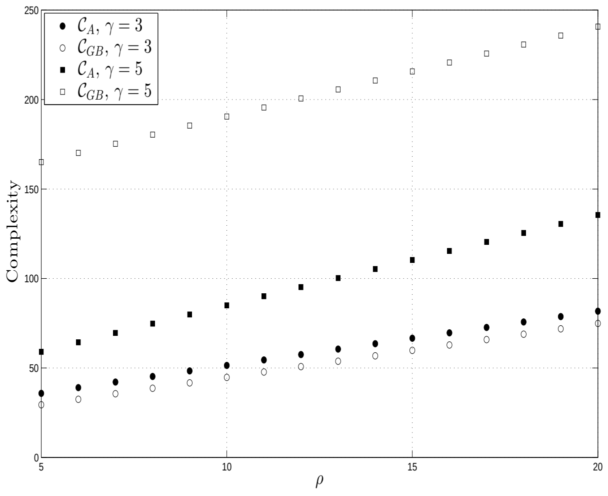

Fig. 8: Complexity comparison of the decoder defined by Algorithm 1 and the Gallager-B decoder.

## VII. Discussion

The mechanisms for breaking trapping sets proposed in this paper investigates all variable nodes in order to identify possible trapping sets and flips variables for which are suspects to be involved in a harmful structure. In contrast to other iterative decoding principles, where a flipping decisions are made based on the information received only by neighboring check nodes, a variable node in our decoder communicates also with other variable nodes. However, we have shown that cost of including additional messages is not high, and that the complexity of our decoder is comparable with the Gallager A/B decoder on column-weight-three codes. It can be built only from XOR and AND logic gates and it is significantly less complex compared to recently proposed PGDBF and TBBF decoders. The complexity of PGDBF decoders is determined by random generators required in each variable node processor, while computational overhead in TBBF decoders originates from two-bit operations required in variable and check node processors.

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

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

[2] M. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *IEEE Transactions on Communications*, vol. 47, no. 5, pp. 673–680, May 1999.

[3] J. Chen, A. Dholakia, E. Eleftheriou, M. Fossorier, and X. Hu, "Reduced complexity decoding of ldpc codes," *IEEE Transactions on Communications*, vol. 53, no. 8, pp. 1288–1299, Aug. 2005.

[4] S. Planjery, D. Declercq, L. Danjean, and B. Vasic, "Finite alphabet iterative decoders, part i: Decoding beyond belief propagation on the bsc," *IEEE Transactions on Communications*, vol. 61, no. 10, pp. 4033–4045, Oct. 2013.

[5] S. Planjery, D. Declercq, B. Vasic, and L. Danjean, "Finite alphabet iterative decoders for LDPC codes surpassing floating-point iterative decoders," *IET Electronics Letters*, vol. 46, no. 16, Aug. 2011.

[6] M. Sipser and D. Spielman, "Expander codes," *IEEE Transactions on Information Theory*, vol. 42, no. 6, pp. 1710–1722, Nov. 1996.

[7] D. Burshtein, "On the error correction of regular LDPC codes using the flipping algorithm," *IEEE Trans. Inf. Theory*, vol. 54, no. 2, pp. 517–530, Feb. 2008.

[8] S. K. Chilappagari, D. V. Nguyen, B. Vasic, and M. W. Marcellin, "On trapping sets and guaranteed error correction capability of LDPC codes and GLDPC codes," *IEEE Trans. Inf. Theory*, vol. 56, no. 4, pp. 1600–1611, Apr. 2010.

[9] ——, "Error correction capability of column-weight-three LDPC codes under the Gallager A algorithm - Part II," *IEEE Trans. Inf. Theory*, vol. 56, no. 5, pp. 2626–2639, June 2010.

[10] S. K. Chilappagari and B. Vasic, "Error-correction capability of column-weight-three LDPC codes," *IEEE Trans. Inf. Theory*, vol. 55, no. 5, pp. 2055–2061, May 2009.

[11] D. V. Nguyen and B. Vasic, "Two-Bit Bit Flipping Algorithms for LDPC Codes and Collective Error Correction," *IEEE Trans. Comm.*, vol. 62, no. 4, pp. 1153–1163, April 2014.

[12] T. Wadayama, K. Nakamura, M. Yagita, Y. Funahashi, S. Usami, and I. Takumi, "Gradient Descent Bit Flipping algorithms for decoding LDPC codes," *IEEE Trans. Comm.*, vol. 58, no. 6, pp. 1610–1614, June 2010.

[13] O. Al Rasheed, P. Ivanis, and B. Vasic, "Fault-tolerant probabilistic gradient-descent bit flipping decoder," *IEEE Communications Letters*, vol. 18, no. 9, pp. 1487–1490, Sept. 2014.

[14] P. Ivanis, O. Al Rasheed, and B. Vasic, “MUDRI: A fault-tolerant decoding algorithm,” in *in IEEE Int. Conf. on Commun. (ICC 2015)*, London, June 2015, pp. 4291–4296.

[15] A. Nouh and A. A. Banihashemi, “Reliability-based schedule for bit-flipping decoding of low-density parity-check codes,” *IEEE Trans. Comm.*, vol. 52, no. 12, pp. 2038–2040, Dec. 2004.

[16] T. Ngatched, M. Bossert, A. Fahrner, and F. Takawira, “Two bit-flipping decoding algorithms for low-density parity-check codes,” *IEEE Trans. Comm.*, vol. 57, no. 3, pp. 591–596, Mar. 2009.

[17] X. Wu, C. Ling, M. Jiang, E. Xu, C. Zhao, and X. You, “New insights into weighted bit-flipping decoding,” *IEEE Trans. Comm.*, vol. 57, no. 3, pp. 591–596, Mar. 2009.

[18] G. Sundararajan, C. Winstead, and E. Boutillon, “Noisy gradient descent bit-flip decoding for LDPC codes,” *IEEE Trans. Comm.*, vol. 62, no. 10, pp. 3385–3400, Oct. 2014.

[19] N. Mobini, A. Banihashemi, and S. Hemati, “A differential binary message-passing LDPC decoder,” *IEEE Trans. Comm.*, vol. 57, no. 9, pp. 2518–2523, Sep. 2009.

[20] L. Sassatelli, S. Chilappagari, B. Vasic, and D. Declercq, “Two-bit message passing decoders for ldpc codes over the binary symmetric channel,” in *Proceedings of IEEE International Symposium on Information Theory (ISIT 2009)*, Seoul, Korea, June–July 2009, pp. 2156–2160.

[21] B. Vasic, P. Ivanis, S. Brkic, and R. V., “Fault-resilient decoders and memories made of unreliable components,” in *Proceedings of 10th Information Theory and Applications Workshop (ITA 2015)*, San Diego, CA, Feb. 2015, paper 273, [Online Available:] http://ita.ucsd.edu/workshop/15/files/paper/paper 273.pdf.

[22] S. Planjery, D. Declercq, M. Diouf, and B. Vasic, “On the guaranteed error-correction of decimation-enhanced iterative decoders,” in *8th International Symposioum on Turbo Codes and Iterative Information Processing (ISTC)*, Bremen, Germany, Aug. 2014, pp. 57–61.

[23] D. V. Nguyen, S. K. Chilappagari, M. W. Marcellin, and B. Vasic, “On the construction of structured LDPC codes free of small trapping sets,” *IEEE Trans. Inf. Theory*, vol. 58, no. 4, pp. 2280–2302, Apr. 2012.

[24] M. Tanner, D. Sridhara, A. Sridharan, T. Fuja, and D. Costello, “LDPC block and convolutional codes based on circulant matrices,” *IEEE Trans. Inf. Theory*, vol. 50, no. 12, pp. 2966–2984, Dec. 2004.

[25] D. Declercq, *List of LDPC codes*, http://www2.engr.arizona.edu/ vasiclab/tool.php?id=7.

[26] M. Margulis, “Explicit group-theoretic constructions for combinatorial designs with applications to expanders and concentrators,” *Probl. Pered. Inform*, vol. 24, no. 1, pp. 51–60, 1988.

[27] B. Vasic, S. Brkic, and P. Ivanis, “Low complexity memory architectures based on LDPC codes: Benefits and disadvantages,” in *Proceedings of 12th Int. Conf. on Telecommun. in Modern Satellite, Cable and Broadcasting Services (TELSIKS 2015)*, Nis, Serbia, 14–17 Oct. 2015, pp. 1–8.