Two-Bit Bit Flipping Algorithms for LDPC Codes and Collective Error Correction

Dung Viet Nguyen and Bane Vasić, Fellow, IEEE

Abstract—A new class of bit flipping algorithms for low-density parity-check codes over the binary symmetric channel is proposed. Compared to the regular (parallel or serial) bit flipping algorithms, the proposed algorithms employ one additional bit at a variable node to represent its “strength.” The introduction of this additional bit allows an increase in the guaranteed error correction capability. An additional bit is also employed at a check node to capture information which is beneficial to decoding. A framework for failure analysis and selection of two-bit bit flipping algorithms is provided. The main component of this framework is the (re)definition of trapping sets, which are the most “compact” Tanner graphs that cause decoding failures of an algorithm. A recursive procedure to enumerate trapping sets is described. This procedure is the basis for selecting a collection of algorithms that work well together. It is demonstrated that decoders which employ a properly selected group of the proposed algorithms operating in parallel can offer high speed and low error floor decoding.

Index Terms—Bit flipping algorithms, low-density parity-check codes, error floor, trapping set.

I. INTRODUCTION

WITH the introduction of high speed applications such as flash memory comes the need for fast and low-complexity error control coding. Message passing algorithms for decoding low-density parity-check (LDPC) codes [1] such as the sum-product algorithm (SPA) offer very attractive error performance, especially for codes with variable degree $d_v \leq 4$. However, the complexity of these algorithms is high or the decoding speed is limited. For 3-left-regular LDPC codes, which allow lower complexity implementation, message passing algorithms (as well as other classes of decoding algorithms) usually suffer from a high error floor [2]. A high error floor is also observed for high-rate 4-left-regular codes [3], [4], which are of practical importance. This weakness of message passing algorithms in 3- and 4-left-regular codes justifies the search for alternatives which offer better trade-offs between complexity, decoding speed and error performance.

Among existing decoding algorithms for LDPC codes on the binary symmetric channel (BSC), bit flipping algorithms are the fastest and least complex, which makes them excellent candidates for high speed applications and applications in fault-tolerant memories [5], [6]. The check node operations of these algorithms are modulo-two additions while the variable node operations are simple comparisons. Most importantly, their decoding speed does not depend on the left- and right-degree of a code. The simplicity of bit flipping algorithms also makes them amenable to analysis. Many important results on the error correction capability of bit flipping algorithms have been derived. The parallel bit flipping algorithm was shown by Zyablov and Pinseker to be capable of asymptotically correcting a linear number of errors (in the code length) for almost all codes in the regular ensemble with left degree $d_v \geq 5$ [7]. In a later work, Sipser and Spielman used expander graph arguments to show that this algorithm and the serial bit flipping algorithm can correct a linear number of errors if the underlying Tanner graph is a good expander [8]. Recently, it was shown by Burshtein that regular codes with left degree $d_v = 4$ are also capable of correcting a linear number of errors under the parallel bit flipping algorithm [9].

Despite being theoretically valuable, the aforementioned capability to correct a linear number of errors does not translate to good error correcting performance on finite-length codes. This is because the fraction of correctable errors is extremely small, which means that a large code length is required to guarantee the correction of a small number of errors. Moreover, the fraction of errors guaranteed to be correctable stays the same when the code length increases. In fact, the error performance of bit flipping algorithms is typically inferior compared to hard-decoding message passing algorithms such as the Gallager A/B algorithm. The weakness of bit flipping decoding is especially visible for 3-left-regular codes for which the guaranteed error correction capability is upper-bounded by $\lceil g/4 \rceil - 1$, where $g$ is the girth of the Tanner graph representation of a code [10]. The fact that a code with $g = 6$ or $g = 8$ cannot correct certain weight-two error patterns indeed makes the bit flipping algorithms impractical regardless of their low complexity.

In recent years, numerous bit-flipping-oriented decoding algorithms have been proposed [11], [12], [13], [14], [15]. However, almost all of these algorithms require some soft information from a channel. This means that the channels assumed in these work have larger capacity than that of the BSC. A few exceptions include the probabilistic bit flipping al-
algorithm (PBFA) proposed by Miladinovic and Fossorier [16]. In that algorithm, whenever the number of unsatisfied check nodes suggests that a variable (bit) node should be flipped, it is flipped with some probability $p < 1$ rather than being flipped automatically. This random nature of the algorithm slows down the decoding, which was demonstrated to be helpful in practical codes whose Tanner graphs contain cycles. The idea of slowing down the decoding can also be found in a bit flipping algorithm proposed by Chan and Kschischang [17]. This algorithm, which is used on the additive white Gaussian noise channel (AWGNC), requires a certain number of decoding iterations between two possible flips of a variable node.

In this paper, we propose a new class of bit flipping algorithms for LDPC codes on the BSC. These algorithms are designed in the same spirit as the class of finite-alphabet iterative decoders (FAID) [18]. In the proposed algorithms, an additional bit is introduced to represent the strength of a variable node. Given a combination of satisfied and unsatisfied check nodes, an algorithm may reduce the strength of a variable node before flipping it. An additional bit is also introduced at a check node to indicate its reliability. Similar to the aforementioned PBFA, our algorithms also slow down the decoding. However they only do so when necessary and in a deterministic manner. We call the proposed algorithms two-bit bit flipping (TBF) algorithms.

For the newly proposed algorithms, we provide a complete decoding failure analysis. More importantly, we give a rigorous procedure to select a collection of algorithms based on their complementariness in correcting different error patterns. Algorithms selected by this procedure have provably good error performance as they can correct a wide range of diverse error configurations. By the nature of bit flipping, these algorithms operate at high speed. Consequently, TBF algorithms can finally be considered practical and suitable for high speed applications.

As one can expect, a TBF algorithm (like other sub-optimal graph-decoding algorithms) fails to correct some low-weight error patterns due to the presence of certain small subgraphs in the Tanner graph. In this paper, we characterize a special class of these subgraphs and refer to them with the common term “trapping sets.” Our definition of a trapping set for a given algorithm readily gives a sufficient condition for successful decoding. The set of all possible trapping sets of a given decoding algorithm constitutes the algorithm’s trapping set profile. A unique property of trapping sets of TBF algorithms is that a trapping set profile may be obtained by a recursive procedure. The diversity among trapping set profiles of different algorithms allows us to select groups of algorithms such that they can collectively correct error patterns that are uncorrectable by individual algorithms. Given a code of certain rate, the method proposed in this paper determines the number of correctable errors and finds a complementary set of two-bit flipping decoders.

The rest of the paper is organized as follows. Section II gives the necessary background. Section III introduces the class of TBF algorithms. In Section IV, we define trapping sets, trapping set profiles and describe the recursive procedure for constructing a trapping set profile. Section V discusses the selection of algorithms. Section VI gives numerical results and Section VII concludes the paper.

II. Preliminaries

Let $C$ denote an $(n,k)$ binary LDPC code. $C$ is defined by the null space of $H$, an $m \times n$ parity-check matrix of $C$. $H$ is the bi-adjacency matrix of $G$, a Tanner graph representation of $C$. $G$ is a bipartite graph with two sets of nodes and a set of edges. The two sets of nodes are: $n$ variable (bit) nodes $V(G) = \{1, 2, \ldots, n\}$ and $m$ check nodes $C(G) = \{1, 2, \ldots, m\}$. $E(G)$ denotes the set of edges. A vector $r = (r_1, r_2, \ldots, r_n)$ is a codeword iff $rH^T = 0$, where $H^T$ is the transpose of $H$. The support of $r$, denoted as supp$(r)$, is defined as the set of all variable nodes (bits) $v \in V$ such that $r_v \neq 0$. A $d_v$-left-regular ($d_c$-right-regular) LDPC code has a Tanner graph $G$ in which all variable nodes (check nodes) have degree $d_v$ ($d_c$). A $(d_v,d_c)$-regular LDPC code is $d_v$-left-regular and $d_c$-right-regular. In this paper, we only consider $(d_v,d_c)$-regular LDPC codes. The length of the shortest cycle in the Tanner graph $G$ is called the girth $g$ of $G$. A subgraph of a bipartite graph $G$ is a bipartite graph $U$ such that $V(U) \subseteq V(G)$, $C(U) \subseteq C(G)$ and $E(U) \subseteq E(G)$. $G$ is said to contain $U$. Furthermore, if $Y$ is a graph which is isomorphic to $U$ then $G$ is also said to contain $Y$. In a bipartite graph $G$, the induced subgraph on a set of variable nodes $V_v \subseteq V(G)$ is a bipartite graph $Y$ with $V(Y) = V_v$, $C(U) = \{c \in C(G) : 2v \in E(c) \text{ such that } (v,c) \in E(G)\}$ and $E(U) = \{(v,c) \in E(G) : v \in V_v\}$.

Denote by $x$ the transmitted codeword. Consider an iterative decoder and let $\hat{x} = (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n)$ be the decision vector after the $l$th iteration, where $l$ is a positive integer. At the end of the $l$th iteration, a variable node $v$ is said to be corrupt if $\hat{x}_v = x_v$, otherwise it is correct. A variable node $v$ is said to be eventually correct if there exists a positive integer $l_c$ such that for all $l$ with $l \geq l_c$, $\hat{x}_v = x_v$.

Assume the transmission of the all-zero codeword over the BSC. Since $x_v = 0 \forall v$, at the end of the $l$th iteration, a variable node is corrupt if $\hat{x}_v = 1$ and is correct if $\hat{x}_v = 0$. Let $y = (y_1, y_2, \ldots, y_n)$ be the channel output vector. A variable node is said to be initially corrupt if $y_v = 1$, otherwise it is initially correct. Furthermore, the support supp$(y)$ is simply the set of variable nodes that are initially corrupt. For the sake of convenience, we let $\hat{x}^0 = y$. Let $s^l = (s_1^l, s_2^l, \ldots, s_m^l)$ denote the syndrome vector of the decision vector after the $l$th iteration, i.e., $s^l = \hat{x}^lH^T$. Also let $s^0 = yH^T$ be the syndrome vector of the channel output vector $y$. A check node $c$ is said to be satisfied at the beginning of the $l$th iteration if $s_{c\setminus -1}^0 = 0$, otherwise it is unsatisfied.

For any variable node $v$ in a Tanner graph $G$, let $\chi^1_v$ and $\chi^0_v$ denote, respectively, the number of satisfied check nodes and unsatisfied check nodes that are connected to $v$ at the beginning of the $l$th iteration. A simple hard decision decoding algorithm for LDPC codes on the BSC, known as the parallel bit flipping algorithm [7], [8] is described as follows: The algorithm iterates until a valid codeword is found or until a maximum number of $l^{th}$ iterations is reached. In each iteration, the algorithm “flips” in parallel the variable nodes that are connected to more unsatisfied than satisfied check nodes, i.e., flip $v$ if $\chi^0_v(v) > \chi^1_v$. 
III. THE CLASS OF TBF ALGORITHMS

The class of TBF algorithms is described in this section. We start with two motivating examples to illustrate the advantages of having more bits to represent variable node and check node states.

A. Motivating Examples: Two-Bit Variable Nodes and Two-Bit Check Nodes

In the first example, ○ and ● denote a correct and a corrupt variable node at the end of the \((l-1)\)th iteration, respectively, while ◯ and ■ denote a satisfied and an unsatisfied check node at the beginning of the \(l\)th iteration, respectively. Let \(C\) be a 3-left-regular LDPC code with \(g = 8\). Assume that the variable nodes \(v_1, v_2, v_3\) and \(v_4\) form an eight-cycle as shown in Fig. 1. Consider decoding with the parallel bit flipping algorithm when only \(v_1\) and \(v_2\) are initially corrupt. Fig. 1(a) and (b) illustrate the decoding in the first and second iteration, respectively. It can be seen that the set of corrupt variable nodes after a decoding iteration alternates between \{\(v_1, v_3\}\} and \{\(v_2, v_4\}\), and thus decoding fails. Decoding fails in the above scenario because the parallel bit flipping algorithm flips variable nodes with either \(\chi^1(v) = 3\) or \(\chi^1(v) = 2\). The algorithm is too “aggressive” when flipping a variable node \(v\) with \(\chi^1(v) = 2\). From this observation, let us consider a modified algorithm which only flips a variable node \(v\) with \(\chi^1(v) = 3\). Under this modified algorithm, decoding succeeds in the above scenario. However, if only \(v_1\) and \(v_2\) are initially corrupt, as demonstrated in Fig. 1(c), then decoding again fails because no variable node would be flipped. The modified algorithm is now too “cautious” to flip a variable node \(v\) with \(\chi^1(v) = 2\).

Both decisions (to flip and not to flip a variable node \(v\) with \(\chi^1(v) = 2\)) can lead to decoding failure. However, we must pick one or the other due the assumption that the state of a variable node \(v\) at the end of the \(l\)th iteration can either be correct (if \(\hat{\chi}^1_v = 0\)) or corrupt (if \(\hat{\chi}^1_v = 1\)). Relaxing this assumption is therefore required for a better bit flipping algorithm.

Let us now assign a variable node to one out of four states. Specifically, in addition to the hard decision, a variable node \(v\) is also either a strong variable node or a weak variable node. We use \(0_s, 1_s, 0_w\), and \(1_w\) to denote the state of a strong zero, strong one, weak zero and weak one variable node, respectively. The set of possible states of a variable node is denoted by \(A_v\). Let \(w = (w_1, w_2, \ldots, w_l)\) be a vector in \(A_v^l\) such that \(w_v\) gives the state of variable node \(v\) at the end of the \(l\)th iteration. The decision vector \(\hat{x}\) is determined upon \(w\) as follows: \(\hat{x}_v = 1\) if \(w_v = 1_s\) or \(w_v = 1_w\), and \(\hat{x}_v = 0\) if \(w_v = 0_s\) or \(w_v = 0_w\) \(\forall v \in V\). The state \(w_v\) of a variable node \(v\) is initialized to \(\Delta^0_v\) if \(y_v = 0\) and to \(\Delta^1_v\) if \(y_v = 1\), where \((\Delta^0_v, \Delta^1_v) \in \{(0_s, 1_s), (0_w, 1_w)\}\). Since \(d_v = 3\), let \(f_1 : A_v \times \{0, 1, 2, 3\} \rightarrow A_v\) be the function defined in Table I. Consider the bit flipping algorithm which we call the TBF Algorithm 1 (TBF1). This algorithm iterates for a maximum of \(l^m_\text{TBF1} > 3\) iterations and has \((\Delta^0_v, \Delta^1_v) = (0_s, 1_s)\).

Algorithm 1 TBF1

\[
\begin{align*}
\forall v : w^0_v & \leftarrow \Delta^0_v, l \leftarrow 1 \\
\text{while } s^l_i \neq 0 \text{ and } l < l^m_\text{TBF1} \text{ do} & \\
\forall v : w^l_v & \leftarrow f_1(w^{l-1}_v, \chi^1_{\hat{x}}(v)); l \leftarrow l + 1; \\
\text{end while}
\end{align*}
\]

Compared to the parallel bit flipping algorithm and its modified version discussed above, TBF1 possesses a gentler treatment for a variable node \(v\) with \(\chi^1_{\hat{x}}(v) = 2\). It tries to reduce the “strength” of \(v\) before flipping it. One can verify that TBF1 is capable of correcting the error configurations shown in Fig. 1. Moreover, TBF1 is also capable of correcting any error pattern with weight up to 3 in a girth \(g = 8\), 3-left-regular LDPC code as stated in the following proposition.

Proposition 1: TBF1 is capable of correcting any weight-three error pattern in a girth \(g = 8\), 3-left-regular LDPC code with minimum distance \(d_{\text{min}} > 6\).

Proof: By enumerating trapping set profiles (to be discussed in Section IV-B).

Remarks: It has been shown that for girth \(g = 8\), 3-left-regular LDPC codes, the Gallager A/B algorithm can only guarantee to correct up to two errors [19]. This means that the guaranteed error correction capability of TBF1 is better for girth \(g = 8\), 3-left-regular LDPC codes.

Now we explore the possibility of using two bits to represent the states of a check node.

In the second example, we use ○, ○, ● and ■ to denote a 0\(_s\) variable node, a 0\(_w\) variable node, a 1\(_s\) variable node and a 1\(_w\) variable node at the end of the \((l-1)\)th iteration, respectively. The symbols ◯ and ■ still represent a satisfied and an unsatisfied check node at the beginning of the \(l\)th iteration.

Assume a decoder that uses TBF1. Fig. 2 illustrates the first, second and third decoding iteration of TBF1 on an error configuration with four initially corrupt variable nodes \(v_1, v_3, v_4\) and \(v_6\). We assume that all variable nodes which are not in this subgraph remain correct during decoding and will not be considered in this analysis. All variable nodes are...
initialized as strong variable nodes. At the beginning of
the third iteration, every variable node is connected to two satisfied
check nodes and one unsatisfied check node. The algorithm
does not change the state of any variable node and hence
decoding fails.

In the above error configuration, a setting of two unsatisfied
and one satisfied check nodes is considered unreliable. Thus
TBFA1 reduces the strength of $v_3, v_7$, and $v_8$ in the first
iteration before it flips them in the second iteration.
However, decoding with TBFA1 fails because the states of $v_3$ and $v_7$ never change. One can realize that if a setting
of two unsatisfied and one satisfied check nodes is considered
unreliable, then a setting of one unsatisfied and two satisfied
check nodes should also be considered unreliable because such
a check node setting prevents the corrupt variable nodes
$v_3$ and $v_7$ from being corrected. Unfortunately, TBFA1 cannot
reduce the strength, and then flip variable nodes with less
than $d_v/2$ unsatisfied check nodes since many other correct
variable nodes in the Tanner graph would also be flipped.

TBFA1 is incapable of evaluating the reliability of check node
settings in which the number of unsatisfied check nodes is close
to $|d_v|/2$. We now demonstrate that such reliability can
be evaluated with a new concept of check node reliability
introduced below.

**Definition 1:** A satisfied check node is called *previously
satisfied* (previously unsatisfied) if it was satisfied (unsatisfied)
in the previous decoding iteration, otherwise it is called *newly
satisfied* (newly unsatisfied).

We use $0_p$, $0_n$, $1_p$ and $1_n$ to denote the states of a
previously satisfied, a newly satisfied, a previously unsatisfied
and a newly unsatisfied check node, respectively. The set
of possible states of a check node is denoted by $\mathcal{A}_c$. Let
$\mathbf{z} = (z_1, z_2, \ldots, z_m)$ be a vector in $\mathcal{A}_c^m$ such that $z_c$ gives
the state of check node $c$ at the beginning of the $l$th iteration.

Recall that $s_c^l$ is the syndrome value associated with check
node $c$. Equivalently, $s_c^{l+1} = \Phi(s_c^l, s_c^l)$, where the check
node update function $\Phi : \{0, 1\}^2 \rightarrow \mathcal{A}_c$ is defined as follows:
$\Phi(0, 0) = 0_p$, $\Phi(0, 1) = 1_n$, $\Phi(1, 0) = 0_n$ and $\Phi(1, 1) = 1_p$. The state $z_c^l$ of a check node $c$ is initialized to $\Delta_c^0$ if $s_c^0 = 0$ and to $\Delta_c^1$ otherwise, where $(\Delta_c^0, \Delta_c^1) \in \{(0_p, 1_p), (0_n, 1_n)\}$.

Let $\chi_{0_p}(v)$, $\chi_{0_n}(v)$, $\chi_{1_p}(v)$ and $\chi_{1_n}(v)$ be the number of
previously satisfied, previously unsatisfied, newly satisfied
and newly unsatisfied check nodes that are connected to
a variable node $v$ at the beginning of the $l$th iteration, respectively. Let $\Xi_d$ denote the set of all ordered 4-tuples
$(\xi_1, \xi_2, \xi_3, \xi_4)$ such that $\xi_i \in \mathbb{N}$ and $\sum_i \xi_i = d_v$. Clearly,$\Xi_{d_v} \subset \{(\chi_{0_p}(v), \chi_{0_n}(v), \chi_{1_p}(v), \chi_{1_n}(v))\}$. Let $f_2 : \mathcal{A}_v \times \Xi_{d_v} \rightarrow \mathcal{A}_c$ be a function defined in (1):

Consider the bit flipping algorithm, which we call the TBF
Algorithm 2 (TBFA2). This algorithm iterates for a maximum
of $l_{TBFA2}$ iterations and has $(h_0, h_1) = (0_p, 1_p)$ and $(h_0^e, h_1^e) = (0_p, 1_p)$.

**Algorithm 2 TBFA2**

$\forall v : w_0^v \leftarrow \Delta_c^0$, $\forall c : z_c^l \leftarrow \Delta_c^0$, $l \leftarrow 1$

while $s_c^l \neq 0$ and $l < l_{TBFA2}$ do

$\forall v : w_0^v \leftarrow f_2(w_0^v, \chi_{0_p}(v), \chi_{0_n}(v), \chi_{1_p}(v), \chi_{1_n}(v))$

$\forall c : z_c^{l+1} \leftarrow \Phi(s_c^l, s_c^l)$; $l \leftarrow l + 1$

end while

**Table I**

<table>
<thead>
<tr>
<th>$w_u$</th>
<th>$x_u(v)$</th>
<th>$w_u$</th>
<th>$x_u(v)$</th>
<th>$w_u$</th>
<th>$x_u(v)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
</tbody>
</table>

Fig. 3. The decoding of the TBFA2 on a weight-four error configuration. All variable nodes are correct at the end of the 9th iteration. (a)–(i) Iteration 1–9.

The TBFA2 considers a setting of one newly unsatisfied,
one newly satisfied and one previously satisfied check node to
be less reliable than a setting of one previously unsatisfied
and two previously satisfied check nodes. Therefore, it will reduce
the strength of $v_3$ and $v_4$ in the third iteration. Consequently,
the error configuration shown in Fig. 2 can now be corrected
after 9 iterations, as illustrated in Fig. 3. In Fig. 3, we use
□, □□□□ and □ to represent a previously satisfied check node,
a newly satisfied check node, a previously unsatisfied
check node and a newly unsatisfied check node at the beginning of
the $l$th iteration, respectively.

**Remarks**: Proposition 1 also holds for the TBFA2.

**B. TBF Algorithms**

**Definition 2:** The class $\mathcal{F}$ of TBF algorithms is given in
Algorithm 2, where $\mathbf{z}^l$, $\mathbf{w}^l$, $\mathcal{A}_v$, $\mathcal{A}_c$, $\Delta_v = (\Delta_0^v, \Delta_1^v)$, $\Delta_c = (\Delta_0^c, \Delta_1^c)$, $\chi_{0_p}(v), \chi_{0_n}(v), \chi_{1_p}(v), \chi_{1_n}(v)$ were defined in the above motivating examples, and $f_2 = f$ is treated as a
general function defining the transition of a variable node from
one state to another. A TBF algorithm $\mathcal{F} = (f, l_{TBFA}, \Delta_v, \Delta_c)$
iteratively updates $\mathbf{z}^l$ and $\mathbf{w}^l$ until all check nodes are satisfied
or until a maximum number of iteration $l_{TBFA}$ is reached.

The check node update function $\Phi : \{0, 1\}^2 \rightarrow \mathcal{A}_c$ is defined as follows: $\Phi(0, 0) = 0_p$, $\Phi(0, 1) = 1_n$, $\Phi(1, 0) = 0_n$ and $\Phi(1, 1) = 1_p$. The variable node update is specified by a
function $f : \mathcal{A}_v \times \Xi_{d_v} \rightarrow \mathcal{A}_c$, where $\Xi_{d_v}$ is the set of all ordered 4-tuples $\xi = (\xi_1, \xi_2, \xi_3, \xi_4)$ such that $\xi_i \in \mathbb{N}$
and $\sum_i \xi_i = d_v$. The function $f$ must satisfy the following conditions: (1) Symmetry: $f$ must be symmetric with respect

to 0 and 1 in the sense that if $\xi = (\xi_1, \xi_2, \xi_3, \xi_4) \in \Xi$, and $(w_{v_1}^t, w_{v_2}^t) \in \{(0_s, 1_s), (0_v, 1_w)\}$, then the following are true:

(a) $f(w_{v_1}^t, \xi_1) = 0_s \iff f(w_{v_2}^t, \xi_1) = 1_s$, 
(b) $f(w_{v_1}^t, \xi_1) = 0_v \iff f(w_{v_1}^t, \xi_1) = 1_v$, 
(c) $f(w_{v_1}^t, \xi_1) = 1_s \iff f(w_{v_2}^t, \xi_1) = 0_s$, and 
(d) $f(w_{v_1}^t, \xi_1) = 1_w \iff f(w_{v_2}^t, \xi_1) = 0_w$.

IV. A FRAMEWORK FOR FAILURE ANALYSIS

To select a good algorithm or a good collection of algorithms, it is necessary to establish a framework to analyze different choices of the variable node update functions. Note that the notion of trapping sets is useful. Hence, we will focus on a subset of variable nodes that are not eventually correct.

A. Trapping Sets and Trapping Set Profiles of TBF Algorithms

Although the term trapping set was originally defined as a set of variable nodes that are not eventually correctable by an iterative decoding algorithm [20], in the literature it has been used more frequently to refer to a combinatorially defined subgraph that may be harmful to decoding. The justification for this less rigorous use of terminology is that the variable node set of a so-called trapping set (a subgraph) would be a set of all eventually-reachable variable nodes if and only if there exists a positive integer $l$ such that $\supp(y^l) = \supp(\xi^l)$ for all $l \geq l$. If $\supp(y) = \emptyset$ and $y'$ is a fixed point, then $\hat{F}(y') = \supp(y')$ is called a fixed set.

Theorem 1 [10]: Let $C$ be an LDPC code with $d_v$-left-regular Tanner graph $G$. Let $T$ be a set consisting of variable nodes with induced subgraph $J$. Let the check nodes in $J$ be partitioned into two disjoint subsets: $C_{\text{odd}}(J)$ consisting of check nodes with odd-degree and $C_{\text{even}}(J)$ consisting of check nodes with even-degree. Then $T$ is a fixed set for the bit flipping algorithms (serial or parallel) iff: (a) Every variable node in $V(J)$ has at least $\lfloor \frac{d_v}{2} \rfloor$ neighbors in $C_{\text{even}}(J)$ and (b) No collection of $\lfloor \frac{d_v}{2} \rfloor + 1$ check nodes of $C_{\text{odd}}(J)$ share a neighbor outside $J$.

By definition, fixed sets are responsible for all the decoding failure events in which the set of corrupt variable nodes remain unchanged after a certain number of decoding iterations. Note that the number of initially corrupt variable nodes in such a decoding failure event can be less than the number of variable nodes in the fixed set. Besides, fixed sets (subgraphs that satisfy particular conditions) are also responsible for some decoding failure events in which the set of corrupt variable nodes oscillates between two different (but not necessary disjoint) sets of variable nodes, as seen previously in Section III-A. Decoding failures caused by fixed sets are the most frequent failures of the parallel bit flipping algorithm, but they are not the sole cause of error floor. In fact, it is easy to find decoding failure events caused by harmful subgraphs that are not fixed sets, such as the ones shown in Fig. 2. However, when additional bits are used in TBF algorithms, not all fixed sets cause decoding failures. Because TBF algorithms in general have a stronger error correction capability than the parallel bit flipping algorithm, it is necessary to include all potentially harmful subgraphs into the analysis and eliminate the ones that are not. To achieve this goal, we therefore redefine the notion of a trapping set for a TBF algorithm, as we now explain. First, we introduce the following definition of failures of a TBF algorithm.

Definition 4: Consider a Tanner graph $G$ and a TBF algorithm $\mathcal{F} = (f, l^2, \Delta_v, \Delta_e)$. Let $V_0$ denote the set of variable nodes that are initially corrupt and let $J$ denote the induced subgraph on $V_0$. If under the algorithm $\mathcal{F}$, decoding fails after $l^2$ iterations, then we say that $\mathcal{F}$ fails because of the subgraph $J$ of $G$.

Example 1: Consider a 3-left-regular code $C$ with bipartite graph $G$ and assume that the induced subgraph on $V_0 = \{v_1, v_2, \ldots, v_7\} \in V(C)$ is depicted in Fig. 2(a). Further assume that no two check nodes in $\{c_1, c_2, \ldots, c_{11}\}$ share a neighboring variable node $\notin V_0$. Let $V_0 = \supp(y) = \{v_1, v_3, v_4, v_6\}$ and let $J$ denote the induced subgraph on $V_0$. Then, the TBF fails because of the subgraph $J$ of $G$.

Let us now assume that during the transmission of the codeword $x$, the BSC makes exactly $t$ errors. Denote by $T$ the set of all $d_v$-left-regular Tanner graphs with $t$ variable nodes. It is clear that the induced subgraph on the set of initially corrupt variable nodes is isomorphic to a graph in $T$. Let $I$ be a Tanner graph in $T$ and let $\mathcal{E}_I(\mathcal{F})$ denote the set of Tanner graphs containing a subgraph $J$ isomorphic to $I$ such that $\mathcal{F}$ fails because of $J$. The following facts follow: (1) If $C$ is represented by $G \in \bigcup_{I \in T} \mathcal{E}_I(\mathcal{F})$, then there exist some weight-$t$ error patterns for which algorithm $\mathcal{F}$ fails to correct. (2) If, on the other hand, $G \notin \bigcup_{I \in T} \mathcal{E}_I(\mathcal{F})$, then algorithm $\mathcal{F}$ is capable of correcting any weight-$t$ error patterns. Although these facts are simple, they are important elements in our analysis. However, the set of Tanner graphs $\mathcal{E}_I(\mathcal{F})$ is undeniably too general to be useful. Hence, we will focus on a subset $\mathcal{E}_{\mathcal{I}}(\mathcal{F})$ of $\mathcal{E}_I(\mathcal{F})$ formulated as follows.

Definition 5: Consider a Tanner graph $S_1 \in \mathcal{E}_I(\mathcal{F})$ such that $\mathcal{F}$ fails because of the subgraph $J_1$ of $S_1$. Then, $\mathcal{E}_{\mathcal{I}}(\mathcal{F})$
consists of all \( S_1 \) for which there does not exist \( S_2 \in \mathcal{E}_1(\mathcal{F}) \) such that: (1) \( \mathcal{F} \) fails because of some subgraph \( J_2 \) of \( S_2 \), and (2) there is an isomorphism between \( S_2 \) and a proper subgraph of \( S_1 \) under which the variable node set \( V(J_2) \) is mapped into the variable node set \( V(J_1) \).

**Example 2:** To elucidate the formulation of \( \mathcal{E}_1^s(\mathcal{F}) \), let us once again revisit the TBFA1. For the sake of convenience, let \( \mathcal{F} \) denote the TBFA1. Let \( S_1 \) be the Tanner graph shown in Fig. 4(a) and denote by \( J_1 \) the induced subgraph on the variable node set \( V(J_1) = \{ v_1, v_3, v_4, v_6 \} \). It is easy to check that \( \mathcal{F} \) fails because of the subgraph \( J_1 \) of \( S_1 \) as the set of corrupt variable nodes oscillates between \( \{ v_1, v_3, v_4, v_6 \} \) and \( \{ v_2, v_3, v_4, v_5 \} \) every two decoding iterations. Let \( I \) be the Tanner graph which is depicted in Fig. 4(c). Since \( J_1 \) is isomorphic to \( I \), it is clear that \( S_1 \in \mathcal{E}_1^s(\mathcal{F}) \).

Consider all non-isomorphic subgraphs of \( S_t \) which have five variable nodes and contain \( I \). There are two such subgraphs, which are depicted in Fig. 4(d) and (e). One can check easily that these subgraphs are not elements of \( \mathcal{E}_1(\mathcal{F}) \). Consequently, there cannot exist a Tanner graph \( S_2 \in \mathcal{E}_1(\mathcal{F}) \) which satisfies condition 2 of Definition 5. Therefore, \( S_1 \in \mathcal{E}_1^s(\mathcal{F}) \).

Now, let \( S_3 \) denote a Tanner graph which is identical to the Tanner graph shown in Fig. 2(a). As we have discussed before, the TBFA1 fails because of the subgraph \( J_3 \) of \( S_3 \), where \( V(J_3) = \{ v_1, v_3, v_4, v_6 \} \). Hence, \( S_3 \in \mathcal{E}_1(\mathcal{F}) \). Note that if we delete \( v_2 \) and \( v_7 \) from \( S_3 \), then \( S_1 \) is obtained. From the above discussion, it is clear that the Tanner graph \( S_1 \in \mathcal{E}_1^s(\mathcal{F}) \) satisfies the two conditions of Definition 5: (1) \( \mathcal{F} \) fails because of the subgraph \( J_1 \) of \( S_1 \), and (2) there is an isomorphism between \( S_1 \) and a proper subgraph of \( S_3 \) under which the variable node set \( V(J_1) \) is mapped into the variable node set \( V(J_3) \). Therefore, at this stage, we cannot conclude that \( S_3 \in \mathcal{E}_1^s(\mathcal{F}) \).

On the other hand, we also cannot conclude at this stage that \( S_3 \notin \mathcal{E}_1^s(\mathcal{F}) \). This is due to the possibility that \( \mathcal{F} \) also fails because of a different induced subgraph \( J_3' \) of \( S_3 \) that is isomorphic to \( I \). In such a case, if there does not exist a Tanner graph in \( \mathcal{E}_1(\mathcal{F}) \) that satisfies the two conditions of Definition 5, then \( S_3 \in \mathcal{E}_1^s(\mathcal{F}) \). Before concluding the example, let us also explore this scenario.

By simple inspections, it can be seen that there are 14 subsets of \( V(S_3) \) on which the induced subgraphs are isomorphic to \( I \). However, due to isomorphism, there are only four distinct cases which correspond to the following subsets: \( V(J_3) = \{ v_1, v_3, v_4, v_6 \} \), \( V(J_3') = \{ v_1, v_3, v_4, v_6 \} \), \( V(J_3'') = \{ v_1, v_3, v_4, v_7 \} \), and \( V(J_3''') = \{ v_1, v_3, v_4, v_7 \} \). It is easy to verify that \( \mathcal{F} \) does not fail on the subgraphs \( J_3', J_3'', J_3''', J_3''' \) of \( S_3 \). Consequently, we can now conclude that \( S_3 \notin \mathcal{E}_1^s(\mathcal{F}) \).

Example 2 not only elucidates the formulation of \( \mathcal{E}_1^s(\mathcal{F}) \), it also illustrates a critical simplification in our analysis. In particular, one can see that the Tanner graph set \( \mathcal{E}_1^s(\mathcal{F}) \) contains \( S_1 \) but not \( S_3 \). In other words, while \( S_1 \) is a Tanner graph of interest and is used to evaluate the performance of \( \mathcal{F} \), \( S_3 \) is omitted from the analysis. The justification for this simplification is as follows. The decodings of the algorithm \( \mathcal{F} \) on \( S_1 \) and \( S_3 \) are almost identical. The only difference is that in the decoding on \( S_3 \) one more variable node, namely \( v_7 \), is potentially corrupted. However, whether or not \( v_7 \) is corrupted does not affect the decoding outcome. Therefore, to understand the mechanism of decoding failure on \( S_1 \) and \( S_3 \), it is sufficient to only consider \( S_1 \). Most importantly, eliminating \( S_3 \), as well as many other graphs in \( \mathcal{E}_1(\mathcal{F}) \) that contain \( S_1 \), makes the analysis manageable as we show in this section.

**Remarks:** The above simplification does not affect the validity and usefulness of the analysis. For example, if the outcomes of the analysis are used in code construction, then it can be seen that the elimination of \( S_1 \) in a Tanner graph also leads to the elimination of \( S_3 \) as well as all subgraphs that contain \( S_1 \). If the outcomes of the analysis are used for designing better algorithms, then it can be seen that an algorithm must be designed to correct \( S_1 \) first before it can be designed to correct \( S_3 \). Due to the importance of graphs in \( \mathcal{E}_1(\mathcal{F}) \), we call them trapping sets of the TBFA1 algorithm \( \mathcal{F} \), as we now formally define.

**Definition 6:** If \( S \in \mathcal{E}_1^s(\mathcal{F}) \) then \( S \) is a trapping set of \( \mathcal{F} \). \( I \) is called an inducing set of \( S \). \( \mathcal{E}_1^s(\mathcal{F}) \) is called the trapping set profile with inducing set \( I \) of \( \mathcal{F} \).

An important property of a trapping set, which enables the construction of a trapping set profile, is stated in the following proposition.

**Proposition 2:** Let \( S \in \mathcal{E}_1^s(\mathcal{F}) \) be a trapping set of \( \mathcal{F} \) with inducing set \( I \). Then, there exists at least one induced subgraph \( J \) of \( S \) which satisfies the following properties: (1) \( J \) is isomorphic to \( I \), and (2) \( \mathcal{F} \) fails because of \( J \) of \( S \), and (3) Consider the decoding of \( \mathcal{F} \) on \( S \) with \( V(J) \) being the set of initially corrupt variable nodes. Then, for any variable node \( v \in V(S) \), there exist an integer \( l \) such that \( 0 \leq l \leq l_m \) and \( w_{vl} \in \{ 0, 1 \} \).

**Proof:** Let \( S \) be a trapping set of \( \mathcal{F} \) with inducing set \( I \). Then, \( S \in \mathcal{E}_1^s(\mathcal{F}) \subseteq \mathcal{E}_1(\mathcal{F}) \). Since \( S \in \mathcal{E}_1(\mathcal{F}) \), \( \mathcal{F} \) fails because of some induced subgraphs \( J_1, J_2, \ldots, J_q \) of \( S \) which is isomorphic to \( I \). All the subgraphs \( J_1, J_2, \ldots, J_q \) satisfy conditions 1 and 2 of Proposition 2. Consider the decoding of \( \mathcal{F} \) on \( S \) with \( V(J) \) being the set of initially corrupt variable nodes. Let \( n = |V(S)| \) and without loss of generality, assume that \( 0 \leq l \leq l_m \). Obviously, there exists a subgraph \( S' \) of \( S \) such that \( S' \) is obtained by deleting \( v_s \) from \( S \) and \( S' \) satisfies condition 1 and 2 of Definition 5. Consequently, if none of the subgraphs \( J_1, J_2, \ldots, J_q \) satisfies condition 3 of Proposition 2, then there always exists a subgraph \( S' \) that satisfies conditions 1 and 2 of Definition 5. Hence, \( S \in \mathcal{E}_1^s(\mathcal{F}) \), which is a contradiction.

From Proposition 2, one can see that the trapping set profile \( \mathcal{E}_1(\mathcal{F}) \) of \( \mathcal{F} \) contains the graphs that are most “compact.” We consider these graphs most compact because for at least one \( J \) isomorphic to \( I \), the decoding of \( \mathcal{F} \) on such a graph with \( V(J) \) being the set of initially corrupt variable nodes could be made successful by removing any variable node of the graph. This special property of trapping sets is the basis for an explicit recursive procedure to obtain all trapping sets up to a certain size, which compensates for the lack of a fully combinatorial characterization of trapping sets. We remark that for certain reasonably good algorithms, the necessary condition for a Tanner graph to be a trapping set can be easily derived. Before describing the recursive procedure for constructing trapping set profiles in the next subsection, we state the following proposition, which gives a sufficient
condition for the convergence of an algorithm $\mathcal{F}$ on a Tanner graph $G$.

**Proposition 3:** Consider decoding with an algorithm $\mathcal{F}$ on a Tanner graph $G$. Let $V_c$ be the set of initially corrupt variable nodes and $I$ be the induced subgraph on $V_c$. Then, algorithm $\mathcal{F}$ will converge to (a codeword) after at most $l_{1}^{\mathcal{F}}$ decoding iterations if there is no subset $V_s$ of $V(G)$ such that $V_s \supset V_c$ and the induced subgraph on $V_s$ is isomorphic to a graph in $\mathcal{E}_{1}^{\mathcal{F}}$.

**Proof:** Follows from the definition of $\mathcal{E}_{1}^{\mathcal{F}}$.

**Remark:** Proposition 3 only gives a sufficient condition because the existence of $V_s \subseteq V(G)$ which satisfies the above-mentioned conditions does not necessarily indicate that $G \in \mathcal{E}_{1}(\mathcal{F})$.

### B. The Construction of a Trapping Set Profile

The recursive procedure for constructing a trapping set profile $\mathcal{E}_{1}^{\mathcal{F}}(\mathcal{F})$ relies on Proposition 2. In particular, consider a trapping set $S$ with an inducing set $I$, Proposition 2 states that for at least one induced subgraph $J$ of $S$ which is isomorphic to $I$, in the decoding $\mathcal{F}$ on $S$ with $V(J)$ being the set of initially corrupt variable nodes, every variable node in $V(S)$ is corrupt at the end of some iteration. As a result, given the knowledge of $J$, it is possible to generate $S$ by simultaneously performing decoding and adding variable nodes to $J$ in one specific manner. Consequently, if we simultaneously perform decoding and add variable nodes to $I$ in all possible ways, then all trapping sets with inducing set $I$ can be generated. We now describe this procedure.

Let us assume that we are only interested in trapping sets with at most $n_{\text{max}}$ variable nodes. Consider the decoding of $\mathcal{F}$ on a Tanner graph $I$ with $V(I)$ being the set of initially corrupt variable nodes. Let $n_{1} = |V(I)|$. If $\mathcal{F}$ fails because of $I$ then $\mathcal{E}_{1}^{\mathcal{F}}(\mathcal{F}) = \{I\}$ and we have found the trapping set profile. If $\mathcal{F}$ does not fail because of $I$, then we expand $I$ by recursively adding variable nodes to $I$ until a trapping set is found. During this process, we only add variable nodes that become corrupt at the end of a certain iteration.

Consider all possible bipartite graphs obtained by adding one variable node, namely $v_{n_{1}+1}$, to the graph $I$ such that when the decoding is performed on these graphs with $V(I)$ being the set of initially corrupt variable nodes, the newly added variable node is a corrupt variable node at the end of the first iteration, i.e., $w_{1_{n_{1}+2}}^{I} \in \{1_{w}, 1_{s}\}$. Let $O_{I}$ denote the set of such graphs. Take one graph in $O_{I}$ and denote it by $U$. Then, there can be two different scenarios in this step: (1) $\mathcal{F}$ does not fail on the subgraph $I$ of $U$. In this case, $U$ is certainly not a trapping set and we put $U$ in a set of Tanner graphs denoted by $E_{1}^{I}$, (2) $\mathcal{F}$ fails because of the subgraph $I$ of $U$. In this case, $U$ is a potential trapping set and a test is carried out to determine if $U$ is indeed one. If $U$ is not a trapping set then it is discarded.

We complete the formation of $E_{1}^{I}$ by repeating the above step for all other graphs in $O_{I}$.

Let us now consider a graph $U \in E_{1}^{I}$. Again, we denote by $O_{U}$: the set of Tanner graphs obtained by adding one variable node, namely $v_{n_{1}+2}$, to the graph $U$ such that when the decoding is performed on these graphs with $V(I)$ being the set of initially corrupt variable nodes, the newly added variable node is a corrupt variable node at the end of the first iteration, i.e., $w_{1_{n_{1}+2}}^{U} \in \{1_{w}, 1_{s}\}$. It is important to note that the addition of variable node $v_{n_{1}+2}$, which is initially correct, cannot change the fact that variable node $v_{n_{1}+1}$ is also corrupt at the end of the first iteration. This is because the addition of correct variable nodes to a graph does not change the states of the existing check nodes and the decoding dynamic until the moment the newly added variable nodes get corrupted.

Similar to what has been discussed before, we now take a graph in $O_{U}$ and determine if it is a trapping set, or it is to be discarded, or it is a member of the set of Tanner graph $E_{2}^{I}$. By repeating this step for all other graphs in $E_{1}^{I}$, all graphs in $E_{2}^{I}$ can be enumerated. In a similar fashion, we obtain $E_{3}^{I}$, $E_{4}^{I}$, ..., $E_{n_{\text{max}}-n_{1}}^{I}$. For the sake of convenience, we also let $E_{n_{\text{max}}-n_{1}+1}^{I} = \{I\}$.

At this stage, we have considered one decoding iteration on $I$. It can be seen that if $S$ is a trapping set with at most $n_{\text{max}}$ variable nodes then either $S$ has been found, or $S$ must contain a graph in $\bigcup_{i=0}^{n_{\text{max}}-n_{1}-1} E_{i}$. Therefore, we proceed by expanding graphs in $E_{i} = \bigcup_{i=0}^{n_{\text{max}}-n_{1}-1} E_{i}$.

Let $K$ denote a Tanner graph in $E_{i} = \bigcup_{i=0}^{n_{\text{max}}-n_{1}-1} E_{i}$.

We now repeat the above graph expanding process starting from $K$. Specifically, we first obtain $O_{K}$ which is defined as the set of all Tanner graphs obtained by adding one variable node $v_{n_{K}+1}$ to the graph $K$ such that when the decoding is performed on these graphs with $V(I)$ being the set of initially corrupt variable nodes, the newly added variable node is a corrupt variable node at the end of the second iteration, but not a corrupt variable node at the end of the first iteration, i.e., $w_{1_{n_{K}+2}}^{K} \in \{0_{w}, 0_{s}\}$ and $w_{2_{n_{K}+2}}^{K} \in \{1_{w}, 1_{s}\}$. The graphs in $O_{K}$ that are not trapping sets are either discarded or form the set $E_{K}$. By iteratively adding variable nodes, we enumerate all graphs in $E_{2}^{K}, E_{3}^{K}, ..., E_{n_{\text{max}}-n_{1}}^{K}$.

One can see that there are two different recursive algorithms. The first algorithm enumerates graphs in $E_{K} = \bigcup_{i=0}^{n_{\text{max}}-n_{1}} E_{K}$ for a given graph $K$ by recursively adding variable nodes. The second algorithm recursively calls the first algorithm to enumerate graphs in $E_{K} = \bigcup_{i=0}^{n_{\text{max}}-n_{1}} E_{K}$ for each graph $K$ in $E_{l} = \bigcup_{i=0}^{n_{\text{max}}-n_{1}-1} E_{l}$. Each recursion of the second algorithm corresponds to a decoding algorithm. As a result, the trapping set profile is obtained after $l_{1}^{\mathcal{F}}$ recursions of the second algorithm. The pseudocodes of the two algorithms, which we call RA1 and RA2, can be found in [22]. The interested readers are referred to [22] for an example which demonstrates the operations of these recursive algorithms.

**Example 3:** In this example, we demonstrate that the trapping sets with fewest number of variable nodes need not be absorbing sets or fixed sets, thereby again justifying the necessity of our analysis. Consider the Tanner graph $I$ that is depicted in Fig. 4(b). There are 7 trapping sets of the TBFA2 with inducing set $I$ and 9 variable nodes. These are trapping sets with fewest number of variable nodes. None of these trapping sets is an absorbing set or a fixed set. One such trapping set is depicted in Fig. 4(f).

In the enumeration of trapping sets using the RA1 and RA2, it is necessary that $\mathcal{F}$ fails because of the subgraph $I$ of $K$ for $K$ to be a trapping set. We remark that this condition,

---

1 More details are to be provided later in this paper.
which can see that if to determine whether \( K \) is a trapping set, one can check, among the set of enumerated graphs, for the existence of one which contains less variable nodes than \( K \) and which satisfies condition 2) of Definition 5. This can be done by deleting certain variable nodes of \( K \) to obtain a smaller graph \( K' \), and then checking for the existence of an isomorphism between \( K' \) and an enumerated graph under which \( V(I) \) is mapped into \( V(I) \). Let \( n^{\min} \) be the smallest number of variable nodes of a Tanner graph in \( \mathcal{E}_I^r(\mathcal{F}) \). One can see that if \( n^{\max} - n^{\min} \) is relatively small, then the check to determine whether \( K \) is a trapping set is fairly simple. Moreover, note that if we set \( n^{\max} = n^{\min} \), then the checking step for a trapping set can be completely eliminated. This is because graphs with \( n^{\min} \) variable nodes cannot contain a smaller trapping set.

V. APPLICATION OF THE FAILURE ANALYSIS IN THE SELECTION OF TBF ALGORITHMS

The failure analysis presented in the previous section enables the enumeration of all relevant uncorrectable error patterns for a given TBF algorithm. Consequently, the selection of one good TBF algorithm becomes simple. Nevertheless, even if the best TBF algorithm can now be found, its error performance might not be attractive enough for certain applications. In such a scenario, because of the simplicity and high throughput of bit flipping decoding, it is natural to consider the use of multiple TBFAs operating in a complimentary manner. In the following discussion, we introduce the concept of collective error correction and emphasize on the ease of selecting complimentary TBF algorithms.

Let us consider a collection \( \mathcal{A} \) of (general) iterative decoding algorithms for LDPC codes. Assume for a moment that the set of all uncorrectable error patterns for each and every algorithm in \( \mathcal{A} \) is known. More precisely, in the context of LDPC codes, we assume that all the induced subgraphs on such error patterns can be enumerated for each decoding algorithm. This naturally suggests the use of a decoder \( \mathcal{D} \) which employs multiple algorithms drawn from \( \mathcal{A} \). The basis for this use of multiple algorithms is rather simple: If different algorithms are capable of correcting different error patterns, then a decoder employing a set of properly selected algorithms can achieve provably better error performance than any single-algorithm decoder. Although the above assumption is not valid for most iterative algorithms, it is valid for the TBF algorithms defined in this paper. This is because the notion of trapping set of a TBF algorithm help determine whether or not an arbitrary error pattern is correctable. The concept and explicit construction of trapping set profiles allow rigorous selections of multiple algorithms which can collectively correct a fixed number of errors with high probability. In particular, a decoder employing algorithms with diverse trapping set profiles is capable of correcting a wide range of error configurations, hence possesses desirable error correction capability. Although the selection of algorithms can be code independent, any knowledge on the Tanner graph can be utilized to refine the selection.

The inputs to the process of selecting a group of TBF algorithms are: (1) A large collection \( \mathcal{A} \) of TBF algorithms. \( \mathcal{A} \) can be the set of all possible algorithms or it can be a subset of algorithms that satisfies certain constraints. (2) A set of Tanner graphs with a small number of variable nodes. These are all possible subgraphs induced on the set of initially corrupt variable nodes. (3) Optional knowledge on the Tanner graph on which the decoder operates.

The main element of the selection process is the enumeration of trapping sets for all input algorithms to generate their trapping set profiles. Once the trapping set profiles of all input algorithms are generated, they will be used as inputs to the algorithm selector. The algorithm selector outputs a set of algorithms drawn from \( \mathcal{A} \) with diverse trapping set profiles. By cycling though these algorithms, or operating them in parallel, it is possible to overcome the most commonly-occurring error configurations, thereby improving the error performance.

A. Selecting One TBF Algorithm

We first discuss an important criterion to select one algorithm among all possible algorithms. Let \( n^{\min}_{I,F} \) be the smallest number of variable nodes of Tanner graphs in \( \mathcal{E}_I^r(\mathcal{F}) \). Then, one should select an algorithm \( \mathcal{F} \) such that \( n^{\min}_{I,F} \) is maximized.
To justify this selection criterion, we rely on results in extremal graph theory. In particular, we are interested in the probability that a bipartite graph $G$ has $S$ as its subgraph, where $G$ is drawn randomly from the $(d_v, d_e)$-regular graph ensemble and $S$ is a $d_v$-left-regular Tanner graph. Characterizing the probability that a graph has another graph as its subgraph is a heavily studied problem. A nice survey of results on this problem is given in [23].

Let $n_G, n_S$ denote the number of variable nodes in $G$ and $S$, respectively. Similarly, let $m_G, m_S$ denote the number of check nodes. Since $S$ is only left-regular, we denote by $\rho = (\rho_1, \rho_2, \ldots, \rho_{m_S})$ the degree sequence of check nodes of $S$ and let $\rho_{\max}$ be the maximum value in $\rho$. The probability that $G$ has $S$ as its subgraph, denoted by $P(S \supset G)$, can be evaluated using [23, Theorem 2.2] as follows.

$$P(S \supset G) = \frac{(d_v!)^{n_S} \left[ (n - n_S) d_v! \right] \prod_{i=1}^{m_S} h(\rho_i)}{(n d_v)!} \times \left[ 1 + O \left( \frac{\Lambda n_S}{n} \right) \right],$$  

where $\Lambda = (d_v + d_e)(2d_e + d_v + \rho_{\max}), h(\rho_i) = d_v(d_v - 1) \ldots (d_v - \rho_i + 1)$, and $O(\Lambda n_S/n)$ is an error term which vanishes when $n \to \infty$.

It can be seen that if the parameters associated with $G$ are fixed, then $P(S \supset G)$ only depends on $n_S$ and the check degree sequence $\rho$. Notice that since $S$ contains $n_S d_v$ edges, the product $\prod_{i=1}^{m_S} h(\rho_i)$ is constituted of $n_S d_v$ factors that are members of the set $\{(d_v - \rho_{\max} + 1), (d_v - \rho_{\max} + 2), \ldots, d_v\}$. Consequently, $d_v^{n_S d_v} \leq \prod_{i=1}^{m_S} h(\rho_i) \leq (d_v - \rho_{\max} + 1)^{n_S d_v}$.

Now, one can see that for sufficiently large values of $n$ and sufficiently small values of $d_v, d_e, n_S$ and $\rho_{\max}$, the probability $P(S \supset G)$ is mostly influenced by $n_S$. In particular, the term $\left[ (n - n_S) d_v! \right]$ is most significant. In such a case, $P(S \supset G)$ decreases when $n_S$ increases. In the context of this paper, $d_v, d_e$ are fixed. Besides, we observe from the enumeration of trapping sets for various algorithms that $n_S$ and $\rho_{\max}$ are usually small. Therefore, it is safe to assume that given two $d_v$-left-regular Tanner graphs $S_1, S_2$ with $0 < |V(S_1)| < |V(S_2)| < |V(G)|$, the probability that $G$ contains $S_2$ is less than the probability that $G$ contains $S_1$.

From the above argument, it is clear that algorithms which fail on larger trapping sets are more desirable. One can also deduce that the larger the number $|V(S)|$ of a given trapping set $S$ is, the easier it would be (if at all possible) to construct a Tanner graph $G$ that does not contain $S$. Therefore, a larger $n_{\min}^{\rho,s}$ means that the sufficient condition for the convergence of $\mathcal{F}$ can be met with higher probability. If for two algorithms $\mathcal{F}_1$ and $\mathcal{F}_2$, $n_{\min}^{\rho,s,\mathcal{F}_1} = n_{\min}^{\rho,s,\mathcal{F}_2}$, then it is also possible to derive other comparison criteria based on $\delta^0_1(\mathcal{F}_1)$ and $\delta^0_1(\mathcal{F}_2)$, and/or compare $\mathcal{F}_1$ and $\mathcal{F}_2$ with a different assumption of $I$. For example, the probability of a graph $G$ containing a trapping set $S$ can be also be evaluated based on $\rho$.

### B. Selecting Multiple TBF Algorithms

We now consider the problem of selecting multiple algorithms. The basis for this selection is that one should select good individual algorithms with diverse trapping set profiles. In this paper, we only consider a decoder $D$ with algorithms $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_p$ operating in parallel, i.e., the received vector of the channel is the input vector for all algorithms. Note that one can also use trapping set profiles to select algorithms that operate in serial, i.e., the output from one algorithm is the input to another. For a decoder $D$ that employs parallel algorithms, the concept of trapping sets and trapping set profiles can be defined in the same manner as trapping sets and trapping set profiles for a single TBF algorithm. One can easily modify the recursive procedures given in Section IV-B to generate trapping set profiles of the decoder $D$. Then, $D$ can be designed with the same criterion discussed in the previous subsection.

### VI. Numerical Results

In this section, we give examples of selecting TBF algorithms and demonstrate the frame error rate (FER) performance of decoders employing these algorithms. For simplicity, we assume the use of 3-left-regular codes with girth $g = 8$. We also let $\Delta_v = (0, 1, 1)$, $\Delta_e = (0, 1, 1, 0)$ and $\ell^w_v = 30$ for all algorithms. Our algorithms are compared with the Gallager A/B algorithm, the 6-bit precision offset min-sum algorithm and the SPA. Note that since the performance of the TBFA is much worse than that of the Gallager A/B algorithm, we do not include its performance for the sake of simplicity. In these comparisons, the Gallager A/B algorithm, the offset min-sum algorithm and the SPA iterate for a maximum of 100 iterations. In the SPA, messages have floating-point precision. The comparisons are made on two codes: (1) $C_1$: The popular (155, 64) Tanner code [24]. $C_1$ has $d_v = 3$, $d_e = 5$, rate $R = 0.4129$ and minimum distance $d_{\min} = 20$; (2) $C_2$: A quasi-cyclic code of length $n = 732$, rate $R = 0.7527$ and minimum distance $d_{\min} = 12$ [2]. Note that the BSC cross-over probabilities that correspond to the capacity of 0.4129 and 0.7527 are $\alpha = 0.141$ and $\alpha = 0.041$, respectively. As previously mentioned, only a subset $\mathcal{F}$ of TBF algorithms should be considered. The constraints on TBF algorithms that define $\mathcal{F}$ are constraints on the images of the variable node update functions $f$. We are considering a set of 41, 472, 000 algorithms that satisfy a specific set of constraints. All the algorithms given in this paper and the constraints on them can be found at [25].

By Proposition 1, it is expected that a single algorithm can guarantee the correction of 3 errors in a girth-8 3-left-regular LDPC code. Let $I_3$ be the set of all Tanner graphs with girth $g \geq 8$ and three variable nodes. We generate the trapping set profiles $\delta^0(F)$ for all $F \in \mathcal{F}$, and all $I \in I_3$. There is a set $\mathcal{F}^3_3$ of 358, 851 algorithms for which $\delta^0(F) = \emptyset$ for all $I \in I_3$. Such an algorithm can guarantee to correct all weight-three error patterns. Our next step is to select an algorithm which is most expected to correct a weight-four error pattern. Let $I_4$ be the set of all Tanner graphs with girth $g \geq 8$ and four variable nodes. Then, $|I_4| = 10$. We generate the trapping set profiles $\delta^0(F)$ for all $F \in \mathcal{F}^3_3$ and all $I \in I_4$. The largest among the smallest sizes of trapping sets in $\delta^0(F)$ for all $F \in \mathcal{F}^3_3$ are $(10, 8, 8, 7, 7, \infty, \infty, \infty, 6, 4)$. There are six algorithms, denoted as $F_{A1}, F_{A2}, \ldots, F_{A6}$ for which the smallest sizes of
trapping sets in $\mathcal{E}_1^r(F)$ are $(10, 8, 5, 6, \infty, \infty, \infty, 5, 4)$. The smallest sizes of trapping sets in $\mathcal{E}_1^r(F)$ for these algorithms overlap the most with $\max_i(n_{1i}^{\min}, n_{2i}^{\min}, \ldots, n_{10i}^{\min})$. Hence, these algorithms have the highest probability to correct a random error pattern of weight-four. Let $D_1$ denote a decoder which uses $F_{A1}$. The FER performance of $D_1$ on $C_1$ and $C_2$ are shown in Fig. 5(a) and Fig. 6, respectively. It can be seen that $D_1$, which operates with a maximum of only 30 iterations, outperforms the Gallager A/B algorithm in both codes.

Let $D'_1$ denote a decoder which uses $F_{A1}, F_{A2}, \ldots, F_{A6}$ in parallel. Fig. 5(b) shows the performance of $D'_1$ in comparison with $D_1$. One can observe that there is no gain in the FER performance of $D'_1$. Simply using in parallel a group of good algorithms does not necessarily result in better error correction capability. We shall now give an example on how to properly selecting a collection of algorithms. Assume that $F_{A1}$ is already in use. We would like to select other TBF algorithms to complement $F_{A1}$. In particular, we need to select more TBF algorithms to complement $F_{A1}$ in correcting weight-four error patterns. From the set of trapping set profiles for all algorithms in $\mathcal{F}$, we can select three algorithms, denoted as $F_{B1}, F_{B2}$ and $F_{B3}$. The smallest sizes of trapping sets in $\mathcal{E}_1^r(F)$ for these algorithms are $(8, 8, 7, 8, 8, 9, \infty, 8, 4), (9, 8, 8, 6, 6, \infty, 11, \infty, 9, 4)$ and $(4, 4, 4, 4, 4, 4, 4, 4, 4, \infty)$, respectively. Note that these three algorithms are not capable of correcting all weight-three error patterns. Let $D'_2$ denote a decoder which uses $F_{A1}$ and $F_{B1}$ in parallel. Let $D''_2$ denote a decoder which uses $F_{A1}, F_{B1}, F_{B2}$ in parallel. Also let $D_2$ denote a decoder which uses $F_{A1}, F_{B1}, F_{B2}, F_{B3}$ in parallel. The FER performance of $D_2$, $D'_2$ and $D''_2$ on $C_1$ in comparison with $D_1$ are shown in Fig. 5(b). The FER performance of $D_2$ on $C_1$ and $C_2$ are also shown in Fig. 5(a) and Fig. 6, respectively. One can observe the gradual improvement in the FER performance when more algorithms are used.

As previously mentioned, knowledge on the Tanner graph can be useful in the selection of algorithms. By using the method of searching for subgraphs proposed in [2], one can conclude that the Tanner graph of $C_1$ is free of the subgraphs listed in Fig. 7. After deleting irrelevant trapping sets (those that contain a Tanner graph listed in Fig. 7 as a subgraph) from the trapping set profiles, we can select a set of four TBF algorithms that are used by a decoder $D_3$. The FER performance of $D_3$ on $C_1$ and $C_2$ are shown in Fig. 5(a) and Fig. 6, respectively. $D_3$’s performance on $C_1$ is also shown in Fig. 5(b) to facilitate the comparison. One can observe that $D_3$ outperforms $D_2$ on $C_1$. Note that although $D_3$ also outperforms $D_2$ on $C_2$, this fact is not expected to be typical.

Finally, we construct all trapping set profiles with inducing sets containing four, five and six variable nodes for each algorithm in $\mathcal{F}$. Note that there are 10 possible inducing sets (Tanner graphs with girth $g \geq 8$) containing four variable nodes, 24 possible inducing sets containing five variable nodes and 57 possible inducing sets containing six variable nodes. Hence, for each algorithm, we construct a total of 92 trapping set profiles. From the trapping set profiles of all algorithms, we select a collection of 35 algorithms. Then, we simulate the performance of a decoder $D_4$ which employs these algorithms in parallel. The maximum total number of iterations of $D_4$ is $35 \times 30 = 1050$. Fig. 5(a) shows the FER performance of $D_4$ on $C_1$. It can be seen that the FER performance of $D_4$ approaches (and might surpass) that of the SPA in the error floor region. It is also important to note that $D_4$ can correct any error pattern up to weight 5 on $C_1$. Fig. 6 also shows the FER performance of $D_4$ on $C_2$. It can be seen that the slope of the FER curve of $D_4$ in the error floor region is higher than that of the SPA. This indicates that the FER performance of $D_4$ might eventually surpass that of the SPA. Finally, we remark that the slope of the FER curve of $D_4$ in the error floor region is between 5 and 6, which indicates that $D_4$ can
correct error patterns of weight 4 and 5 with high probability. This is why in our simulation, no weight-four error pattern that leads to decoding failure of $D_4$ was observed.

VII. CONCLUSIONS

We proposed a novel class of bit flipping algorithms for LDPC codes. More importantly, we gave a framework for the analysis and selection of algorithms that can together correct a fixed number of errors with high probability. Since in TBF algorithms variable nodes take more than two states, it would be interesting to consider the use of the proposed algorithms on channels which output more than two signal levels. It is also interesting to asymptotically analyze multi-bit bit flipping algorithms. These are left as our future works.

REFERENCES


Bane Vu Nguyen (S’92–M’14) received the B.S. degree and the Ph.D. degree in electrical engineering from the University of Arizona, Tucson, AZ, in 2007 and in 2012, respectively. He is currently an employee of Marvell Semiconductor Inc, Santa Clara, CA. His research interests include digital communications and information theory.

Dung Viet Nguyen (S’92–M’93–SM’02–F’11) is a Professor of Electrical and Computer Engineering and Mathematics at the University of Arizona. He is affiliated with BIO5, the Institute for Collaborative Biomedical Research, and is a Director of the Error Correction Laboratory.

Dr. Vasic is an inventor of the soft error-event decoding algorithm, and the key architect of a detector/decoder for Bell Labs magnetic recording read channel chips which were regarded as the best in industry. Different variants of this algorithm were implemented in virtually all magnetic hard drives. His pioneering work on structured low-density parity-check (LDPC) error correcting codes and invention of codes has enabled low-complexity iterative decoder implementations. Structured LDPC codes are today adopted in a number of communications standards, and are used in extremely high-density magnetic hard drives. Dr. Vasic’s theoretical work in error correction coding theory and codes on graphs has led to analytical characterization of the hard decision iterative decoders of low-density parity check (LDPC) codes, and design of codes and decoders with best error-floor performance known today.

His editorial and conference organizing responsibilities include: 2003 IEEE Communication Theory Workshop, 2004 DIMACS Workgroup and Workshop on Theoretical Advances in Information Recording, 2004 Los Alamos Workshop on Applications of Statistical Physics to Coding Theory, Communication Theory Symposium of the 2006 ICC, and 2007 Communication Theory Workshop. Dr. Vasic currently serves as a Member of the Editorial Board of the IEEE TRANSACTIONS ON MAGNETICS and Guest Editor of the JSAC Special Issue on Data Communication Techniques for Storage Channels and Networks. He is an IEEE Fellow.