Failure Analysis of Two-Bit Flipping Decoding Algorithms

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

Dung Viet Nguyen
Marvell Semiconductor Inc.
University of Arizona
Santa Clara, CA 95054 USA
Email: nguyendv@ece.arizona.edu

***Abstract*—We consider a class of bit flipping algorithms for low-density parity-check codes over the binary symmetric channel in which one additional bit at a variable and check nodes is employed. For these two-bit flipping algorithms, we give and illustrate through examples a recursive procedure for finding all uncorrectable error patters and corresponding induced subgraphs, referred as a trapping set profile. This procedure is used to select a small collection of good algorithms that in a decoding diversity approach, run in parallel or serial, outperform Gallager A/B, min-sum and sum product algorithm in the error floor region.**

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

I. Introduction

In many applications such as flash memory, fiber and free-space optical communications, due to high data transmission speed, a decoder can utilize only hard decisions from the channel. In addition, power consumption constraints may dictate using simple decoding algorithms. The simplest and fastest algorithms for decoding low-density parity-check (LDPC) codes on the binary symmetric channel (BSC), are bit flipping algorithms. While bit flipping is shown to be able to correct a number of errors linear in code length on several LDPC code ensembles, [1]–[3], it does not perform well on short and medium length codes. An extreme case are column-weight-three codes for which the guaranteed error correction capability is upper-bounded by $\lceil g/4 \rceil - 1$, where g is the girth of a code [4], resulting in the fact for $g = 6$ or $g = 8$ not even all weight-two error patterns can be corrected. It was believed that bit flipping is inferior to Gallager A/B algorithm, until Nguyen and Vasic have introduced a class of bit flipping algorithms [5] capable of surpassing Gallager A/B and even the sum-product algorithm (SPA). Unlike other recently proposed bit-flipping algorithms (for a comprehensive list of references the reader is referred to a journal version of this paper [5]) our algorithm does not use any soft information from the channel. It uses two bits to represent the state of variable and check nodes. Hence the name - two-bit flipping (TBF) algorithm. In [5] we gave a systematic procedure for selecting a set of TBF algorithms which can collectively correct more error patterns than any single TBF algorithm, and gave a framework for determining uncorrectable error patterns. In this paper we provide the detailed analysis of decoding failures of the TBF algorithms on column weight three LDPC codes.

As any other sub-optimal decoder operating on a Tanner graph of a code, the TBF also fails to converge due to the presence of certain small subgraphs in the Tanner graph, referred to as "trapping sets." We define a trapping set for a given algorithm so that it provides sufficient conditions for decoding convergence. The set of all possible trapping sets of a given decoding algorithm is referred to as trapping set profile of a decoder. We analyze the recursive procedure, to determine a trapping set profile of any TBF algorithm.

The rest of the paper is organized as follows. In Section II we introduce the TBF update rules, provide theorems establishing sufficient conditions for a decoding failure and introduce a notion of a trapping set profile of a given TBF. In Section III we present two recursive algorithms **RA1** and **RA2** used to enumerate graphs in a trapping set profile. We also give a detailed example elucidating intuition behind these algorithms. Section IV concludes the paper.

II. Two-Bit Flipping Algorithms

In this paper, we consider $(d_{\rm v}, d_{\rm c})$-regular LDPC codes given by the parity check matrix H. The length of the shortest cycle in the Tanner graph G is called the girth g of G. Denote by $\mathbf{x}$ the transmitted codeword. Consider an iterative decoder and let $\hat{\mathbf{x}}^l = (\hat{x}_1^l, \hat{x}_2^l, \ldots, \hat{x}_n^l)$ be the decision vector after the lth iteration, where l is a positive integer. At the end of the lth iteration, a variable node is said to be *corrupt* if $\hat{x}_v^l \neq x_v$, otherwise it is *correct*. Let $\mathbf{s}^l = (s_1^l, s_2^l, \ldots, s_m^l)$ denote the syndrome vector of the decision vector after the lth iteration, i.e., $\mathbf{s}^l = \hat{\mathbf{x}}^l H^{\rm T}$. Also let $\mathbf{s}^0 = \mathbf{y}H^{\rm T}$ be the syndrome vector of the channel output vector $\mathbf{y}$. A check node c is said to be satisfied at the beginning of the lth iteration if $s_c^{l-1} = 0$, otherwise it is unsatisfied.

In [5] we have introduced a class of bit flipping algorithms in which both variable nodes as well as check nodes have four different states and thus require two bits to represent them. A variable node v in addition to its value zero or one, can be also either *strong* or *weak*. We use $0_{\rm s}$, $1_{\rm s}$, $0_{\rm w}$ and $1_{\rm w}$ to denote these states. The set of possible states of a variable node is thus $\mathcal{A}_{\rm v}$, i.e., $\mathcal{A}_{\rm v} = \{0_{\rm s}, 0_{\rm w}, 1_{\rm w}, 1_{\rm s}\}$.

We call a satisfied check *previously satisfied* if it was satisfied in the previous iteration, otherwise it is called *newly satisfied*. We define *previously unsatisfied* and *newly unsatisfied* similarly, and use 0_p, 0_n, 1_p and 1_n to denote these states. The set of possible states of a check node is thus $\mathcal{A}_\mathrm{c}$, i.e., $\mathcal{A}_\mathrm{c} = \{0_\mathrm{p}, 0_\mathrm{n}, 1_\mathrm{p}, 1_\mathrm{n}\}$.

The class $\mathscr{F}$ of TBF algorithms is given in Algorithm 1, where the vector $\mathbf{z}^l = (z_1^l, z_2^l, \ldots, z_m^l)$ gives the states of the check nodes at the beginning of the lth iteration, while the vector $\mathbf{w}^l = (w_1^l, w_2^l, \ldots, w_n^l)$ gives the states of the variable nodes at the end of the lth iteration.

The state w_v^0 of a variable node v is initialized to Δ_v^0 if $y_v = 0$ and to Δ_v^1 if $y_v = 1$. The state z_c^1 of a check node c is initialized to Δ_c^0 if $s_c^0 = 0$ and to Δ_c^1 otherwise. $\Delta_\mathrm{v} = (\Delta_\mathrm{v}^0, \Delta_\mathrm{v}^1)$ and $\Delta_\mathrm{c} = (\Delta_\mathrm{c}^0, \Delta_\mathrm{c}^1)$ take their values from the sets $\{(0_\mathrm{s}, 1_\mathrm{s}), (0_\mathrm{w}, 1_\mathrm{w})\}$ and $\{(0_\mathrm{p}, 1_\mathrm{p}), (0_\mathrm{n}, 1_\mathrm{n})\}$, respectively. A TBF algorithm $\mathcal{F} = (f, L, \Delta_\mathrm{v}, \Delta_\mathrm{c})$ iteratively updates $\mathbf{z}^l$ and $\mathbf{w}^l$ until all check nodes are satisfied or until a maximum number of iteration L is reached. The check node update function $\Phi : \{0,1\}^2 \to \mathcal{A}_\mathrm{c}$ is defined as follows: $\Phi(0,0) = 0_\mathrm{p}, \Phi(0,1) = 1_\mathrm{n}, \Phi(1,0) = 0_\mathrm{n}$ and $\Phi(1,1) = 1_\mathrm{p}$. The variable node update is specified by a function $f : \mathcal{A}_\mathrm{v} \times \Xi_{d_\mathrm{v}} \to \mathcal{A}_\mathrm{v}$, where Ξ_{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 $\sum_i \xi_i = d_\mathrm{v}$. In Algorithm 1, $\chi_{0_\mathrm{p}}^l(v), \chi_{0_\mathrm{n}}^l(v), \chi_{1_\mathrm{p}}^l(v)$ and $\chi_{1_\mathrm{n}}^l(v)$ give the number of check nodes with states $z_c^l = 0_\mathrm{p}, 0_\mathrm{n}, 1_\mathrm{p}$ and 1_n, respectively, that are connected to v. The function $f : \mathcal{A}_\mathrm{v} \times \Xi_3 \to \mathcal{A}_\mathrm{v}$ which defines the transition of a variable node from one state to another, must satisfy the conditions of symmetry and irreducibility defined in [5]. An example of f is given in Eq. 1, where $f_1 : \mathcal{A}_\mathrm{v} \times \{0,1,2,3\} \to \mathcal{A}_\mathrm{v}$ is the function defined in Table I, where $\chi_\mathrm{u}^l(v) \in \{0, 1, \ldots, d_\mathrm{v}\}$ denote the number of unsatisfied check nodes that are connected to the variable node v at the beginning of the lth iteration.

Algorithm 1 TBF Algorithm

$\forall v:\ w_v^0 \leftarrow \Delta_\mathrm{v}^{y_v}$, $\forall c:\ z_c^1 \leftarrow \Delta_\mathrm{c}^{s_c^0}$, $l \leftarrow 1$
while $\mathbf{s}^l \neq \mathbf{0}$ and $l < l_{\mathcal{F}}^\mathrm{m}$ **do**
  $\forall v:\ w_v^l \leftarrow f(w_v^{l-1}, \chi_{0_\mathrm{p}}^l(v), \chi_{0_\mathrm{n}}^l(v), \chi_{1_\mathrm{p}}^l(v), \chi_{1_\mathrm{n}}^l(v))$;
  $\forall c:\ z_c^{l+1} \leftarrow \Phi(s_c^{l-1}, s_c^l)$;
  $l \leftarrow l + 1$;
end while

TABLE I
$f_1 : \mathcal{A}_\mathrm{v} \times \{0,1,2,3\} \to \mathcal{A}_\mathrm{v}$

w_v^l	$\chi_\mathrm{u}^l(v)$				w_v^l	$\chi_\mathrm{u}^l(v)$			
	0	1	2	3		0	1	2	3
0_s	0_s	0_s	0_w	1_s	1_w	1_s	0_w	0_s	0_s
0_w	0_s	1_w	1_s	1_s	1_s	1_s	1_s	1_w	0_s

A. Good TBF Algorithms

Not all update rules lead to good decoding algorithms. We consider a subset of TBF algorithms satisfying certain constraints on on the images of the variable node update functions f. $\mathscr{F}$ denotes the set of these TBF algorithms.

The above mentioned constraints are given in Table II, which gives all possible values of $(f(0_\mathrm{s}, \xi), f(0_\mathrm{w}, \xi))$ for all ξ. To conserve space, we sort the tuples ξ in Ξ_3 in lexicographic order with the "$<$" relation and index these tuples from $i = 1$ to $|\Xi_3| = 20$. Then, an ordered tuple $\xi_i = (\chi_{0_\mathrm{p}}^l(v), \chi_{0_\mathrm{n}}^l(v), \chi_{1_\mathrm{p}}^l(v), \chi_{1_\mathrm{n}}^l(v))$ can be conveniently represented by its lexicographic index i.

TABLE II
POSSIBLE VALUES OF $f(0_\mathrm{s}, \xi)$ AND $f(0_\mathrm{w}, \xi)$.

i	$(f(0_\mathrm{s}, \xi_i), f(0_\mathrm{w}, \xi_i))$	Total
$1, 3$	$(0_\mathrm{w}, 1_\mathrm{w}), (0_\mathrm{w}, 1_\mathrm{s}), (1_\mathrm{w}, 1_\mathrm{w}), (1_\mathrm{w}, 1_\mathrm{s}), (1_\mathrm{s}, 1_\mathrm{s})$	5
$2, 4$	$(1_\mathrm{w}, 1_\mathrm{s}), (1_\mathrm{s}, 1_\mathrm{s})$	2
$5, 7, 11$	$(0_\mathrm{w}, 0_\mathrm{w}), (0_\mathrm{w}, 1_\mathrm{w}), (1_\mathrm{w}, 1_\mathrm{w})$	3
6	$(0_\mathrm{s}, 0_\mathrm{w}), (0_\mathrm{s}, 1_\mathrm{w}), (0_\mathrm{w}, 0_\mathrm{w}), (0_\mathrm{w}, 1_\mathrm{w}), (1_\mathrm{w}, 1_\mathrm{w})$	5
8	$(0_\mathrm{s}, 0_\mathrm{s}), (0_\mathrm{s}, 0_\mathrm{w}), (0_\mathrm{w}, 0_\mathrm{w})$	3
9	$(0_\mathrm{w}, 0_\mathrm{w})$	1
$10, 16, 20$	$(0_\mathrm{s}, 0_\mathrm{s})$	1
12	$(0_\mathrm{w}, 1_\mathrm{s}), (0_\mathrm{w}, 1_\mathrm{s}), (1_\mathrm{w}, 1_\mathrm{w}), (1_\mathrm{w}, 1_\mathrm{s})$	4
13	$(0_\mathrm{w}, 1_\mathrm{w}), (1_\mathrm{w}, 1_\mathrm{w})$	2
14	$(0_\mathrm{w}, 0_\mathrm{w}), (0_\mathrm{w}, 1_\mathrm{w})$	2
$15, 18$	$(0_\mathrm{s}, 0_\mathrm{w}), (0_\mathrm{s}, 1_\mathrm{w}), (0_\mathrm{w}, 0_\mathrm{w}), (0_\mathrm{w}, 1_\mathrm{w})$	4
17	$(0_\mathrm{s}, 0_\mathrm{w}), (0_\mathrm{w}, 0_\mathrm{w})$	2
19	$(0_\mathrm{s}, 0_\mathrm{s}), (0_\mathrm{s}, 0_\mathrm{w})$	2

It can be seen that the total number of algorithms in consideration is $|\mathscr{F}_\mathrm{r}| = \prod_i |\{(f(0_\mathrm{s}, \xi), f(0_\mathrm{w}, \xi))\}| = 41,472,000$. For each i, we index the elements in $\{(f(0_\mathrm{s}, \xi_i), f(0_\mathrm{w}, \xi_i))\}$ from 1 to $|\{(f(0_\mathrm{s}, \xi_i), f(0_\mathrm{w}, \xi_i))\}|$ according to their orders in Table II. Then, an algorithm $\mathcal{F} \in \mathscr{F}_\mathrm{r}$ can be represented by an ordered tuple $(o_1, o_2, \ldots, o_{20})$, where o_i is the index of $(f(0_\mathrm{s}, \xi_i), f(0_\mathrm{w}, \xi_i))$. We sort these tuples in lexicographic ordering and assign to each an index from 1 to $41,472,000$. Consequently, each algorithm in $\mathscr{F}$ now corresponds to a number between 1 and $41,472,000$.

To select a good algorithm from this huge number of possibly good algorithms, it is necessary to establish uncorrectable error patterns for different choices of the variable node update functions f.

III. Failure Analysis of TBF Algorithms

The main components of the failure analysis include the notion of trapping sets and trapping set profiles, and the procedure for finding a trapping set profile. We begin by defining trapping sets and decoding failures.

Definition 1 ([6]): For the channel output vector $\mathbf{y}$, let $\mathbf{F}(\mathbf{y})$ denote the set of variable nodes that are not eventually correct. For transmission over the BSC, $\mathbf{y}'$ is a fixed point of the decoding algorithm if and only if there exists a positive integer l_f such that $\mathrm{supp}(\mathbf{y}') = \mathrm{supp}(\hat{\mathbf{x}}^l)$ for all $l \geq l_\mathrm{f}$. If $\mathbf{F}(\mathbf{y}) \neq \emptyset$ and $\mathbf{y}'$ is a fixed point, then $\mathbf{F}(\mathbf{y}) = \mathrm{supp}(\mathbf{y}')$ is called a fixed set.

Theorem 1 ([4]): Let $\mathcal{C}$ be an LDPC code with d_v-left-regular Tanner graph G. Let $\mathbf{T}$ be a set consisting of variable nodes with induced subgraph J. Let the check nodes in J be partitioned into two disjoint subsets; $C_\mathrm{odd}(J)$ consisting of check nodes with odd-degree and $C_\mathrm{even}(J)$ consisting of check nodes with even-degree. Then $\mathbf{T}$ is a fixed set for the bit

$$f(w_v^l, \chi_{0_\mathrm{p}}^l, \chi_{0_\mathrm{n}}^l, \chi_{1_\mathrm{p}}^l, \chi_{1_\mathrm{n}}^l) = \begin{cases} f_1(w_v^l, \chi_{\mathrm{u}}^l) & \text{if} \quad (\chi_{0_\mathrm{p}}^l, \chi_{0_\mathrm{n}}^l, \chi_{1_\mathrm{p}}^l) \notin \{(2,0,0),(1,1,0)\} \\ w_v^l & \text{if} \quad (\chi_{0_\mathrm{p}}^l, \chi_{0_\mathrm{n}}^l, \chi_{1_\mathrm{p}}^l) = (2,0,0) \\ 0_\mathrm{w} & \text{if} \quad (\chi_{0_\mathrm{p}}^l, \chi_{0_\mathrm{n}}^l, \chi_{1_\mathrm{p}}^l) = (1,1,0), w_v^l \in \{0_\mathrm{s}, 0_\mathrm{w}\} \\ 1_\mathrm{w} & \text{if} \quad (\chi_{0_\mathrm{p}}^l, \chi_{0_\mathrm{n}}^l, \chi_{1_\mathrm{p}}^l) = (1,1,0), w_v^l \in \{1_\mathrm{s}, 1_\mathrm{w}\} \end{cases} \tag{1}$$

flipping algorithms (serial or parallel) iff : (a) Every variable node in $V(J)$ has at least $\lceil \frac{d_\mathrm{v}}{2} \rceil$ neighbors in $C_{\mathrm{even}}(J)$ and (b) No collection of $\lfloor \frac{d_\mathrm{v}}{2} \rfloor + 1$ check nodes of $C_{\mathrm{odd}}(J)$ share a neighbor outside J.

We define a failure of a TBF algorithm as follows.

Definition 2 ([5]): Consider a Tanner graph G and a TBF algorithm $\mathcal{F} = (f, L, \Delta_\mathrm{v}, \Delta_\mathrm{c})$. Let V_e denote the set of variable nodes that are initially corrupt and let J denote the induced subgraph on V_e. If the algorithm $\mathcal{F}$ does not converge on G after $l_{\mathcal{F}}^{\mathrm{m}}$ iterations, then we say that $\mathcal{F}$ *fails on the subgraph J of G*.

It can be seen that the decoding failure of $\mathcal{F}$ is defined with the knowledge of the induced subgraph J on the set of initially corrupt variable nodes, thus we need to enumerate all induced subgraphs J. While this is difficult in general, for practically important cases of small numbers of initial errors (less than 8) and small column-weight codes ($d_\mathrm{v} = 3$ or 4), the enumeration of such induced subgraphs is tractable. To illustrate this, Table III gives the number of non-isomorphic, column-weight $d_\mathrm{v} = 3, 4$, girth $g = 6, 8, \infty$ Tanner graphs with a small number of variable nodes.

TABLE III
NUMBER OF NON-ISOMORPHIC, LEFT-REGULAR TANNER GRAPHS WITH A SMALL NUMBER OF VARIABLE NODES.

$\lvert V\rvert$	$d_\mathrm{v}=3$			$d_\mathrm{v}=4$		
	$g=6$	$g=8$	$g=\infty$	$g=6$	$g=8$	$g=\infty$
2	0	0	2	0	0	2
3	1	0	4	1	0	4
4	6	1	9	6	1	9
5	32	4	20	47	4	21
6	209	23	50	441	26	53
7	1619	113	133	6549	145	146

To determine the uncorrectable t-error patterns, we start with by $\mathcal{I}$, the set of all d_v-variable-regular Tanner graphs with t variable nodes. Induced subgraph on the set of any t-set of initially corrupt variable nodes is isomorphic to some graph in $\mathcal{I}$. Let I be a Tanner graph in $\mathcal{I}$ and let $\mathscr{E}_I(\mathcal{F})$ denote the set of Tanner graphs containing a subgraph J isomorphic to I such that $\mathcal{F}$ fails on J. Then,if a code $\mathcal{C}$ has its Tanner graph $G \in \bigcup_{I \in \mathcal{I}} \mathscr{E}_I(\mathcal{F})$, then there exist some uncorrectable weight-t error patterns and if $G \notin \bigcup_{I \in \mathcal{I}} \mathscr{E}_I(\mathcal{F})$, then algorithm $\mathcal{F}$ can correct any weight-t error patterns. Let us assume that we are only interested in trapping sets with at most $n^{\max}$ 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 $n_I = |V(I)|$. We focus on a subset $\mathscr{E}_I^{\mathrm{r}}(\mathcal{F})$ of $\mathscr{E}_I(\mathcal{F})$, which is formulated as follows.

Definition 3 ([5]): Consider a Tanner graph $S_1 \in \mathscr{E}_I(\mathcal{F})$ such that $\mathcal{F}$ fails on the subgraph J_1 of S_1. Then, $S_1 \in \mathscr{E}_I^{\mathrm{r}}(\mathcal{F})$ if there *does not* exist $S_2 \in \mathscr{E}_I(\mathcal{F})$ such that:

1) $\mathcal{F}$ fails on the 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 $V(J_1)$.

The graphs in $\mathscr{E}_I^{\mathrm{r}}(\mathcal{F})$ are referred to as trapping sets of the TBF algorithm $\mathcal{F}$.

Definition 4 ([5]): If $S \in \mathscr{E}_I^{\mathrm{r}}(\mathcal{F})$ then S is a trapping set of $\mathcal{F}$. I is called an inducing set of S. $\mathscr{E}_I^{\mathrm{r}}(\mathcal{F})$ is called the trapping set profile with inducing set I of $\mathcal{F}$.

What enables the recursive procedure to determine a trapping set profile, is the two following propositions proven in [5].

Proposition 1: Let S 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 on 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 $0 \le l \le l_{\mathcal{F}}^{\mathrm{m}}$ and $w_v^l \in \{1_\mathrm{s}, 1_\mathrm{w}\}$.

Proposition 2: Consider decoding with an algorithm $\mathcal{F}$ on a Tanner graph G. Let V_e be the set of initially corrupt variable nodes and I be the induced subgraph on V_e. Then, algorithm $\mathcal{F}$ will converge after at most $l_{\mathcal{F}}^{\mathrm{m}}$ decoding iterations if there is no subset V_s of $V(G)$ such that $V_\mathrm{s} \supset V_\mathrm{e}$ and the induced subgraph on V_s is isomorphic to a graph in $\mathscr{E}_I^{\mathrm{r}}(\mathcal{F})$.

The recursive procedure for constructing a trapping set profile $\mathscr{E}_I^{\mathrm{r}}(\mathcal{F})$ with at most $n^{\max}$ variable nodes can be now defined as follows.

Let I be induced by initially corrupt variable nodes, and let $\mathbf{O}_I$ be a set of graphs obtained by adding a single, initially corrupt, variable node, to the graph I. If for some graph in $\mathbf{O}_I$, the decoder either fails, it is a potential trapping set. If, on the other hand, the decoder succeeds, we put it to $\mathbf{E}_I^1$ (1 indicates the first iteration), a set of subgraphs we know are not trapping sets. $\mathbf{E}_I^1$ is formed by repeating the above check for all other graphs in $\mathbf{O}_I$. Determining the trapping set profile involves maintaining the list of $\mathbf{E}_I^i$, $1 \le i \le \max - n_I$. The algorithm proceeds by expanding graphs in $\mathbf{E}_I = \bigcup_{i=0}^{(n^{\max}-n_I-1)} \mathbf{E}_I^i$.

For all Tanner graphs in $\mathbf{E}_I = \bigcup_{i=0}^{(n^{\max}-n_I-1)} \mathbf{E}_I^i$, denoted by K, the expansion of K is repeated resulting in $\mathbf{O}_K$, the set of all Tanner graphs obtained by adding a single initially corrupt to the graph K induced by all initially corrupt variable nodes, so that the added variable nodes remain corrupt at the end of second iteration but not a corrupt variable node at the end of the first iteration. By recursively adding variable nodes, all graphs in $\mathbf{E}_K^i$ are enumerated.

We note that the algorithm proceeds by a double recursion: The algorithm **RA1** enumerates graphs in $\mathbf{E}_K$ by recursively

adding variable nodes. The algorithm **RA2** recursively calls **RA1** in order to enumerate graphs in $\mathbf{E}_K$ for each graph K in $\mathbf{E}_I$. The trapping set profile is obtained after L recursions of the second algorithm. The pseudocodes of the two algorithms, which we call the **RA1** and the **RA2**, are given in Algorithms 2 and 3, where let $\mathbf{E}_I^0 = \{I\}$.

Algorithm 2 RA1

Input: A Tanner graph K which contains the induced subgraph I, the current decoding iteration l and $\mathscr{E}_I^{\mathrm{r}}(\mathcal{F})$
Output: $\mathbf{E}_K = \bigcup_{i=0}^{(n^{\max}-n_I-1)} \mathbf{E}_K^i$ and $\mathscr{E}_I^{\mathrm{r}}(\mathcal{F})$
Execute:

```
E_K = ∅
if F fails on I of K then
    if K is a trapping set then
        E_I^r(F) ← E_I^r(F) ∪ {K}
    end if
else
    E_K ← E_K ∪ {K}
    Add one variable node to K in different ways to obtain the set of Tanner graphs O_K. For each graph
    U ∈ O_K, the newly added variable node v has states
    w_v^0, w_v^1, ..., w_v^{l-1} ∈ {0_w, 0_s} and w_v^l ∈ {1_w, 1_s}
    for all U ∈ O_K do
        Call the RA1 with U, l as the inputs to obtain E_U
        E_K ← E_K ∪ E_U
    end for
end if
```

Algorithm 3 RA2

Input: A set of Tanner graphs $E_I = \bigcup_{i=0}^{(n^{\max}-n_I-1)} \mathbf{E}_I^i$, the current decoding iteration l and $\mathscr{E}_I^{\mathrm{r}}(\mathcal{F})$
Output: $\mathscr{E}_I^{\mathrm{r}}(\mathcal{F})$
Execute:

```
for all K ∈ ⋃_{i=0}^{(n^max - n_I - 1)} E_I^i do
    Perform one decoding iteration on K
    l ← l + 1
    if l < L then
        Call the RA1 with K, l and E_I^r(F) as the inputs to obtain E_K = ⋃_{i=0}^{(n^max - n_I - 1)} E_K^i
        Call the RA2 with E_K, l and E_I^r(F) as the inputs
    else
        Call the RA1 with K, l and E_I^r(F) as the inputs
    end if
end for
```

The following example demonstrates the operations of these recursive algorithms.

Example 1:

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 lth iteration, respectively.

Let I be the Tanner graph which is depicted in Fig. 1(a). Assume the use of the TBFA2 called the "Algorithm 2" in [5]. Also assume that we are only interested in trapping sets with at most 8 variable nodes, i.e., $n^{\max} = 8$, in a regular column-weight-three code with girth $g = 8$. The construction of the trapping set profile $\mathscr{E}_I^{\mathrm{r}}(\mathcal{F})$ starts with the initializations $\mathscr{E}_I^{\mathrm{r}}(\mathcal{F}) = \emptyset$, $l = 0$, and $w_v^0 = 1_{\mathrm{s}}\ \forall v \in V(I)$. Then, the **RA2** is called with $\{I\}$, l and $\mathscr{E}_I^{\mathrm{r}}(\mathcal{F})$ as the inputs.

The **RA2** first performs one decoding iteration on I. Fig. 1(b) show the states of variable nodes in I at the end of the first iteration and the states of the check nodes at the beginning of the second iteration. The **RA2** then calls the **RA1**, passing I, $l = 1$ and $\mathscr{E}_I^{\mathrm{r}}(\mathcal{F})$ as the inputs.

The **RA1** initializes $E_I = \emptyset$, then checks to see if $\mathcal{F}$ fails on I of I. Since $\mathcal{F}$ does not fail on I of I, E_I is replace by $E_I \cup \{I\}$. The **RA1** now tries to add one variable node v_5 to I such that $w_{v_5}^1 \in \{1_{\mathrm{s}}, 1_{\mathrm{w}}\}$. Since $w_{v_5}^0 = 0_{\mathrm{s}}$, v_5 would have to connect to three unsatisfied check nodes. However, since the set of unsatisfied check nodes at the beginning of the first iteration is $\{c_1, c_2, c_4, c_6, c_8, c_9\}$, connecting v_5 to any combination of three unsatisfied check nodes would violate the girth condition. Consequently, no variable node can be added to I and $E_I = \{I\}$ is returned to the **RA2**.

The **RA2** then calls itself, with E_I, $l = 1$, and $\mathscr{E}_I^{\mathrm{r}}(\mathcal{F}) = \emptyset$ as the inputs. The 2nd level **RA2** picks one graph in E_I. There is only one such graph. It then performs one decoding iteration on I. Fig. 1(c) shows the states of variable nodes and check nodes of I at the end of the second iteration and at the beginning of the third iteration, respectively. The **RA2** then call the **RA1**, passing I, $l = 2$ and $\mathscr{E}_I^{\mathrm{r}}(\mathcal{F})$ as the inputs.

Since $\mathcal{F}$ does not fails on I of I, E_I is replace with $E_I \cup \{I\}$, hence $E_I = \{I\}$. The **RA1** now tries to add one variable node v_5 to I such that $w_{v_5}^2 \in \{1_{\mathrm{s}}, 1_{\mathrm{w}}\}$. Due to the girth restriction , v_5 can connect to at most two check nodes in $\{c_i : 1 \le i \le 9\}$. It is easy to see that v_5 must connect to exactly two check nodes in $\{c_1, c_2, c_4, c_6, c_8, c_9\}$, which is the set of unsatisfied check nodes. Fig. 1(d) and 1(e) show two different graphs obtained by two different ways of adding v_5. These two graphs form the set of Tanner graphs O_I.

The **RA1** will now call itself $|O_I|$ times, with the inputs for each time include a graph in O_I. Consider the instance in which the graph shown in Fig. 1(e) is an input. Let U denotes this graph. By adding one variable node to U in different ways, the **RA1** obtains three graphs, which are shown in Fig. 1(d), (f) and (g). These graphs form the set O_U.

Now, let us consider the instance in which the graph shown in Fig. 1(g), denoted by K, is an input to the **RA1**. Since $\mathcal{F}$ fails on I of K, the **RA1** determines whether or not K is a trapping set. It is easily to check that K is indeed a trapping set and hence the **RA1** stops expanding K.

Finally, let us consider the graph shown in Fig. 1(i), denoted by K'. K' is obtained by adding one variable node to the graph shown in Fig. 1(f). One can see that $\mathcal{F}$ fails on I of K'. However, K' is not a trapping set, since K' contains K. This is because the graph obtained by deleting v_6 from K' is isomorphic to K. Therefore, the **RA1** will discard K'.

When expanding a graph by adding variable nodes, it is critical that the **RA1** results in distinct graphs to ensure a

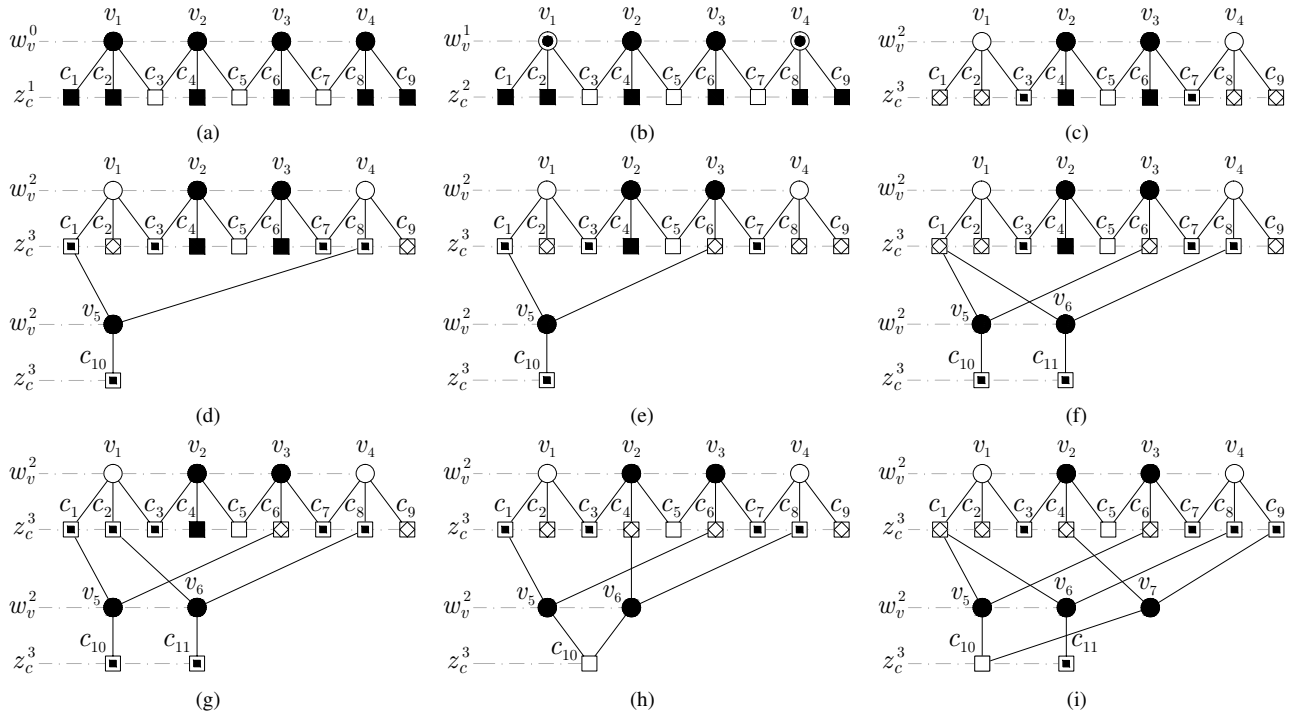

Fig. 1. Figures used in Example 1.

manageable complexity. In other words, although there can be many possible ways to add a new variable node, some result in equivalent graphs. In the operation of the **RA1**, two Tanner graphs are equivalent if there exists an isomorphism between them such that under this isomorphism, the variable node set $V(I)$ is mapped into $V(I)$. In the enumeration of a trapping set profile, it is sufficient to consider only one graph in a set of equivalent graphs. Consider a Tanner graph K which contains the subgraph I. Then, two subsets of check nodes of K are equivalent if the two graphs obtained by adding a variable node connecting to check nodes in each subset are equivalent. In the enumeration of graphs in O_K, the **RA1** should only add a new variable node that connects to check nodes in only one among all equivalent subsets of check nodes of K.

The determination of equivalent check node subsets is facilitated with the following definitions as well as Proposition 3, which we state without proof.

Proposition 3: In the operation of the **RA1**, two subsets C_1, C_2 of check nodes of K are equivalent if there is a one-to-one correspondence between the set of variable nodes that connect to at least one check node in C_1 and the set of variable nodes that connect to at least one check node in C_2, under which a variable node is equivalent to its image.

IV. Conclusions

We gave a systematic procedure for the analysis and selection of algorithms that can together correct a fixed number of errors with high probability. Using this procedure, we were able to construct TBF algorithms for column-weight-three codes with girth $g = 8$ outperforming the Gallager A/B algorithm, the min-sum algorithm and the SPA for different code rates and lengths. We have constructed all trapping set profiles with inducing sets containing four, five and six variable nodes for each algorithm in $\mathscr{F}$, and designed a set of TBF algorithms that can collectively correct error patterns of weight 4 and 5 with high probability.

Acknowledgment

This work was supported by the National Science Foundation under Grants CCF-0963726 and CCF-1314147 and in part by Seventh Framework Programme of the European Union, under Grant Agreement number 309129 (i-RISC project). Part of the material in this paper will appear in IEEE Transaction on Communications. The work of D. V. Nguyen was performed when he was with the Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ, USA.

References

[1] V. Zyablov and M. Pinsker, "Estimation of the error-correction complexity for Gallager low-density codes," *Probl. Inf. Transm.*, vol. 11, no. 6, pp. 18–26, 1976.

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

[3] 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.

[4] 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.

[5] D. V. Nguyen and B. Vasic, "Two-bit bit flipping algorithms for LDPC codes and collective error correction (accepted)," *IEEE Trans. Comm.*, Jan. 2014.

[6] D. V. Nguyen, S. K. Chilappagari, B. Vasic, and M. W. Marcellin, "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.