Exhaustive Search for Small Fully Absorbing Sets and The Corresponding Low Error-Floor Decoder

Gyu Bum Kyung and Chih-Chun Wang
School of Electrical and Computer Engineering
Purdue University, West Lafayette, IN 47907, USA
E-mail: {gkyung, chihw}@purdue.edu

***Abstract*—This work provides an exhaustive search algorithm for finding small fully absorbing sets (FASs) of arbitrary low-density parity-check (LDPC) codes. In particular, given any LDPC code, the problem of finding all FASs of size less than t is formulated as an integer programming problem, for which a new branch-&-bound algorithm is devised. New node selection and the tree-trimming mechanisms are designed to further enhance the efficiency of the algorithm. The proposed algorithm is capable of finding all FASs of size ≤ 11 with no larger than 2 induced odd-degree check nodes for LDPC codes of length ≤ 1000.**

The resulting exhaustive list of small FASs is then used to devise a new post-processing decoder. Numerical results show that by taking advantage of the exhaustive list of small FASs, the proposed decoder significantly lowers the error floor for codes of practical lengths and outperforms the state-of-the-art low-error-floor decoders.

I. Introduction

Low-Density Parity-Check (LDPC) codes [1] have received much attention thanks to their near Shannon limit performance and to the efficient belief propagation (BP) decoders. When compared to the prohibitively costly optimal maximum likelihood decoders, the suboptimality of BP leads to the performance gaps both in the *waterfall region* and in the *error-floor* region. Generally, the error floor of LDPC codes is at around 10^{-5} to 10^{-7}, which is detrimental to many important applications that require very low error floor (10^{-12} to 10^{-15}) such as the digital storage devices. The error-floor performance of LDPC codes is affected by error-prone substructures (EPSs) such as stopping sets (SSs) [2], near-codewords [3], trapping sets (TSs) [4], pseudocodewords [5], and absorbing sets (ASs) [6]. Since the EPSs of LDPC codes are generally not valid codewords, the error floor can be lowered with improved BP decoders [7], [8].

The characterization and enumeration of EPSs are important for estimating the error floor of LDPC codes using importance sampling [4], [9]. Recent results show that even a partial list of EPSs can be used to significantly lower the error floor [9]. However, the problem of searching *all* small EPSs turns out to be a non-trivial task. The problems of determining the minimum size of the SSs are proven to be an NP-complete problem [10]. This result is extended for the k-out TSs in [11]. Furthermore, [12] generalized this NP-hardness result and showed that even approximating the size of the "cover SSs" and different kinds of TSs is also NP-hard. On the other hand, it is still possible to develop an efficient algorithm that can reduce the coefficient of exponential complexity so that we can identify the minimum size of EPSs and also find the exhaustive list of them for codes of practical length ($1000 \sim 10000$). Moreover, like any NP-hardness results, [12] focuses on the worst case analysis with asymptotically large codeword length, and thus may not describe the average behavior when focusing on practical LDPC codes of special structures (say Quasi-Cyclic LDPC codes). In spite of the hardness of finding the minimum Hamming distance, some practical efficient algorithms have been proposed recently to find the minimum distance of LDPC codes of moderate lengths ≤ 2000 [13]. These results are very encouraging especially considering the fact that the minimum size of EPSs is much smaller than the minimum Hamming distance, which, in principle, makes the problem of exhaustively searching for small EPSs easier (i.e., with smaller coefficients or with smaller exponential growth rate) than the problem of finding the minimum Hamming distance. There are recently some results on finding the minimal EPSs. [11] proposed the first such exhaustive search algorithm of small SSs and k-out TSs based on tree-pruning techniques. An even more efficient exhaustive SS search algorithm was proposed in [14] based on extended iterative decoding and linear programming techniques.

Several works were proposed to lower the error floor by improving the suboptimal BP decoder. One such approach is to directly modify the BP decoder: [15] suggested *averaged BP decoding* to prevent the errors being trapped during decoding. Another approach is based on *post-processing decoding*. [9] used the bit pattern subtraction after BP decoding by checking whether the unsatisfied check nodes after BP decoding match those of a TS in an inexhaustive TS list. A bit-guessing approach after the decoding failure is developed to lower the error floor in [16], and is refined in [7]. [8] proposed a bi-mode syndrome-erasure decoder and a generalized LDPC decoder, which combined several problematic check nodes to a generalized constraint node and applied the corresponding BCJR algorithm. [17] used post-processing decoding which lowers the error floor by increasing the reliability of the messages from unsatisfied check nodes and decreasing the reliability of the messages from *mis-satisfied* check nodes.

In this work, we first propose an efficient algorithm to find all small fully ASs (FASs), originally defined in [6], for arbitrary LDPC codes. We then use the exhaustive list to develop a new post-processing decoder that substantially lowers the error floor. Our exhaustive search is based on the branch-&-bound principle, which was previously used in [11] and [14] for exhaustively finding SSs. During the bounding step, we formulate a new integer programming (IP) problem to decide the minimum size of FASs under given constraints and solve the problem using the branch-&-cut algorithm, which can be viewed as a generalization of [13] that finds the minimum Hamming distance using IP problems. Note that the IP approach was also used by [18] to approximate the minimum Hamming distance, which is conceptually quite different from the exact/exhaustive determination in this work and in [11], [13], [14]. A new branching policy and other techniques are used to further improve the efficiency. For practical codes of length ≤ 1000, our algorithm is able to find all small FASs with size ≤ 11 and no larger than 2 induced odd-degree check nodes. We also propose a new post-processing decoder which takes advantage of the exhaustive list of small FASs and is based on new soft log-likelihood ratio (LLR) manipulation and joint identification of problematic variable and check nodes. Numerical results on the PEGR504 and the Margulis2640 codes [19] show that the error floor can be lowered by a couple of orders of magnitude when compared to the BP decoder, and the proposed decoder outperforms the state-of-the-art low-error-floor decoders.

II. Notation and Definition

We define a code by an $m \times n$ sparse parity check matrix $\boldsymbol{H}$ where n is the codeword length and m is the number of parity check equations. For convenience, we denote a code by its $\boldsymbol{H}$ matrix. Also, the code $\boldsymbol{H}$ can be represented by its bipartite Tanner graph which has two sets of nodes: the variable nodes $V = \{v_1, \ldots, v_n\}$ and the check nodes $C = \{c_1, \ldots, c_m\}$. Let $\mathcal{N}(v)$ and $\mathcal{N}(c)$ be the sets of neighboring nodes of a variable node v and a check node c, respectively. Also, let $d_v = |\mathcal{N}(v)|$ and $d_c = |\mathcal{N}(c)|$ be the corresponding node degrees.

The definitions of the AS and its variants follow from [6]. For any $A \subseteq V$, let $E_A \subseteq C$ and $O_A \subseteq C$ be the sets of check nodes that are connected to A for an even number (≥ 0) and an odd number of times, respectively. We also define $E_A(v)$ as the set of check nodes in E_A which are connected to v. $O_A(v)$ can be defined similarly.

Definition 1 (AS and FAS [6]): A subset $A \subseteq V$ is called an AS if $|E_A(v)| > |O_A(v)|$ for all $v \in A$ and a subset $A \subseteq V$ is called a FAS if $|E_A(v)| > |O_A(v)|$ for all $v \in V$.

An AS A of size s is also called an (s,t) AS if $s = |A|$ is its size and $t = |O_A|$. An AS is different from the operational definitions of TSs [4] and is also different from the graphical definitions of SSs [2], k-out TSs [11], and near-codewords [3]. For the following, we will exhaustively search all small FASs of a given LDPC code.

III. A New Exhaustive Search Algorithm for Small FASs

A. Main Branch-&-Bound Algorithm

Let $\mathbf{S} \subseteq \{0, 1, *\}^n$ be a collection of strings where $*$ is the "unconstrained" symbol that is not fixed to "0" or "1." We say that a binary string $\mathbf{s}_1 = (s_{11}, \cdots, s_{1n})$ is *compatible* to $\mathbf{s}_0 = (s_{01}, \cdots, s_{0n}) \in \mathbf{S}$ if $s_{1i} = s_{0i}$ for all i satisfying $s_{0i} \neq *$. Let $\mathsf{Supp}(\mathbf{s}_0)$ be the set of variable nodes corresponding to the 1's in a string $\mathbf{s}_0 \in \mathbf{S}$. Conversely, $\mathsf{Inc}(A)$ is a binary incidence vector which has 1's in the positions for all $v \in A$ and 0's in the other positions $V \backslash A$. In addition, we use $\mathcal{A}$ to denote a collection of FASs. The main branch-&-bound algorithm is described as follows.

Algorithm 1 An exhaustive search algorithm for FASs.

```
1:  Input: the parity check matrix $\boldsymbol{H}$, two integers $s_{\max}$, and $t_{\max}$.
2:  Set $\mathbf{S} \leftarrow \{(*, *, \cdots, *)\}$ and set $\mathcal{A} \leftarrow \emptyset$.
3:  while $\mathbf{S} \neq \emptyset$ do
4:      Take one string $\mathbf{s}_0$ from $\mathbf{S}$, and let $\mathbf{S} \leftarrow \mathbf{S} \backslash \mathbf{s}_0$.
5:      if $|\mathsf{Supp}(\mathbf{s}_0)| \leq s_{\max}$ then
6:          if $\mathsf{Supp}(\mathbf{s}_0)$ is a FAS and $|O_{\mathsf{Supp}(\mathbf{s}_0)}| \leq t_{\max}$ then
7:              $\mathcal{A} \leftarrow \mathcal{A} \cup \{\mathsf{Supp}(\mathbf{s}_0)\}$.
8:          end if
9:          Compute a lower bound $b$ such that $b \leq |A|$ for all $A \subseteq V$ such that $A$ is a FAS, $|O_A| \leq t_{\max}$, and $\mathsf{Inc}(A)$ is compatible to $\mathbf{s}_0$.
10:         if $b \leq s_{\max}$ then
11:             Choose one unconstrained location of $\mathbf{s}_0$ and generate two new strings $\mathbf{s}_1$ and $\mathbf{s}_2$ by setting the unconstrained location to 1 and 0, respectively.
12:             $\mathbf{S} \leftarrow \mathbf{S} \cup \{\mathbf{s}_1, \mathbf{s}_2\}$.
13:         end if
14:     end if
15: end while
16: Output: $\mathcal{A}$ is the exhaustive list of all $(s,t)$ FASs with $s \leq s_{\max}$ and $t \leq t_{\max}$.
```

B. Bounding Step in Line 9

We use an IP solver to find a lower bound b in Line 9. In our IP problem, the integer variables are denoted by x_{v_i}, w_{c_j}, and z_{c_j} where the subscripts $v_i \in V$ and $c_j \in C$. For notational simplicity, we often use x_i, w_j, and z_j as shorthand.

$$\text{Minimize} \quad \sum_{v_i \in V} x_i \tag{1}$$

$$\text{subject to} \quad \sum_{v_i \in \mathcal{N}(c_j)} x_i = 2w_j + z_j \text{ for all } c_j \in C \tag{2}$$

$$\sum_{c_j \in \mathcal{N}(v_i)} z_j \leq \left\lfloor \frac{d_{v_i}}{2} \right\rfloor \text{ for all } v_i \in V \tag{3}$$

$$\sum_{v_i \in V} x_i \geq 1, \text{ and } \sum_{c_j \in C} z_j \leq t_{\max} \tag{4}$$

$$x_i, z_j \in \{0,1\} \text{ for all } v_i \in V \text{ and } c_j \in C$$

$$w_j \text{ is a non-negative integer for all } c_j \in C.$$

In this IP problem, x_i decides whether v_i is part of a FAS. (2) uses z_j to capture the resulted parity of c_j. (3) follows from the definition of FASs. (4) is used for eliminating the all-zero codeword and for limiting the number of the check codes of odd degree in a FAS. The objective function (1) minimizes the size of the FAS satisfying the given constraints. When a constraint string $\mathbf{s}_0$ is given, we hardwire the corresponding positions of x_i to be 0 or 1 depending on the locations of 0's and 1's in $\mathbf{s}_0$. The IP program can be solved by linear programming (LP) relaxation and the branch-&-cut principle. In our implementation, we used CPLEX 10.2 [20] with the additional user-defined *cuts* proposed in [13] to reduce the number of branching nodes.

It is worth noting that during the LP relaxation, there is no need to identify the optimal integer solution as we are only interested in a lower bound b as in Line 9. Therefore, any LP-relaxed minimal objective value (1) during the branch-&-cut process can be used as a lower bound b. By solving a relaxed LP problem instead of an IP problem, we avoid spending too much time on a single IP problem and the efficiency of the exhaustive search algorithm is greatly improved. In our implementation, we also define the "branch node limit." That is, if the number of branching nodes is over the predefined node limit, we stop the branch-&-cut process of the current IP problem, use the optimized objective value of the LP relaxation as our lower bound b, and proceed to Line 11.

C. Further Enhancing The Efficiency

As with any branch-&-bound algorithm, the efficiency depends dramatically on the branching policy in Line 11. We use the following branching policy that significantly improves the efficiency of the branch-&-bound algorithm. During the LP relaxation, the values of x_i are chosen from the interval $[0,1]$ rather than from $\{0,1\}$. For any *soft* x_i values found in the LP relaxation, let z'_j be the distance of $\mathsf{sum}_j \triangleq \sum_{v_i\in\mathcal{N}(c_j)} x_i$ to the closest even number, i.e. $z'_j = |\mathsf{sum}_j - 2\lfloor 0.5+0.5\mathsf{sum}_j\rfloor|$. Let $\mathcal{N}(A)$ be $\bigcup_{v_i\in A}\mathcal{N}(v_i)$, and let B be the set of variable nodes corresponding to the unconstrained positions in $\mathbf{s}_0$. For each $v_k \in \mathcal{N}(\mathcal{N}(A))\cap B$, compute

$$\xi_k \triangleq \min\left(\max\left(\left(\sum_{c_j\in\mathcal{N}(v_k)} z'_{j,x_k=0}\right) - \left\lfloor\frac{d_{v_k}}{2}\right\rfloor, 0\right),\right.$$
$$\left.\max\left(\left(\sum_{c_j\in\mathcal{N}(v_k)} z'_{j,x_k=1}\right) - \left\lfloor\frac{d_{v_k}}{2}\right\rfloor, 0\right)\right), \quad (5)$$

where $z'_{j,x_k=0}$ and $z'_{j,x_k=1}$ are the z'_j values computed when hardwiring the x_k value to 0 and 1, respectively. The larger value of ξ_k means that it is more likely that using x_k as a branching choice is going to violate (3), which means that the IP solver can quickly refine the solution. Among all $v_k \in \mathcal{N}(\mathcal{N}(A))\cap B$, we choose the one with the largest ξ_k as the branching position in Line 11.

For comparison, in our implementation, if we choose the branch location randomly, it takes 45 hours 43 minutes to find all FASs with $s_{\max} = t_{\max} = 5$ of the PEGR504 code [19]. However, it only takes 6 hours 55 minutes using the branching policy described in this subsection.

IV. Low Error-Floor Post-Processing Decoding

In this section, we propose new post-processing decoding using the exhaustive list of FASs obtained from the algorithm in Section III to lower the error floor under the additive white Gaussian noise (AWGN) channel. Let $L^{(0)}$ and $L^{(l)}$ denote the initial LLR vector of the output of the AWGN channel and the final LLR vector after l iterations of BP, respectively. Let $Y^{(0)}$ and $Y^{(l)}$ denote the hard decisions based on $L^{(0)}$ and $L^{(l)}$, respectively. The main idea is to generate a new LLR vector L based on the exhaustive list of small FASs and $Y^{(0)}$ and $Y^{(l)}$. L is then used as new initial LLR messages to rerun the BP decoder. The construction of L can be fine tuned by changing the values of the parameters $(\beta_{\text{th}}, d, \alpha, \Delta_\alpha, l')$. The new post-processing decoder works as follows.

Algorithm 2 A post-processing decoding algorithm.

```
1: Input: the parity check matrix $\boldsymbol{H}$, the exhaustive list of FASs $\mathcal{A}$, $(L^{(0)}, L^{(l)})$, $(Y^{(0)}, Y^{(l)})$, and the tuning parameters $(\beta_{\text{th}}, d, \alpha, \Delta_\alpha, l')$
2: if $|O_{\mathsf{Supp}(Y^{(l)})}| > 0$ then
3:   Set the binary vector $U \leftarrow Y^{(0)}$.
4:   Run BP for 1 iteration based on $L^{(0)}$ and denote the resulting check-to-variable message as $m^{(1)}_{c_j\to v_i}$.
5:   Calculate the soft parity message $m_{c_j}$ for all $c_j \in C$ after the 1-iteration BP decoding.
6:   For each $A \in \mathcal{A}$, compute $\beta_A = \sum_{c_j\in O_A}\left(1 + 1_{\{c_j\in O_{\mathsf{Supp}(Y^{(0)})}\cap O_{\mathsf{Supp}(Y^{(l)})}\}}\right) m_{c_j}$, where $1_{\{\cdot\}}$ is the indicator function.
7:   Sort $\mathcal{A}$ according to $\beta_A$ in the ascending order, that is, $\beta_{A_1} \le \beta_{A_2} \le \cdots$. And set $k \leftarrow 1$
8:   while $\beta_{A_k} < \beta_{\text{th}}$ do
9:     Set the LLR vector $L \leftarrow L^{(0)}$.
10:    Set $A \leftarrow \bigcup_{i=k}^{k+d-1} A_i$, $E \leftarrow \bigcup_{i=k}^{k+d-1} E_{A_i}$, and $O = \bigcup_{i=k}^{k+d-1} O_{A_i}$.
11:    Compute the following message scaling function $f(c_j, v_i)$. In the case that $v_i \in A_{k+h}$ for some[1] $h \in \{0,\cdots,d-1\}$, we set $f(c_j,v_i) = \alpha - h\Delta_\alpha$ if $c_j \in O \cap O_{\mathsf{Supp}(U)}$ and set $f(c_j,v_i) = \frac{1}{\alpha - h\Delta_\alpha}$ if $c_j \in E\backslash O_{\mathsf{Supp}(U)}$. Otherwise, set $f(c_j,v_i) = 1$.
12:    Denote the $i$-th component of $L$ by $m_{v_i}$. For those $v_i \in A$, replace the corresponding $m_{v_i}$ by $\left(L^{(0)}\right)_{v_i} + \sum_{c_j\in\mathcal{N}(v_i)} f(c_j,v_i) m^{(1)}_{c_j\to v_i}$.
13:    Using the final $L$ vector as the initial LLR, run BP decoding for $l'$ iterations and let $\bar{Y}$ be the binary decision after $l'$ iterations.
14:    If $\bar{Y}$ is a valid codeword, then RETURN $\bar{Y}$ as the final output.
15:    $k \leftarrow k + 1$.
16:  end while
17: end if
```

The detailed explanation of the above algorithm is as follows. Line 2 focuses only on the case that the BP decoder fails to generate a valid codeword, in which we perform post-processing decoding. Lines 3 and 9 state that we use the initial messages as the starting points: the L and U vectors. Line 4 prepares some $m^{(1)}_{c_j\to v_i}$ values that will be used later in the

[1]If v_i is in several A_{k+h}'s with different h's, then we use the smallest h for computing $f(c_j, v_i)$.

algorithm. The soft parity message in Line 5 is computed by $m_{c_j} = 2\tanh^{-1}\left(\prod_{v_k \in \mathcal{N}(c_j)} \tanh\left(\frac{m_{v_k \to c_j}}{2}\right)\right)$. In Line 6, we sum up the m_{c_j} values for $c_j \in O_A$. The more negative is the sum, the more likely that those odd parity c_j's are caused by errors in the given FAS. To emphasize the repeated observations from both $Y^{(0)}$ and $Y^{(l)}$, those $c_j \in O_{\mathsf{Supp}(Y^{(0)})} \cap O_{\mathsf{Supp}(Y^{(l)})}$ are emphasized by assigning a weight of 2 instead of 1 when computing β_A. This β_A is then used to sort the exhaustive FAS list $\mathcal{A}$. A_1, having the smallest (the most negative) β_{A_1}, is then the most likely FAS candidate in our post-processing decoder. We then search over a sliding window of d A_i's, from A_k to A_{k+d-1}. To limit the number of trials, we require $\beta_{A_k} < \beta_{\text{th}}$ to avoid searching over too many FASs. For each window, we use an algorithm that is modified from the increase and decrease (ID) algorithm proposed in [17]. The main idea of the ID algorithm is to increase the check-to-variable (c_j-to-v_i) LLR that contains the correcting power (emitting from an unsatisfied check node) while decreasing the c_j-to-v_i LLR that emits from a mis-satisfied check node. However, without the knowledge of the exhaustive FAS list, the method in [17] has only the knowledge about which c_j is unsatisfied but does not know which $v_i \in \mathcal{N}(c_j)$ needs a stronger correcting power. In our scheme, we take advantage of both the c_j and the v_i information from our exhaustive FAS list. In Line 11, our scaling factor takes into account both v_i and c_j by comparing it to the FAS in the exhaustive list. The scaling factor $f(c_j, v_i)$ depends on h because we want to use stronger $f(c_j, v_i)$ for those $v_i \in A_{k+h}$ with more negative $\beta_{A_{k+h}}$ and use weaker $f(c_j, v_i)$ for FASs with more positive $\beta_{A_{k+h}}$. In Line 12, we use $\left(L^{(0)}\right)_{v_i}$ to denote the received LLR values for v_i. Line 12 then replaces part of the L vector by the newly increased/decreased messages.

Note that the proposed algorithm is based on the initial LLR $L^{(0)}$. One can easily extend the proposed algorithm by considering the final LLR $L^{(l)}$ as well. That is, we replace the Line 3 by $U \leftarrow Y^{(l)}$, replace Line 9 by $L \leftarrow L^{(l)}$, delete Line 4, and compute Lines 5 and 12 directly from the $m^{(l)}_{c_j \to v_i}$ messages after l BP iterations. In our implementation, we concatenate these two algorithms (one based on $L^{(0)}$ and one based on $L^{(l)}$, respectively) in sequence, i.e., we run the $L^{(0)}$-based scheme first and if it succeeds, then stop. If it does not succeed, we then continue to run the $L^{(l)}$-based scheme to further lower the error floor of the original $L^{(0)}$-based scheme.

V. Numerical Results

A. Results of The Exhaustive FAS Search Algorithm

1) Randomly constructed regular LDPC codes: Due to the space limit, we only report results on two randomly generated regular $(3, 6)$ LDPC codes from [19]. PEGR504 is of length 504 and is constructed by the PEG algorithm [21]. M1008 is also a regular $(3, 6)$ LDPC code constructed by MacKay. For these two codes, our algorithm is capable of finding all small FASs when setting the search range $(s_{\max}, t_{\max})$ to $(6, 6)$ and to $(5, 5)$, respectively. Moreover, if we use a smaller $t_{\max} = 2$, we find all FASs of size ≤ 13 for PEGR504 and size ≤ 11 for M1008. For example, there are exactly 12 different types of FAS in PEGR504 with $s + t \leq 12$ (See Table I). We can also find more FASs of larger sizes using our algorithm. For example, there are 111 $(7, 3)$ FASs in M1008.

PEGR504	(4, 4)	760	(5, 3)	14	(5, 5)	10156
	(6, 4)	849	(6, 6)	66352	(7, 3)	47
	(7, 5)	22430	(8, 2)	4	(8, 4)	2270
	(9, 1)	1	(9, 3)	146	(10, 2)	6
	(12, 2)	26	(13, 1)	1		
PEGI504	(3, 2)	219	(3, 3)	1517	(3, 4)	380
	(4, 2)	208	(4, 3)	3870	(4, 4)	27968
	(4, 5)	5131	(5, 2)	198	(5, 3)	6913
	(6, 2)	205	(7, 2)	274	(8, 2)	468

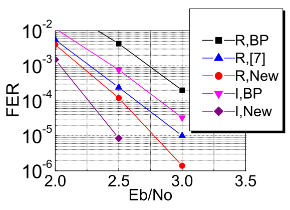
TABLE I
THE NUMBER OF (s, t) FASS FOR DIFFERENT LDPC CODES. IN EACH SUPER COLUMN, THE VECTOR IN THE FIRST SUB-COLUMN REPRESENTS THE (s, t) VALUE AND THE SECOND SUBCOLUMN CORRESPONDS TO THE NUMBER OF SUCH FASS.

Without taking advantage of special structures of the codes, our algorithm is actually computationally slower than the result in [22] when searching for small FASs with $s_{\max} \leq 5$. However, our scheme can easily find much larger FASs (see Table I) for which there are no analytical search results in the existing literature. Moreover, this computational search can avoid derivation mistakes. For example, [22] finds FASs exhaustively based on counting cycles and discussing the relationship between different cycles. Some very small derivation mistakes[2] in [22] lead to incorrect conclusion that the numbers of $(3, 3)$, $(4, 4)$, and $(5, 3)$ FASs of M1008 are 165, 1215, and 14, while our search algorithm shows that the actual numbers[2] should be 153, 1130, and 92, respectively, with additional 6 $(4, 2)$ FASs that are previously not discovered in [22].

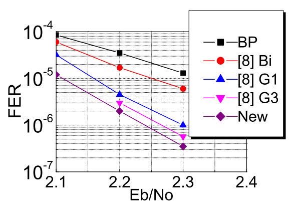
2) Algebraically Constructed LDPC Codes: We also apply our exhaustive algorithm to the algebraically constructed (155, 64, 20) Tanner code (Tanner155) [23] and Margulis2640 of $p = 11$ and length 2640 [24]. Our algorithm exhaustively find all FASs with the search range $(s_{\max}, t_{\max})$ set to $(7, 7)$ and to $(5, 5)$, respectively. In addition, if we use a smaller $t_{\max} = 2$, we find all FASs of size ≤ 15 for Tanner155 and size ≤ 9 for Margulis2640. Our algorithm also exhaustively finds FASs of larger (s, t) values that were not reported (as a 2-out TSs) in [11]. For example, there are 930 $(12, 2)$ FASs in Tanner155 and there is no $(9, 1)$ FASs in Margulis2640.

3) Irregular LDPC codes: The proposed algorithm can also be applied to irregular LDPC codes. Nonetheless, for irregular LDPC codes, there exist usually a lot of variable nodes of degree 2. By the definition of FASs, a degree-2 variable node v in a FAS must satisfy $|O_A(v)| < 0.5 d_v = 1$, which implies $|O_A(v)| = 0$. This constraint that no degree-2 variable node can be connected to an unsatisfied check node is too strong for a meaningful error-prone pattern. For irregular LDPC codes, we thus relax the definition of a FAS to $|E_A(v)| \geq |O_A(v)|$ for all $v \in V$ of degree 2 and keep the original constraint $|E_A(v)| > |O_A(v)|$ for all v of degree ≥ 3. We then apply our algorithm to PEGI504, a randomly constructed irregular LDPC code in [19]. Table I lists all its FASs of with $s+t \leq 8$.

[2]We thank Dr. S. K. Chilappagari of University of Arizona for valuable discussion.



(a) PEGR504 and PEGI504



(b) Margulis2640

Fig. 1. The FER curves of the proposed decoder compared to decoders in [7], [8]. In (a), we use 'R' and 'I' as shorthand for 'PEGR504' and 'PEGI504'. 'R,BP' is the performance of standard BP for PEGR504. 'R, [7]' is the performance of [7] on PEGR504. 'R,New' and 'I,New' are the results of our proposed decoder. In (b), we use 'Bi', 'G1', and 'G3' as shorthand for the 'Bimode' and the two 'Generalized' decoders in [8]. The results of our proposed decoder are denoted by 'New'.

B. Post-processing decoding

In Fig. 1(a), we show the frame error rate (FER) performance of our new post-processing decoder for PEGR504 and PEGI504 over the AWGN channel. For comparison, we also plot the best existing result on PEGR504 [7]. We use $l = 200$ and the additional $l' = 20$ iterations for post-processing decoding. In addition, we use Algorithm 2 with $\beta_{\text{th}} = 3$, $d = 4$, $\alpha = 1.5$, and $\Delta_\alpha = 0.1$. We employ all FASs obtained from $s_{\max} = t_{\max} = 7$ except the $(7, 7)$ FASs for PEGR504 and from $s_{\max} = 4$ and $t_{\max} = 5$ for PEGRI504 as the input $\mathcal{A}$. Our performance for PEGR504 is better than the best bit guessing scheme in [7]. Moreover, the proposed decoder lowers the error floor by orders of magnitude for both PEG-type LDPC codes. The performance enhancement is especially significant in the high signal-to-noise ratio (SNR) regime.

We also apply our decoder to Margulis2640, which were also used to evaluate the performance of the three existing post-processing decoders in [8]. For this code, we use $l = 50$ and $l' = 20$ iterations of BP decoding. We use $\beta_{\text{th}} = 3$, $d = 5$, $\alpha = 1.5$, and $\Delta_\alpha = 0.1$ in Algorithm 2. We employ all FASs of Margulis2640 with $s_{\max} = t_{\max} = 5$. The performance of the proposed decoder outperforms the state-of-the-art bi-mode and generalized-LDPC decoders in [8]. It is worth mentioning that with the sorting step in Line 7 of Algorithm 2, often we do not need to go through the entire FAS list, which means that the complexity of Algorithm 2 is kept low. In our experiments, the average number of FASs used in Algorithm 2 is 13.56 at 2.3 dB for Margulis2640. Half of the time, the number of FASs used by Algorithm 2 is only 1, which shows the efficiency of the proposed decoder when compared to the results in [8].

VI. Conclusions

We have proposed an efficient algorithm to find all small FASs of any given LDPC code using a branch-&-bound algorithm. We have then used the resulting exhaustive list to design a new post-processing decoder that substantially lowers the error floor of any LDPC code. By taking advantage of the exhaustive FAS list, the post-processing decoder can be made very effective and results on example LDPC codes show that our approach lowers the error floor by orders of magnitude and outperforms the state-of-the-art post-processing decoders.

References

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

[2] C. Di, D. Proietti, E. Telatar, T. Richardson, and R. Urbanke, "Finite length analysis of low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 48, no. 6, pp. 1570–1579, Jun. 2002.

[3] D. MacKay and M. Postol, "Weakness of Margulis and Ramanujan-Margulis low-density parity-check codes," *Electronic Notes in Theoretical Computer Science*, vol. 74, 2003.

[4] T. Richardson, "Error floors of LDPC codes," in *Proc. 41st Annu. Allerton Conf. on Commun. Contr. and Computing*. Monticello, IL, Oct. 2003.

[5] P. Vontobel and R. Koetter, "Graph-cover decoding and finite-length analysis of message-passing iterative decoding of LDPC codes," *IEEE Trans. Inform. Theory*, preprint-arXiv:cs.IT/0512078, to be published.

[6] L. Dolecek, Z. Zhang, V. Anantharam, M. Wainwright, and B. Nikolic, "Analysis of absorbing sets and fully absorbing sets of array-based LDPC codes," *IEEE Trans. Inform. Theory*, vol. 56, no. 1, pp. 181–201, Jan. 2010.

[7] N. Varnica, M. P. C. Fossorier, and A. Kavcic, "Augmented belief propagation decoding of low-density parity-check codes," *IEEE Trans. Commun.*, vol. 55, no. 7, pp. 1308–1317, Jul. 2007.

[8] Y. Han and W. E. Ryan, "Low-floor decoders for LDPC codes," *IEEE Trans. Commun.*, vol. 57, no. 6, pp. 1663–1673, Jun. 2009.

[9] E. Cavus and B. Daneshrad, "A performance improvement and error floor avoidance technique for belief propagation decoding of LDPC codes," in *Proc. IEEE Int'l. Symp. on Pers., Indoor and Mobile Radio Comm.* Berlin, Germany, Sept. 2005, pp. 2386–2390.

[10] K. Krishnan and P. Shankar, "On the complexity of finding stopping set size in Tanner graphs," in *Proc. 40th Conf. Inform. Sciences and Systems*. Princeton, NJ, Mar. 2006.

[11] C.-C. Wang, S. R. Kulkarni, and H. V. Poor, "Finding all small error-prone substructures in LDPC codes," *IEEE Trans. Inform. Theory*, vol. 55, no. 5, pp. 1976–1998, May 2009.

[12] A. McGregor and O. Milenkovic, "On the hardness of approximating stopping and trapping sets," *IEEE Trans. Inform. Theory*, vol. 56, no. 4, pp. 1640–1650, Apr. 2010.

[13] A. Keha and T. M. Duman, "Minimum distance computation of LDPC codes using a branch and cut algorithm," *IEEE Trans. Commun.*, vol. 58, no. 4, pp. 1072–1079, Apr. 2010.

[14] E. Rosnes and Ø. Ytrehus, "An efficient algorithm to find all small-size stopping sets of low-density parity-check matrices," *IEEE Trans. Inform. Theory*, vol. 55, no. 9, pp. 4167–4178, Sept. 2009.

[15] S. Ländner and O. Milenkovic, "Algorithmic and combinatorial analysis of trapping sets in structured LDPC codes," in *Proc. 2005 Int'l. Conf. Wireless Networks, Commun., Mobile Comp.*, Jun. 2005, pp. 630–635.

[16] H. Pishro-Nik and F. Fekri, "Improved decoding algorithms for low-density parity-check codes," in *Proc. 3rd Int'l. Symp. Turbo Codes and Related Topics*. Brest, France, Aug. 2003.

[17] Z. Zhang, L. Dolecek, B. Nikolic, V. Anantharam, and M. Wainwright, "Lowering LDPC error floors by postprocessing," in *Proc. IEEE Glob. Telecom. Conf.* Cannes, France, Dec. 2008, pp. 1–6.

[18] X. Y. Hu, M. P. C. Fossorier, and E. Eleftheriou, "On the computation of the minimum distance of low-density parity-check codes," in *Proc. IEEE Int'l. Conf. on Commun.* Paris, France, Jun. 2004, pp. 767–771.

[19] D. J. C. MacKay, "Encyclopedia of spase graph codes," [Online]. Available: http://www.inference.phy.cam.ac.uk/mackay/codes/data.html.

[20] *ILOG CPLEX 10.2 User's Manual*. Sunnyvale, CA: ILOG SA, 2007.

[21] X. Hu, M. Fossorier, and E. Eleftheriou, "Regular and irregular progressive edge-growth Tanner graph," *IEEE Trans. Inform. Theory*, vol. 51, no. 1, pp. 386–398, Jan. 2005.

[22] S. K. Chilappagari, S. Sankaranarayanan, and B. Vasic, "Error floors of LDPC codes on the binary symmetric channel," in *Proc. IEEE Int'l. Conf. on Commun.* Istanbul, Turkey, Jun. 2006, pp. 1089–1094.

[23] R. Tanner, D. Sridhara, A. Sridharan, T. Fuja, and D. Costello, "LDPC block and convolutional codes based on circulant matrices," *IEEE Trans. Inform. Theory*, vol. 50, no. 12, pp. 2966–2984, Dec. 2004.

[24] G. A. Margulis, "Explicit constructions of graphs without short cycles," *Combinatorica*, vol. 2, no. 1, pp. 71–78, 1982.