This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE Globecom 2011 proceedings.

Efficient Algorithms to Find All Small Error-Prone Substructures in LDPC Codes

Xiaojie Zhang and Paul H. Siegel
Department of Electrical and Computer Engineering
University of California, San Diego, La Jolla, CA 92093, USA
Email:{ericzhang, psiegel}@ucsd.edu

***Abstract*—Exhaustively enumerating all small error-prone substructures in arbitrary, finite-length low-density parity-check (LDPC) codes has been proven to be NP-complete. In this paper, we present two exhaustive search algorithms to find such small error-prone substructures of an arbitrary LDPC code given its parity-check matrix. One algorithm is guaranteed to find all error-prone substructures including stopping sets, trapping sets, and absorbing sets, which have no more than a_{max} variable nodes and up to b_{max} induced odd-degree neighboring check nodes. The other algorithm is specially designed to find fully absorbing sets (FAS). Numerical results show that both of our proposed algorithms are more efficient in terms of execution time than another recently proposed exhaustive search algorithm [13]. Moreover, by properly initialization of the algorithm, the efficiency can be further improved for quasi-cyclic (QC) codes.**

***Index Terms*—Low-density parity-check (LDPC) codes, exhaustive search, branch-and-bound, error floors, trapping sets.**

I. Introduction

Low-density parity-check (LDPC) codes have been the focus of much research over the past decade thanks to their near Shannon limit performance and to their efficient belief propagation (BP) decoders [1], [2]. However, some properties of LDPC codes are still not fully understood. One of the most important open problems is the phenomenon of the error floor. Roughly speaking, the error floor is an abrupt change in the error-rate performance of a BP decoder in the high SNR region. It is known that the error floor performance of a finite-length LDPC code is dominated by certain error-prone substructures (EPS) within the Tanner graph used to represent the code. For the binary erasure channel (BEC), the class of EPS known as *stopping sets* determine the error floor performance as well as the error-rate performance [3]. For general memoryless binary-input output-symmetric (MBIOS) channels, the EPS that dominate the error floor performance have not yet been fully characterized, although some classes of EPS have been identified and, depending upon the context and performance analysis techniques being used, given names such as *near-codewords* [4], *trapping sets* [5], *absorbing sets*, *fully absorbing sets* [6], and *pseudocodewords* [7].

Generally, the error floor of LDPC codes occurs at a bit error rate (BER) of around 10^{-5} to 10^{-7}, but many important applications, such as data storage, require a very low error floor, in the range of 10^{-12} to 10^{-15}. This makes estimating error floors by Monte Carlo simulation virtually impossible. With importance sampling, the error floor of LDPC codes can be semi-analytically estimated by characterizing and enumerating the trapping sets [5]. Moreover, with even a partial list of trapping sets of an LDPC code, new LDPC codes having a low error floor can be designed [8]; alternatively post-processing decoding can be used to reduce the error floor [9].

In [10], it was proven that exhaustively enumerating all small *k-out trapping sets* in an arbitrary, finite-length LDPC code is NP-complete. This result was extended to different kinds of EPS [11]. In spite of the hardness of finding all EPS, some practical search algorithms have been proposed recently for limited-length LDPC codes. In [12], a non-exhaustive search algorithm based upon the impulse approach was proposed. It can find a portion of the trapping sets efficiently, but can not guarantee a complete enumeration of all the trapping sets. An exhaustive search algorithm based on a *branch-and-bound* approach was proposed in [13], which can find all small-sized fully absorbing sets for LDPC codes with moderate length (< 1000).

In this paper, we introduce two practical, exhaustive search algorithms based upon the branch-and-bound principle which was previously used in [13] and [14]. During the bounding step, we formulate two new and different linear programming (LP) problems for our EPS enumeration algorithm and FAS enumeration algorithm, respectively. By solving the LP problem, we can decide the minimum size of EPS and FAS under given constraints, and thereby bound the search. In comparison to the EPS search algorithm in [12], our EPS enumeration algorithm can find all small EPS rather than only a subset of them. Moreover, as we will show later in this paper, our FAS enumeration algorithm is more efficient than a recently proposed exhaustive FAS search algorithm [13]. Finally, for quasi-cyclic (QC) LDPC codes, we can further improve the efficiency for both algorithms by exploiting the cyclicity properties of the code.

The remainder of the paper is organized as follows. In Section II, we give some basic notation and definitions. Section III describes our proposed exhaustive search algorithms. In Section IV, we apply our algorithms to some well-documented LDPC codes, and compare the efficiency and accuracy with that of other algorithms. Section V concludes the paper.

978-1-4244-9268-8/11/$26.00 ©2011 IEEE

II. Notation and Definitions

We define a linear code of length n by an $m \times n$ parity-check matrix $\mathbf{H}$. Its corresponding Tanner graph has n variable nodes (VN) $V = \{v_1, \ldots, v_n\}$ and m check nodes (CN) $C = \{c_1, \ldots, c_m\}$. Let $N(v)$ and $N(c)$ be the sets of neighboring nodes of a variable node v and a check node c, respectively. Let $d_v = |N(v)|$ denote the degree of variable node v, and let $d_c = |N(c)|$ denote the degree of check node c, where $|\cdot|$ denotes the cardinality of a set.

Several classes of EPS have been identified in the literature. On the BEC, it is stopping sets, clearly defined substructure within the Tanner graph of an LDPC code, that dominate the error-rate performance of the message-passing (MP) decoder: the decoder fails if and only if the set of erased bits contains a stopping set [3]. However, for general MBIOS channels such as the binary symmetric channel (BSC) and the additive white Gaussian noise channel (AWGNC), the exact substructures that cause the error-floor are still not fully understood. The most prevalent term for such a substructure is *trapping set* (TS), which is operationally defined as a subset of variable nodes that is susceptible to errors under iterative decoding. However, this concept depends on both the channel and the decoding algorithm.

To facilitate our discussion, we define the term *error-prone substructure* from a graph-theoretic perspective, independent of the channel and the decoder.

Definition 1 (EPS): A subset of V is an (a, b) *error-prone substructure (EPS)* if in the induced subgraph, there are a variable nodes and b odd degree check nodes as neighbors. Correspondingly, a binary indicator vector $\mathbf{t} \in \{0,1\}^n$ is also called an EPS if it has weight a and its syndrome $\mathbf{s}$ has weight b, where $\mathbf{s} = \mathbf{H}\mathbf{t}$.

Remark 1: Unlike the conventional definition of trapping set, the subgraph corresponding to an EPS is not necessarily a connected graph.

For a set of variable nodes $A \subseteq V$, denote by $E_A \subseteq C$ and $O_A \subseteq C$ the sets of check nodes connected to A an even number or an odd number of times, respectively. Let $E_A(v)$ and $O_A(v)$ be the sets of neighboring check nodes of v in E_A and O_A, respectively.

Definition 2 (AS and FAS [6]): A set of variable nodes $A \subseteq V$ is an *absorbing set* (AS) if $|E_A(v)| > |O_A(v)|$ for all $v \in A$; and a set of variable nodes $A \subseteq V$ is a *fully absorbing set* (FAS) if $|E_A(v)| > |O_A(v)|$ for all $v \in V$. An AS (or FAS) is also called an (a, b) AS (or FAS) where $a = |A|$ and $b = |O_A|$.

From Definitions 1 and 2, it is easy to see that all trapping sets are EPS, and all the absorbing sets and fully absorbing sets defined in [6] are also EPS. Actually, any subset of variable nodes can be considered as an (a, b) EPS, but the EPS of most interest are those with small a and b values, and/or small b/a ratio.

III. Algorithm Description

In the following description of the proposed algorithms, we use an extension of the notion of indicator vector that we call a *candidate vector*. Specifically, a candidate vector $\mathbf{f} = [f_1, \ldots, f_n]^T \in \{0, 1, *\}^n$ is a vector of length n, where 1 and 0 indicate that the corresponding VN is included or excluded from a subset of V, respectively, and $*$ means the position is *unconstrained*, i.e., not yet set to a "0" or "1" value. Let $\mathsf{supp}(\mathbf{f}) = \{i : f_i = 1\}$ be the support set of the candidate vector $\mathbf{f}$, and $\mathrm{wt}(\mathbf{f}) = |\mathsf{supp}(\mathbf{f})|$ be the weight of $\mathbf{f}$. Let $\mathbf{f}|_{p,0}$ and $\mathbf{f}|_{p,1}$ be the candidate vectors obtained by setting the p-th position of $\mathbf{f}$ to 0 and 1 while keeping other positions unchanged, respectively. Let $\mathcal{S}$ be the set of row indexes whose corresponding check nodes have at least one unconstrained neighbor VN and an odd number of 1-valued neighbor VNs, according to $\mathbf{f}$. Finally, let $\mathcal{T}$ be the set of column indexes corresponding to the unconstrained positions of $\mathbf{f}$. Define the operation $\otimes$ between a binary vector $\mathbf{h} = [h_1, \ldots, h_n] \in \{0,1\}^n$ and $\mathbf{f}$ as

$$\mathbf{h} \otimes \mathbf{f} = \begin{cases} 0, & \text{if } \exists i : f_i = * \text{ and } h_i = 1; \\ \displaystyle\sum_{i:h_i=1} h_i f_i \text{ (over } \mathbb{F}_2), & \text{otherwise.} \end{cases}$$

Note that $\mathbf{h} \otimes \mathbf{f} \in \{0, 1\}$. We can extend this operation to a matrix and a vector, i.e., applying $\otimes$ on each row of the matrix, and get a binary vector, such that $\mathbf{H} \otimes \mathbf{f} = \{\mathbf{h}_1 \otimes \mathbf{f}, \ldots, \mathbf{h}_m \otimes \mathbf{f}\}^T \in \{0,1\}^m$ where $\mathbf{h}_i$ is the i-th row of $\mathbf{H}$. Hence, the weight of $\mathbf{H} \otimes \mathbf{f}$ is the number of unsatisfied CNs which have no unconstrained neighbor VNs.

Algorithm 1 Exhaustive Search Algorithm for EPS

Input: parity-check matrix $\mathbf{H}$, integer $a_{\max}$ and $b_{\max}$.
Output: the exhaustive list $\mathcal{L}$ of all (a, b) EPS with $a \le a_{\max}$ and $b \le b_{\max}$.

1: $\mathcal{L} \leftarrow \varnothing$, $\mathsf{STACK} \leftarrow \varnothing$, and push $(*, \ldots, *)$ into STACK.
2: **while** $\mathsf{STACK} \neq \varnothing$ **do**
3: $\quad\mathbf{f} \leftarrow$ pop STACK.
4: $\quad\alpha \leftarrow |\mathsf{supp}(\mathbf{f})|$ and $\beta \leftarrow \mathrm{wt}(\mathbf{H} \otimes \mathbf{f})$.
5: $\quad$**if** $\alpha \le a_{\max}$ and $\beta \le b_{\max}$ **then**
6: $\quad\quad$**if** there is no unconstrained position in $\mathbf{f}$ **then**
7: $\quad\quad\quad\mathcal{L} \leftarrow \mathcal{L} \cup \{\mathsf{supp}(\mathbf{f})\}$
8: $\quad\quad$**else**
9: $\quad\quad\quad$Compute lower bound $\omega(\mathbf{f})$.
10: $\quad\quad\quad$**if** $\alpha + \beta + \omega(\mathbf{f}) \le a_{\max} + b_{\max}$ **then**
11: $\quad\quad\quad\quad$Choose position $p \in \{i : f_i = *\}$.
12: $\quad\quad\quad\quad$Push $\mathbf{f}|_{p,0}$ and $\mathbf{f}|_{p,1}$ into STACK.
13: $\quad\quad\quad$**end if**
14: $\quad\quad$**end if**
15: $\quad$**end if**
16: **end while**

A. EPS Search Algorithm

The proposed exhaustive search procedure for EPS is given in Algorithm 1, which finds all (a, b) EPS such that $a \le a_{\max}$ and $b \le b_{\max}$. The object denoted STACK is a last-in first-out (LIFO) stack that keeps candidate vectors. The tightness of the computed lower bound in line 9 and the selection of an unconstrained position in line 11 determine the number of

candidate sets (or the number of nodes in the binary search tree in the branch-and-bound approach) considered in the algorithm.

1) Computing Lower Bound in line 9 (Bound Step):

The quantity $\omega(\mathbf{f})$ is a lower bound on the minimum increase in the value of $\alpha+\beta$, given all fixed positions in $\mathbf{f}$, where α and β are defined in line 4. Define $\hat{\mathbf{H}} \triangleq [\mathbf{H}]_{\mathcal{S},\mathcal{T}}$, the submatrix of $\mathbf{H}$ consisting of elements in the rows and columns determined by the sets $\mathcal{S}$ and $\mathcal{T}$, respectively. If set $\mathcal{S}$ is empty, we set lower bound $\omega(\mathbf{f})$ to be zero. Let $\mathbf{x}=\{x_1,\ldots,x_{|\mathcal{T}|}\}$ be the optimization variables. The lower bound on the increase of $\alpha+\beta$ can be found by solving the following integer programming (IP) problem:

$$\begin{aligned} \min \quad & \sum_{i\in\mathcal{T}} x_i \\ \text{s. t.} \quad & \hat{\mathbf{H}}\mathbf{x} \geq (1,1,\ldots,1)^T, \\ & x_i \in \{0,1\}. \end{aligned} \tag{1}$$

The constraint $\hat{\mathbf{H}}\mathbf{x} \geq (1,1,\ldots,1)^T$ arises from the fact that each row of $\hat{\mathbf{H}}$ corresponds to an unsatisfied CN which has at least one unconstrained neighbor VN. Consider an unsatisfied CN which has only one unconstrained neighbor VN. Such a CN will increase the value of $\alpha+\beta$ by one, because if we include this VN into the set the value of α will increase by one, and if we do not include it, the value of β will increase by one. Therefore, the purpose of the IP problem above is to find a lower bound of the minimum number of unconstrained VNs that could eliminate all the unsatisfied CNs in $\mathcal{S}$ if they are set to the value 1.

The IP problem (1) can be relaxed by allowing x_i to be a nonnegative real value, transforming the optimization problem into an LP problem:

$$\begin{aligned} \min \quad & \sum_{i\in\mathcal{T}} x_i \\ \text{s. t.} \quad & \hat{\mathbf{H}}\mathbf{x} \geq (1,1,\ldots,1)^T, \\ & \mathbf{x} \geq 0. \end{aligned} \tag{2}$$

Therefore, the lower bound $\omega(\mathbf{f})$ is the optimal value of the objective function of LP problem (2), i.e., $\omega(\mathbf{f}) = \sum_{i\in\mathcal{T}} x_i^* = \sum_{i\in\mathcal{T}^+} x_i^*$, where $\mathcal{T}^+ = \{i : i\in\mathcal{T}, x_i^* > 0\}$.

2) Position Selection in line 11 (Branch Step):

Following [14], the principle of the position selection in line 11 is to find a VN that has the largest number of neighbor CNs connected to the smallest number of unconstrained VNs. Therefore, we choose position $p\in\mathcal{T}^+$ such that

$$(N_p(1),N_p(2),\ldots,N_p(d_p)) \succeq (N_q(1),N_q(2),\ldots,N_q(d_q)) \tag{3}$$

for all $q\in\mathcal{T}^+$ and $q\neq p$, where $N_q(k)$ is the number of neighboring CNs of the q-th VN that are connected to k unconstrained VNs, and $\succeq$ denotes *lexicographical order* under which $(x_1,x_2,\ldots,x_l)\succeq(y_1,y_2,\ldots,y_l)$ if $x_j > y_j$ and $x_i = y_i$ for $1\leq i\leq j-1$, or $x_i=y_i$ for $1\leq i\leq l$. If $d_p\neq d_q$, the vector in (3) corresponding to the smaller of the two is padded with zeros. If there is more than one position p satisfying (3), then we choose the one with greater column weight in $\hat{\mathbf{H}}\mathbf{x}$. If $\mathcal{S}=\emptyset$, we choose an unconstrained position with maximum column weight in $\mathbf{H}$.

Algorithm 2 Exhaustive Search Algorithm for FAS

Input: parity-check matrix $\mathbf{H}$, integer $a_{\max}$ and $b_{\max}$.
Output: the exhaustive list $\mathcal{L}$ of all (a,b) FAS with $a\leq a_{\max}$ and $b\leq b_{\max}$.

1: $\mathcal{L}\leftarrow\varnothing$, STACK $\leftarrow\varnothing$, and push $(*,\ldots,*)$ into STACK.
2: **while** STACK $\neq\varnothing$ **do**
3: $\quad\mathbf{f}\leftarrow$ pop STACK.
4: $\quad\alpha\leftarrow|\mathsf{supp}(\mathbf{f})|$ and $\beta\leftarrow\mathrm{wt}(\mathbf{H}\otimes\mathbf{f})$.
5: $\quad$**if** $\alpha\leq a_{\max}$ and $\beta\leq b_{\max}$ **then**
6: $\quad\quad$**if** there is no unconstrained position in $\mathbf{f}$ **then**
7: $\quad\quad\quad\mathcal{L}\leftarrow\mathcal{L}\cup\{\mathsf{supp}(\mathbf{f})\}$
8: $\quad\quad$**else**
9: $\quad\quad\quad$Compute lower bound $\mu(\mathbf{f})$.
10: $\quad\quad\quad$**if** $\alpha+\mu(\mathbf{f})\leq a_{\max}$ **then**
11: $\quad\quad\quad\quad$Choose position $p\in\{i : f_i=*\}$.
12: $\quad\quad\quad\quad$Push $\mathbf{f}|_{p,0}$ and $\mathbf{f}|_{p,1}$ into STACK.
13: $\quad\quad\quad$**end if**
14: $\quad\quad$**end if**
15: $\quad$**end if**
16: **end while**

As we noted previously, our definition of EPS may include unconnected subgraphs especially when b_{max} increases. However, since most EPS of interested are of small b where the number of such "degenerated" substructures is small if they exist, it is easy to add an additional step after each EPS being found to filter out these undesired substructures, and the computational complexity of such filtering is negligible.

B. FAS Search Algorithm

The exhaustive search procedure for FAS is given in Algorithm 2, which is based on the EPS search algorithm. The key differences are in line 9 and line 10, where we compute and apply a lower bound, $\mu(\mathbf{f})$, on the minimum number of unconstrained positions in $\mathbf{f}$ that should be set to 1 in order to get a valid FAS.

In line 9, before computing the lower bound, we first check the unsatisfied CNs with all known neighboring VNs to see whether the subgraph is a FAS, i.e., no VN has more unsatisfied neighboring CNs than satisfied neighboring CNs. If the definition of FAS has already been violated by $\mathbf{f}$, the lower bound $\mu(\mathbf{f})$ is set to be $a_{\max}$, implying that the condition in line 10 will not hold; otherwise, the following approach is applied to compute the lower bound.

Let $C_1(\mathbf{f})$ be the set of all unsatisfied CNs represented by $\mathbf{f}$, i.e., the set of CNs that connect to VNs in $\mathsf{supp}(\mathbf{f})$ an odd number of times. Let $U_k \triangleq N(k)\cap C_1(\mathbf{f})$ be the set of unsatisfied neighboring CNs of the k-th VN. The lower bound

$\mu(\mathbf{f})$ can be found by solving the following LP problem:

$$\begin{aligned} \min \quad & \sum_{i\in\mathcal{T}} x_i \\ \text{s. t.} \quad & \theta(x_k) + \sum_{j\in U_k} \sum_{i\in N(j)\cap\mathcal{T}/\{k\}} x_i \\ & \geq 1 + |U_k| - \lceil \frac{d_k}{2} \rceil \quad \text{for all } k \in V, \qquad (4) \\ & x_i \geq 0 \quad \text{for all } i \in \mathcal{T} \end{aligned}$$

where $\theta(x_k)$ is defined as

$$\theta(x_k) = \begin{cases} (2|U_k| - d_k)\, x_k, & \text{if } k \in \mathcal{T}; \\ 0, & \text{otherwise.} \end{cases}$$

Note that we can remove the constraints in (4) if the right-hand side is less than or equal to zero. The optimal value of the LP problem is the lower bound, i.e., $\mu(\mathbf{f}) = \sum_{i\in\mathcal{T}+} x_i^*$, and the subsequent position selection step in line 11 is the same as in Algorithm 1.

From the definition of FAS, each VN has to connect to more satisfied neighboring CNs than the unsatisfied. Given candidate vector $\mathbf{f}$ and a VN position k, $1 \leq k \leq n$, the VN has $|U_k|$ unsatisfied neighbor CNs, and we have to have

$$d_k - |U_k| > |U_k| \qquad (5)$$

in order to get a valid FAS. If this inequality does not hold for a VN, we have to set itself and/or some of its neighboring VNs (2-step neighbors on the Tanner graph) to one to let the inequality hold. The LP problem in (4) gives a lower bound on the minimum number of unconstrained positions in $\mathbf{f}$ needed to further be set to one in order to have (5) hold for every VN.

By comparing Algorithm 1 and Algorithm 2, we can see that the value of $a_{\max} + b_{\max}$ dominates the computational complexity of Algorithm 1 in the search for general EPS, whereas in Algorithm 2, where we restrict the search to FAS, it is $a_{\max}$ alone that determines the running time.

C. The Exhaustiveness of Proposed Algorithms

The searches for EPS and FAS preformed in Algorithm 1 and Algorithm 2 can be seen as a search on a binary tree, whose root is the first VN position that the algorithms pick to start with. If no branch-and-bound is performed, the leaves of the binary tree are all 2^n binary vectors of length n. The proposed exhaustive search algorithms traverse the binary tree in an efficient way such that they compute the lower bound on $a+b$ for EPS or a for FAS on all children of the current node. If the lower bound on such parameters exceeds thresholds on a certain node, the algorithms cut the branch from this node off the binary tree. With branch-and-bound, the search algorithms avoid going through all 2^n leaves of the binary tree, but still traverse the whole binary tree; therefore, our proposed algorithms is an exhaustive search. Note that, using different ways to compute the lower bounds and to select next position p does not affect the exhaustiveness of the search algorithm, it only affects the efficiency of the search.

TABLE I
(a, b) EPS ENUMERATORS FOR THE (155,64) TANNER CODE

(a,b)	count	(a,b)	count	(a,b)	count
(5,3)	155	(6,4)	2790	(7,3)	930
(8,2)	465	(8,4)	14415	(9,3)	5580
(10,2)	1395	(10,4)	83235	(11,3)	17360
(12,2)	930	(12,4)	36280	(13,3)	43245

D. Efficiency Improvement for QC Codes

Consider a vector of length βt where both β and t are positive integers,

$$\mathbf{v} = (\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_t) = (v_{1,1}, \ldots, v_{1,\beta}, \ldots, v_{t,1}, \ldots, v_{t,\beta}).$$

Let $\mathbf{v}_i^{(l)} = (v_{i,\beta-l+1}, \ldots, v_{i,\beta-l})$ be the (right) cyclic-shift of $\mathbf{v}_i$ by l positions. The vector $\mathbf{v}^{[l]} = (\mathbf{v}_1^{(l)}, \ldots, \mathbf{v}_t^{(l)})$ is called the *t-section* cyclic-shift of $\mathbf{v}$.

Definition 3 (QC codes): Let β and t be positive integers. A linear block code $\mathcal{C}_{qc}$ of length βt is called a *quasi-cyclic (QC)* code if the following conditions hold: (1) each codeword consists of t sections of β bits each; and (2) every t-section cyclic-shift of a codeword in $\mathcal{C}_{qc}$ is also a codeword in $\mathcal{C}_{qc}$. Such a QC code is also called a *t-section* QC code.

From the definition above, we can see that if an indicator vector $\mathbf{s}$ of length βt corresponds to an (a, b) EPS (or FAS) of a t-section QC code, the t-section cyclic-shift of $\mathbf{s}$ is also an (a, b) EPS (or FAS). In this case, we can initialize the STACK with t candidate vectors, $\mathbf{f}^i = (f_1^i, \ldots, f_{\beta t}^i)$, $0 \leq i \leq t-1$, where

$$f_j^i = \begin{cases} 0, & \text{if } j = k\beta + l + 1,\ 0 \leq k \leq i-1,\ 0 \leq l \leq \lambda \\ 1, & \text{if } j = i\beta + \lambda + 1 \\ *, & \text{otherwise} \end{cases}$$

and $\lambda \triangleq \max\left(\lceil \frac{\beta}{a_{\max}} \rceil - 1, 0\right)$.

Moreover, we can further reduce the search space by initialize the STACK with more properly designed candidate vectors. The EPS (or FAS) found using such an initialization are then quasi-cyclically shifted and the number of distinct ones are counted. In comparison to the initialization proposed in [14], our method generates initialization vectors with more fixed positions, and therefore, it more efficiently reduces the search space.

IV. NUMERICAL RESULTS

In this section, we provide some numerical results for several well-documented LDPC codes. By comparing our results with existing results in the literature, we demonstrate the completeness and efficiency of our proposed search algorithms.

A. Results of EPS Search Algorithm

1) The (3,5)-Regular (155,64) QC Tanner Code [15]:

In Table I, we list the number all (a, b) EPS where $a \leq 13$ and $b \leq 4$. To demonstrate the efficiency of the proposed algorithm, consider the case of finding all EPS for $a_{\max} = 9$

TABLE II
(a, b) EPS ENUMERATORS FOR THE M816 CODE

(a, b)	EPS	FAS	analytical search [17]
(3,3)	132	126	132
(4,2)	3	3	0
(4,4)	3459	1350	1372
(5,3)	120	86	41

TABLE III
(a, b) EPS ENUMERATORS FOR THE M1008 CODE

(a, b)	EPS	FAS	analytical search [17]
(3,3)	165	153	165
(4,2)	6	6	0
(4,4)	3701	1130	1215
(5,3)	160	92	14

and $b_{\max} = 3$. With a brute force exhaustive search, we would be required to check $\sum_{i=1}^{9} \binom{155}{i} \approx 1.2 \times 10^{14}$ sets in order to find all EPS of size up to 9. In contrast, the proposed algorithm only searchers through 6.8×10^6 candidate vectors if initialized with an empty STACK. If we further take advantage of the QC property of the Tanner (155,64) code and initialize the STACK with 5 candidate vectors, the number of candidate vectors the algorithm has to consider drops to 6.0×10^5.

2) The (3,6)-Regular (816,408) M816 and (1008,504) M1008 LDPC Codes [16]:

M816 and M1008 are two random (3,6)-regular LDPC codes generated by MacKay. In Table II and Table III, we compare the number of EPS and FAS ($a_{\max} \leq 5$) found by our proposed algorithms with the number of trapping sets found by the analytical search method in [17] for the M816 and M1008 code, respectively. According to the definition of trapping set in [17] (i.e., if an error vector stays the same after one iteration of Gallager B decoding, it corresponds to a trapping set), although all EPS in the tables above can not be correctly decoded by the Gallager B decoder, some of them are not trapping sets since they decode to error vectors corresponding to trapping sets of other sizes. Interestingly, for the parameters listed in Table II and Table III, the FAS correspond precisely to the set of all trapping sets. The analytical method in [17], which counts cycles and cycle interactions on the Tanner graph, is intended to enumerate all the trapping sets, but as can be seen, the results obtained are not always accurate.

3) The (256, 128) CCSDS QC-LDPC Code [12]:

Table IV compares the number of EPS found by our algorithm with that found by a non-exhaustive approach proposed in [12], which tries to find as many EPS as possible. This code has irregular variable degree, such that some VNs are of degree 3 and others have degree 5. We can see that the non-exhaustive approach works well when searching for EPS with small a and b, but the accuracy decreases dramatically as the size of EPS increases.

TABLE IV
(a, b) EPS ENUMERATORS FOR (256,128) LDPC CODE

(a, b)	exhaustive	non-exhaustive [12]
(5,3)	32	32
(7,3)	32	32
(8,2)	32	32
(8,4)	544	540
(9,3)	192	191
(10,4)	2304	1600
(11,3)	128	117
(12,2)	32	32
(12,4)	6304	1645
(13,3)	864	457
(14,2)	96	91
(14,4)	28896	2466
(19,1)	64	38

TABLE V
SIMULATION TIME COMPARISON FOR FAS OF PEGR504 CODE
($a_{\max} = b_{\max} = 5$)

Algorithm 1	Algorithm 2	EFSA [13]
59 min	3 min	6 hr 55 min

B. Results of FAS Search Algorithm

Due to space limitations and the fact that the FAS form a subset of EPS, we do not include here the number of FAS found using our exhaustive search algorithm. We provide instead only some numerical results that demonstrate the efficiency of our proposed algorithms. The algorithms were implemented with VC++ and GLPK as the LP solver [18]. Note that we *do not* use any coding optimization techniques to accelerate the execution of our C++ code. The running times were obtained on a standard desktop PC with a 2.67GHz processor.

For the comparison, we examined the PEGR504 code, which is a (3,6)-regular (504,252) LDPC code taken from [16]. We found the same number of FAS for this code as was reported in [13]. Table V compares the running times of our proposed algorithms with that of the exhaustive FAS search algorithm (EFSA) in [13] for the case where $a_{\max} = b_{\max} = 5$. We can see that both of our proposed algorithms are significantly faster than EFSA. Algorithm 2, in particular, reduced the execution time by more than two orders of magnitude. The improvement on efficiency comes from two parts. One is that our algorithms have a more efficient ways to estimate the lower bound and to select next constraint position. The other is that, when computing the lower bound, our algorithms only need to solve an LP problem which can be efficiently solved, but EFSA has to solve an IP problem which is generally more computational complicated and requires more CPU running time. Note that the PEGR504 code is not quasi-cyclic, so we could not take the advantage of QC codes as described in Section III-D. However, we can expect even more efficiency improvement with our proposed

algorithms for QC-LDPC codes.

V. Conclusion

In this paper, we proposed efficient, exhaustive search algorithms for EPS and FAS. By properly initializing the algorithms, we further improved the execution-time efficiency for QC-LDPC codes. A comparison of our results to those obtained with previously proposed full and partial search algorithms confirms the relative efficiency of our algorithms, while identifying weaknesses in some of the others.

Acknowledgment

This work was supported in part by the Center for Magnetic Recoding Research at University of California, San Diego and by the National Science Foundation under Grant CCF-0829865.

References

[1] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, vol. 8, pp. 21-28, Jan. 1962.

[2] D. J. C. MacKay and R. M. Neal, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, vol. 45, no. 2, pp. 399-431, Mar. 1999.

[3] C. Di, D. Proietti, E. Telatar, T. Richardson, and R. Urbanke, "Finite length analysis of low-density parity-check codes on the binary erasure channel," *IEEE Trans. Inf. Theory*, vol. 48, no. 6, pp. 1570-1579, Jun. 2002.

[4] D. MacKay and M. Postol, "Weakness of Margulis and Ramanujan-Margulis low-density parity check codes," *Electron. Notes Theor. Comp. Sci.*, vol. 74, 2003.

[5] T. Richardson, "Error-floors of LDPC codes," *Proc. of the 41st Annual Allerton Conference on Communication, Control, and Computing, Monticello*, IL, Oct. 1-3, 2003, pp. 1426-1435.

[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] P. O. Vontobel and R. Koetter, "Graph-cover decoding and finite-length analysis of message-passing iterative decoding of LDPC codes," *IEEE Trans. Inform. Theory*, accepted for publication.

[8] M. Ivkovic, S. K. Chilappagari, and B. Vasic, "Eliminating trapping sets in low-density parity-check codes by using Tanner graph covers," *IEEE Trans. Inform. Theory*, vol. 54, no. 8, pp. 3763-3768, Aug. 2008.

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

[10] 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. 1979-1999, May 2009.

[11] 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.

[12] S. Abu-Surra, D. DeClerq, D. Divsalar, and W. Ryan, "Trapping set enumerators for specific LDPC codes," *Information Theory and Applications Workshop (ITA)*, La Jolla, CA, Jan. 31-Feb. 5, 2010.

[13] G. Kyung and C. Wang, "Exhaustive search for small fully absorbing sets and the corresponding low error-floor decoder," in *Proc. IEEE Int. Symp. Inform. Theory (ISIT)*, Austin, TX, Jul. 2010, pp. 739-743.

[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. 4167-178, Sep. 2009.

[15] R. M. Tanner, D. Sridhara, and T. Fuja, "A class of group-structured LDPC codes," in *Proc. Int. Symp. Communication Theory and Applications (ISCTA)*, Ambleside, U.K., Jul. 2001.

[16] D. J. C. MacKay, *Encyclopedia of Sparse Graph Codes*. [Online]. Available: http://www.inference.phy.cam.ac.uk/mackay/codes/data.html

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

[18] GNU Linear Programming Kit, http://www.gnu.org/software/glpk