An Efficient Algorithm for Finding Dominant Trapping Sets of LDPC Codes*

Mehdi Karimi, *Student Member, IEEE* and Amir H. Banihashemi, *Senior Member, IEEE*

Abstract

This paper presents an efficient algorithm for finding the dominant trapping sets of a low-density parity-check (LDPC) code. The algorithm can be used to estimate the error floor of LDPC codes or to be used as a tool to design LDPC codes with low error floors. For regular codes, the algorithm is initiated with a set of short cycles as the input. For irregular codes, in addition to short cycles, variable nodes with low degree and cycles with low approximate cycle extrinsic message degree (ACE) are also used as the initial inputs. The initial inputs are then expanded recursively to dominant trapping sets of increasing size. At the core of the algorithm lies the analysis of the graphical structure of dominant trapping sets and the relationship of such structures to short cycles, low-degree variable nodes and cycles with low ACE. The algorithm is universal in the sense that it can be used for an arbitrary graph and that it can be tailored to find a variety of graphical objects, such as absorbing sets and Zyablov-Pinsker (ZP) trapping sets, known to dominate the performance of LDPC codes in the error floor region over different channels and for different iterative decoding algorithms. Simulation results on several LDPC codes demonstrate the accuracy and efficiency of the proposed algorithm. In particular, the algorithm is significantly faster than the existing search algorithms for dominant trapping sets.

I. INTRODUCTION

ESTIMATING the error floor performance of low-density parity-check (LDPC) codes under iterative message-passing decoding, and the design of LDPC codes with low error floors have attracted a great amount of interest in recent years. The performance of LDPC codes under iterative decoding algorithms in the error floor region is closely related to the structure of the code's Tanner graph. For the binary erasure channel (BEC), the problematic structures are *stopping sets* [10]. In the case of the binary symmetric channel (BSC) and the additive white Gaussian noise (AWGN) channel, the error-prone patterns are called *trapping sets* [30], *near codewords* [24] or *pseudo codewords* [38]. Among the trapping sets, the so-called *elementary trapping sets* are shown to be the main culprits [30], [17], [7], [26], [16], [48]. Related to this, it is demonstrated in [8] that for some structured LDPC codes decoded by iterative algorithms over the AWGN channel, a subset of trapping sets, called *absorbing sets*, determine the error

---

*A preliminary version of part of this work appeared in Proc. *6th International Symposium on Turbo Codes & Iterative Information Processing,* Brest, France, Sept. 6 - 10, 2010.

floor performance. In fact, in an overwhelmingly large number of cases, dominant absorbing sets appear to be elementary trapping sets.

For a given LDPC code, the knowledge of dominant trapping sets is important. On one hand, efficient methods for estimating the error floor of an LDPC code, which rely on the importance sampling technique, operate by biasing the noise toward the dominant trapping sets of the code, see, e.g., [7]. On the other hand, by knowing the dominant trapping sets, several decoder modifications can be applied to improve the error floor performance (see, e.g., [4], [11]). Furthermore, the knowledge of dominant trapping sets can be used to design LDPC codes with low error floor. Related to this, Ivkovic *et al.* [14] applied the technique of edge swapping between two copies of a base LDPC code to eliminate the dominant trapping sets of the base code over the BSC. This was then generalized by Asvadi *et al.* [1] to cyclic liftings of higher degree to construct quasi-cyclic LDPC codes with low error floor. While the knowledge of the problematic sets that dominate the error floor performance is most helpful in the design and analysis of LDPC codes, attaining such knowledge, regardless of differences in the graphical structure of these sets, is a hard problem. For instance, it was shown in [27], [20], [23] that the problem of finding a minimum size stopping set is NP hard. McGregor and Milenkovic [23] showed that not only the problem of finding a minimum size trapping set, but also the problem of approximating the size of a minimal trapping set is NP hard, regardless of the sparsity of the underlying graph.

It should be noted that while the majority of the literature on estimating the error floor of LDPC codes rely on finding the dominant trapping sets, as an eventual result of decoder failure, in [42], [44], [43], Xiao and Banihashemi took a different approach. By focussing on the input error patterns that cause the decoder to fail, they developed a simple technique to estimate the frame error rate (FER) and the bit error rate (BER) of finite-length LDPC codes over the BSC [42]. The complexity of this technique was then reduced in [44], and the estimation technique was extended to the AWGN channel with quantized output in [43]. In this work, unlike [42], [44], [43], we are particularly interested in the examination of the graphical structure of the problematic sets which dominate the error floor performance. This information is then used to efficiently search for these sets.

The complexity of the exhaustive brute force search method for finding problematic structures of size t in a code of length n becomes quickly infeasible as n and t increase. Efficient search algorithms have been devised to find small (dominant) stopping and trapping sets [40], [7], [32], [44], [41], [2], [21]. The reach of these algorithms however is still very limited. For example, the complexity of the algorithm of [40], [41] is only affordable for codes with lengths up to ~ 500. Even for these lengths, the algorithm

can only find trapping sets of maximum size 11 with only one or two unsatisfied check nodes. This is while for many codes, some of the dominant trapping sets may have larger size and/or more than two unsatisfied check nodes. In [37] and [29], the authors proposed to build a database of all possible configurations for trapping sets of different sizes in a graph with specific degree distribution and girth. They then used a parent-child relationship between the trapping sets of different sizes to simplify the search of the larger trapping sets. This method was used to find the dominant trapping sets of left regular LDPC codes with left degree 3 [37]. The method proposed in [37] however becomes very complex when the degree of variable nodes, and in turn the number of possible configurations increases. The application of this method becomes even more difficult when dealing with irregular LDPC codes as for such codes, there may be a large number of possible configurations for each type of trapping set, due to the variety of variable node degrees. Even for the left regular graphs with small left degrees, the number of possible configurations becomes quite large for the larger trapping sets. It is therefore important to look for more efficient algorithms to find the problematic structures that dominate the error floor performance of LDPC codes.

In this paper, we study the problematic graphical structures that dominate the error floor performance of LDPC codes, collectively referred to as trapping sets, and demonstrate that they all contain at least one short cycle (with a small exception of some of the trapping sets of irregular LDPC codes with degree-2 variable nodes). By examining the relationships between cycles and trapping sets, we devise an efficient algorithm to find dominant trapping sets of an LDPC code. The algorithm is initiated by a set of short cycles as input. Each cycle is then expanded recursively to trapping sets of increasing size in a conservative fashion, i.e., the expanded sets all have the smallest size larger than the size of the current set, and each of them will be used as a new input to the next step of the algorithm. It should be mentioned that although our algorithm uses the topological relationships between the small trapping sets and the larger ones, it is different from the method of [37] in several ways. Our algorithm is not based on the knowledge of the exact structure of trapping sets, and hence does not need to build a database. In fact, instead of checking all the possible configurations to find the existing ones in a graph, it directly and efficiently finds those existing configurations, and so it is much faster. Moreover, unlike the method of [37], our algorithm uses a general framework for all the degree distributions and girths (with a small exception of some of the trapping sets of irregular LDPC codes with degree-2 variable nodes). The proposed algorithm is applicable to any Tanner graph (structured or random) and can be tailored to find a variety of graphical structures, such as elementary trapping sets and absorbing sets among others. For structured graphs, such as those of

quasi-cyclic or protograph codes, one can use the existing automorphisms in the graph to further simplify the search. Results on several LDPC codes verify the high efficiency and accuracy of the algorithm. For example, for the tested codes, the search speed is improved by a factor of 10 to 100 compared to the methods of [2] and [7].

The remainder of this paper is organized as follows. Basic definitions and notations are provided in Section II. In Section III, we develop the proposed algorithm. Section IV presents the modification needed for irregular LDPC codes. Section V includes some numerical results. Section VI concludes the paper.

II. DEFINITIONS AND NOTATIONS

Let $G = (L \cup R, E)$ be the bipartite graph, or Tanner graph, corresponding to the LDPC code $\mathcal{C}$, where L is the set of variable nodes, R is the set of check nodes and E is the set of edges. The notations L and R refer to "left" and "right", respectively, pointing to the side of the bipartite graph where variable nodes and check nodes are located, respectively. The degree of a node $v \in L$ (*or* R) is denoted by $d(v)$. For a subset $\mathcal{S} \subset L$, $\Gamma(\mathcal{S})$ denotes the set of neighbors of $\mathcal{S}$ in R. The *induced subgraph* of $\mathcal{S}$, represented by $G(\mathcal{S})$, is the graph containing nodes $\mathcal{S} \cup \Gamma(\mathcal{S})$ with edges $\{(u, v) \in E : u \in \mathcal{S}, v \in \Gamma(\mathcal{S})\}$. The set of check nodes in $\Gamma(\mathcal{S})$ with odd degree in $G(\mathcal{S})$ is denoted by $\Gamma_o(\mathcal{S})$. Similarly, $\Gamma_e(\mathcal{S})$ represents the set of check nodes in $\Gamma(\mathcal{S})$ with even degree in $G(\mathcal{S})$. The subgraph resulting from removing the nodes of $\Gamma_o(\mathcal{S})$ and their edges from $G(\mathcal{S})$ is denoted by $G'(\mathcal{S})$. In this paper, we interchangeably use the terms *satisfied check nodes* and *unsatisfied check nodes* to denote the check nodes in $\Gamma_e(\mathcal{S})$ and $\Gamma_o(\mathcal{S})$, respectively. Given a Tanner graph $G = (L \cup R, E)$, the following objects play an important role in the error floor performance of the corresponding LDPC code:

Definition 1:

i) A set $\mathcal{S} \subset L$ is an (a, b) *trapping set* if $|\mathcal{S}| = a$ and $|\Gamma_o(\mathcal{S})| = b$. The integer a is referred to as the *size* of the trapping set $\mathcal{S}$.

ii) An (a, b) trapping set $\mathcal{S}$ is called *elementary* if all the check nodes in $G(\mathcal{S})$ have degree one or two.

iii) A set $\mathcal{S} \subset L$ is an (a, b) *absorbing set* if $\mathcal{S}$ is an (a, b) trapping set and if all the nodes in $\mathcal{S}$ are connected to more nodes in $\Gamma_e(\mathcal{S})$ than to nodes in $\Gamma_o(\mathcal{S})$.

iv) A set $\mathcal{S} \subset L$ is an (a, b) *fully absorbing set* if $\mathcal{S}$ is an (a, b) absorbing set and if all the nodes in $L \backslash \mathcal{S}$ have strictly more neighbors in $R \backslash \Gamma_o(\mathcal{S})$ than in $\Gamma_o(\mathcal{S})$.

v) A set $\mathcal{S} \subset L$ is a k-*out trapping* set [40] if $\Gamma_o(\mathcal{S})$ contains exactly k nodes of degree one in $G(\mathcal{S})$.

vi) Let $G = (L \cup R, E)$ be a left-regular bipartite graph with left degree l. A set $\mathcal{S} \subset L$ is a *Zyablov-Pinsker (ZP) trapping set* [23] if every node of $\mathcal{S}$ is connected to less than $l - \lfloor (l-1)/2 \rfloor$ nodes in

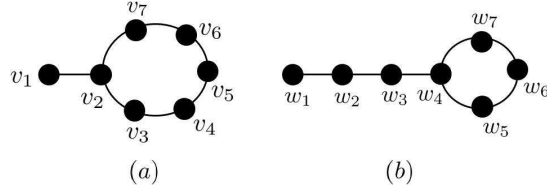Fig. 1. Two lollipop walks of length 7.

$\Gamma_o(\mathcal{S})$.

The ZP trapping sets are the trapping sets of the Zyablov-Pinsker bit-flipping algorithm [49] over the BSC [23]. It should also be noted that for odd values of l, the definitions of ZP trapping sets and absorbing sets are identical.

It is important to note that Definitions 1(ii) – 1(vi) are all special cases of a trapping set in Definition 1(i). In the rest of the paper, therefore, we collectively refer to them as trapping sets. Distinctions will be made as necessary. Trapping sets with smaller values of a and b are generally believed to be more harmful to iterative decoding. Loosely speaking, such trapping sets are called *dominant*. To measure how harmful a trapping set really is, one can use techniques such as importance sampling [7] to measure the contribution of the trapping set to the error floor. This contribution and the dominance of a trapping set (compared to others) would also depend on the channel model and the iterative decoding algorithm, as well as the detailed structure of the Tanner graph (not just the values of a and b).

In a graph $G = (V, E)$ with the set of nodes V and the set of edges E, a *lollipop walk* of length k is defined as a sequence of nodes $v_1, v_2, \ldots, v_{k+1}$ in V such that $v_1, v_2, \ldots, v_k$ are distinct, $v_{k+1} = v_m$ for some $m \in [2, k]$, and $(v_i, v_{i+1}) \in E$ for all $i \in \{1, \ldots, k\}$. Fig. 1 shows two lollipop walks of length 7. The lollipop walk in Fig. 1(a) is represented as $v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_2$. A *cycle* can be considered as a special lollipop walk if the definition is extended to $m = 1$. The length of the shortest cycle in a graph G is denoted by g and is called the *girth* of G.

III. DEVELOPMENT OF THE PROPOSED ALGORITHM

A. Graphical Structure of Trapping Sets

Without loss of generality, we assume that the induced subgraph of a trapping set is connected. Disconnected trapping sets can be considered as the union of connected ones. Moreover, to the best of our knowledge, almost all the structures reported as dominant trapping sets (of regular LDPC codes) in the literature have the property that every variable node is connected to at least two satisfied check nodes in the induced subgraph. We thus focus on trapping sets with this property except for irregular LDPC
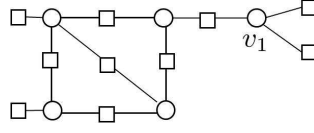
Fig. 2. The induced subgraph of a trapping set.

codes, where we relax this condition for degree-2 variable nodes. As an example, the subgraph in Fig. 2 does not satisfy this condition. (Variable nodes and check nodes are represented by circles and squares, respectively.) Removal of node v_1 and its edges however makes the subgraph satisfy the condition. The following lemma proves this property for certain absorbing and ZP trapping sets.

Lemma 1: Suppose that $\mathcal{S} \subset L$ is an absorbing set (ZP trapping set) in $G = (L \cup R, E)$, and that for all variable nodes $v \in \mathcal{S}$, we have $d(v) \geq 2$ ($d(v) \geq 3$). Then each variable node $v \in \mathcal{S}$ is connected to at least two satisfied check nodes in $G(\mathcal{S})$.

Proof: The proof follows from the definition of absorbing and ZP trapping sets. ∎

For small trapping sets, which dominate the error floor performance, it is unlikely to see check nodes of degree larger than 2 in their subgraphs, i.e., most of the dominant trapping sets are elementary [7], [30]. Related to this, almost all the trapping sets reported as the dominant trapping sets of practical LDPC codes are elementary. In fact, it can be shown that the sizes of non-elementary trapping sets for left-regular graphs are generally larger than those of elementary ones (cf. Lemma 7 in Appendix A).

Example 1: For left-regular LDPC codes with left degree 4 and girths 6, 8 and 10, lower bounds on the sizes of non-elementary trapping sets $\mathcal{S}$ with less than 3 unsatisfied check nodes and with at least one satisfied check node of degree larger than 2 in $G(\mathcal{S})$, are 7, 14 and 22, respectively (based on Lemma 7 in Appendix A, and by choosing $b = 2$). Moreover, for the same conditions, the minimum sizes of non-elementary trapping sets $\mathcal{S}$ with at least one unsatisfied check node of degree larger than 1 in $G(\mathcal{S})$ (and without satisfied check nodes of degree larger than 2 in $G(\mathcal{S})$) are at least 5, 11 and 17, respectively. This is while for the same scenario, the code can have elementary trapping sets of size 5, 8 and 17, respectively.

In the following, we develop our search algorithm mainly for elementary trapping sets, and then present simple modifications to tailor the algorithm to find non-elementary trapping sets.

In the rest of the paper, we use the notation $\mathcal{T}$ to denote the set of all trapping sets $\mathcal{S}$ in a graph G whose induced subgraph $G(\mathcal{S})$ is connected and for which every node $v \in \mathcal{S}$ is connected to at least two nodes in $\Gamma_e(\mathcal{S})$. Notation $\mathcal{T}^a$ is used for the set of all elements in $\mathcal{T}$ with size a and $\mathcal{T}_\mathcal{S}$ denotes the set of all elements in $\mathcal{T}$ that contain the set $\mathcal{S}$. Naturally, $\mathcal{T}_\mathcal{S}^a$ denotes the set of all elements in $\mathcal{T}$ of size a
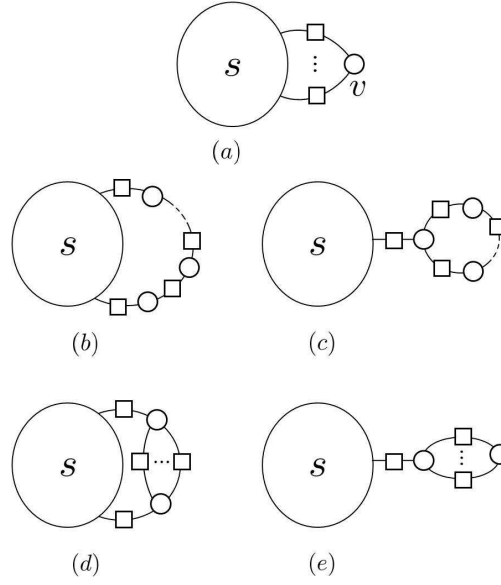
Fig. 3. Possible expansions of an elementary trapping set $\mathcal{S}$ to a larger elementary trapping set $\mathcal{S}'$. (Unsatisfied check nodes of $G(\mathcal{S}')$ are not shown.)

that contain the set $\mathcal{S}$. In the following, we also assume that the Tanner graph G has no parallel edges and no node of degree less than 2.

B. Expansion of Elementary Trapping Sets

The main idea of the proposed algorithm is to start from a relatively small set of small elementary trapping sets, which are easy to enumerate, and then recursively expand them to larger elementary trapping sets. To achieve this, we first characterize the expansion of an elementary trapping set to a larger elementary trapping set through the following lemmas.

Lemma 2: Let $\mathcal{S}$ be an elementary trapping set of size a in $\mathcal{T}$. Then for each elementary trapping set $\mathcal{S}' \in \mathcal{T}_{\mathcal{S}}^{a+1}$ (if any), the variable node in $\mathcal{S}'\backslash\mathcal{S}$ is only connected to unsatisfied check nodes of $\mathcal{S}$ (i.e., to the check nodes in $\Gamma_{\mathrm{o}}(\mathcal{S})$).

Proof: If the node in $\mathcal{S}'\backslash\mathcal{S}$ is connected to any satisfied check nodes of $\mathcal{S}$, then $\mathcal{S}'$ will have unsatisfied check nodes in $\Gamma_{\mathrm{o}}(\mathcal{S}')$ connected to 3 variable nodes of $\mathcal{S}'$. This contradicts $\mathcal{S}'$ being an elementary trapping set. ∎

Fig. 3(a) depicts an example of the set $\mathcal{S}'$ discussed in Lemma 2. (It should be noted that in all the configurations of Fig. 3, including 3(a), unsatisfied check nodes of $G(\mathcal{S}')$ are not shown.)

Lemma 3: Suppose that $A = \{a_1, ..., a_i, a_{i+1}, ..., a_I\}$ is the sorted set of sizes of the elementary trapping sets in $\mathcal{T}$ in increasing order. Let $\mathcal{S}$ be an elementary trapping set of size a_i in $\mathcal{T}$. If $a_{i+1} > a_i + 2$, then for each elementary trapping set $\mathcal{S}' \in \mathcal{T}_{\mathcal{S}}^{a_{i+1}}$ (if any), the set $\mathcal{S}'\backslash\mathcal{S}$ is connected to zero satisfied check

nodes of $\mathcal{S}$ (i.e., nodes in $\Gamma_e(\mathcal{S})$) and to only one or two unsatisfied check nodes of $\mathcal{S}$ (i.e., nodes in $\Gamma_o(\mathcal{S})$). If the set $\mathcal{S}'\backslash\mathcal{S}$ is connected to two nodes in $\Gamma_o(\mathcal{S})$, then there is no cycle in $G(\mathcal{S}'\backslash\mathcal{S})$. If the set $\mathcal{S}'\backslash\mathcal{S}$ is connected to only one node in $\Gamma_o(\mathcal{S})$, then there is exactly one cycle in $G(\mathcal{S}'\backslash\mathcal{S})$.

Proof: If any variable node in $\mathcal{S}'\backslash\mathcal{S}$ is connected to $\Gamma_e(\mathcal{S})$, then $G(\mathcal{S}')$ contains satisfied check nodes of degree 4 or higher, or unsatisfied check nodes of degree greater than 1. Both are in contradiction with $\mathcal{S}'$ being an elementary trapping set.

Since $\mathcal{S}'$ is an elementary trapping set, there cannot be more than one connection between each node of $\Gamma_o(\mathcal{S})$ and the nodes in $\mathcal{S}'\backslash\mathcal{S}$. To see this, consider a node $v \in \mathcal{S}'\backslash\mathcal{S}$ which is connected to $\Gamma_o(\mathcal{S})$. Node v can have only one connection to $\Gamma_o(\mathcal{S})$ because otherwise, all the other nodes in $\mathcal{S}'\backslash\mathcal{S}$ can be removed and we will end up with an elementary trapping set of size $a_i + 1$ in $\mathcal{T}$, which is in contradiction with the assumption of the lemma.

Now suppose that there are at least 3 connections between the variable nodes in $\mathcal{S}'\backslash\mathcal{S}$ and $\Gamma_o(\mathcal{S})$. Based on the discussion in the previous paragraph, this means that there are at least 3 variable nodes in $\mathcal{S}'\backslash\mathcal{S}$ each with a single connection to a different check node in $\Gamma_o(\mathcal{S})$. If $G(\mathcal{S}'\backslash\mathcal{S})$ is not connected, then one can remove one of its components and obtain an elementary trapping set of size smaller than a_{i+1}, which results in a contradiction. If $G(\mathcal{S}'\backslash\mathcal{S})$ is connected, then one can find the shortest paths in $G(\mathcal{S}'\backslash\mathcal{S})$ between every two variable nodes of $\mathcal{S}'\backslash\mathcal{S}$ that are connected to $\Gamma_o(\mathcal{S})$, and among them select the one with the least number of nodes. By keeping the nodes on the selected path and removing all the other nodes in $\mathcal{S}'\backslash\mathcal{S}$, one can then obtain an elementary trapping set of size smaller than a_{i+1}, which is again a contradiction. We therefore conclude that the number of connections between the variable nodes in $\mathcal{S}'\backslash\mathcal{S}$ and $\Gamma_o(\mathcal{S})$ must be strictly less than 3.

For the case that $\mathcal{S}'\backslash\mathcal{S}$ is connected to exactly two nodes in $\Gamma_o(\mathcal{S})$, there must be two different variable nodes v and v' of $\mathcal{S}'\backslash\mathcal{S}$ corresponding to those connections. Also, there must be no cycles in $G(\mathcal{S}'\backslash\mathcal{S})$. Otherwise, one can remove all the variable nodes on the cycle except those on the shortest path between v and v', and obtain an elementary trapping set larger than a_i but smaller than a_{i+1}. This contradicts the lemma's assumption. Fig. 3(b) is an example of the case where $\mathcal{S}'\backslash\mathcal{S}$ is connected to exactly two nodes in $\Gamma_o(\mathcal{S})$.

The proof for the case with one connection is similar and omitted. Fig. 3(c) is an example of this case, where the expansion of set $\mathcal{S}$ is through a lollipop walk. In both Figs. 3(b) and (c), the dashed line indicates that more variable and check nodes can be part of the chain. $\blacksquare$

Lemma 4: Suppose that $A = \{a_1, ..., a_i, a_{i+1}, ..., a_I\}$ is the sorted set of sizes of the elementary trapping

sets in $\mathcal{T}$ in increasing order and that $a_{i+1} = a_i + 2$. Let $\mathcal{S}$ be an elementary trapping set of size a_i in $\mathcal{T}$. If the girth of the graph is larger than 4, then for each elementary trapping set $\mathcal{S}' \in \mathcal{T}_{\mathcal{S}}^{a_{i+1}}$ (if any), the only possible configuration for $G(\mathcal{S}')$ is that of Fig. 3(b), described in Lemma 3, with only 2 variable nodes in $\mathcal{S}' \backslash \mathcal{S}$. If the girth is 4, then the only possible configurations are those in Figs. 3(b) (with only 2 variable nodes in $\mathcal{S}' \backslash \mathcal{S}$) 3(d) and 3(e).

Proof: The proof is similar to that of Lemma 3 and is omitted. ■

C. Proposed Algorithm

The basic idea behind the proposed algorithm is to construct larger elementary trapping sets by expanding smaller ones. More precisely, given an elementary trapping set $\mathcal{S}$ of size a_i at the input, the algorithm finds all the elementary trapping sets $\mathcal{S}'$ containing $\mathcal{S}$, with the property that their size a_{i+1} is the smallest size greater than a_i. The algorithm then continues by using the sets found in the current step as the inputs to the next step and finds the next set of larger elementary trapping sets. Each step of the algorithm is performed by using Lemmas 2 - 4. The pseudo-code for one step of the proposed algorithm is given in Algorithm 1.

Algorithm 1: Expansion of input elementary trapping sets to larger ones of size up to k with the number of unsatisfied check nodes up to T in $G = (L \cup R, E)$.

($\mathcal{L}_{in}$ and $\mathcal{L}_{out}$ are the lists of input and output trapping sets, respectively.)

1: **Inputs:** G and $\mathcal{L}_{in}$.

2: **Initialization:** $\mathcal{L}_{out} \leftarrow \emptyset$.

3: **repeat**

4: Select an element of $\mathcal{L}_{in}$ and denote it as t_j.

5: Construct a new graph G' by removing all the nodes in $\Gamma_e(t_j)$ and their neighbors from G.

6: $i_{max} \leftarrow (k - |t_j|)$ and $\mathcal{G} \leftarrow \emptyset$.

7: **for** each node c in $\Gamma_o(t_j)$ **do**

8:   Examine the neighborhood of c in G' one layer at a time and to the maximum of i_{max} layers in search for paths with $i \leq i_{max}$ variable nodes between c and the other nodes of $\Gamma_o(t_j)$, and lollipop walks with $i \leq i_{max}$ variable nodes starting from c.

9:   Denote $\mathcal{G}_c$ as the set of all such paths/lollipop walks of shortest length i (if any).

10:        **if** $i < i_{max}$ **then**

11:          $i_{max} \leftarrow i$.

12:          $\mathcal{G} \leftarrow \mathcal{G}_c$.

13:        **else**

14:          $\mathcal{G} \leftarrow \mathcal{G} \cup \mathcal{G}_c$.

15:        **end if**

16:      **end for**

17:      **for** each element $\mathcal{S}$ in $\mathcal{G}$ **do**

18:        $t' \leftarrow t_j \cup \mathcal{S}$.

19:        **if** $(t' \notin \mathcal{L}_{out})$ and $(|\Gamma_{\mathrm{o}}(t')| \leq T)$ **then**

20:          $\mathcal{L}_{out} \leftarrow \mathcal{L}_{out} \cup \{t'\}$.

21:        **end if**

22:      **end for**

23: **until** all the elements of $\mathcal{L}_{in}$ are selected.

24: **Output:** $\mathcal{L}_{out}$.

Remark 1: Note that in Line 5 of Algorithm 1, all the satisfied check nodes in $G(t_j)$, i.e., the set $\Gamma_{\mathrm{e}}(t_j)$, and their neighboring variable nodes are removed from the graph. This is because, based on Lemmas 2 - 4, such nodes cannot be part of the expansion of an elementary trapping set.

Remark 2: In Line 19 of the algorithm, the threshold value T on the number of unsatisfied check nodes is needed to keep the complexity of the overall search algorithm, which involves multiple applications of Algorithm 1, low. A proper choice of T has negligible effect on the ability of the algorithm to find the larger trapping sets (a, b) with small values of b. This is explained in the following example.
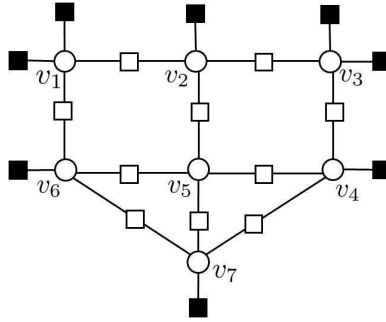

Fig. 4.   An example of a $(7, 8)$ trapping set (satisfied and unsatisfied check nodes are shown by empty and full squares, respectively)

Example 2: Consider the $(7, 8)$ trapping set $\mathcal{S}$, shown in Fig. 4. This set belongs to $\mathcal{T}$ and contains 4

trapping sets of size 6, all also in $\mathcal{T}$. These four trapping sets can be each obtained by removing one of the nodes v_1, v_3, v_5 and v_7. As a result, we have $(6, 8)$, $(6, 8)$, $(6, 12)$ and $(6, 10)$ trapping sets, respectively. Among these trapping sets, the $(6, 8)$ ones have a smaller number of unsatisfied check nodes. Starting from each of these two trapping sets, Algorithm 1 finds $\mathcal{S}$. Hence, ignoring the $(6, 12)$ (or even $(6, 12)$ and $(6, 10)$) trapping set(s) does not impair the ability of the algorithm to find $\mathcal{S}$.

Remark 3: Based on Lemmas $2 - 4$, it can be proved that starting from an (a, b) elementary trapping set $\mathcal{S}$, Algorithm 1 will find all the (a', b') elementary trapping sets of the smallest size a' larger than a that contain $\mathcal{S}$ (this requires the removal of the condition $|\Gamma_{\mathrm{o}}(t')| \leq T$ in Line 19). Note that this does not imply that by the recursive application of Algorithm 1 one can obtain all the elementary trapping sets containing $\mathcal{S}$. The following example demonstrates this.

Example 3: Consider the $(6, 6)$ elementary trapping set $\mathcal{S}' = \{v_1,\, v_2,\, v_3,\, v_4,\, v_5,\, v_6\}$ in Fig. 5. Assume that Algorithm 1 starts from the elementary trapping set $\mathcal{S} = \{v_1,\, v_2,\, v_6\}$. Using this input, the output of the algorithm is $\{v_1,\, v_2,\, v_6,\, v_5\}$. By subsequent applications of the algorithm, the next outputs are $\{v_1,\, v_2,\, v_6,\, v_5,\, v_7\}$ and $\{v_1,\, v_2,\, v_6,\, v_5,\, v_7,\, v_3,\, v_4\}$, respectively. This means that the algorithm does not find the trapping set $\mathcal{S}'$, although $\mathcal{S}'$ contains $\mathcal{S}$. (It is however easy to see that if the algorithm starts from the set $\{v_2,\, v_3,\, v_4,\, v_5\}$, it will find $\mathcal{S}'$.)
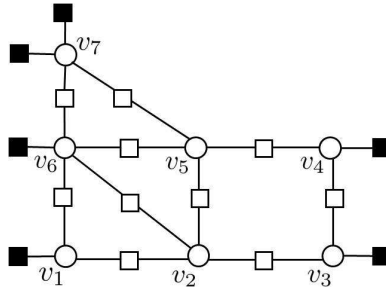

Fig. 5. An example explaining that the algorithm cannot find all the elementary trapping sets containing a specific elementary trapping set

In fact, the sufficient condition for the algorithm to find a trapping set $\mathcal{S}'$ of size a_j, starting with one of its subsets $\mathcal{S}$ of size $a_i < a_j$, is that $\mathcal{S}'$ has at least one subset in $\mathcal{T}^a_{\mathcal{S}}$ for all $a \in A$, $a_i < a < a_j$, where A is defined in Lemma 3.

The following example shows that despite the limitation explained in Remark 3 and Example 3, for many cases, the proposed algorithm in fact finds (in a guaranteed fashion) *all* the trapping sets (a, b) with a and b up to certain values.

Example 4: For a left-regular graph with left degree 4 and girth larger than 4, initiating the proposed algorithm with the set of cycles of length g and $g{+}2$, one can guarantee to find *all* the elementary trapping

sets of size less than 9 with less than 5 unsatisfied check nodes. This can be seen by the inspection of all the possible structures for such trapping sets and verifying that for each structure, the removal of only one variable node will result in another trapping set in $\mathcal{T}$. Subsequent removals of such nodes from an (a, b) elementary trapping set with $a < 9$ and $b < 5$ in $\mathcal{T}$, thus, results in a sequence of embedded elementary trapping sets in $\mathcal{T}$, each with size only one less than that of its parent. The sequence will always end with a cycle of length g or $g + 2$. This implies that all such trapping sets satisfy the sufficient condition, mentioned earlier, for being found by the algorithm starting from a cycle of length g or $g + 2$.

Similar results to those of Example 4 can be found for other left-regular graphs. For irregular graphs however, it is very difficult to provide such guarantees. This is due to the fact that the number of possible structures for a trapping set of a given size could be very large in this case.

Remark 4: For irregular LDPC codes, the variable nodes with large degrees cannot be part of small trapping sets. This is formulated in the following lemma.

Lemma 5: In a graph G with girth $g > 4$, if an (a, b) trapping set $\mathcal{S}$ contains a variable node v of degree $d(v) > b$, then $a \geq d(v) + 1 - b$.

Proof: The proof is provided in Appendix A. ∎

Based on Lemma 5, for example, for an irregular code with girth larger than 4, a variable node of degree 15 can not participate in an (a, b) trapping set with $a < 13$ and $b < 4$. Such results can be used to simplify the algorithm by removing the large degree variable nodes and their edges from the graph.

Remark 5: It is easy to see that for the left-regular graphs with left degree 3 or 4, all the trapping sets found by Algorithm 1 are ZP trapping sets. For the left-regular graphs with left degree 3, the obtained trapping sets are also absorbing sets.

Remark 6: Our simulations for many practical LDPC codes show that in almost all the cases, $a_{i+1} \leq a_i + 3$.

In the following, we discuss the selection of the initial set of elementary trapping sets.

D. Initial Set of Trapping Sets

One of the graphical objects that plays an important role in the structure of trapping sets is a cycle. Tian *et al.* [36] showed that every stopping set includes the variable nodes of at least one cycle. Related to this, the induced graph of the support of a pseudo-codeword always contains at least one cycle [18]. In [7], [44], [48], it was shown that an overwhelming majority of dominant trapping (absorbing) sets are combinations of short cycles. Short cycles are also easy to enumerate [44]. We thus use short cycles as

the initial inputs to the proposed algorithm. The following lemma provides more justifications for this choice.

Lemma 6:

i) In a left-regular graph G with left degree $d_l \geq 2$, if the induced subgraph $G(\mathcal{S})$ of an (a, b) trapping set $\mathcal{S}$ does not contain any cycle, then $b \geq a(d_l - 2) + 2$. The inequality is satisfied with equality for elementary trapping sets.

ii) The variable nodes in any shortest cycle (of length g) of a Tanner graph form an elementary trapping set.

iii) Let $\mathcal{T}$ be the set of all trapping sets $\mathcal{S}$ of a graph G, whose induced subgraph $G(\mathcal{S})$ is connected and for which every node $v \in \mathcal{S}$ is connected to at least two nodes in $\Gamma_e(\mathcal{S})$. Then for every $\mathcal{S} \in \mathcal{T}$, its induced subgraph $G(\mathcal{S})$ contains at least one cycle.

iv) Suppose that $\mathcal{S} \subset L$ is an absorbing set of a left-regular Tanner graph $G = (L \cup R, E)$ with left node degrees at least 2. Then $G(\mathcal{S})$ contains at least one cycle.

v) Suppose that $\mathcal{S} \subset L$ is a ZP trapping set of a Tanner graph $G = (L \cup R, E)$ with node degrees at least 3. Then $G(\mathcal{S})$ contains at least one cycle.

Proof: The proof of Part (i) is provided in Appendix A. The proofs for Parts (ii) and (iii) are simple and thus omitted. Parts (iv) and (v) follow from Lemma 1 and Part (iii). ■

It can be shown that Part (i) of Lemma 6 can be generalized to the case where variable node degree distribution is irregular. In this case, the result is modified as $b \geq a(\bar{d}_{\mathcal{S}} - 2) + 2$, where $\bar{d}_{\mathcal{S}}$ is the average degree of variable nodes in $\mathcal{S}$. The following example, based on Part (i) of Lemma 6, demonstrates that cycle-free (a, b) trapping sets have relatively large values of b.

Example 5: For a left-regular graph G with left degree 4, any cycle-free (a, b) trapping set satisfies $b \geq 2(a+1)$. Such large values of b for a given a would imply that the (a, b) trapping set is not dominant.

Our simulation results indicate that for denser graphs, the set of short cycles of length g, or g and $g+2$, where g is the girth, is sufficient to find almost all the small (with, say, $a \leq 10$) dominant trapping sets. In this case, adding short cycles of larger lengths to the input set has negligible effect on the performance of the algorithm, while increasing its complexity. For example, we examined a number of randomly constructed codes with rates larger than 0.4. The codes had block length 1000 and left-regular Tanner graphs with left degree 5 and girth 6. In all cases, the trapping sets obtained by Algorithm 1 using cycles of length 6 and 8 as input were identical to those obtained by using cycles of length 6, 8 and 10 as the input set.

For sparser graphs, however, one may need to use short cycles of larger lengths (e.g., g, $g + 2$, and $g + 4$) as the initial set.

E. Complexity of the Algorithm

The complexity of the algorithm is highly dependent on the short cycle distribution of the graph, which itself is mostly a function of the degree distribution of the graph (code) [19]. As a result, in general, the complexity increases much faster with the increase in the average variable and check node degrees of the graph than it does with increasing the block length. To have a more detailed analysis of the complexity of Algorithm 1, we note that the total complexity can be divided into two parts: a) Finding the initial input set and b) Expanding the input set to larger trapping sets.

Regarding the complexity of Part (a), assuming that an exhaustive brute force search is used to find cycles of length k, say for $g \le k \le g+4$, the complexity is $O(nd_v^{k/2} d_c^{k/2})$, for a (d_v, d_c) regular graph with n variable nodes. This is obtained by considering all the possible paths of length k starting from all the n variable nodes in the graph. The memory required for the storage of all the k-cycles is of order $O(kN_k)$, where N_k is the number of k-cycles in the graph. To the best of our knowledge, there is no theoretical result on how N_k scales with n or the degree distribution of the Tanner graph. Empirical results of [19] however suggest that N_k is mainly a function of the degree distribution and is rather independent of n.

Regarding the complexity of expanding the input trapping sets to larger ones, consider the expansion of an (a, b) trapping set $\mathcal{S}$ of a (d_v, d_c) graph. Depending on the size $a' > a$ of the smallest trapping set(s) $\mathcal{S}'$ that contain $\mathcal{S}$, the complexity and memory requirements for finding and storing the sets $\mathcal{S}'$ would differ. For $a' = a+1, a+2$ and $a+3$, the complexity is $O(bd_c), O(bd_v d_c^2)$ and $O(bd_v^2 d_c^3)$, respectively. The memory requirement for these cases are respectively $O(abd_c), O(abd_v d_c^2)$ and $O(abd_v^2 d_c^3)$. To see this, for example, consider the case where $a' = a + 1$. To find $\mathcal{S}'$, one needs to check at most $b(d_c - 1)$ variable nodes as possible candidates, which corresponds to $O(bd_c)$ complexity. The memory required to store all possible trapping sets of size $a + 1$ obtained through such a search is thus upper bounded by $(a + 1)bd_c$, which is of order $O(abd_c)$.

Based on the above discussions, assuming that the initial set is limited to cycles of length up to $g + 4$, and that we only consider trapping sets $\mathcal{S}'$ of size up to $a' = a + 3$ in the expansion process, the complexity of Algorithm 1 will be $O(d_v^2 d_c^3(T \sum_{i=g/2}^{g/2+2} N_{2i} + nd_v^{g/2} d_c^{g/2-1}))$ and the memory requirement will be $O(Td_v^2 d_c^3 \sum_{i=g/2}^{g/2+2} 2iN_{2i})$, where T is the maximum number of unsatisfied check nodes in Algorithm 1. It is however important to note that the actual complexity and memory requirements are much less than what these complexity bounds may suggest. In particular, our simulation results show that codes with

block lengths up to about 10,000 with a wide variety of degree distributions can be managed using the proposed algorithm on a regular desktop computer.

F. Expansion of Non-Elementary Trapping Sets

According to the general definition of trapping sets, any arbitrary set of variable nodes can be considered as a trapping set. Hence, to expand a connected trapping set $\mathcal{S}$ of size a, one just needs to select a variable node from the neighboring variable nodes, and add it to $\mathcal{S}$ to obtain a new trapping set $\mathcal{S}'$ with size $a' = a + 1$. This method of expansion leads to an exponentially growing search space. Even by limiting the search space to the trapping sets in $\mathcal{T}$, i.e., connected trapping sets for which every variable node is connected to at least two satisfied check nodes, there are still too many configurations for $\mathcal{S}'$, especially when $a' \gg a$. For practical LDPC codes with $g > 4$, however, considering a nested sequence of trapping sets, the size of the next larger trapping set a' is almost always less than $a + 3$.

The search for non-elementary trapping sets of size $a' \leq a + 3$ in a graph with girth $g > 4$, can be performed similar to what was described for the elementary trapping sets with a number of small differences. For non-elementary trapping sets, since there is no limitation on the degrees of the check nodes in $G(\mathcal{S})$, only the variable nodes of $\mathcal{S}$ and their edges are removed from the graph. Then the shortest paths between different check nodes of $G(\mathcal{S})$ or the shortest lollipop walks starting from different check nodes of $G(\mathcal{S})$ are found. However, it should be mentioned that not all such structures will necessarily satisfy the condition that each variable node is connected to at least two satisfied check nodes. After finding a candidate trapping set, one should thus check for this condition. In summary, to find the non-elementary trapping sets of size $a' \leq a + 3$, the only modifications needed to be applied to Algorithm 1 are the followings:

5: Construct a new graph G' by removing all the nodes of $\mathcal{S}$ from G.

7: **for** each node c in $\Gamma(t_j)$ **do**

8: Examine the neighborhood of c in G' one layer at a time and to the maximum of i_{max} layers in search for paths with $i \leq i_{max}$ variable nodes between c and the other nodes of $\Gamma(t_j)$, and lollipop walks with $i \leq i_{max}$ variable nodes starting from c.

19: **if** $(t' \in \mathcal{T})$ and $(t' \notin \mathcal{L}_{out})$ and $(|\Gamma_o(t')| \leq T)$

IV. IRREGULAR LDPC CODES

For the irregular LDPC codes which do not have variable nodes of degree 2, Algorithm 1 without any modification can be used to find the dominant trapping sets. As mentioned in Remark 4 of Section III.C,

based on the desired sizes of trapping sets, one may also remove the high-degree variable nodes and their edges from the graph to simplify the algorithm. In the case that the code has variable nodes of degree 2, some modifications are needed for the initial input set of the algorithm. In this section, we study the effect of degree-2 variable nodes on the structure of trapping sets in irregular LDPC codes, and present simple steps to find the corresponding trapping sets.[1]

A. On the Degree-2 Variable Nodes

It is known that degree-2 variable nodes play an important role in the performance of irregular LDPC codes. On one hand, to have codes with asymptotic performance close to the capacity, the proportion of degree-2 variable nodes should be as large as possible. This is usually a considerable fraction of the total variable nodes of the code. On the other hand, having a large proportion of degree-2 variable nodes results in a small minimum distance and a high error floor [35]. Cycles containing only degree-2 variable nodes are codewords. Hence, to have a large minimum distance, it is desirable to avoid such cycles, especially the shorter ones. To avoid all cycles of any length containing only degree-2 variable nodes, the number of these nodes n_{v_2} must be strictly less than the number of check nodes m (i.e., $n_{v_2} < m$). Based on this fact, a class of irregular LDPC codes with $n_{v_2} = m - 1$, called *extended irregular repeat accumulate* (eIRA) codes was proposed in [46]. It was shown in [46] that these codes exhibit relatively better error floor performance compared to the codes constructed by the optimized degree distributions without applying this restriction on n_{v_2}. Related to this, it was proved in [33] that for the case where $n_{v_2} > m$, the minimum distance grows only logarithmically with the code length. For the special case where $n_{v_2} = m$ and all the degree-2 variable nodes are part of a single cycle, the minimum distance is a sub-linear power function of the block length [35]. In the following, we study the effect of having a large fraction of degree-2 variable nodes on the structure of trapping sets in irregular LDPC codes.

Example 6: For all the degree distributions optimized for rate-1/2 LDPC codes on the binary-input AWGN (BIAWGN) channel [31], 43% to 55% of variable nodes are of degree 2. This implies that, on average, every check node in the corresponding codes is connected to about 2 variable nodes of degree 2. The average number of degree-2 variable nodes connected to each check node becomes even larger for the optimized codes of higher rate. This is explained in the next example.

Example 7: For the optimized degree distribution of rate $8/9$ over the BIAWGN channel with the maximum variable node degree 10 [31], 31% of variable nodes are of degree 2. This implies that, on

---

[1]In case that the graph contains degree-1 variable nodes as well, a similar approach to the one described in Section IV.B (for finding dominant trapping sets which include degree-2 variable nodes) can be used to find the dominant trapping sets containing degree-1 variable nodes.
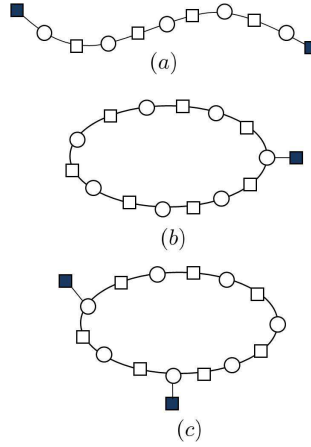
Fig. 6. Typical trapping sets constructed mostly by the degree-2 variable nodes.

average, every check node in a Tanner graph with this degree distribution is connected to about 6 variable nodes of degree 2.

Consequently, it is very likely to see chains of degree-2 variable nodes, referred to as *2-chains*, in the Tanner graph of LDPC codes with optimized degree distributions. The length of a 2-chain is defined as the number of the edges in the subgraph induced by the degree-2 variable nodes of the chain. That is, the length of a 2-chain containing k variable nodes of degree 2 is $2k$. A 2-chain of length $2k$ is a $(k, 2)$ trapping set (with the exception of the case where the chain is closed and forms a cycle; in that case, we refer to the 2-chain as a *2-cycle*. A 2-cycle of length $2k$, is a $(k, 0)$ trapping set). Having only 2 unsatisfied check nodes, 2-chains of length $2k$ are among the most dominant trapping sets of size k. Fig. $6(a)$ shows a 2-chain of length 10 (a $(5, 2)$ trapping set). Note that this trapping set also contains two $(4, 2)$, three $(3, 2)$ and four $(2, 2)$ trapping sets as its subsets. It is worth noting that although for the cases where $n_{v_2} = m - 1$ and $n_{v_2} = m$, the graph may have no or only one 2-cycle, it can have many 2-chains of different lengths. For example, it is easy to see that for the case where $m = n_{v_2}$ and all the degree-2 variable nodes are contained in a single cycle, there are m 2-chains of length $2k$, $1 \leq k \leq m - 1$.

Another aspect of having 2-chains in the Tanner graph of irregular LDPC codes is that they might participate in short cycles with other variable nodes of higher degrees. These cycles have low approximate cycle extrinsic message degree (ACE) (ACE is defined as $\sum_i d_i - 2$, where the summation is taken over all the variable nodes of the cycle, and d_i is the degree of the ith variable node in the cycle [36]). It has been shown that cycles with low ACE deteriorate the error rate performance, and that avoiding them in the construction of irregular LDPC codes generally improves the error rate [45], [39].

Example 8: Consider the case where $m = n_{v_2}$ and all the degree-2 variable nodes are contained in a single cycle. In this case, there exist two 2-chains between any two check nodes of the graph. This

implies that every variable node of degree $d_v > 2$ along with the 2-chains connecting its check nodes form several trapping sets with at most $d_v - 2$ unsatisfied check nodes.

Example 9: Fig. 6(b) shows a $(7, 1)$ trapping set composed of one variable node of degree 3 and a chain of six variable nodes of degree 2.

Example 10: The $(12, 1)$ trapping sets of the $(1944, 972)$ LDPC code adopted in the IEEE 802.11 standard [13] are single cycles of length 24, each consisting of a 2-chain of length 22 and one degree-3 variable node.

Even in the cases where $n_{v_2} < m$ (but not much smaller), it is likely to see cycles mostly constructed by 2-chains.

Example 11: Fig. 6(c) shows a $(7, 2)$ trapping set composed of two variable nodes of degree 3 and five variable nodes of degree 2.

Due to the important role that 2-chains (and 2-cycles) play in the formation of dominant trapping sets, we study the necessary condition to avoid these structures in the following theorem.

Theorem 1: Let m be the number of check nodes and n_{v_2} be the number of degree-2 variable nodes in the graph G corresponding to an irregular code $\mathcal{C}$. If G has no 2-chains of length $2k$ or larger, for $k \geq 2$ (and no 2-cycles of length less than or equal to $2k$) then

$$m \geq n_{v_2}(1 + \frac{1}{\sum_{i=0}^{k-2} (d_{c,max} - 1)^{\lfloor \frac{i+1}{2} \rfloor}}),$$

where $d_{c,max}$ is the maximum check node degree in G.

Proof: Let G_{v_2} denote the induced subgraph of degree-2 variable nodes of the graph G. This subgraph contains no cycle. Otherwise, the length of such a cycle would be at least $2k + 2$, which would imply the existence of a 2-chain of length $2k$ in G_{v_2}, and thus in G. This contradicts the assumption of the theorem. The subgraph G_{v_2} is thus composed of some tree-like components. For each component, the number of check nodes is always larger than the number of variable nodes by one. Therefore the total number of check nodes of the graph is more than the number of degree-2 variable nodes by at least the number of disjoint components in G_{v_2} (some check nodes of G may not appear in G_{v_2}). To avoid 2-chains of length $2k$ or larger, the maximum number of variable nodes in each component is $\sum_{i=0}^{k-2} (d_{c,max} - 1)^{\lfloor \frac{i+1}{2} \rfloor}$(Appendix A, Lemma 8). The minimum number of components in G_{v_2} is thus $\lceil n_{v_2} / \sum_{i=0}^{k-2} (d_{c,max} - 1)^{\lfloor \frac{i+1}{2} \rfloor} \rceil$. ∎

Theorem 1 can be used to determine the maximum number of degree-2 variable nodes in an irregular graph to avoid 2-chains (and 2-cycles) of a specific length.

Example 12: For an irregular code with 1000 check nodes of degree $d_c = 6$, to avoid $(4, 2)$ trapping

sets corresponding to 2-chains of length 8, the number of variable nodes of degree 2 must be at most 910.

Theorem 1 can be also used to obtain some information about the existing trapping sets in a code.

Example 13: For the same scenario as that of Example 12 (i.e., $m = 1000$, $d_c = 6$), the eIRA construction [46] results in $n_{v_2} = m - 1 = 999$. For these parameters, the smallest value of k which satisfies the inequality of Theorem 1 is $k = 9$. This implies that the eIRA code will have 2-chains of length 16 and smaller, corresponding to $(k, 2)$ trapping sets for all values of $k < 9$.

B. Finding Trapping Sets of Irregular LDPC Codes

In this section, we present a simple process to find the dominant trapping sets involving degree-2 variable nodes. The process can be used in combination with Algorithm 1 to find the dominant trapping sets of irregular graphs containing degree-2 variable nodes. It is important to note that according to the definition of absorbing sets, any variable node of degree 2 in these sets is connected to 2 satisfied check nodes. Also, for the trapping sets found by Algorithm 1, each variable node is connected to at least 2 satisfied check nodes. Therefore, 2-chains and other trapping sets containing variable node(s) of degree 2 with one satisfied check node are neither absorbing sets nor found by Algorithm 1. In fact, it appears that being connected to 2 satisfied check nodes is too strong of a condition for a variable node of degree 2 to be part of a dominant trapping set. For this reason, we consider also trapping sets whose variable nodes of degree 2 are connected to only one satisfied check node. To obtain such trapping sets using the expansion of smaller trapping sets, we consider an $(a - 1, b)$ trapping set $\mathcal{S}$ which is expanded to a trapping set $\mathcal{S}'$ by the connection of a variable node v of degree 2 to an unsatisfied check node of $\mathcal{S}$. Three cases are possible:

a) v is not connected to any other check node of $\Gamma(\mathcal{S})$. In this case, $\mathcal{S}' = \mathcal{S} \cup \{v\}$ is an (a, b) trapping set. If $\mathcal{S}$ is elementary, so is $\mathcal{S}'$.

b) v is also connected to a satisfied check node of $\mathcal{S}$. In this case, $\mathcal{S}' = \mathcal{S} \cup \{v\}$ is an (a, b) non-elementary trapping set.

c) v is also connected to another unsatisfied check node of $\mathcal{S}$. In this case, $\mathcal{S}' = \mathcal{S} \cup \{v\}$ is an $(a, b-2)$ trapping set. If $\mathcal{S}$ is elementary (or is in the set $\mathcal{T}$), so is $\mathcal{S}'$.

Such an expansion of a trapping set can be performed multiple times by adding one neighboring variable node of degree 2, each time. This is summarized in Algorithm 2. In a general case, Algorithm 2 can be used with Algorithm 1 to expand the trapping sets found by Algorithm 1. This is summarized in Algorithm 3.

Algorithm 2: Finding trapping sets of size up to k with the number of unsatisfied check nodes up to T constructed by adding degree-2 variable nodes to the input trapping sets for an irregular LDPC code with the Tanner graph $G = (L \cup R, E)$.

($\mathcal{L}_{in}$ and $\mathcal{L}_{out}$ are the lists of input and output trapping sets, respectively)

1: **Inputs:** G, $\mathcal{L}_{in}$

2: $\mathcal{L}_{out} \leftarrow \emptyset$.

3: **repeat**

4:     Select an element of $\mathcal{L}_{in}$ with size less than k, and denote it as t.

5:     Form the set $N_2(t)$ which contains variable nodes of degree 2 in $L \backslash t$ that are connected to at least one unsatisfied check node of t, i.e., to $\Gamma_\mathrm{o}(t)$.

6:     **for** each node v in $N_2(t)$ **do**

7:         $t' \leftarrow t \cup \{v\}$.

8:         **if** $(t' \in \mathcal{T})$ [2] and $(t' \notin \mathcal{L}_{out})$ and $(|\Gamma_\mathrm{o}(t')| \leq T)$ **then**

9:             $\mathcal{L}_{out} \leftarrow \mathcal{L}_{out} \cup \{t'\}$.

10:       **end if**

11:     **end for**

12: **until** all the elements of $\mathcal{L}_{in}$ are selected.

13: **Output:** $\mathcal{L}_{out}$.

Remark 7: Note that in Algorithm 2, the number of unsatisfied check nodes of the resultant trapping sets never increases. Hence, to find trapping sets of size a with less than b unsatisfied check nodes, one should consider all the (a', b') trapping sets with $a' < a$, $b' < b$.[3] It should be mentioned that since every single variable node of degree d_v can be regarded as a $(1, d_v)$ trapping set, to find the trapping sets with less than b unsatisfied check nodes, we consider also all the variable nodes of degree $d_v \leq b$ as part of the initial set. For example, for the case of $b = 3$, starting with a single variable node of degree $d_v = 2$ or $d_v = 3$, two typical structures of the resultant trapping sets are shown in Figs. 7(a) and 7(b), respectively. Note that starting from a degree-2 variable node and performing the above steps results in finding a

---

[2]This condition ensures that each variable node of degree larger than 2 is connected to at least 2 satisfied check nodes. The condition has no bearing on degree-2 variable nodes.

[3]Although this condition may not cover all the trapping sets discussed in Part c of Section IV.B, our simulations show that for the tested codes, almost all the trapping sets are in fact found by Algorithm 2. The trapping sets that are missed by Algorithm 2 are the ones that can *only* be obtained by starting from trapping sets with larger number of unsatisfied check nodes.
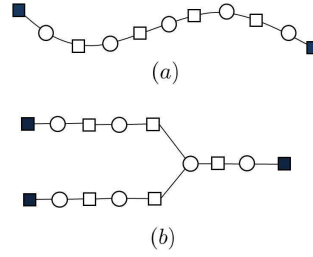
Fig. 7. Typical expansions of degree-2 and degree-3 variable nodes by adding the neighboring degree-2 variable nodes.

2-chain.

Algorithm 3: Finding trapping sets of size up to k with the number of unsatisfied check nodes up to T for an irregular LDPC code with the Tanner graph $G = (L \cup R, E)$.

($\mathcal{L}_{in}$ and $\mathcal{L}_{out}$ are the lists of input and output trapping sets, respectively.)

1: **Inputs:** G, $\mathcal{L}_{in}$

2: Use $\mathcal{L}_{in}$ as the input of Algorithm 1

3: $\mathcal{L}1_{out}$ = trapping sets found by Algorithm 1

4: $\mathcal{L}2_{in} = \mathcal{L}1_{out} \bigcup \{\text{low degree variable nodes}\}$

5: Use $\mathcal{L}2_{in}$ as the input of Algorithm 2

6: $\mathcal{L}_{out}$ = trapping sets found by Algorithm 2

7: **Output:** $\mathcal{L}_{out}$.

Remark 8: For irregular codes, in addition to short cycles, cycles with low ACE are also considered as part of the initial input set of Algorithm 1. This is because these cycles may not be found using the expansion process of Algorithm 1. Algorithm 1 finds the smallest trapping sets containing the input, which are usually the combination of the input and a short cycle (or a structure described in Lemmas 2 – 4). Since variable nodes of large degree are more likely to be part of such structures, the outputs of Algorithm 1 are usually the combinations of the input and variable node(s) of large degree. This is while cycles with low ACE are generally constructed by low degree variable nodes. Cycles with low ACE can be easily found by monitoring the ACE value during the execution of a cycle finding algorithm.

Remark 9: As an alternative approach to using Algorithm 3, one can only use Algorithm 2 with the variable nodes of low degree and cycles with low ACE as the initial input set, and then recursively expand them to larger trapping sets. It should however be noted that for the irregular LDPC codes with a small

fraction of degree-2 variable nodes, this approach may not find all the dominant trapping sets of the code.

V. NUMERICAL RESULTS

For the simulations, we assume binary phase-shift keying (BPSK) modulation over the AWGN channel with coherent detection. Notations E_b and N_0 are used for the average energy per information bit and the one-sided power spectral density of the AWGN, respectively.

A. Regular Codes

We have applied the proposed algorithm successfully to a large number of regular LDPC codes. Here, we only present the results for four of them. The first three examples are random and structured LDPC codes whose dominant trapping sets have already been reported in the literature and thus provide us with a reference for comparison. The fourth example is a random LDPC code of rate $1/2$ with variable node degree 4. To verify the trapping sets found by the proposed algorithm for this code, we estimate the error floor using importance sampling [7] based on the obtained trapping sets and demonstrate that the estimation is practically identical to the results of Monte Carlo simulations. The reported running times in the following examples are for a desktop computer with 2-GHz CPU and 1 GB of RAM.

Example 14: We consider an LDPC code constructed by the progressive edge growth (PEG) algorithm [12] (*PEGReg252x504* of [50]). This code is left-regular with the left degree 3, and girth 8. The same code was also investigated in [21] and the distribution of its fully absorbing sets was determined. For Algorithm 1, the short cycles of length g, $g+2$ and $g+4$ were used as the initial input set. The algorithm was limited to finding trapping sets of maximum size 13, and the threshold T was selected such that only the trapping sets with the two smallest values of b for each size were considered. (Using a larger T has no effect on the accuracy of the results reported here.) Since all the variable nodes have degree 3, all the trapping sets found by Algorithm 1 are absorbing sets. Fully absorbing sets were found by examining the obtained absorbing sets and testing them for the definition of a fully absorbing set. Table I shows the absorbing sets and the fully absorbing sets found by the proposed algorithm and their multiplicities. In the table, we have also reported the results obtained by the exhaustive search algorithm of [21], for comparison. (Note that the hyphen notation "-" in the table means that no data was reported.) As can be seen from Table I, for many classes of trapping sets, the proposed algorithm found exactly the same number of fully absorbing sets as the exhaustive search algorithm of [21] did. For the other classes, the difference between the two sets of results is rather small. Moreover, the proposed algorithm found $(11,3)$, $(13,3)$, $(10,4)$ and $(12,4)$ fully absorbing sets which are out of the reach of the exhaustive search

algorithm. It is worth mentioning that the exhaustive search algorithm of [21] took about 7 hours to find only the first three rows of Table I [21] (needless to say, the larger the size of the absorbing sets, the longer the running time of the algorithm). This is while Algorithm 1 took only 10 minutes to find all the absorbing sets listed in Table I.

TABLE I
DOMINANT ABSORBING SETS(ABS) AND FULLY ABSORBING SETS OF THE *PEGReg252x504* CODE OBTAINED BY THE
PROPOSED ALGORITHM AND THE EXHAUSTIVE SEARCH ALGORITHM OF [21]

Trapping Set	Proposed Algorithm (ABS)	Proposed Algorithm (Fully ABS)	Exhaustive Search [21] (Fully ABS)
(4, 4)	802	760	760
(5, 3)	14	14	14
(5, 5)	11279	10156	10156
(6, 4)	985	849	849
(6, 6)	86391	66352	66352
(7, 3)	57	47	47
(7, 5)	27176	21810	22430
(8, 2)	5	4	4
(8, 4)	2610	2258	2270
(9, 1)	1	1	1
(9, 3)	156	146	146
(10, 2)	6	6	6
(10, 4)	7929	6691	-
(11, 3)	605	558	-
(12, 2)	25	24	26
(12, 4)	23668	19959	-
(13, 1)	1	1	1
(13, 3)	2124	1954	-

Example 15: In this example, we consider the Tanner $(155, 64)$ code [34]. This code was also investigated in [41]. The exhaustive search algorithm of [41] showed that this code has no trapping set of length less than 8 with 2 unsatisfied check nodes and has no trapping set of length up to 11 with 1 unsatisfied check node. It was also shown in [41] that the code has 465 $(8, 2)$ trapping sets.

The girth for the Tanner graph of this code is $g = 8$. The short cycles of length g, $g + 2$ and $g + 4$ were used as the initial inputs to Algorithm 1. The algorithm was limited to only find trapping sets of maximum size 12 and the threshold T was selected such that only the trapping sets with the two smallest values of b for each size were considered. Table II shows the trapping sets found by the proposed algorithm and their multiplicity. As can be seen in the table, the algorithm found all the 465 $(8, 2)$ trapping sets among others. All the trapping sets in Table II were found in less than 2 minutes.

To further verify that the obtained trapping sets do in fact include the dominant ones, we performed

Monte Carlo simulations on the code with a 4-bit quantized min-sum decoder over the AWGN channel at signal-to-noise ratio (SNR) of 6.5 dB (which is in the error floor region of this code). Among the 300 error patterns, about 90% were $(8, 2)$ trapping sets, about 8% were $(10, 2)$ trapping sets, and only 2 did not belong to the sets reported in Table II.

TABLE II
DOMINANT TRAPPING SETS OF THE TANNER $(155, 64)$ CODE OBTAINED BY THE PROPOSED ALGORITHM

Trapping Set	Multiplicity
(4,4)	465
(5,3)	155
(6,4)	930
(7,3)	930
(8,2)	465
(9,3)	1395
(10,2)	1395
(11,3)	1860
(12,2)	930

Example 16: As the third example, we consider the Margulis $(2640, 1320)$ code [25], [50]. It is known that the most dominant trapping sets of this code are 1320 $(12, 4)$ and 1320 $(14, 4)$ trapping sets [30]. The Tanner graph of this code has girth $g = 8$. The set of short cycles of length g, $g + 2$ and $g + 4$ was used as the input set of the proposed algorithm. The algorithm was limited to use only the trapping sets with the two smallest values of b for each size. Since the degree of all the variable nodes of this code is 3, all the trapping sets found by Algorithm 1 are also absorbing sets. The first column in Table III shows the dominant absorbing sets found by Algorithm 1. For comparison, the dominant trapping sets obtained by the algorithm of [2] are listed in the last column of Table III. It should be noted that in [2] there is no condition on the number of satisfied check nodes connected to each variable node. Thus to have a fair comparison, we also consider the trapping sets constructed by the combination of trapping sets found by Algorithm 1 and one of their neighboring variable nodes. The second column of Table III shows the number of such trapping sets.[4] As can be seen, for all the trapping set classes, the proposed algorithm performs at least as well as the algorithm of [2]. Moreover, the required time for the algorithm of [2] was 7 days on a 2.8 GHz PC [2], while the proposed algorithm took about 5 hours to finish. As another comparison for the running time of the proposed algorithm, it took the algorithm 55 minutes to find all the absorbing sets of size less than 15, while the same task took 8.2 hours for the impulse method of [7] on a comparable computer (2.2-GHz CPU with 1 GB RAM).

---

[4]Our simulations indicate that the effect of extra trapping sets found by removing the constraint on the number of satisfied check nodes connected to each variable node of the trapping set on the error floor performance of the code is negligible.

TABLE III
DOMINANT TRAPPING SETS OF THE MARGULIS (2640, 1320) CODE OBTAINED BY THE PROPOSED ALGORITHM AND THE
ALGORITHM OF [2]

Trapping Set	Proposed Algorithm (Absorbing)	Proposed Algorithm (Trapping)	Algorithm of [2] (Trapping)
(7, 5)	7920	7920	-
(8, 6)	106920	>106920	-
(9, 5)	2640	2640	-
(10, 6)	117480	>117480	-
(11, 5)	5280	5280	9
(12, 4)	1320	1320	1320
(13, 5)	2640	26400	2699
(14, 4)	1320	1320	1320
(15, 5)	0	26400	7938
(16, 6)	0	258347	21153
(17, 5)	5280	5280	0
(18, 6)	0	132000	2642

Example 17: For this example, we consider a $(1008, 504)$ random code with variable node degree 4 and check node degree 8 constructed by the program of [50].[5] This code has one cycle of length 4 (C_4). In addition to that, the short cycles of length 6 to 10 were used as the initial input set for Algorithm 1. The algorithm was constrained to find trapping sets of size up to 12 and to use only the trapping sets with the two smallest values of b for each size. Table IV shows the dominant trapping sets found by Algorithm 1 and their multiplicities. It is worth mentioning that none of the trapping sets listed in Table IV contains any of the variable nodes participating in C_4. The trapping sets reported in Table IV were used to estimate the error floor of the code using the importance sampling technique described in [7]. Fig. 8 shows the Monte Carlo simulation results for the frame error rate (FER) and the corresponding error floor estimation based on importance sampling. The results are for a 3-bit min-sum decoder with a maximum number of 50 iterations. As can be seen in Fig. 8, the estimation closely matches the Monte Carlo simulation, verifying the dominance of the trapping sets found by Algorithm 1. Monte Carlo simulations also revealed that the most harmful trapping set of this code is the $(6, 4)$ trapping set. In fact, in almost all the decoding failures, the decoder converged to the (6,4) trapping set. As can be seen in Table IV, all the trapping sets have at least 4 unsatisfied check nodes. This makes the exhaustive search methods of [40], [41], [21] ineffective for finding the dominant trapping sets of this code. This is while all the trapping sets in Table IV were found in less than 5 minutes by the proposed algorithm.

[5]Using *code6.c* with seed=380.

TABLE IV
DOMINANT TRAPPING SETS OF THE $(1008, 504)$ REGULAR LDPC CODE $(d_v = 4, d_c = 8)$ OBTAINED BY THE PROPOSED
ALGORITHM

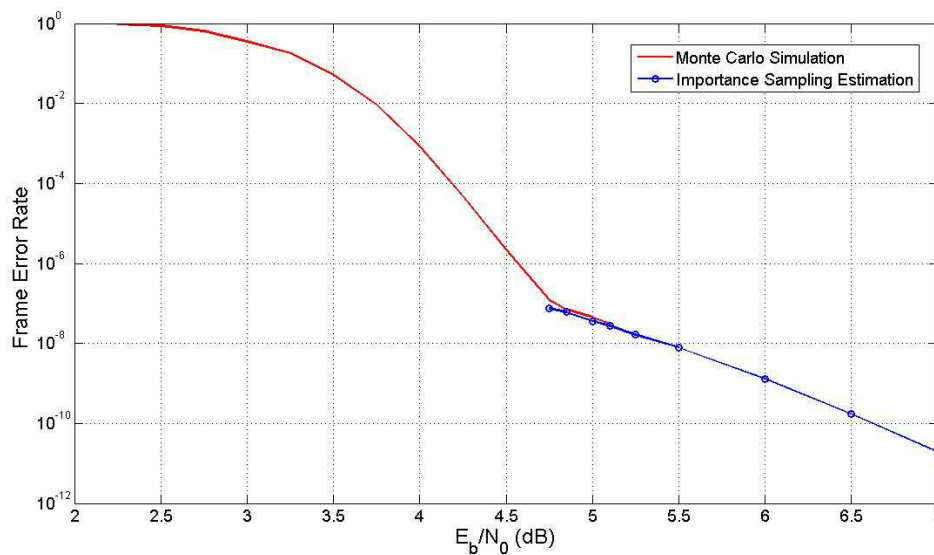| Trapping Set	Multiplicity
(5,6)	15
(6,4)	1
(6,6)	36
(7,5)	13
(8,6)	5
(9,6)	5
(10,6)	3
(11,6)	3
(12,8)	75

Fig. 8. Error floor estimation and Monte Carlo simulation for the $(1008, 504)$ regular LDPC code $(d_v = 4, d_c = 8)$.

B. Irregular Codes

In this section, we present the results of applying the proposed algorithm to three irregular LDPC codes. To find the dominant trapping sets of the irregular codes, we used two approaches. In the first approach, we used Algorithms 1 and 2 in the framework described in Algorithm 3. In this approach, as the first step, we used the short cycles of the codes, as well as the low ACE cycles as the initial input set, and applied Algorithm 1. We then used the trapping sets found by Algorithm 1 along with the variable nodes of low degree, and applied Algorithm 2 to expand them. As the second approach, we only used the variable nodes of low degree and cycles with low ACE as the initial input set, and then used Algorithm 2 to recursively expand them to larger trapping sets. Interestingly, for all three codes, the results of the second approach were very close to those of the first one.

Example 18: For this example, we consider the irregular LDPC code constructed by the PEG algorithm (*PEGirReg252x504* code [50]). This code was also investigated in [21] for its fully absorbing sets. For Algorithm 1, the short cycles of length g, $g + 2$, and the cycles with length less than 20 and ACE less than 4 were used as the initial input set. The algorithm was constrained to find only trapping sets of size less than 12 and the threshold T was selected such that only the trapping sets with the four smallest values of b for each size were considered. The resultant trapping sets and variable nodes of degree 2 and 3 were then expanded by adding neighboring degree-2 variable nodes, and finally were examined to find the fully absorbing sets. Table V shows the fully absorbing sets found by Algorithm 3 and the exhaustive search algorithm of [21]. It should be noted that, similar to [21], we relaxed the condition that degree-2 variable nodes of (fully) absorbing sets must be connected to two satisfied check nodes. As can be seen from Table V, the proposed algorithm found almost all the fully absorbing sets of this code.[6] Moreover, the proposed algorithm found a number of $(a, 1)$ trapping sets for $a \geq 9$, which were not reported in [21]. For the second approach, the cycles of length up to 20 with ACE lower than 4 and the variable nodes of degree 2 and 3 were used as the initial inputs, and the algorithm found almost the same trapping sets as in the first approach. For the running time, the first and the second approaches took 15 minutes and 5 minutes, respectively.

TABLE V
DOMINANT FULLY ABSORBING SETS OF THE *PEGirReg252x504* CODE OBTAINED BY THE PROPOSED ALGORITHM AND THE ALGORITHM OF [21]

Trapping Set	Proposed Algorithm	Exhaustive Search [21]
(3, 2)	219	219
(4, 2)	208	208
(5, 2)	198	198
(6, 2)	205	205
(7, 1)	2	2
(7, 2)	271	272
(8, 1)	8	8
(8, 2)	458	460
(9, 1)	16	-
(9, 2)	855	-
(10, 1)	22	-
(10, 2)	1533	-
(11, 1)	36	-

Example 19: For this example, we used the $(1944, 972)$ structured irregular code with rate $1/2$, adopted

---

[6]The multiplicity for trapping sets $(7, 2)$ and $(8, 2)$ are reported as 274 and 468 in [21], respectively. Moreover, no $(7, 1)$ or $(8, 1)$ trapping set is reported in [21]. The values reported for these four trapping sets in the last column of Table V are based on [22].

in the IEEE 802.11 standard [13]. We used the same parameters as in the previous example for the two approaches. Table VI shows the number of dominant trapping sets of different sizes found by the algorithm of [2] and the proposed approaches. For this code, both of our approaches found exactly the same set of trapping sets. In fact, all the trapping sets listed in Table VI have one of the following three structures: a 2-chain, a single cycle with low ACE, and the combination of a 2-chain and a single cycle of low ACE. For example, all the trapping sets of size less than 7 listed in Table VI are 2-chains, and all the $(12, 1)$ trapping sets are single cycles of eleven degree-2 variable nodes and one degree-3 variable node. As can be seen in Table VI, for all classes of trapping sets, the proposed algorithms found at least as many trapping sets as the algorithm of [2] did. The first and the second approaches took 45 and 5 minutes, respectively, to find all the trapping sets in Table VI. This is while the algorithm of [2] took 5 days (on a 2.8-GHz CPU) to find the results reported in Table VI.

TABLE VI
DOMINANT TRAPPING SETS OF THE $(1944,972)$ CODE OBTAINED BY THE PROPOSED ALGORITHM

Trapping Set	Proposed Algorithm	Algorithm of [2]
(2, 2)	810	-
(3, 2)	729	-
(4, 2)	648	648
(5, 2)	567	567
(6, 2)	486	486
(7, 2)	486	485
(8, 2)	648	637
(9, 2)	972	-
(10, 2)	1377	1210
(11, 2)	1944	1635
(12, 1)	81	81
(12, 2)	2754	2166
(13, 1)	162	162
(14, 1)	162	162
(15, 1)	162	-
(16, 1)	162	-
(17, 1)	162	-
(18, 1)	81	-

Based on the importance sampling technique of [7], the trapping sets in Table VI with size l, $6 \leq l \leq 12$, were used to estimate the error floor of this code for a 3-bit quantized min-sum decoder over the AWGN channel. Fig. 9 shows the error floor estimation and the Monte Carlo simulation results for this code. As can be seen in Fig. 9, the importance sampling estimation closely matches the Monte Carlo simulation, further verifying the dominance of the trapping sets found by the proposed algorithm.
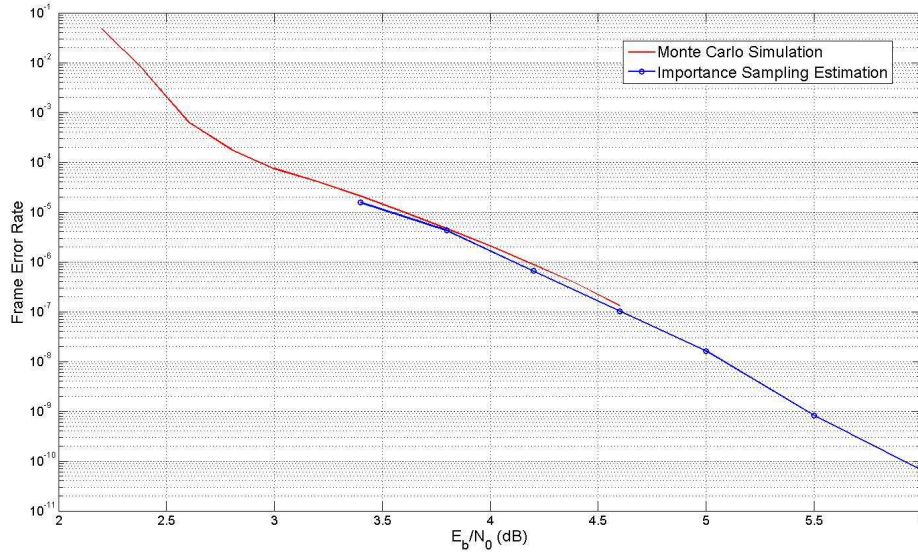
Fig. 9. Error floor estimation and Monte Carlo simulation for the $(1944, 972)$ irregular LDPC code.

Example 20: As the last example, we use the following degree distribution optimized for the min-sum algorithm in [6] and construct a $(1000, 499)$ LDPC code using the PEG algorithm: $\lambda(x) = .30370x + .27754x^2 + .02843x^5 + .20014x^6 + .19019x^{19}$ and $\rho(x) = .0160x^5 + .9840x^6$. The girth of the resultant graph is 6, and we use the short cycles of length 6 and 8, and cycles of length up to 20 with ACE less than 4 as the initial input set of Algorithm 2. It takes 1 minute for the algorithm to find the trapping sets of size up to 10. Based on the obtained trapping sets and using the importance sampling, we estimate the error floor of the code. Fig. 10 shows the estimation and Monte Carlo simulations for this code. As can be seen in this figure, the estimation closely matches the Monte Carlo simulation results, verifying that the dominant trapping sets of the code have been found by the algorithm.

VI. CONCLUSIONS

In this paper, we proposed an efficient algorithm for finding the dominant trapping sets of an LDPC code. The algorithm starts from an initial set of trapping sets and recursively and greedily expands them to trapping sets of larger size. The initial set for regular codes is a set of short cycles, and for irregular codes, it also includes variable nodes of small degree and cycles with low ACE values. To devise the expansions, the structure of dominant trapping sets is carefully studied for both regular and irregular codes. The efficiency and accuracy of the proposed algorithm was demonstrated through a number of examples. It was observed that the proposed algorithm is faster by up to about two orders of magnitude compared to similar search algorithms.
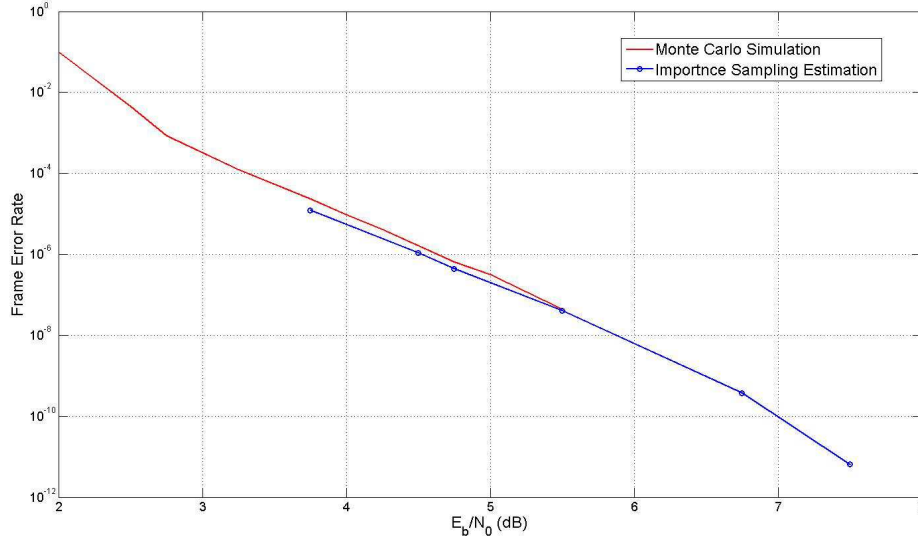
Fig. 10. Error floor estimation and Monte Carlo simulation for the $(1000, 499)$ irregular LDPC code.

APPENDIX A

In this appendix, we present Lammas 7 and 8, used in Sections III and IV, respectively, along with their proofs. The appendix also contains the proofs for Lemmas 5 and 6(i).

Lemma 7: For a left-regular graph G with left degree $d_l \geq 3$ and girth $g > 4$,[7] consider an (a, b) trapping set with $b < a$. If such a trapping set is elementary, let the notation a_e denote its size, and consider the case where $d_l(d_l - 1) > b$. Otherwise, for non-elementary trapping sets with $b < a$, let the notations a_{n1} and a_{n2} denote the size of the trapping set if it has at least one unsatisfied check node of degree $d_o > 1$ and one satisfied check node of degree $d_e > 2$ in $G(\mathcal{S})$, respectively. For the two latter cases, suppose that $d_o(d_l - 1) > b$ and $d_e(d_l - 1) > b$, respectively. Then depending on the value of g, we have the following two sets of inequalities:

a) For $g = 4k$, where k is an integer larger than 1, we have:

$$a_e \geq 1 + d_l + (d_l(d_l - 1) - b) \sum_{i=0}^{k-3} (d_l - 1)^i + \frac{(d_l(d_l - 1) - b)(d_l - 1)^{k-2}}{d_l} ,$$

$$a_{n1} \geq d_e + (d_e(d_l - 1) - b) \sum_{i=0}^{k-2} (d_l - 1)^i ;$$

---

[7]For the case of $d_l = 2$, it is easy to see that any (a, b) elementary trapping set has $b = 0$ or $b = 2$. For $b = 0$, the smallest value of a is $g/2$, which corresponds to the trapping set being a shortest cycle. For an elementary trapping set with $b = 2$, the smallest value of a is one, which corresponds to a single variable node. For a non-elementary (a, b) trapping set however, if $b = 0$, the smallest value of a is g. If $b = 2$, the minimum value of a for such a trapping set is 3.

$$a_{n2} \geq d_o + (d_o(d_l - 1) - b + 1) \sum_{i=0}^{k-2} (d_l - 1)^i .$$

b) For $g = 4k + 2$, where k is a positive integer, we have:

$$a_e \geq 1 + d_l + (d_l(d_l - 1) - b) \sum_{i=0}^{k-2} (d_l - 1)^i ,$$

$$a_{n1} \geq d_e + (d_e(d_l - 1) - b) \sum_{i=0}^{k-2} (d_l - 1)^i + \frac{(d_e(d_l - 1) - b)(d_l - 1)^{k-1}}{d_l} ;$$

$$a_{n2} \geq d_o + (d_o(d_l - 1) - b + 1) \sum_{i=0}^{k-2} (d_l - 1)^i + \frac{(d_o(d_l - 1) - b + 1)(d_l - 1)^{k-1}}{d_l} .$$

Proof: Here, we just present the sketch of the proof. For this, we first need the following lemma, whose proof follows later in the appendix.

Lemma 6(i): In a left-regular graph G with left degree $d_l \geq 2$, if the induced subgraph $G(\mathcal{S})$ of an (a, b) trapping set $\mathcal{S}$ does not contain any cycle, then $b \geq a(d_l - 2) + 2$. The inequality is satisfied with equality for elementary trapping sets.

Based on Lemma 6(i), it is clear that a trapping set with $b < a$ has at least one cycle. Therefore, considering any variable (or check) node of $\mathcal{S}$ as the root, and growing $G(\mathcal{S})$ from that node, one can construct a tree of at least $g/2$ layers, where the layers contain either variable or check nodes alternately, with no repetition of nodes. The number of variable nodes in this tree can be used as a lower bound on the number of variable nodes in $\mathcal{S}$. In this tree, the number of check nodes in layer $i > 1$ of the tree, N_c^i, is $N_c^i = (d_l - 1)N_v^{i-1}$, where N_v^{i-1} is the number of variable nodes in layer $i - 1$. Similarly, $N_v^i = \sum (d_{c_j^{i-1}} - 1)$, where $d_{c_j^{i-1}}$ is the degree (within $G(\mathcal{S})$) of the jth check node in layer $i - 1$, and the summation is over all the check nodes in layer $i - 1$. To minimize the number of variable nodes in the tree, one needs to make $\sum (d_{c_j^{i-1}} - 1)$ as small as possible in each check node layer of the tree. In particular, this should be done at the upper layers of the tree if possible, since these layers contribute the most in the total number of variable nodes in the tree. In addition, to obtain a lower bound on the size of the trapping sets, we assume that even for the non-elementary case, except for one check node, the degrees of all the other check nodes in $G(\mathcal{S})$ are either 1 or 2. Moreover, we assume that all the check nodes of degree 1 are in the first (upper) layer(s) of check nodes after the root layer.

For the case of an elementary trapping set, according to the assumption of $b < a$, there is at least one variable node that is not connected to any unsatisfied check nodes. Considering such a variable

node as the root node, all the check nodes in the first layer are satisfied check nodes. That is, $N_v^0 = 1$ (root node), $N_c^1 = N_v^2 = d_l$, $N_c^3 = d_l(d_l - 1)$, $N_v^4 = d_l(d_l - 1) - b$ and $N_c^{i-1} = (d_l - 1)N_v^{i-2}$, $N_v^i = N_c^{i-1}$, for $i = 6, 8, \ldots$.[8] Therefore, the total number of variable nodes in the constructed tree is $1 + d_l + (d_l(d_l - 1) - b) + (d_l(d_l - 1) - b)(d_l - 1) + \ldots$. Distinction should be made between the cases of $g = 4k + 2$ and $g = 4k$. While in the former, the last layer of the tree consists of variable nodes, in the latter, it consists of check nodes. In this case, for each set of d_l check nodes in the last layer of the tree, there must be at least one other variable node in $\mathcal{S}$. The sketch of the proofs for the non-elementary cases are similar to that of the elementary case, with the difference that the check node of degree d_o or d_e is used as the root node. ∎

Proof of Lemma 5:

Consider the $d(v)$ neighbors of v in $G(\mathcal{S})$. At least $d(v) - b$ of them are in $\Gamma_e(\mathcal{S})$ and are thus connected to other variable nodes in $\mathcal{S}$. None of such variable nodes can share more than one check node from $\Gamma_e(\mathcal{S})$ with v, because of the condition $g > 4$. This implies that there are at least $d(v) - b$ variable nodes in $\mathcal{S} \backslash \{v\}$. ∎

Proof of Lemma 6(i):

Since $G(\mathcal{S})$ does not contain any cycle, it forms a tree (note that $G(\mathcal{S})$ is connected). Suppose that $G(\mathcal{S})$ is grown from a variable node of $\mathcal{S}$ as the root, one layer at a time, until along each path, the growth is terminated by reaching a check node as a leaf. These nodes are the unsatisfied check nodes of degree one. In the tree, each variable node, except the root, has a parent which is a check node of degree ≥ 2. In the case that $\mathcal{S}$ is elementary, the degree of the parent check nodes is 2, and hence each check node is the parent to one variable node. There are thus exactly $a - 1$ check nodes of degree 2 in $G(\mathcal{S})$. Since $G(\mathcal{S})$ is a tree, the number of its nodes is more than the number of its edges by one. The total number of nodes in the graph is $a + (a - 1) + b_1$ and the total number of edges is $a \cdot d_v$, where b_1 is the number of unsatisfied check nodes of degree one. For an elementary trapping set, we thus have $2a + b_1 - 1 = ad_v + 1$, which implies $b = b_1 = a(d_v - 2) + 2$. In the case that $\mathcal{S}$ is not elementary, some variable nodes may share the same parent. The number of parent check nodes is thus less than $a - 1$, and therefore $b \geq b_1 > a(d_v - 2) + 2$. ∎

Lemma 8: Let $G = (L \cup R, E)$ be a left-regular bipartite graph with left degree 2. Consider a set $\mathcal{S} \in L$, for which the induced subgraph is a tree and has the longest path of length $2k - 2$. Then

---

[8]Here, based on the statement of the lemma, we have assumed that all the unsatisfied check nodes can fit in the third layer of the tree. In the case that $d_l(d_l - 1) - b \leq 0$, some of the unsatisfied check nodes have to be located in the next layer(s), and the above equations and the claims of the lemma will have to be accordingly revised.

$|\mathcal{S}| \leq \sum_{i=0}^{k-2} (d_{c,max} - 1)^{\lfloor \frac{i+1}{2} \rfloor}$, where $|\mathcal{S}|$ is the number of nodes in $\mathcal{S}$ and $d_{c,max}$ is the maximum degree of the nodes in R.

Proof: The upper bound is derived by counting the number of variable nodes in a tree where the number of check nodes is maximized with the constraint that the longest path has $2k - 2$ edges. This implies that there is a path of length $2k - 2$ between any two leaf check nodes of $G(\mathcal{S})$. In addition, to maximize $|\mathcal{S}|$, the degree of all the check nodes in $G(\mathcal{S})$ is assumed to be $d_{c,max}$. ∎
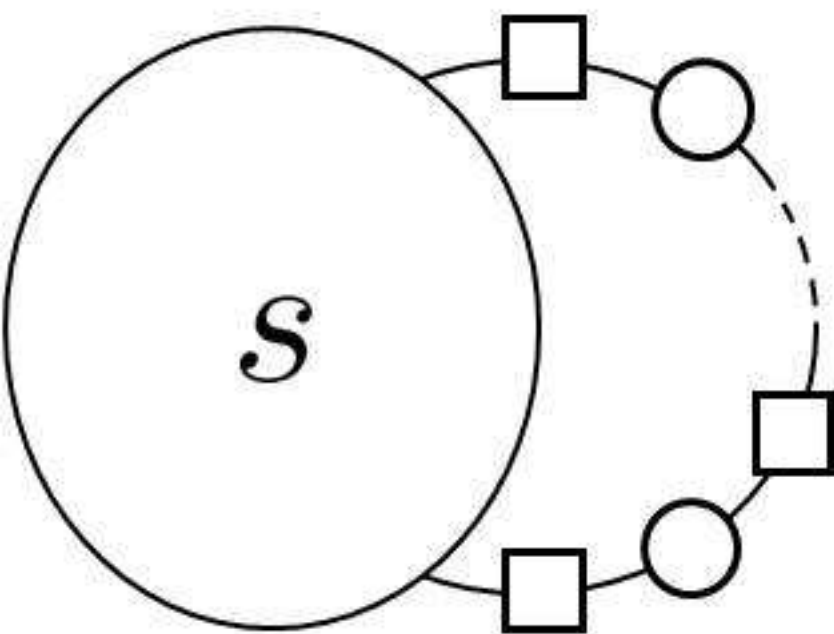
ACKNOWLEDGEMENT

REFERENCES

[1] R. Asvadi, A. H. Banihashemi and M. Ahmadian-Attari, "Lowering the error floor of LDPC codes using cyclic liftings," *IEEE Trans. Inform. Theory*, vol. 57, no. 4, pp. 2213-2224, Apr. 2011.

[2] S. Abu-Surra, D. DeClercq, D. Divsalar, and W. Ryan, "Trapping set enumerators for specific LDPC codes," Proc. *Inform. Theory and Applications Workshop,* San Diego, CA, Jan. 31- Feb. 5, 2010, pp. 1-5.

[3] E.T. Bax, "Algorithms to count paths and cycles," *Information Processing Letters*, vol. 52, no. 5, pp. 249252, Dec. 1994.

[4] E. Cavus and B. Daneshrad, "A performance improvement and error floor avoidance technique for belief propagation decoding of LDPC codes," Proc. *16th IEEE International Symposium Pers., Indoor Mobile Radio Communications,* Sept. 2005, vol. 4, pp. 2386-2390.

[5] E. Cavus, C. Haymes, and B. Daneshrad, "An IS simulation technique for very low BER performance evaluation of LDPC codes," Proc. *IEEE International Conference on Communications*, Istanbul, June 11-15, 2006, pp. 1095-1100.

[6] S. Y. Chung, "On the construction of some capacity-approaching coding schemes," Ph.D. thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Sept. 2000.

[7] C. Cole, S. Wilson, E. Hall, and T. Giallorenzi, "A general method for finding low error rates of LDPC codes," *CoRR, arxiv.org/abs/cs/0605051*.

[8] L. Dolecek, Z. Zhang, M. Wainwright, V. Anantharam, and B. Nikolic, "Evaluation of the low frame error rate performance of LDPC codes using importance sampling," Proc. *IEEE Inform. Theory Workshop*, Lake Tahoe, CA, Sept. 2-6, 2007, pp. 202-207.

[9] L. Dolecek, P. Lee, Z. Zhang, V. Anantharam, B. Nikolic, and M. Wainwright, "Predicting error floors of LDPC codes: deterministic bounds and estimates," *IEEE Journal on Selected Areas in Communications*, vol. 27, no. 6, pp. 908-917, Aug. 2009.

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

[11] Y. Han, and W. E. Ryan, "LDPC decoder strategies for achieving low error floors," Proc. *Inform. Theory and Applications Workshop*, Jan. 2008, pp. 277-286.

[12] X.-Y. Hu, E. Eleftheriou, and D. M. Arnold, "Regular and irregular progressive edge-growth Tanner graphs," *IEEE Trans. Inform. Theory,* vol. 51, no. 1, pp. 386-398, Jan. 2005.

[13] IEEE-802.11n. Wireless LAN medium access control and physical layer specifications: enhancements for higher throughput. *P802.11n/D3.07*, Mar. 2008.
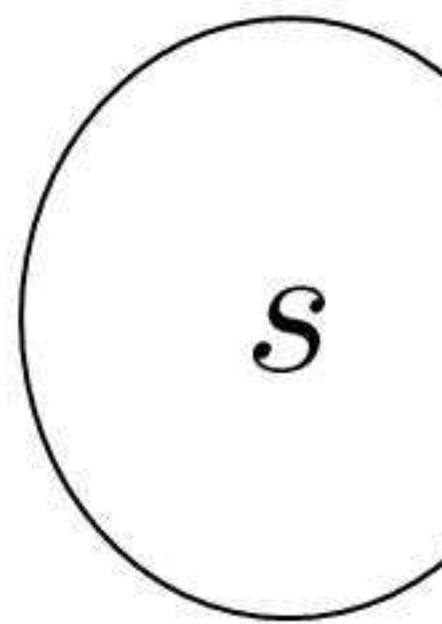
[14] M. Ivkovic, S. K. Chilappagari, and B. Vasic, "Eliminating trapping sets in low-density parity-check codes by using Tanner graph covers," *IEEE Trans. Inform. Theory*, vol. 54, no. 8, pp. 3763-3768, Aug. 2008.

[15] D. B. Johnson, "Find all the elementary circuits of a directed graph," *Society for Industrial and Applied Mathematics (SIAM) Journal on Computing*, vol. 4, no. 1, pp. 77-84, Mar. 1975.

[16] S. Laendner, T. Hehn, O. Milenkovic and J.B. Huber, "The trapping redundancy of linear block codes," *IEEE Trans. Inform. Theory*, vol. 55, no. 1, pp. 53-63, Jan. 2009.

[17] S. Laendner and O. Milenkovic, "Algorithmic and combinatorial analysis of trapping sets in structured LDPC codes," Proc. *IEEE International Conference on Wireless Networks, Communications and Mobile Computing*, Hawaii, USA, June 13-16, 2005, pp. 630-635.

[18] R. Koetter and P. O. Vontobel, "Graph-covers and iterative decoding of finite length codes," Proc. *IEEE International Symposium on Turbo Codes and Applications,* Brest, France, Sept. 2003, pp. 75-82.

[19] M. Karimi and A. H. Banihashemi, "A message-passing algorithm for counting short cycles in a graph," *CoRR, arxiv.org/abs/1004.3966*.

[20] K. M. Krishnan and P. Shankar, "Computing the stopping distance of a Tanner graph is NP-hard," *IEEE Trans. Inform. Theory*, vol. 53, no. 6, pp. 2278-2280, June 2007.

[21] G. B. Kyung and C.-C. Wang, "Exhaustive search for small fully absorbing sets and the corresponding low error-floor decoder," Proc. *IEEE International Symposium on Inform. Theory,* Austin, TX, June 2010, pp. 739-743.

[22] G. B. Kyung, "private communication," Jan. 4, 2011.

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

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

[25] G. A. Margulis, "Explicit constructions of graphs without short cycles and low density codes," *Combinatorica*, vol. 2, no. 1, pp. 71-78, 1982.

[26] O. Milenkovic, E. Soljanin, and P. Whiting, "Asymptotic spectra of trapping sets in regular and irregular LDPC code ensembles," *IEEE Trans. Inform. Theory*, vol. 53, no. 1, pp. 39-55, Jan. 2007.

[27] K. Murali Krishnan and L. Sunil Chandran, "Hardness of approximation results for the problem of finding the stopping distance in Tanner graphs," Proc. *26th International Conference on Foundations of Software Technology and Theoretical Computer Science* , Dec. 13-15, 2006, pp. 69-80.

[28] B. D. McKay, N. C. Wormald and B. Wysocka, "Short cycles in random regular graphs," *Electronic Journal of Combinatorics*, vol. 11, no. 1, 2004.

[29] D. V. Nguyen, S. K. Chilappagari, M. W. Marcellin, and B. Vasic, "On the construction of structured LDPC codes free of small trapping sets," *IEEE Trans. Inform. Theory*, to appear.

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

[31] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low density parity-check codes," *IEEE Trans. Inform. Theory,* vol. 47, no. 2, pp. 619-637, Feb. 2001.

[32] E. Rosnes and O. 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.

[33] A. Otmani, J.-P. Tillich, I. Andriyanova, "On the minimum distance of generalized LDPC codes," Proc. *International Symposium on Inform. Theory*, Nice, France, June 24-29, 2007, pp. 751-755.

[34] R. M. Tanner, D. Sridhara and T. Fuja, "A class of group-structured LDPC codes," Proc. *International Symposium on Communication Theory and Applications*, Ambleside, U.K., July 2001, pp. 365-370.

[35] J.-P. Tillich and G. Zemor, "On the minimum distance of structured LDPC codes with two variable nodes of degree 2 per parity-check equation," Proc. *IEEE International Symposium on Inform. Theory,* Seattle, WA, July 9-14, 2006, pp. 1549-1553.

[36] T. Tian, C. Jones, J. D. Villasenor, and R. D. Wesel, "Selective avoidance of cycles in irregular LDPC code construction," *IEEE Trans. Communications*, vol. 52, no. 8, pp. 1242-1248, Aug. 2004.

[37] B. Vasic, S. Chilappagari, D. Nguyen, and S. Planjery, "Trapping set ontology," Proc. *47th Annual Allerton Conference on Communication, Control and Computing,* Monticello, IL, Sept. 2009, pp. 1-7.

[38] P. O. Vontobel and R. Koetter, "Graph-cover decoding and finite-length analysis of message-passing iterative decoding of LDPC codes," *CoRR, arxiv.org/abs/cs/0512078.*

[39] D. Vukobratovic and V. Senk, "Generalized ACE constrained progressive edge growth LDPC code design," *IEEE Communications Letters*, vol. 12, no. 1, pp. 32-34, Jan. 2008.

[40] C.-C. Wang, "On the exhaustion and elimination of trapping sets: Algorithms & the suppressing effect," Proc. *International Symposium on Inform. Theory*, Nice, France, June 24-29, 2007, pp. 2271-2275.

[41] C.-C. Wang, S.R. Kulkarni, 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.

[42] H. Xiao and A. H. Banihashemi, "Estimation of bit and frame error rates of low-density parity-check codes on binary symmetric channels," *IEEE Trans. Communications*, vol. 55, pp. 2234-2239, Dec. 2007.

[43] H. Xiao and A. H. Banihashemi, "Error rate estimation of finite-length low-density parity-check codes decoded by soft-decision iterative algorithms," Proc. *International Symposium on Inform. Theory*, Toronto, Ontario, July 6-11, 2008, pp. 439-443.

[44] H. Xiao and A. H. Banihashemi, "Error rate estimation of low-density parity-check codes on binary symmetric channels using cycle enumeration," *IEEE Trans. Communications*, vol. 57, pp. 1550-1555, June 2009.

[45] H. Xiao and A. H. Banihashemi, "Improved progressive-edge-growth (PEG) construction of irregular LDPC codes," *IEEE Communications Letters*, vol. 8, no. 12, pp. 715-718, Dec. 2004.

[46] M.Yang, W. E. Ryan, and Y. Li, "Design of efficiently-encodable moderate-length high-rate irregular LDPC codes," *IEEE Trans. Communications*, vol. 52, no. 4, pp. 564-571, Apr. 2004.

[47] Z. Zhang, L. Dolecek, M. Wainwright, V. Anantharam, and B. Nikolic, "Quantization effects in low-density parity-check decoders," Proc. *IEEE International Conference on Communications*, Glasgow, Scotland, June 24-28, 2007, pp. 6231-6237.

[48] Y. Zhang and W. E. Ryan, "Toward low LDPC-code floors: a case study," *IEEE Trans. Communications*, vol. 57, no. 6, pp. 1566-1573, June 2009.

[49] V. Zyablov and M. Pinsker, "Estimates of the error-correction complexity of Gallager's low-density codes," *Problems of Inform. Transmission,* vol. 11, no. 1, pp. 18-28, Jan. 1976.
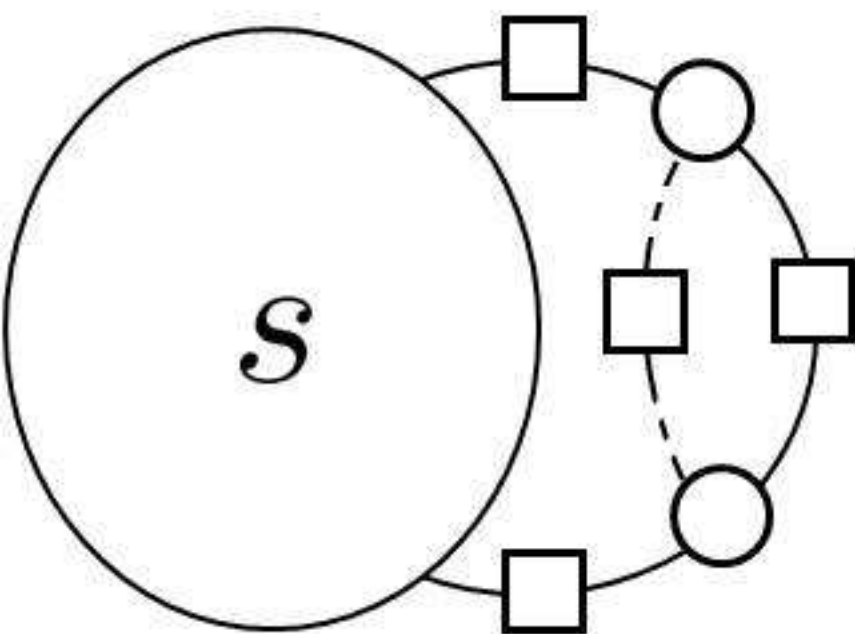
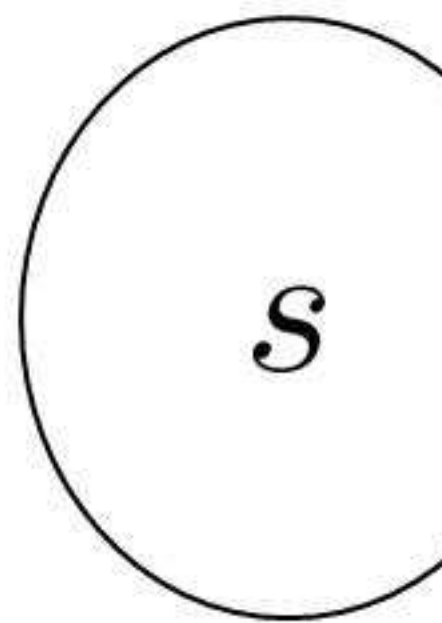[50] [Online] Available: http://www.inference.phy.cam.ac.uk/mackay/codes/.

(a)

(b)

(c)

(d)

(e)