An Efficient Exhaustive Low-Weight Codeword Search for Structured LDPC Codes

Seyed Mehrdad Khatami, Ludovic Danjean, Dung V. Nguyen, Bane Vasić
Department of Electrical and Computer Engineering
University of Arizona
Tucson, AZ 85721, USA
Emails: {khatami,danjean,nguyendv,vasic}@ece.arizona.edu

***Abstract*—In this paper, we present an algorithm to find all low-weight codewords in a given quasi-cyclic (QC) low-density parity-check (LDPC) code with a fixed column-weight and girth. The main idea is to view a low-weight codeword as an $(a, 0)$ trapping sets, and then show that each topologically different $(a, 0)$ trapping set can be dissected into smaller trapping sets. The proposed search method relies on the knowledge of possible topologies of such smaller trapping sets present in a code ensemble, which enables an efficient search. Combined with the high-rate QC LDPC code construction which successively adds blocks of permutation matrices, the algorithm ensures that in the code construction procedure all codewords up to a certain weight are avoided, which leads to a code with the desired minimum distance. The algorithm can be also used to determine the multiplicity of the low-weight codewords with different trapping set structure.**

I. Introduction

It is now established that the presence of certain structures in Tanner graphs of low-density parity-check (LDPC) codes leads to decoder failures and error-floor phenomenon in the high signal-to-noise ratio region. For iterative decoding these structures are generally referred to as *trapping sets*. An iterative decoding algorithm initialized with errors located in variable nodes of a trapping set does not converge to a codeword. If we view an iterative decoder as a dynamical system [1], successful decoding corresponds to a "trajectory" which leads to the original codeword, thus codewords act as strong *attractor points*, while oscillatory behavior occurs in the neighborhood of weaker attractor points corresponding to trapping sets.

Our experimental study has shown that a random, finite length, regular LDPC code free of certain trapping sets has large minimum distance with high probability. Moreover, in [2] we have observed that the short Tanner code with $d_{min} = 20$ [3] contains only three types of minimal-weight codewords and that the subgraphs induced by the the support of any of these codewords contain trapping sets. Moreover, the three codeword types differ in their *trapping set composition*, and the corresponding attractor strengths. These observations lead to a plausible conjecture that, in general, small trapping sets are nested in the support of codewords, and more importantly that the graph induced by the support of a codewords may be decomposed into a union of subgraphs which are mainly trapping sets. Therefore, it seems natural to study the connection between trapping sets and the structure of codewords as well as the relations between the trapping sets present in a code and the minimum distance d_{min} of the code.

Finding the minimum distance of a given code is known to be hard [4], and during the past decade several practical techniques have been proposed to search for low-weight codewords of a code hence providing upper bounds on d_{min}. The first technique, called the error impulse method, has been introduced by Berrou *et al.* [5] in the context of turbo codes [6] and adapted for LDPC codes by Declerq and Fossorier (see [7] and references therein). The impulse method aims at adding a controlled noise to the all-zero codeword in order to make the decoder converge to a nearest non-zero codeword, and then give an upper bound on d_{min} based on the weight of such non-zero codeword. Stern [8] proposed a probabilistic approach to determine low-weight codewords of binary linear codes, which has been modified by Hirotomo *et al.* and used for LDPC codes in [9]. This approach operates on the systematic representation of the parity-check matrix in order to find low-weight codewords. More recently Keha and Duman introduced a branch and cut algorithm [10] to find exact minimum distance of short-length LDPC codes. The algorithm solves two integer programming problems (one to find codewords based on the generator matrix, and another to find codewords in the null space of the parity-check matrix). However the complexity of this algorithm restricts its application to moderate-length codes. None of the above approaches consider the topology of the subgraphs induced by codewords.

Since codewords are just a particular type of trapping set, namely $(a, 0)$ trapping sets (to be introduced precisely later), a naive code construction procedure would involve an exhaustive search for all the subgraphs corresponding to codewords up to a certain size in a Tanner graph, and then removing such structures from a graph in order to obtain a code with a good minimum distance. However the exhaustive search of low-weight codewords or $(a, 0)$ trapping sets is impractical as soon as the codeword weight exceeds 12. Previous work on exhaustive search for trapping sets ([11], [12]) has demonstrated that search for (a, b) trapping sets is practical only when $a \leq 11$ and a Tanner graph has no more than 1000 variable nodes. Efficient algorithms to find trapping sets with a dominant contribution to the error floor have been proposed by Karimi and Banihashemi [13]. Although these algorithms

can handle trapping set of larger size, they may miss low-weight codewords.

On the code design front, the most common approach taken in the literature to achieve large minimum distance when the column-weight is small (three or four) is ensuring that the designed code has a large girth. Indeed, it has been shown that a linear increase in the girth leads to an exponential increase in the lower bound on the minimum distance for codes whose column-weight is greater than three [14]. Large girth is beneficial to iterative decoders because it reduces the statistical dependence of the messages passed along edges of the Tanner graph. However increasing the girth either drastically increases the code length or reduces the row-weight of the code, hence reducing the code rate. Numerous approaches based on this idea have been proposed. The bit-filling [15] consists of adding columns in the parity-check matrix while satisfying the constraints on the girth. Significant effort has also been made in increasing girth of structured codes using algebraic codes [16], array codes [17], [18] or quasi-cyclic (QC) LDPC codes [19], [20], [21], [22].

MacKay and Davey [23] and Fossorier [19] established an upper bound of $(d_v + 1)!$ on the minimum distance for (d_v, d_c)-regular QC-LDPC codes. Bocharova *et al.* [24] used computer search to obtain several low-rate codes ($R = 2/5$ and $R = 3/6$) that achieve this upper bound. Theses codes are based on convolutional LDPC codes and are also obtained by increasing the girth of the Tanner graph. In [24] the authors give an overview of results on improving minimum distance of QC codes by increasing the girth. By generalizing the work presented in [19] Smarandache and Vontobel [25] derived upper bounds for a special class of regular QC codes using protographs. The construction of protograph-based LDPC codes with a linear minimum distance has also been considered in [26] and[27].

In this paper, we consider an ensemble of (d_v, d_c)-regular LDPC codes of length n and girth g, denoted $\mathcal{C}^n(d_v, d_c, g)$, and give an algorithm which determines the trapping set composition of all non-isomorphic Tanner graphs corresponding to weight-w codewords present in a given code in $C^n(d_v, d_c, g)$. The algorithm utilizes three key expansion operations to search for subgraphs by adding nodes or trapping sets to existing subgraphs. Combined with the code construction that we have recently developed [28], the algorithm ensures that in the code construction procedure all codewords of weights up to w are avoided, which leads to a code whose minimum distance is at least $w + 2$. The algorithm can be also used to determine the multiplicity of the lowest weight codewords with different trapping set structure.

The paper gives then the preliminary insights of the structure of codewords of left-regular LDPC codes with a given girth and the link between the presence of trapping sets and the minimum distance of a code. To provide these insights, we present a case study for codes belonging to $C^n(3, d_c, 8)$, and the minimum distance up to 16. We present the search method for weight-14 codewords of the $C^n(3, d_c, 8)$ ensemble. Generalization of the method to LDPC codes with different column-weight and/or girth can be done by modifications of several steps at the cost of increasing the computational complexity of the search.

The rest of the paper is organized as follows. Section II provides the necessary preliminaries regarding LDPC codes, trapping sets and check coloring of structured LDPC codes. In Section III we review the trapping set ontology and we introduce the three expansion operations which are used throughout this paper. Section IV gives the main contribution of the paper, which is a search method that guarantee to find all the low-weight codewords in the $\mathcal{C}^n(d_v, d_c, g)$ ensemble. We conclude the paper by a discussion in Section V.

II. Preliminaries

In this section, we provide some background and notations on LDPC codes [29]. Let $\mathcal{C}$ denote a (n, k) LDPC code over the binary field GF(2). $\mathcal{C}$ is defined by the null space of an $m \times n$ parity-check matrix H over GF(2) where m is the number of constraints on the n bits of the code. $\mathcal{C}$ can be represented by a bipartite graph G, known as Tanner graph [14], which consists of two sets of nodes: the set of the m parity-check nodes (or simply check nodes) $C = \{c_1, c_2, ..., c_m\}$ and the set of the n variable nodes $V = \{v_1, v_2, ..., v_n\}$. An edge connects a check node i to a variable node j if the entry at the ith row and jth column in H is 1. A vector $\mathbf{x} = (x_1, x_2, ..., x_n)$ is a codeword if and only if $\mathbf{x}H^T = 0$. The support of $\mathbf{x}$ denoted by $supp(\mathbf{x})$ is the set of all variable nodes $v \in V$ such that $x_v \neq 0$. The weight of the vector $\mathbf{x}$ is given by $|supp(\mathbf{x})|$. The minimum distance d_{min} of a code $\mathcal{C}$ is given by the minimum weight of the non-zero codewords of $\mathcal{C}$. An LDPC code is said to be d_v-left regular if each variable node in G has degree d_v. Similarly, an LDPC code is said to be d_c-right regular if each check node in G has degree d_c. d_v and d_c are also called column-weight and row-weight, respectively. A (d_v, d_c)-regular LDPC code is both d_v-left regular and d_c-right regular and the rate of such a code is given by $R \geq 1 - d_v/d_c$. The length of the shortest cycle in G is called the *girth* of G and is denoted by g. In this paper, $\mathcal{C}^n(d_v, d_c, g)$ denotes the ensemble of (d_v, d_c)-regular LDPC codes with girth g.

Let $V' \subseteq V$ be a subset of variable nodes of G. The subgraph induced by V' contains all the edges of G that connect the variable nodes of V' to their neighboring check nodes of and is denoted by $\mathbf{I}(V')$.

A. Trapping set

Let $\mathbf{x} = (x_1, \ldots, x_n)$ be a transmitted codeword. Consider an iterative decoder and let $\hat{\mathbf{x}}^l = (\hat{x}_1^l, \ldots, \hat{x}_n^l)$ be the decision vector after the lth iteration. A variable node $v \in V$ is *eventually correct* if there exists a positive integer l_c such that for all $l \geq l_c$, $x_v = \hat{x}_v^l$. The decoding is successful if all the variables in V are eventually correct. We adopt the definition of trapping sets from [30]. Let $\mathbf{r}$ be the input of an iterative decoder and suppose the decoding was not successful.

Definition 1. *[30] A trapping set $\mathbf{T}(\mathbf{r})$ is defined as a non-empty set of variable nodes in G which are not eventually correct. Since two different decoder input vectors can result*

in the same trapping set, a trapping set $\mathbf{T}(\mathbf{r})$ is denoted simply by $\mathcal{T}$. A trapping set $\mathcal{T}$ is an (a, b) trapping set if the subgraph induced by $\mathcal{T}$, $\mathbf{I}(\mathcal{T})$, has a variable nodes and b odd-degree check nodes.

The following theorem for trapping sets of d_v-left-regular LDPC code is adopted from [31] and [28] which will be needed in the remainder of the paper.

Theorem 1. *Let $\mathcal{C}$ be a d_v-left-regular LDPC code. Let $\mathbf{T}$ be a set of variable nodes. Let $\mathbf{O}$ and $\mathbf{E}$ be two disjoint sets consisting of odd- and even-degree check nodes of $\mathbf{I}(\mathbf{T})$, respectively. $\mathbf{T}$ is a trapping set for Gallager A/B decoder if every variable node in $\mathbf{I}(\mathbf{T})$ has at least $\lceil \frac{d_v}{2} \rceil$ check nodes in $\mathbf{E}$ and no collection of $\lfloor \frac{d_v}{2} \rfloor + 1$ check nodes of $\mathbf{O}$ share a neighbor outside $\mathbf{I}$.*

Throughout the paper, the term trapping set is used according to Theorem 1. Moreover, the trapping set ontology [32] used in this paper is based on this theorem for Gallager A/B decoder. The *union* of trapping sets $\mathbf{T}_1$ of type $\mathcal{T}_1$ and $\mathbf{T}_2$ of type $\mathcal{T}_2$ is an induced subgraph of all the variable nodes which are in both $\mathbf{T}_1$ and $\mathbf{T}_2$. $\mathbf{I}(\mathbf{T}_1 - \mathbf{T}_2)$ denotes an induced subgraph of variables which are in $\mathbf{T}_1$, but not in $\mathbf{T}_2$.

Let $\mathbf{x}$ be a non-zero codeword of $\mathcal{C}$ and $\mathbf{T} = supp(\mathbf{x})$. Thus $\mathbf{T}$ is an $(a, 0)$ trapping set. Conversely, if there exists an $(a, 0)$ trapping set in $\mathcal{C}$, then $\mathcal{C}$ has a codeword of weight a. Therefore, a code $\mathcal{C}$ has minimum distance d_{min} if and only if for $a < d_{min}$, $\mathcal{C}$ contains no $(a, 0)$ trapping set and there exists at least one $(d_{min}, 0)$ trapping set in $\mathcal{C}$. We denote elementary codewords as codewords whose induced subgraph only contain degree-2 check nodes. Elementary trapping sets are trapping sets whose subgraphs only contain degree-1 or degree-2 check nodes. From now on, the term codeword refers to an $(a, 0)$ trapping set which is the induced subgraph on the support of the codeword.

Fig. 1 depicts three trapping sets present in $\mathcal{C}^n(3, d_c, 8)$, namely the $(4, 4)$ trapping set, the $(5, 3)$ trapping set and the only weight-6 codeword for this ensemble of LDPC codes, also referred to as the $(6, 0)$ trapping set. In this figure and throughout the paper, ● denotes a variable node, □ denotes an even-degree check node, and ■ represents an odd-degree check node.

B. Check coloring for structured LDPC codes

The definition of structured LDPC codes is adopted from [28]. In particular, we consider LDPC codes whose parity-check matrices are arrays of permutation matrices. A permutation matrix is a square binary matrix with only one entry binary one in each row and column and zeros elsewhere.

There exists an intrinsic property of check nodes in the Tanner graph of the structured LDPC codes. This property is referred to as *coloring* and is given by the following definition.

Definition 2. *Let $K = \{1, 2, ..., d_v\}$ be a set of colors. A coloring of check nodes in a Tanner graph G of a d_v-left regular LDPC code is a function $f : C \rightarrow K$, such that*

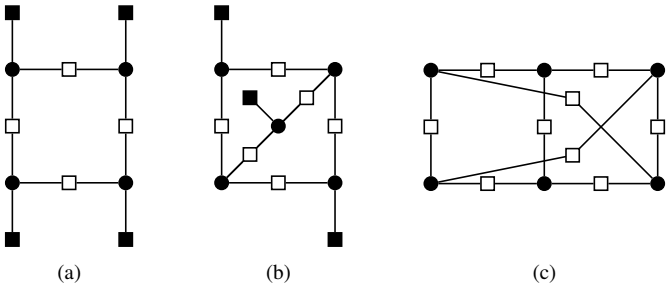

Figure 1. Tanner graph representation of trapping sets for column-weight-3 and girth-8 LDPC codes : (a) the $(4, 4)$ trapping set. (b) the $(5, 3)$ trapping set. (c) $(6, 0)$ trapping set (codeword of weight 6).

$\forall\ c_i, c_j \in C$, $i \neq j$ and $v \in V$ connected to c_i and c_j, $f(c_i) \neq f(c_j)$.

Proposition 1. *There exists a coloring for a d_v-left regular structured LDPC code.*

Proof: The basic blocks of the structured LDPC code are permutation matrices. Since the code is d_v-left regular each column of the parity-check matrix consists of d_v permutation matrices. Let us assign colors $1, \ldots d_v$ to the check nodes corresponding to the rows of the parity-check matrix such that all the check nodes in the same permutation matrix have the same color. For this coloring, suppose there exist distinct check nodes $c_i, c_j \in C$ connected to a common variable node such that $f(c_i) = f(c_j)$. Therefore, the rows corresponding to c_i and c_j are in the same permutation matrix. Since c_i and c_j are connected to a common variable node, the column corresponding to that variable node has entry 1 in c_i and c_j which is a contradiction since c_i and c_j are in the same permutation matrix and each permutation matrix can have only one non-zero entry in each row and column. ■

The coloring property will be used to bound the number of possible codewords which may be generated for a given weight w. Exploiting coloring property results in huge computational complexity reduction of the search algorithms.

III. Trapping set ontology and expansion operations

In this section, we present the basic components of the search algorithm. These components are adopted from the trapping set ontology (TSO) [32] specifically for column-weight-3 and girth-8 LDPC codes. The TSO is the database of trapping sets and it gives the relations between their topological structures. In other words, in addition to the identification of the trapping sets present in an ensemble of codes, the TSO also provides guidance through the search procedure for trapping sets using the knowledge of the topological relationships between them.

The TSO greatly simplifies the enumeration of trapping sets in a code using the underlying parent-child relation between them compared to the exhaustive search of individual trapping sets. The following definition gives the formal concept of the parent-child relations between trapping sets.

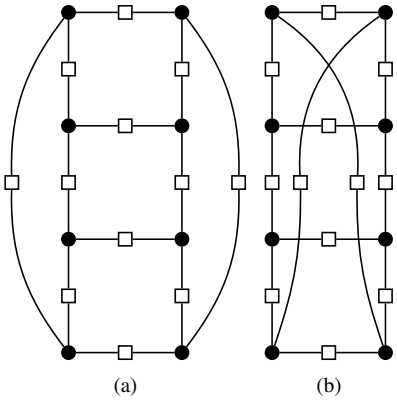

Figure 2. Tanner graph representation of the two topologies for weight-8 codewords for column-weight-3 and girth-8 LDPC codes.

Definition 3. *[28] A trapping set $\mathcal{T}_1$ is said to be a successor of a trapping set $\mathcal{T}_2$ if there exists a subset V_2 of variable nodes of $\mathcal{T}_1$, where $\mathbf{I}(V_2)$ (the subgraph induced by V_2) is isomorphic to $\mathcal{T}_2$. Then $\mathcal{T}_2$ is called a predecessor of $\mathcal{T}_1$.*

For example, the $(5,3)$ trapping set of Fig. 1(b) is obtained by addition of one variable node to the $(4,4)$ trapping set of Fig. 1(a). Thus, the $(5,3)$ trapping set is a successor of the $(4,4)$ trapping set and the $(4,4)$ trapping set is the predecessor of the $(5,3)$ trapping set. Also we can see that the $(6,0)$ trapping set of Fig. 1(c) is a successor of the $(4,4)$ trapping sets. Similarly we have depicted in Fig. 2 the only two topologies corresponding to a weight-8 codeword for column-weight-3 and girth-8 LDPC code. One can easily see that trapping sets (namely $(4,4)$, $(5,3)$, $(6,4)$) are predecessors of these $(8,0)$ trapping sets.

The main result of this paper is an algorithm to search for all the codewords up to certain weight. Let G be the Tanner graph corresponding to a code $\mathcal{C}$. The search algorithm has G as an input and verifies whether any codeword up to certain weight is contained in G. To search for the codewords, it is simpler to search for the smaller trapping set which are the predecessors of the codewords. TSO facilitates an efficient search of trapping sets. The search begins with enumeration of cycles with a variable nodes (which can be seen as a (a,a) trapping set for column-weight-3 codes) since every trapping set contains at least one cycle. Then larger trapping sets can be found by expanding these smaller trapping sets. The term *Expansion Operations* used in this paper refers to three easily programmable subroutines which accept subgraph $\mathcal{T}_1$ and return subgraph $\mathcal{T}_2$, successor of $\mathcal{T}_1$. In other words, $\mathcal{T}_2$ is searched for by checking if the expansion of subgraph $\mathcal{T}_1$ exists in G. We show later that these three Expansion Operations are sufficient to search for all the successor of trapping sets leading to a codeword.

The expansion operations allow us to search for all the codewords up to certain weight without searching for them individually. These operations are adopted from [32] and they are used to search for larger trapping sets from smaller trapping sets. These operations provide a method to search for a trapping set from its parent by adding one variable, or another trapping set to a parent trapping set.

Expansion Operations

1) **Expansion Operation 1 ($\mathcal{EO}_1$):**
Searching for (c,d) trapping set generated by the union of (a_1,b_1) and (a_2,b_2) trapping sets:

Let $\mathcal{T}_1$ be a (a_1,b_1) trapping set and let $\mathcal{T}_2$ be a (a_2,b_2) trapping set. Let $\mathcal{T}_3$ be a (c,d) trapping set where $\mathcal{T}_3$ is a successor of $\mathcal{T}_1$ and $\mathcal{T}_2$. Let $\mathbf{T}_1$, $\mathbf{T}_2$ and $\mathbf{T}_3$ be the trapping sets of types $\mathcal{T}_1$, $\mathcal{T}_2$ and $\mathcal{T}_3$, respectively and let V_1 and V_2 be the set of variable nodes of $\mathbf{T}_1$ and $\mathbf{T}_2$, respectively. Then, to search for $\mathbf{T}_3$, it is sufficient to check if the induced subgraph $\mathbf{I}(\mathbf{V})$ is of type $\mathcal{T}_3$ where $V = V_1 \cup V_2$.

2) **Expansion Operation 2 ($\mathcal{EO}_2$):**
Searching for $(a+1,b-1)$ trapping sets generated by (a,b) trapping sets:

Let $\mathcal{T}_1$ be a (a,b) trapping set and let $\mathcal{T}_2$ be a $(a+1,b-1)$ trapping set where $\mathcal{T}_2$ is a successor of $\mathcal{T}_1$. $\mathcal{T}_2$ can be obtained by adjoining one variable node to $\mathcal{T}_1$. Let $\mathbf{T}_1$ and $\mathbf{T}_2$ be the trapping sets of types $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively. So in order to search for $\mathbf{T}_2$, it is more convenient to search for a variable node connected to two odd-degree check nodes of the smaller subgraph $\mathbf{I}(\mathbf{T}_1)$.

3) **Expansion Operation 3 ($\mathcal{EO}_3$):**
Searching for $(a+1,b-3)$ trapping sets generated by (a,b) trapping sets:

Let $\mathcal{T}_1$ be a (a,b) trapping set and let $\mathcal{T}_2$ be a $(a+1,b-3)$ trapping set where $\mathcal{T}_2$ is a successor of $\mathcal{T}_1$. $\mathcal{T}_2$ can be obtained by adjoining one variable node to $\mathcal{T}_1$. Let $\mathbf{T}_1$ and $\mathbf{T}_2$ be the trapping sets of types $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively. Then, to search for $\mathbf{T}_2$, it is sufficient to search for a variable node connected to three odd-degree check nodes of the smaller subgraph $\mathbf{I}(\mathbf{T}_1)$.

The search for $(a+1,b-3)$ trapping sets is very similar to the search procedure for $(a+1,b-1)$ trapping sets. Fig. 3 illustrates the expansion operations. In these figures, $\widetilde{\mathcal{T}}$ denotes as the set of variable nodes and even-degree check nodes contained in the trapping set $\mathcal{T}$. Fig. 3(a) shows $\mathcal{EO}_1$ where the trapping sets $\mathcal{T}_1$ and $\mathcal{T}_2$ share a certain number of odd-degree check nodes to form the successor trapping set $\mathcal{T}_3$. Fig. 3(b) depicts the $\mathcal{EO}_2$ where one variable node connecting two odd-degree checks nodes of the trapping set $\mathcal{T}_1$ leads to form the successor trapping set $\mathcal{T}_2$. $\mathcal{EO}_3$ is illustrated in Fig. 3(c) in which one variable node connects three odd-degree checks nodes of the trapping set $\mathcal{T}_1$ to form the successor trapping set $\mathcal{T}_2$.

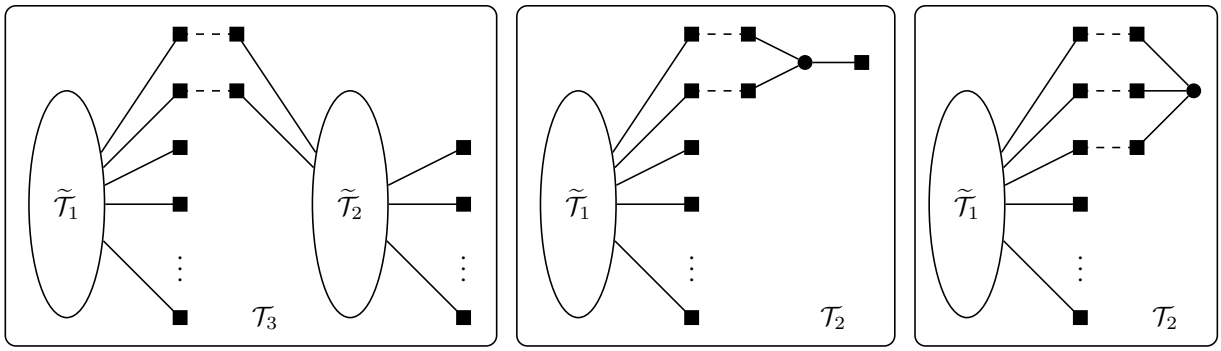

Figure 3. Illustration of the expansion operations: $\widetilde{\mathcal{T}}$ denotes the set of variable nodes and even-degree check nodes contained in the trapping set $\mathcal{T}$. (a) $\mathcal{EO}_1$: Searching for $\mathcal{T}_3$ as a union of $\mathcal{T}_1$ and $\mathcal{T}_2$. (b) $\mathcal{EO}_2$: Searching for $(a+1, b-1)$ trapping set ($\mathcal{T}_2$) starting from a (a, b) trapping set ($\mathcal{T}_1$). (c) $\mathcal{EO}_3$: Searching for $(a+1, b-3)$ trapping set ($\mathcal{T}_2$) starting from a (a, b) trapping set ($\mathcal{T}_1$).

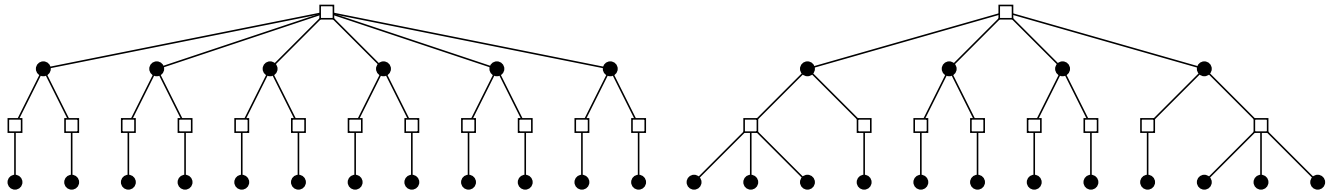

Figure 4. Computation tree of a column-weight-three and girth-eight codeword with (a) one degree-6 check node, (b) three degree-4 check nodes.

IV. Search method

In this section, we present the method to search for all the low-weight codewords in an LDPC code $\mathcal{C}$ with Tanner graph G. In the remainder of the paper, all the statements pertain to $\mathcal{C}^n(d_v = 3, d_c, g = 8)$ unless stated otherwise. This choice has been governed by the availability of the TSO for such LDPC codes. This method can be generalized to any column-weight and girth, provided the TSO for this LDPC code ensemble is known. The search method is presented for the codewords of weight 14. The generalization of this method for codewords with larger weight is straightforward by adding extra steps to the search method. The proof proceeds by classifying the codewords into a finite number of categories and dealing with each category separately. In each case, we prove that the search procedure is able to search for all the codewords in that category. The proofs are constructive and provided along the paper as they illustrate the search algorithm. The proofs also give insights in possible extensions to LDPC codes with different column-weights and girths.

Since the number of possible codewords of weight 14 is very large, it is not possible to search for all of them one by one. Even if possible it would be highly inefficient since a given code does not contain all possible codewords. Our search method categorizes the codewords in such a way that every category of codewords can be searched for in a finite number of steps. Lemma 1 and 2 are presented in order to bound the categories of weight-14 codewords in $\mathcal{C}^n(3, d_c, 8)$.

Lemma 1. *In the $\mathcal{C}^n(3, d_c, 8)$ ensemble, there is no weight-14 codeword with a check node of degree 6.*

Proof: Assume there exists a weight-14 codeword with *girth 8* with a check node of degree-6. We start building the subgraph of this codeword from the degree-6 check node as a root (hence connected to six variable nodes). Each variable node branches to two other check nodes since the column-weight is three. None of these check nodes can be the same since it would results in a 4-cycle which violates the girth-8 constraint. Then each check node is connected to at least one other variable node to have even-degree check nodes. By a similar argument, none of these variable nodes can be the same because it would create a 6-cycle. As it is shown in Fig. 4(a), in order to have a codeword with a degree-6 check node, at least 18 variable nodes are required to satisfy the girth-8 constraint. Thus, it is not possible to have a weight-14 codeword with a degree-6 check node. ■

Lemma 2. *In the $\mathcal{C}^n(3, d_c, 8)$ ensemble, there is no weight 14 codeword with three check nodes of degree 4.*

Proof: By the same argument used in the proof of Lemma 1, we start building the subgraph of a codeword using a degree-4 check node as a root. Similarly to Lemma 1, we expand the subgraph while satisfying the girth-8 constraint. There are two possible subgraphs. Fig. 4(b) shows the first resulting subgraph which does not violate the girth-8 constraint. Therefore, in order to have a weight-14 codeword with three degree-4 check nodes, at least 16 variable nodes are required. The second

possible subgraph (not shown) is a subgraph which the degree-4 check nodes does not share a common variable. However it is not possible to obtain a weight-14 codeword while satisfying the girth-8 constraint by the same reasoning. ■

As it was stated in the previous sections, codewords of weight 14 are $(14, 0)$ trapping sets. The following proposition categorizes the $(14, 0)$ trapping sets.

Proposition 2. *Any (14,0) trapping sets of $\mathcal{C}^n(3, d_c, 8)$ belongs to one of the following three categories.*

1) *Elementary codewords with only degree-2 check nodes.*
2) *Codewords with only one degree-4 check node, and every other check nodes have degree 2.*
3) *Codewords with only two degree-4 check nodes, and every other check nodes have degree 2.*

Proof: To prove Proposition 2, it is sufficient to show that

1) there exists no $(14, 0)$ graph with one check node of degree greater than 4.
2) there exists no $(14, 0)$ graph with more than two degree 4 check nodes.

These statements directly come from Lemma 1 and 2. ■

Before explaining the search algorithm, we present the rationale behind the steps of the search. Since all the codewords of $\mathcal{C}^n(3, d_c, 8)$ have an 8-cycle, the starting point of the procedure is to search for the 8-cycles, which are $(4, 4)$ trapping sets. Our goal is to expand the $(4, 4)$ trapping sets to reach a $(14, 0)$ trapping sets with the use of expansion operations explained in Sec. III. Then, in the subsequent steps, we will be dealing with (a, b) trapping sets found in the previous step. In general, let G be the input Tanner graph to the search algorithm. Let $\mathcal{T}$ be a (a, b) trapping and $\mathbf{T}$ be a trapping set of type $\mathcal{T}$ in G. There are three cases to expand $\mathbf{T}$ in order to search for the successors of $\mathcal{T}$. These three cases perfectly fit to the previously mentioned expansion operations.

Case 1: There exists no odd-degree check node in $\mathbf{I}(\mathbf{T})$ connected to a common variable node in G . Therefore, every odd-degree check node of $\mathbf{T}$ is connected to different variable nodes of G. In this case, all the $(14, 0)$ codewords satisfying this condition can be found by searching for the union of $\mathbf{T}$ and $(14 - a, b)$ trapping sets using the $\mathcal{EO}_1$.

Proposition 3. *Let $\mathcal{T}$ be a $(14, 0)$ trapping set in $\mathcal{C}^n(3, d_c, 8)$, and $\mathcal{T}'$ be a (a, b) trapping set where $\mathcal{T}'$ is a successor of $\mathcal{T}$. Let $\mathbf{T}$ and $\mathbf{T}'$ be the trapping sets of types $\mathcal{T}$ and $\mathcal{T}'$, respectively, and suppose there is no variable node in $\mathbf{I}(\mathbf{T} - \mathbf{T}')$ connected to two or more odd-degree check nodes of $\mathbf{I}(\mathbf{T}')$. Then the subgraph introduced by $\mathbf{I}(\mathbf{T} - \mathbf{T}')$ is a $(14 - a, b)$ trapping set.*

Proof: For a 3-left-regular LDPC code, Theorem 1 states that $\mathbf{T}$ is a trapping set for Gallager A/B decoder if every variable node in $\mathbf{I}(\mathbf{T})$ has at least two even-degree check nodes. In other words, each variable node must have at most one odd-degree check node. The induced subgraph $\mathbf{I}(\mathbf{T} - \mathbf{T}')$ has $(14 - a)$ variable nodes and by removing the induced subgraph $\mathbf{I}(\mathbf{T}')$ from $\mathbf{I}(\mathbf{T})$, it is clear that it has b odd-degree check nodes. Since there are no odd-degree check nodes in the induced subgraph $\mathbf{I}(\mathbf{T}')$ connected to a common variable node in $\mathbf{I}(\mathbf{T} - \mathbf{T}')$, thus each variable node in $\mathbf{I}(\mathbf{T} - \mathbf{T}')$ has at most one odd-degree check node. Therefore, $\mathbf{I}(\mathbf{T} - \mathbf{T}')$ is a $(14 - a, b)$ trapping set. ■

Case 2: There exists at least one variable node in G connected to two odd-degree check nodes of $\mathbf{I}(\mathbf{T})$. In this case, $\mathbf{T}$ can be extended to an $(a+1, b-1)$ trapping set using the $\mathcal{EO}_2$.

Case 3: There exists at least one variable node in G connected to three odd-degree check nodes of $\mathbf{I}(\mathbf{T})$. In this case, $\mathbf{T}$ can be extended to an $(a+1, b-3)$ trapping set using the $\mathcal{EO}_3$.

These 3 cases cover all the possible categories that leads to a $(14, 0)$ trapping set. These are the basic blocks in our search procedure. We check all three cases in each stage of the search to make sure all the $(14, 0)$ trapping sets are searched for.

A. Searching for elementary codewords

The search procedure for all the elementary codewords in the graph G is given below:

Algorithm 1 Search procedure steps for elementary codewords.

1) Search for all the $(4, 4)$ trapping sets (8-cycles) in G.
2) For all the $(4, 4)$ trapping sets which satisfy Case 1, use the $\mathcal{EO}_1$ to search for the union of $(4, 4)$ and all the elementary $(10, 4)$ trapping sets.
3) For all the $(4, 4)$ trapping sets which satisfy Case 2, use the $\mathcal{EO}_2$ to search for all the $(5, 3)$ trapping sets.
   Remark 1. There is no $(5, 1)$ trapping set of girth 8 to be searched for using the $\mathcal{EO}_3$.
4) For all the $(5, 3)$ trapping sets which satisfy Case 1, use the $\mathcal{EO}_1$ to search for the union of $(5, 3)$ and all the elementary $(9, 3)$ trapping sets.
5) For all the $(5, 3)$ trapping sets which satisfy Case 2, use the $\mathcal{EO}_2$ to search for all the $(6, 2)$ trapping sets.
   Remark 2. Using $\mathcal{EO}_3$, it is possible to search for $(6, 0)$ trapping sets. However, starting from a $(6, 0)$ trapping set, it is not possible to search for $(14, 0)$ because there is no odd-degree check node left to adjoin to any other subgraph.
6) For all the $(6, 2)$ trapping sets which satisfy Case 1, use the $\mathcal{EO}_1$ to search for the union of $(6, 2)$ and all the elementary $(8, 2)$ trapping sets.
7) For all the $(6, 2)$ trapping sets which satisfy Case 2, use the $\mathcal{EO}_2$ to search for all the $(7, 1)$ trapping sets.
8) For all the $(7, 1)$ trapping sets which satisfy Case 1, use the $\mathcal{EO}_1$ to search for the union of $(7, 1)$ and all the elementary $(7, 1)$ trapping sets.

Fig. 5 shows the steps 2, 4, 6 and 8 in Algorithm 1 which lead to the elementary weight-14 codewords. The dashed line on these figures represent the connection which has to be made in order to obtain a $(14, 0)$ trapping set. For instance, Fig. 5(a) shows the union of a $(4, 4)$ trapping set (upper subgraph) and

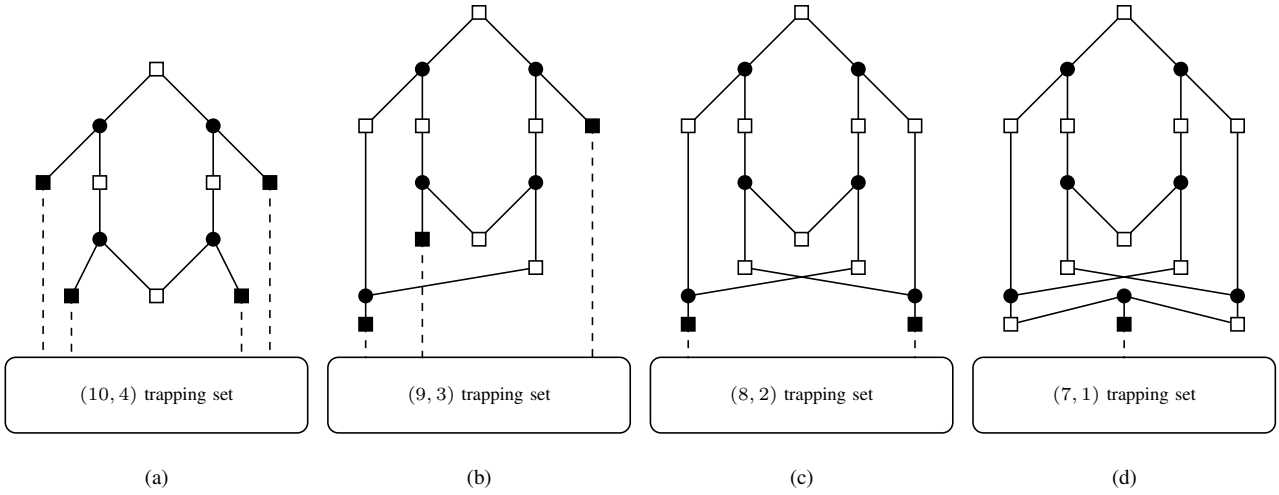

Figure 5. The steps 2, 4, 6 and 8 in Algorithm 1 in which $\mathcal{EO}_1$ is used to search for the elementary codewords of weight 14.

a $(10, 4)$ trapping set (connected topology) which leads to a weight-14 codeword. Similarly Fig. 5(b) shows that the union of a $(5, 3)$ trapping set (upper subgraph) and a $(9, 3)$ trapping set (connected topology) leads to a weight-14 codeword. Note that the TSO ([32]) is the guideline in searching for trapping sets needed in the search procedure.

B. Search procedure for the codewords with one degree-4 check node

The presence of a degree-4 check node introduces some complications to the search procedure. The starting point of the search is a subgraph including the degree-4 check node. Since there is only one degree-4 check node in the codewords and it is already in the starting search subgraphs, all the adjoining trapping sets in the search procedure are elementary trapping sets.

Proposition 4. *The two subgraphs shown in Fig. 6(a) and 6(b) are the predecessors of all the codewords in $\mathcal{C}^n(3, d_c, 8)$ with only one degree-4 check node.*

Proof: Omitted due to page limitations. ■

The subgraphs in Fig. 6(a) is not a trapping set. Nevertheless, the expansion operations are still applicable to search for larger subgraphs from smaller subgraphs. For the reminder of the paper, the term (a, b) *subgraph* is used for a subgraph with a variable node and b odd-degree check node. The search proceeds to two separate cases. The first step is to search for these two subgraphs. Let $\mathcal{S}$ be the $(7, 7)$ subgraph of Fig. 6(a) and $\mathcal{T}$ be the $(8, 8)$ trapping set of Fig. 6(b). Then, two separate algorithms are presented to search for the codewords containing subgraphs of type $\mathcal{S}$ and $\mathcal{T}$. The first step is to find subgraph $\mathcal{S}$. To search for $\mathcal{S}$, $\mathcal{EO}_1$ is used to search for the union of two (4,4) trapping sets in a way which results in $\mathcal{S}$ as a (7,7) subgraph. Searching for the weight-14 codewords containing a subgraph of type $\mathcal{S}$ is given by the following algorithm.

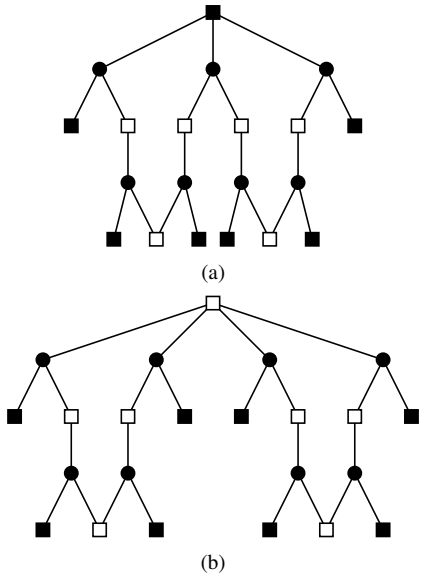

Figure 6. First steps of the search procedure for all the elementary codewords of weight 14 with only one degree-4 check node (a) First case. (b) Second case.

Algorithm 2 Search procedure steps for codewords with one degree-4 check node containing a subgraph of type $\mathcal{S}$.

1) For all the subgraphs of type $\mathcal{S}$ which satisfy Case 1, $\mathcal{EO}_1$ is used to search for the union of the $(7, 7)$ subgraph and all the elementary $(7, 7)$ trapping sets.
2) For all the subgraphs of type $\mathcal{S}$ which satisfy Case 2, $\mathcal{EO}_2$ is used to search for all the $(8, 6)$ subgraphs.
   Remark 3. There is no subgraph of girth 8 to be searched for using $\mathcal{EO}_3$. It is easy to see that by adjoining one variable node to any 3 odd-degree check node of $\mathcal{S}$ in Fig. 6(a), a 6-cycle is created.
3) For all the $(8, 6)$ subgraphs which satisfy Case 1, use the $\mathcal{EO}_1$ to search for the union of the $(8, 6)$ subgraphs and all the elementary $(6, 6)$ trapping sets.

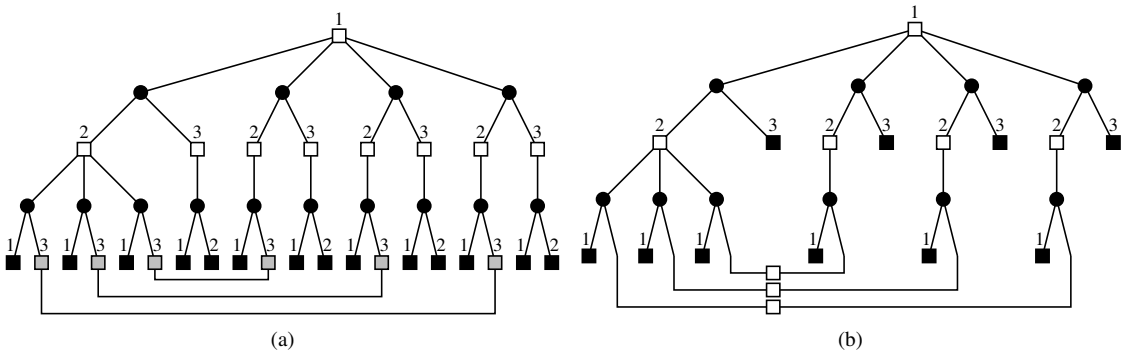

Figure 7. First steps of the search procedure for all the elementary codewords of weight 14 with two degree-4 check node sharing one variable node.

4) For all the $(8,6)$ subgraphs which satisfy Case 2, use the $\mathcal{EO}_2$ to search for all the $(9,5)$ subgraphs.
5) For all the $(8,6)$ subgraphs which satisfy Case 3, use the $\mathcal{EO}_3$ to search for all the $(9,3)$ subgraphs.
6) For all the $(9,5)$ subgraphs which satisfy Case 1, use the $\mathcal{EO}_1$ to search for the union of the $(9,5)$ subgraphs and all the elementary $(5,5)$ trapping sets.
7) For all the $(9,3)$ subgraphs which satisfy Case 1, use the $\mathcal{EO}_1$ to search for the union of the $(9,3)$ subgraphs and all the elementary $(5,3)$ trapping sets.
8) For all the $(9,5)$ subgraphs which satisfy Case 2, use the $\mathcal{EO}_2$ to search for all the $(10,4)$ subgraphs.
9) For all the $(10,4)$ subgraphs which satisfy Case 1, use the $\mathcal{EO}_1$ to search for the union of the $(10,4)$ subgraphs and all the elementary $(4,4)$ trapping sets.

To search for the weight-14 codewords containing a trapping set of type $\mathcal{T}$, the first step is to find $\mathcal{T}$. To search for $\mathcal{T}$, $\mathcal{EO}_1$ is used to search for the union of two (4,4) trapping sets in a way which results in $\mathcal{T}$ as a (8,8) trapping set with one degree-4 check node (see Fig. 6(b)). The rest of the algorithm is as follows:

Algorithm 3 Search procedure steps for codewords with one degree-4 check node containing a subgraph of type $\mathcal{T}$.

1) For all the trapping sets of type $\mathcal{T}$ which satisfy Case 2, use the $\mathcal{EO}_2$ to search for all the $(9,7)$ trapping sets.
2) For all the trapping sets of type $\mathcal{T}$ which satisfy Case 3, use the $\mathcal{EO}_3$ to search for all the $(9,5)$ trapping sets.
3) For all the $(9,7)$ trapping sets which satisfy Case 3, use the $\mathcal{EO}_3$ to search for all the $(10,4)$ trapping sets.
4) For all the $(9,5)$ trapping sets which satisfy Case 1, use the $\mathcal{EO}_1$ to search for the union of the $(9,5)$ trapping sets and all the elementary $(5,5)$ trapping sets.
5) For all the $(9,5)$ trapping sets which satisfy Case 2, use the $\mathcal{EO}_2$ to search for all the $(10,4)$ trapping sets.
6) For all the $(10,4)$ trapping sets which satisfy Case 1, use the $\mathcal{EO}_1$ to search for the union of the $(10,4)$ trapping sets and all the elementary $(4,4)$ trapping sets.

From Proposition 4, all the codewords of this category can be searched using Algorithm 2 and Algorithm 3 which concludes the search for all weight-14 codewords with exactly one degree-4 check nodes. The next category of weight-14 codewords concerns codewords with exactly two degree-4 check nodes.

C. Search procedure for the codewords with two degree-4 check node

In the $\mathcal{C}^n(3,d_c,8)$ code ensemble, the codewords with two degree-4 check nodes are categorized into two groups given below:

1) Codewords whose two degree-4 check nodes are connected to a common variable node.
2) Codewords whose two degree-4 check nodes are not connected to a common variable node.

Proposition 5. *There is no codeword of weight 14 in* 3*-left regular structured LDPC codes with girth 8 with two degree-4 check nodes connected to a common variable node.*

Proof: The proof is involved but gives insight into the restrictions of the coloring property for the structure of the codewords. The starting point of the search is a subgraph containing two degree-4 check nodes connected to a common variable node. We set one of degree-4 check nodes as the root of the starting subgraph, and then expanding it to the next levels satisfying the column-weight-3 and girth-8 constraints and the constraint that there are only two degree-4 check nodes. In order to preserve the girth-8 constraint, no check nodes or variable nodes can be connected until the fifth level. This is illustrated in Fig. 7(a). Since we already have 14 variable nodes at this level, all the check nodes must be shared between the variable nodes. In order to bound the number of cases possible, we start by coloring the check nodes of Fig. 7(a) . Without loss of generality, a coloring is set starting with 1 as the color of the root check node. Continuing the coloring until the last level, one can see that there are three check nodes with color 3 branching out from the secondary degree-4 check node. These three check nodes must be distinct, otherwise a 4-cycle would be created . Therefore, these three check nodes must be connected to the three remaining check nodes with color 3 (this is depicted in Fig. 7(a) by the edge between the check nodes with color 3 whose filling color has been changed to gray to show that they are shared). Because of the symmetry in the branches containing the remaining check nodes with

color 3, connecting the check nodes in any way results in just one non-isomorphic graph which is shown in Fig. 7(a). Then, we remove all the variable nodes connected to two odd-degree check nodes which results in the $(10, 10)$ trapping set of Fig. 7(b) denoted by $\mathcal{T}$. Since there is no $(4, 10)$ trapping set, $\mathcal{EO}_1$ can not be used to search for the codewords which are the successors of $\mathcal{T}$. As it is shown in Fig. 7(a), there is no odd-degree check node with color 3 left after the necessary sharing of the color-3 check nodes. Therefore, no variable node can be connected to three odd-degree check nodes due to coloring constraint as it would require one odd-degree check node in each color. So $\mathcal{EO}_3$ can not be used. Therefore we can only use $\mathcal{EO}_2$ to search for all the $(11, 9)$ trapping sets. The only possible way to continue the search is to use $\mathcal{EO}_3$ three consecutive times. However in order to use $\mathcal{EO}_3$, the check nodes with colors 1, 2 and 3 must connect to a variable node. Therefore, three check nodes of three colors are needed which it is not possible because at this stage we are left with five check nodes with color 1, one check node with color 2 and three check nodes with color 3. ■

We now detail the search procedure for the weight-14 codewords whose induced subgraph have two-odd-degree check nodes which are not connected to common variable node. The starting subgraph in the search for the this category is shown in Fig. 8. The two degree-4 check nodes in codewords in this case are already in the starting search subgraph, therefore all the adjoining trapping sets in the search procedure are elementary trapping sets.

Proposition 6. *In the $\mathcal{C}^n(3, d_c, 8)$ code ensemble, all the codewords with two degree-4 check nodes not connected to a common variable node are successors of a $(6, 8)$ subgraph shown in Fig. 8.*

Proof: Omitted due to page limitations. ■

Let $\mathcal{S}$ be the $(6, 8)$ subgraph of Fig. 8. One can see that $\mathcal{S}$ can be searched by the union of two $(4, 4)$ trapping set using $\mathcal{EO}_3$.

The following search algorithm searches for all the codewords containing a subgraph of type $\mathcal{S}$:

Algorithm 4 Search procedure steps for codewords with two degree-4 check node containing a subgraph of type $\mathcal{S}$.

1) For all the subgraphs of type $\mathcal{S}$ which satisfy Case 1, use $\mathcal{EO}_1$ to search for the union of the subgraphs of type $\mathcal{S}$ and all the elementary $(8, 8)$ trapping sets.
2) For all the subgraphs of type $\mathcal{S}$ which satisfy Case 2, use $\mathcal{EO}_2$ to search for all the $(7, 7)$ subgraphs.

   Remark 4. There is no subgraph of girth 8 to be searched for using $\mathcal{EO}_3$. It is easy to see that by adjoining one variable node to any 3 odd-degree check node of $\mathcal{S}$ in Fig. 8, a 6-cycle is created.
3) For all the $(7, 7)$ subgraphs which satisfy Case 1, use $\mathcal{EO}_1$ to search for the union of the $(7, 7)$ subgraphs and all the elementary $(7, 7)$ trapping sets.

   Remark 5. Expansion operation 2 can be used to search for all the $(8, 6)$ subgraphs from the $(7, 7)$ subgraphs. However, we observed that no codeword can be generated from the $(8, 6)$ subgraphs since there is no coloring possible for the generated codewords.

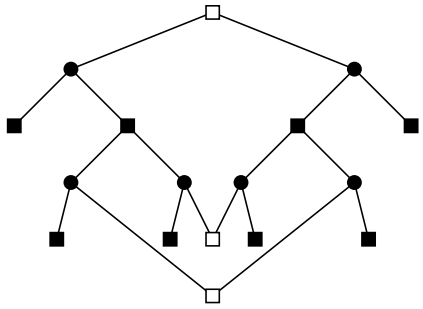

Figure 8. Starting search subgraph for the weight-14 codewords with two odd-degree check nodes which are not connected to common variable node.

In all the previous sections, we have covered all possible cases to obtain all weight-14 codewords in $\mathcal{C}^n(3, d_c, 8)$. Using LDPC code construction technique from [28], we can avoid these structures during the code construction to guarantee a minimum distance of at least 16.

V. Conclusion

In this paper, we have presented a method to search for all the codewords up to a certain weight in $\mathcal{C}^n(d_v = 3, d_c, g = 8)$. This method is based on the decomposition of the codewords into smaller subgraphs. The search method starts by searching for the smallest (parent) subgraphs containing in all the codeword categories. Then, by using three key expansion operations, we expand the parent subgraph to search for larger (child) subgraphs. The expansion continues until reaching all the codewords of certain weight present in the code. We showed that the codewords of weight 14 can be enumerated in a finite number of steps. Avoiding such codewords during the code construction ensures a code with guaranteed minimum distance. An extension of this method to higher column-weight and/or different girth is straightforward (with a price of increase in complexity) provided the trapping set ontology of the given code ensemble is available.

Acknowledgement

This work was funded by NSF under grant CCF-0963726.

References

[1] D. Declercq, B. Vasić, S. Planjery, and E. Li, "Finite alphabet iterative decoders approaching maximum likelihood performance on the binary symmetric channel," in *Proc. Inf. Theory and Applications Workshop*, San Diego, CA USA, Feb. 2012, pp. 23–32.

[2] D. Declercq, L. Danjean, E. Li, S. Planjery, and B. Vasić, "Finite alphabet iterative decoding (FAID) of the (155,64,20) Tanner code," in *Proc. Int. Symp. on Turbo Codes and Iterative Inf.*, Brest, France, Sep. 2010, pp. 11–15.

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

[4] A. Vardy, "The intractability of computing the minimum distance of a code," *IEEE Trans. Inf. Theory*, vol. 43, no. 6, pp. 1757–1766, Nov 1997.

[5] C. Berrou and S. Vaton, "Computing the minimum distances of linear codes by the error impulse method," in *Proc. IEEE Int. Symp. Inf. Theory*, Lausanne, Switzerland, Jun. 30–Jul. 5 2002, p. 5.

[6] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding: Turbo-codes," in *Proc. IEEE Int. Conf. Commun*, vol. 2, Geneva, Switzerland, May 1993, pp. 1064–1070.

[7] D. Declercq and M. Fossorier, "Improved impulse method to evaluate the low weight profile of sparse binary linear codes," in *Proc. IEEE Int. Symp. Inf. Theory*, Toronto, Canada, Jul, 2008, pp. 1963–1967.

[8] J. Stern, "A method for finding codewords of small weight," in *Coding Theory and Applications G. Cohen and J. Wolfmann, Eds. New York: Springer-Verlag*, 1989, pp. 106–113.

[9] M. Hirotomo, M. Mohri, and M. Morii, "A probabilistic computation method for the weight distribution of low-density parity-check codes," in *Proc. IEEE Int. Symp. Inf. Theory*, Adelaide, Australia, Sep. 2005, pp. 2166–2170.

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

[11] C.-C. Wang, S. Kulkarni, and H. Poor, "Finding all small error-prone substructures in LDPC codes," *IEEE Trans. Inf. Theory*, vol. 55, no. 5, pp. 1976–1999, May 2009.

[12] G. B. Kyung and C.-C. Wang, "Exhaustive search for small fully absorbing sets and the corresponding low error-floor decoder," in *IEEE Int. Symp. Inf. Theory*, Jun. 2010, pp. 739–743.

[13] M. Karimi and A. Banihashemi, "Efficient algorithm for finding dominant trapping sets of LDPC codes," *IEEE Trans. Inf. Theory,*, vol. 58, no. 11, pp. 6942–6958, Nov. 2012.

[14] R. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inf. Theory*, vol. 27, no. 5, pp. 533–547, Sep. 1981.

[15] J. Campello, D. Modha, and S. Rajagopalan, "Designing LDPC codes using bit-filling," in *Proc. IEEE Int. Conf. Commun.*, vol. 1, Helsinki, Finland, Jun. 2001, pp. 55–59.

[16] M. O'Sullivan, "Algebraic construction of sparse matrices with large girth," *IEEE Trans. Inf. Theory*, vol. 52, no. 2, pp. 718–727, Feb. 2006.

[17] O. Milenkovic, N. Kashyap, and D. Leyba, "Shortened array codes of large girth," *IEEE Trans. Inf. Theory*, vol. 52, no. 8, pp. 3707–3722, Aug. 2006.

[18] J. Lu, J. Moura, and U. Niesen, "Grouping-and-shifting designs for structured LDPC codes with large girth," in *Proc. IEEE Int. Symp. Inf. Theory*, Chicago, IL USA, Jun. 2004, p. 236.

[19] M. Fossorier, "Quasicyclic low-density parity-check codes from circulant permutation matrices," *IEEE Trans. Inf. Theory*, vol. 50, no. 8, pp. 1788–1793, Aug. 2004.

[20] S. Kim, J.-S. No, H. Chung, and D.-J. Shin, "Quasi-cyclic low-density parity-check codes with girth larger than 12," *IEEE Trans. Inf. Theory*, vol. 53, no. 8, pp. 2885–2891, Aug. 2007.

[21] Y. Wang, J. Yedidia, and S. Draper, "Construction of high-girth QC-LDPC codes," in *Proc. Int. Symp. on Turbo Codes and Related Topics*, Lausanne, Switzerland, Sep. 2008, pp. 180–185.

[22] Y.-K. Lin, C.-L. Chen, Y.-C. Liao, and H.-C. Chang, "Structured LDPC codes with low error floor based on PEG tanner graphs," in *Proc. IEEE Int. Symp. on Circuits and Systems*, Seattle, WA USA, May 2008, pp. 1846–1849.

[23] D. J. MacKay and M. C. Davey, "Evaluation of gallager codes for short block length and high rate applications," in *In Codes, Systems and Graphical Models*. Springer-Verlag, 1999, pp. 113–130.

[24] I. Bocharova, F. Hug, R. Johannesson, B. Kudryashov, and R. Satyukov, "New low-density parity-check codes with large girth based on hypergraphs," in *Proc. IEEE Int. Symp. Inf. Theory*, Austin, TX USA, Jun. 2010, pp. 819–823.

[25] R. Smarandache and P. O. Vontobel, "Quasi-cyclic LDPC codes: Influence of proto- and tanner-graph structure on minimum Hamming distance upper bounds," *IEEE Trans. Inf. Theory*, vol. 58, no. 2, pp. 585–607, Feb. 2012.

[26] D. Divsalar, S. Dolinar, and C. Jones, "Construction of protograph LDPC codes with linear minimum distance," in *Proc. IEEE Int Symp. Inf. Theory*, Seattle, WA USA, Jul. 2006, pp. 664–668.

[27] D. Mitchell, A. Pusane, and D. Costello, "Minimum distance and trapping set analysis of protograph-based LDPC convolutional codes," *IEEE Trans. Inf. Theory,*, vol. 59, no. 1, pp. 254–281, Jan. 2013.

[28] D. V. Nguyen, S. Chilappagari, M. Marcellin, and B. Vasić, "On the construction of structured LDPC codes free of small trapping sets," *IEEE Trans. Inf. Theory*, vol. 58, no. 4, pp. 2280–2302, Apr. 2012.

[29] R. Gallager, "Low-density parity-check codes," *IRE Trans. Inf. Theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962.

[30] T. Richardson, "Error floors of LDPC codes," in *Proc. Allerton Conf. on Commun., Control and Computing*, Monticello, IL USA, Oct. 2003.

[31] S. Chilappagari, D. V. Nguyen, B. Vasić, and M. Marcellin, "On trapping sets and guaranteed error correction capability of LDPC codes and GLDPC codes," *IEEE Trans. Inf. Theory*, vol. 56, no. 4, pp. 1600–1611, Apr. 2010.

[32] B. Vasić, S. K. Chilappagari, D. V. Nguyen, and S. K. Planjery, "Trapping set ontology," in *Proc. Allerton Conf. on Commun., Control, and Computing*, Monticello, IL, USA, Sep. 30–Oct. 2 2009, pp. 1–7.