METHODS OF SEARCHING FOR TRAPPING SETS OF QUASI-CYCLIC LDPC CODES AND THEIR APPLICATIONS IN CODE CONSTRUCTION

Graduate Student: Dung Viet Nguyen
Advisors: Bane Vasic and Michael W. Marcellin
Department of Electrical and Computer Engineering
University of Arizona, Tucson, Arizona, 85721 USA.
Emails: {nguyendv, vasic, marcellin}@ece.arizona.edu.

Abstract

We present a methodology to search for harmful subgraphs in the Tanner graph of a regular column-weight-three low-density parity-check (LDPC) code. These harmful subgraphs, which are more commonly known under the name "trapping sets," cause a degradation in the error correcting performance of LDPC codes in the high signal-to-noise ratio (SNR) region. Our method of searching for trapping sets utilizes the Trapping Set Ontology, a database in which trapping sets are arranged according to their topological relationships. Based on the topological relationships, larger trapping sets are searched for by expanding smaller ones. This reduces the complexity and the run-times of the search algorithms and allows large trapping sets to be searched for efficiently. Besides, if the LDPC code is quasi-cyclic then our search algorithms can be futher simplified. Our method of searching for trapping sets is useful in the performance analysis of LDPC codes and in the construction of good LDPC codes. In this paper, we demonstrate the use of the method in the construction of good LDPC codes.

I. INTRODUCTION

Low-density parity-check (LDPC) codes [1], discovered by Gallager in the 1950s, are error correcting codes with remarkable performance under iterative decoding. Many of these codes have become error correcting codes in standards for various communication systems such as satellite transmission of digital television and 10 Gigabit Ethernet. LDPC codes are also the most promising candidates to be used in many recently developed high speed communication systems such as flash memory, optical communication and free space optics. Much research effort has been invested in LDPC codes in the last decade. Nevertheless, there remain many challenging problems, among which the ones related to the *error floor phenomenon* are most important. This paper addresses one such problem.

The error floor phenomenon can be described as an abrupt degradation in the error correcting performance of LDPC codes in the high SNR region. For example, let us consider Figure 1 which shows the frame error rate (FER) performance of an LDPC code on the binary symmetric channel (BSC), decoded with the Gallager A algorithm. One can observe that as the cross over probabilty α of the channel decreases, the probability of a frame error (or the FER) also decreases. However, once α is less than 0.02, the decreasing rate slows down significantly. This abrupt change in the slope of the FER curve equates to a performance loss. The range of α for which this performance loss can be observed is called *the error floor region.*

It is now well-known that the error floor is caused by certain harmful subgraphs in the Tanner graph representation of the code. These subgraphs are commonly known as trapping sets. The performance of a code in the error floor region depends on the types of trapping sets which are present in the Tanner graph, as well as the number of each type. As a result, enumerating trapping sets is an important

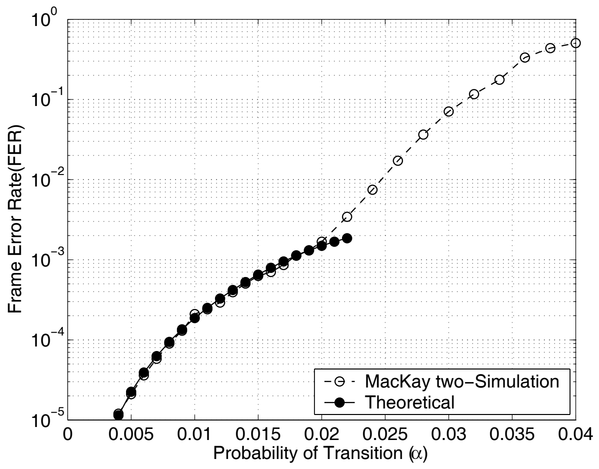

Fig. 1. The error floor of an LDPC code.

problem. Firstly, if the most harmful types of trapping sets that are present in the Tanner graph of a code can be identifed and enumerated, then the performance of the code in the error floor region can be estimated. Secondly, being able to search for trapping sets enables the construction of good LDPC codes. This will be demonstrated later in the paper.

The problem of enumerating trapping sets is NP hard [2], [3]. Previous work on this problem includes exhaustive [4], [5] and non-exhaustive approaches [6], [7]. The main drawback of existing exhaustive approaches is high complexity. Consequently, constraints must be imposed on trapping sets and on the Tanner graph in which the search for trapping sets is carried out. For example, the method in [4] can only search for certain trapping sets in a Tanner graph with less than 500 variable nodes. The complexity is much lower for non-exhaustive approaches. However, these approaches can not guarantee that all trapping sets are enumerated.

The search method proposed in this paper is exhaustive. However, it utilizes the topological relationships among trapping sets to reduce complexity. The topological relationships among trapping sets are given in a database of trapping sets known as the Trapping Set Ontology (TSO) [8]. In our method, trapping sets are searched for in a way similar to how they have evolved in the TSO. A larger trapping set can be found in a Tanner graph by expanding a smaller trapping set. This approach is especially efficient for quasi-cyclic codes, a class of practical codes which we will now discuss.

Practicality requires that an LDPC code is a quasi-cyclic code. An LDPC code is quasi-cylic if its parity check matrix can be represented as an array of circulant permutation matrices. The quasi-cyclicity property of a code allows simple hardware implementation. Conveniently, this property also reduces the complexity of searching the Tanner graph of a code for trapping sets. In this paper, we utilize the quasi-cyclicity property in our trapping set searching techniques. To demonstrate the efficiency of these techniques, we apply them in the construction of a good regular column-weight-three quasi-cyclic LDPC code.

The rest of the paper is organized as follows. Section II gives preliminaries. Section III presents the search method. In Section IV, we discuss the use of the search method in code construction. Finally, Section V presents numerical results.

II. Preliminaries

In this section, we give the necessary background on LDPC codes. Then, we define and discuss the notion of trapping set. Finally, we describe the Trapping Set Ontology.

A. LDPC Codes

LDPC codes belong to a class of codes that can be represented by sparse graphs. Let $\mathcal{C}$ denote an (n,k) LDPC code over the binary field GF(2). $\mathcal{C}$ is defined by the null space of H, an $m \times n$ *parity check matrix* of $\mathcal{C}$. H is the bi-adjacency matrix of G, a Tanner graph representation of $\mathcal{C}$. G is a bipartite graph with two sets of nodes: n variable (bit) nodes $V = \{1, 2, \ldots, n\}$ and m check nodes $C = \{1, 2, \ldots, m\}$. A vector $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ is a codeword if and only if $\mathbf{y}H^{\mathrm{T}} = 0$, where H^{T} is the transpose of H. The support of $\mathbf{y}$, denoted as $\mathrm{supp}(\mathbf{y})$, is defined as the set of all variable nodes (bits) $v \in V$ such that $y_v \neq 0$. A d_v-left-regular LDPC code has a Tanner graph G in which all variable nodes have degree d_v. Similarly, a d_c-right-regular LDPC code has a Tanner graph G in which all check nodes have degree d_c. A (d_v, d_c) regular LDPC code is d_v-left-regular and d_c-right-regular. Such a code has rate $R \geq 1 - d_v/d_c$ [1]. The degree of a variable node (check node, resp.) is also referred to as the left degree (right degree, resp.) or the column weight (row weight, resp.). The length of the shortest cycle in the Tanner graph G is called the girth g of G. $\mathcal{C}$ is defined by the null space of H, a parity check matrix of C. H is the incidence matrix of G, a Tanner graph representation of $\mathcal{C}$.

B. Trapping Sets and Trapping Set Ontology

A trapping set for an iterative decoding algorithm is defined as a non-empty set of variable nodes in a Tanner graph G that are not eventually corrected by the decoder [9]. A set of variable nodes $\mathbf{T}$ is called an (a, b) trapping set if it contains a variable nodes and the subgraph induced by these variable nodes has b odd degree check nodes.

For the Gallager A/B algorithm on the BSC there is no explicit combinatorial characterization of trapping sets. However we can characterize fixed sets, as defined below, which form a subclass of trapping sets. These have been studied in [10]–[13] and shown to be the cause of the error floor phenomenon in this setting.

Consider an iterative decoder on the BSC. Assume the transmission of an all-zero codeword[1]. With this assumption, a variable node is correct if it is 0 and corrupt if it is 1. Let $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ be the input to the decoder. Let $\mathbf{F}(\mathbf{y})$ denote the set of variable nodes that are not eventually correct.

Definition 1 ([11]): For transmission over the BSC, $\mathbf{y}$ is a fixed point of the Gallager A/B algorithm if the messages passed from variable nodes to check nodes along the edges are the same in every iteration. If $\mathbf{F}(\mathbf{y}) \neq \emptyset$ and $\mathbf{y}$ is a fixed point, then $\mathbf{F}(\mathbf{y}) = \mathrm{supp}(\mathbf{y})$ is a fixed set. A fixed set (trapping set) is *an elementary fixed set (trapping set)* if all check nodes in its induced subgraph have degree one or two[2]. Otherwise, it is a non-elementary fixed set (trapping set).

The following theorem gives necessary and sufficient conditions for a set of variable nodes to be a fixed set. Note that condition (b) is trivially satisfied for elementary sets.

Theorem 1 ([11]): Let $\mathcal{C}$ be a code with Tanner graph G in the ensemble of $(3, d_c)$ regular LDPC codes. Let $\mathbf{T}$ be a set of variable nodes with induced subgraph $\mathbf{I}$. Let the check nodes in $\mathbf{I}$ be partitioned into two disjoint subsets; $\mathbf{O}$ consisting of check nodes with odd degree and $\mathbf{E}$ consisting of check nodes

[1]The all-zero-codeword assumption can be applied if the channel is output symmetric and the decoding algorithms satisfied certain symmetry conditions (see Definition 1 and Lemma 1 in [14]). The Gallager A/B algorithm, the bit flipping algorithms and the SPA all satisfy these symmetry conditions.

[2]This classification was given in [15].

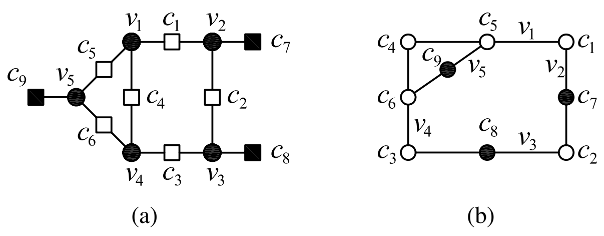

Fig. 2. Graphical representation of the $(5,3)\{1\}$ trapping set: (a) Tanner graph representation, (b) Line-point representation.

with even degree. Then $\mathbf{T}$ is a fixed set iff: a) every variable node in $\mathbf{I}$ is connected to at least two check nodes in $\mathbf{E}$ and b) no two check nodes of $\mathbf{O}$ are connected to a common variable node outside $\mathbf{I}$.

Before presenting the method of searching for trapping sets, we need a complete list of trapping sets up to a certain size and also the topological relations between them. In the following subsection we give some details of how this database, the Trapping Set Ontology (TSO), is represented. From now on we will only be studying fixed sets and for the sake of simplicity will refer to them by the more general term trapping sets.

1) Graphical representation: The induced subgraph of a trapping set (or any set of variable nodes) is a bipartite graph. In the Tanner graph (bipartite graph) representation of a trapping set, we use ● to represent variable nodes, ■ to represent odd degree check nodes and □ to represent even degree check nodes. There exists an alternate graphical representation of trapping sets which allows their topological relations to be established more conveniently. In this lines and points (henceforth line-point) representation of trapping sets, variable nodes correspond to lines and check nodes correspond to points. A point is shaded black if it has an odd number of lines passing through it, otherwise it is shaded white. An (a,b) trapping set is thus an incidence structure with a lines and b black shaded points. To differentiate among (a,b) trapping sets that have non-isomorphic induced subgraphs when necessary, we index (a,b) trapping sets in an arbitrary order and assign the notation $(a,b)\{i\}$ to the (a,b) trapping set with index i.

Depending on the context, a trapping set can be understood as a set of variable nodes in a given code with a specified induced subgraph or it can be understood as a specific subgraph independent of a code. To differentiate between these two cases, we use the letter $\mathbf{T}$ to denote a set of variable nodes in a code and use the letter $\mathcal{T}$ to denote a type of trapping set which corresponds to a specific subgraph. If the induced subgraph of a set of variable nodes $\mathbf{T}$ in the Tanner graph of a code $\mathcal{C}$ is isomorphic to the subgraph of $\mathcal{T}$ then we say that $\mathbf{T}$ is a $\mathcal{T}$ trapping set or that $\mathbf{T}$ is a trapping set of type $\mathcal{T}$. $\mathcal{C}$ is said to contain $\mathcal{T}$ trapping set(s).

Example 1: The $(5,3)\{1\}$ trapping set $\mathcal{T}_1$ is a union of a six cycle and an eight cycle, sharing two variable nodes. The Tanner graph representation of $\mathcal{T}_1$ is shown in Fig. 2(a) and the line-point representation is shown in Fig. 2(b).

2) Topological relationships: The following definition gives the topological relations among trapping sets.

Definition 2: A trapping set $\mathcal{T}_2$ is a successor of a trapping set $\mathcal{T}_1$ if there exists a proper subset of variable nodes of $\mathcal{T}_2$ that induce a subgraph isomorphic to the induced subgraph of $\mathcal{T}_1$. If $\mathcal{T}_2$ is a successor of $\mathcal{T}_1$ then $\mathcal{T}_1$ is a parent of $\mathcal{T}_2$. Furthermore, $\mathcal{T}_2$ is a direct successor of $\mathcal{T}_1$ if it does not have a parent $\mathcal{T}_3$ which is a successor of $\mathcal{T}_1$.

The topological relation between $\mathcal{T}_1$ and $\mathcal{T}_2$ is solely dictated by the topological properties of their subgraphs. In the Tanner graph of a code $\mathcal{C}$, the presence of a trapping set $\mathbf{T_1}$ does not indicate the presence of a trapping set $\mathbf{T_2}$. If $\mathbf{T_1}$ is indeed a subset of a trapping set $\mathbf{T_2}$ in the Tanner graph of $\mathcal{C}$ then we say that $\mathbf{T_1}$ *generates* $\mathbf{T_2}$, otherwise we say that $\mathbf{T_1}$ does not generate $\mathbf{T_2}$.

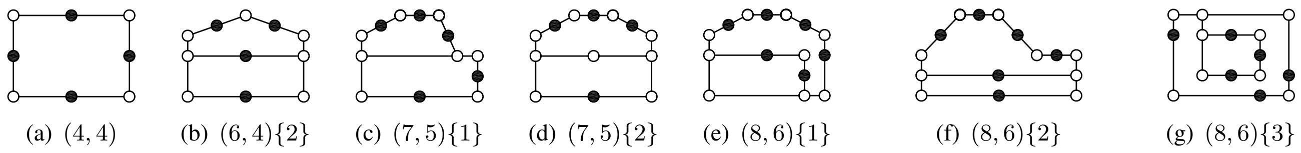

Fig. 3. The $(4,4)$ trapping set and its direct successors of size less than 8 in girth 8 LDPC codes (excluding the $(5,3)\{1\}$ and $(6,4)\{1\}$ trapping sets shown in Fig. 2(b) and 5(a), respectively).

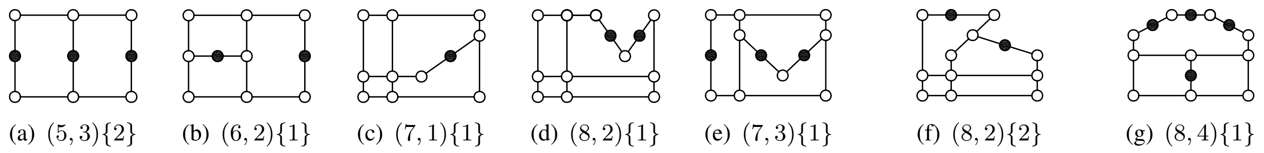

Fig. 4. The $(5,3)\{2\}$ trapping set and its successors of size less than 8 in girth 8 LDPC codes.

Example 2: The trapping sets of size less than 8 in regular column-weight-three LDPC codes of girth $g = 8$ are listed in Fig. 3, 4 and 5. For a more complete list of trapping sets in the TSO, readers are referred to [8], [16]. Note that if $\mathbf{y}$ is a codeword of $\mathcal{C}$ with $a = |\text{supp}(\mathbf{y})|$ then $\mathbf{T} = \text{supp}(\mathbf{y})$ is an $(a, 0)$ trapping set. Thus we can determine the minimum distance of a code by looking at which $(a, 0)$ trapping sets it contains.

III. Searching for Trapping Sets

In this section, we describe our techniques of searching for elementary trapping sets from the TSO in the Tanner graph of a regular column-weight-three LDPC code and describe how the quasi-cyclicity property can be used to reduce complexity. An efficient search of the Tanner graph for trapping sets relies on the topological relations among trapping sets defined in the TSO and/or carefully analyzing their induced subgraphs. Trapping sets are searched for in a way similar to how they have evolved in the TSO. A bigger trapping set can be found in a Tanner graph by expanding a smaller trapping set. More precisely, given a trapping set $\mathbf{T_1}$ of type $\mathcal{T}_1$ in the Tanner graph of a code $\mathcal{C}$, our techniques search for a set of variable nodes such that the union of this set with $\mathbf{T_1}$ form a trapping set $\mathbf{T_2}$ of type $\mathcal{T}_2$, where $\mathcal{T}_2$ is a successor of $\mathcal{T}_1$. Our techniques are sufficient to efficiently search for a large number of trapping sets in the TSO. They can be easily expanded to search for other trapping sets as well.

A. Subroutines

We assume that the following simple subroutines are used in our search algorithms.

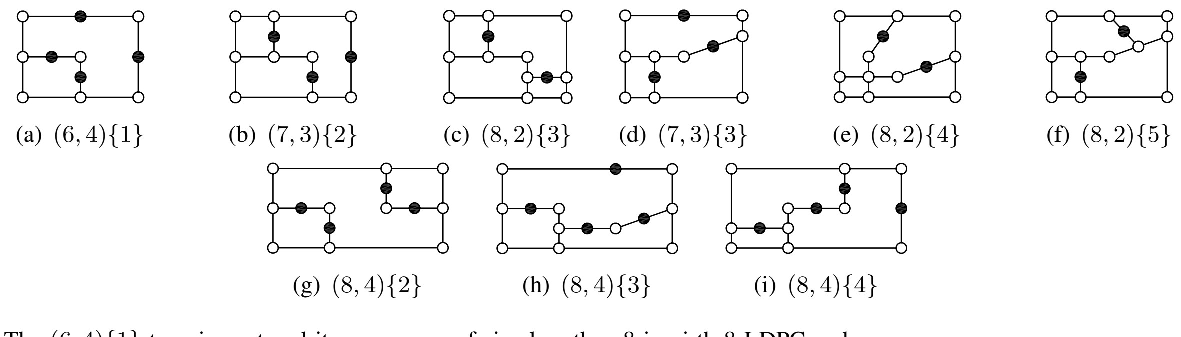

Fig. 5. The $(6,4)\{1\}$ trapping set and its successors of size less than 8 in girth 8 LDPC codes.

- $\mathbf{Y}$ = **RowIntersectIndex**$(\mathbf{X_1}, \mathbf{X_2}, \vartheta)$.
  Let $\mathbf{X_1}$ and $\mathbf{X_2}$ be matrices. $\mathbf{Y}$ is a matrix of two columns. If (i_1, i_2) is a row of $\mathbf{Y}$ then the $i_1{}^{\text{th}}$ row of $\mathbf{X_1}$ and the $i_2{}^{\text{th}}$ row of $\mathbf{X_2}$ share ϑ common entries.
- $\mathbf{Y}$ = **OddDegreeChecks**$(H, \mathbf{X})$.
  Let H be the parity check matrix corresponding to a Tanner graph G of an LDPC code. Let $\mathbf{X}$ be a matrix with each row of $\mathbf{X}$ giving a set of variable nodes. Assume that all the subgraphs induced by variable nodes in rows of $\mathbf{X}$ have the same number of odd degree check nodes. $\mathbf{Y}$ is a matrix with the same number of rows as $\mathbf{X}$. Elements of the i^{th} row of $\mathbf{Y}$ are odd degree check nodes in the subgraph induced by the variable nodes in the i^{th} row of $\mathbf{X}$.
- $\mathbf{Y}$ = **TotalChecksOfDegreeK**$(H, \mathbf{X}, \vartheta)$.
  Let H be the parity check matrix corresponding to a Tanner graph G. Let $\mathbf{X}$ be a matrix whose elements are variable nodes in G. $\mathbf{Y}$ is a one-column matrix with the same number of rows as $\mathbf{X}$. The element in the i^{th} row of $\mathbf{Y}$ is the number of check nodes with degree ϑ in the subgraph induced by the variable nodes in the i^{th} row of $\mathbf{X}$.
- $\mathbf{Y}$ = **IsTrappingSet**$(H, \mathbf{X})$.
  Let H be the parity check matrix corresponding to a Tanner graph G. Let $\mathbf{X}$ be a matrix whose elements are variable nodes in G. $\mathbf{Y}$ is a one-column matrix with the same number of rows as $\mathbf{X}$. The element in the i^{th} row of $\mathbf{Y}$ is 1 if the variable nodes in the i^{th} row of $\mathbf{X}$ form a trapping set and is 0 otherwise.

The above subroutines can be implemented using simple sparse matrix operations and hence are of low complexity.

B. Searching for Cycles of Length ϑ

Since every trapping set contains at least one cycle, the search for trapping sets always starts with finding cycles in the Tanner graph. All cycles of length ϑ that contain variable node v can be found by performing the following steps.

1) Construct the tree of depth $\vartheta/2 - 1$, taking v as the root using the breadth-first search algorithm [17]. Let $N_1, N_2, \ldots, N_{d_v}$ be sets of leaf nodes of depth $\vartheta/2 - 1$ such that all the nodes in N_i are descendants of the i^{th} neighbor of v. It can be shown that $|N_i| \le (d_v - 1)^{t_1}(d_c - 1)^{t_2}$ where
$$\begin{aligned} t_1 &= \frac{\vartheta}{4} - \frac{3}{2}, t_2 = t_1 + 1 \text{ if } \vartheta/2 \text{ is odd} \\ t_1 &= t_2 = \frac{\vartheta}{4} - 1 \text{ if } \vartheta/2 \text{ is even.} \end{aligned}$$
2) For every pair of nodes $o_i \in N_i$, $o_j \in N_j$ and $i \neq j$, determine if they share a common neighbor. If so then a cycle of length ϑ has been found. If o_i and o_j are check nodes then the cycle is induced by the variable nodes that are ancestors of o_i and o_j and their common neighbor. If o_i and o_j are variable nodes then the cycle is induced by o_i and o_j as well as the variable nodes that are ancestors of o_i and o_j. The maximum number of possible pairs o_i, o_j is $\sum_{i \neq j} d_v(d_v - 1)|N_i||N_j|$.

The two steps described above are executed for every variable node. To further simplify the search, after all the cycles containing v are found, v can be marked so that it is no longer included in Step 1 of the search at other variable nodes. The complexity of searching for cycles is polynomial in the degree of the variable nodes and check nodes but increases only linearly in the code length. Note that our search algorithm not only counts the number of cycles but also records the variable nodes that each cycle contains. For this reason, existing efficient algorithms to count number of cycles in a bipartite graph (for example those proposed in [18], [19]) can not be applied directly.

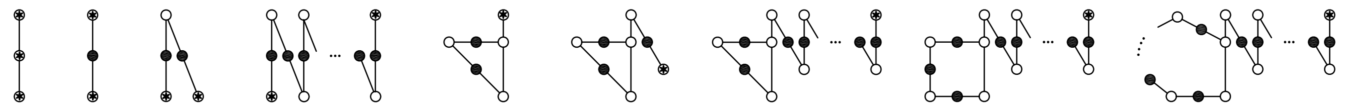

Fig. 6. Possible incidence structures formed by u new lines for elementary trapping sets.

C. *Searching for* $(a+1, b-1)$ *Trapping Sets Generated by* (a, b) *Trapping Sets*

Let $\mathcal{T}_1$ be an (a, b) trapping set, $\mathcal{T}_2$ be an $(a+1, b-1)$ trapping set and let $\mathcal{T}_1$ be a parent of $\mathcal{T}_2$. Further, let $\mathbf{T_1}$ be a trapping set of type $\mathcal{T}_1$ in the Tanner graph of a code $\mathcal{C}$ and assume that $\mathbf{T_1}$ generates a trapping set $\mathbf{T_2}$ of type $\mathcal{T}_2$. $\mathcal{T}_2$ is obtained by adjoining one variable node to $\mathcal{T}_1$. The line-point representation of $\mathcal{T}_2$ is obtained by merging two black shaded nodes in the line-point representation of $\mathcal{T}_1$ with two $\circledast$ nodes in Fig. 6(b). Therefore, to search for $\mathbf{T_2}$, it is sufficient to search for a variable node that is connected to two odd degree check nodes in the subgraph induced by variable nodes in $\mathbf{T_1}$.

Let $\mathbf{X}$ be a matrix whose each row contains variable nodes of a $\mathcal{T}_1$ trapping set in the Tanner graph G. H is the parity check matrix which defines $\mathcal{C}$. All $\mathcal{T}_2$ trapping sets can be found by performing the following steps.

1) Find all odd degree check nodes of all $\mathcal{T}_1$ trapping sets:
   $\mathbf{Y_1}$ = **OddDegreeChecks**$(H, \mathbf{X})$.
2) Form $\mathbf{X_1}$, a one-column matrix with n rows where the element in the i^{th} row is variable node i.
3) Form a matrix $\mathbf{Y_2}$ whose i^{th} row gives all check nodes neighboring to the variable node i:
   $\mathbf{Y_2}$ = **OddDegreeChecks**$(H, \mathbf{X_1})$.
4) Find all pairs (i, j) such that the i^{th} row of $\mathbf{Y_1}$ and the j^{th} row of $\mathbf{Y_2}$ share 2 common entries.
   $\mathbf{Y_3}$ = **RowIntersectIndex**$(\mathbf{Y_1}, \mathbf{Y_2}, 2)$.
5) If (i, j) is the l^{th} row of $\mathbf{Y_3}$, adjoin variable node j to the i^{th} row of $\mathbf{X}$ to form the l^{th} row of $\mathbf{Y_4}$.
6) Determine the number of degree one check nodes in the subgraph induced by variable nodes in each row of $\mathbf{Y_4}$ and eliminate the rows of $\mathbf{Y_4}$ that do not have $b-1$ degree one check nodes. The matrix $\mathbf{Y}$ obtained has each row containing variable nodes that induce a $\mathcal{T}_2$ trapping set in the Tanner graph of the code.
   $\mathbf{Y}$ = $\mathbf{Y_4}$(**TotalChecksOfDegreeK**$(H, \mathbf{Y_4}, 1)$== $b-1$).

D. *Searching for* $(a+2, b)$ *Trapping Sets Generated by* (a, b) *Trapping Sets*

Let $\mathcal{T}_1$ be an (a, b) trapping set, $\mathcal{T}_2$ be an $(a+2, b)$ trapping set and let $\mathcal{T}_1$ be a parent of $\mathcal{T}_2$. Further, let $\mathbf{T_1}$ be a trapping set of type $\mathcal{T}_1$ in the Tanner graph of a code $\mathcal{C}$ and assume that $\mathbf{T_1}$ generates a trapping set $\mathbf{T_2}$ of type $\mathcal{T}_2$. Consider two variable nodes that share a check node. $\mathcal{T}_2$ is obtained by adjoining these two variable nodes to $\mathcal{T}_1$. The line-point representation of $\mathcal{T}_2$ is obtained by merging two black shaded nodes in the line-point representation of $\mathcal{T}_1$ with two $\circledast$ nodes in Fig. 6(c). Therefore, to search for $\mathbf{T_2}$, it is sufficient to search for a pair of variable nodes that share a common neighboring check node and each node is connected to one odd degree check node in the subgraph induced by variable nodes in $\mathbf{T_1}$.

The search for $(a+2, b)$ trapping sets is very similar to the search for $(a+1, b-1)$ trapping sets described in the previous subsection. In particular, the following modifications should be made:

- In Step 2, $\mathbf{X_1}$ is a two column matrix, each row contains a pair of variable nodes that share a common neighboring check node.

- In Step 3, the i^{th} row of $\mathbf{Y_2}$ gives all degree one check nodes in the subgraph induced by variable nodes in the i^{th} row of $\mathbf{X_1}$.
- In Step 5, variable nodes in the j^{th} row of $\mathbf{X_1}$ are adjoined to the i^{th} row of $\mathbf{X}$.
- In Step 6, $b-1$ is replaced by b.

Since Step 4 does not take into account the case in which two check nodes of a new variable node are merged with two odd degree check nodes of $\mathbf{T_1}$, the subroutine **IsTrappingSet** is used afterward to eliminate rows of $\mathbf{Y}$ that do not contain variable nodes that form a trapping set.

E. Using the Quasi-Cyclicity Property to Reduce the Complexity of the Search Algorithms

In this subsection, we discuss how to use the property of a quasi-cyclic LDPC code to reduce the complexity of the search algorithms. Let $\mathcal{C}$ be a quasi-cyclic LDPC code with a Tanner graph representation G. Then the bi-adjacency matrix H of G can be represented as an array of circulant permutation matrices. It can be shown that H takes the following form:

$$H = \begin{bmatrix} I & I & I & \cdots & I \\ I & P_{22} & P_{23} & \cdots & P_{2d_c} \\ I & P_{32} & P_{33} & \cdots & P_{3d_c} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ I & P_{d_v 2} & P_{d_v 2} & \cdots & P_{d_v d_c} \end{bmatrix}, \tag{1}$$

where P_{ij} is a circulant permutation matrix of size $p \times p$ for $2 \leq i \leq d_v, 2 \leq j \leq d_c$.

Let π be a function from $V \times \{0, 1, \ldots, p-1\}$ to V defined as follows:

$$\pi(v, s) = q(v) \times p + (r(v) + s) \text{mod } p \tag{2}$$

where $q(v)$ and $r(v)$ are the quotient and remainder after the devision of v by p, respectively.

From properties of the parity check matrix, one can show the following:

Quasi-cyclicity property Let $\mathbf{S} = \{v_1, v_2, \ldots, v_{|\mathbf{S}|}\} \subset V$ be a subset of variable nodes, then the subgraph induced by all the variable nodes in $\mathbf{S}$ is isormorphic to the subgraph induced by all the variable nodes in $\mathbf{S_s} = \{\pi(v_1, s), \pi(v_2, s), \ldots, \pi(v_{|\mathbf{S}|}, s)\}$, $\forall 0 \leq s \leq p-1$.

Now we can use this property to reduce the complexity of the search algorithms. Let $\mathbf{T}$ be a trapping set of type $\mathcal{T}$ which is present in the Tanner graph of a code $\mathcal{C}$ then one can obtain $p-1$ other type $\mathcal{T}$ trapping sets, namely $\mathbf{T_1}, \mathbf{T_2}, \ldots, \mathbf{T_{p-1}}$, by shifting variable nodes of $\mathbf{T}$ using the map π. Assume that we want to find all trapping sets of type $\mathcal{T}'$ by expanding trapping sets $\mathbf{T}, \mathbf{T_1}, \mathbf{T_2}, \ldots, \mathbf{T_{p-1}}$. Instead of expanding every single trapping set, one can choose to expand only one trapping set in the set $\{\mathbf{T}, \mathbf{T_1}, \mathbf{T_2}, \ldots, \mathbf{T_{p-1}}\}$ to obtain $\mathcal{T}'$ trapping sets. The map π can then be used on the variable nodes of the obtained type $\mathcal{T}'$ trapping sets to obtain the other type $\mathcal{T}'$ trapping sets. This significantly reduces the complexity of the search algorithms since the complexity of expanding a trapping set is much greater than the complexity of shifting variable nodes.

F. Remarks

The above search procedures may not differentiate among different (a, b) trapping sets. For example, all $(5,3)\{1\}$ and $(5,3)\{2\}$ trapping sets are found if they are searched for as trapping sets generated by the $(4,4)$ trapping sets. Similarly, all $(7,3)\{2\}$ and $(7,3)\{3\}$ are found as trapping sets generated by the $(6,4)\{1\}$ trapping sets. If searching for a specific type of trapping set is required, then it is necessary to further analyze the induced subgraph. For example, notice that the $(5,3)\{1\}$ trapping set is a union of a 6-cycle and an 8-cycle, sharing two variable nodes while the $(5,3)\{1\}$ trapping set is a union of two 8-cycles, sharing three variable nodes. Therefore, to search for all $(5,3)\{2\}$ trapping sets from a list of $(4,4)$ trapping sets, one would find all pairs of $(4,4)$ trapping sets that share three variable nodes, using the **RowIntersectIndex** subroutine. The union of each pair of $(4,4)$ trapping sets is then a set of five variable nodes. Each set of variable nodes forms a $(5,3)\{2\}$ trapping set if its induced subgraph contains three degree one check nodes. Similarly, all $(7,3)\{2\}$ can be found by noticing that they are unions of two $(6,4)\{1\}$ trapping sets, sharing five variable nodes.

IV. Applications in Code Construction

The method of searching for trapping sets can be utilized to construct quasi-cyclic codes which are free of harmful trapping sets by progressively building the Tanner graphs. The algorithm is based on a check and select-or-disregard procedure. Let τ specify which graphical structures should not be present in the Tanner graph G. τ may specify the girth of G and may also specify the minimum distance of the code. The Tanner graph of a code is built progressively in ρ stages, where ρ is the row weight of the parity check matrix. Usually, ρ is not pre-specified, and codes are constructed to have rate as high as possible. At each stage, a set of q variable nodes are introduced, initially not connected to check nodes of the Tanner graph. Blocks of edges are then added to connect the new variable nodes and the check nodes. Each block of edges corresponds to a circulant permutation matrix. A circulant matrix may be chosen randomly, or it may be chosen in a predetermined order. After a block of edges is added, the Tanner graph is checked for condition τ. If the condition τ is violated, then that block of edges is removed and replaced by a different block. The algorithm proceeds until no block of edges can be added without violating condition τ.

V. Numerical Results

To illustrate the search algorithms, we list the number of cycles and trapping sets in the popular Tanner code, as well as the run-times of the algorithms on a 2.4 GHz computer in Table I.

TABLE I
Number of Cycles and Trapping Sets of the Tanner Code and Run-time of the Searching Algorithms on a 2.4 GHz Computer

Trapping Sets	Total	Run-time without QC Property(Seconds)	Run-time with QC Property(Seconds)
6-cycles	0	$< 10^{-9}$	$< 10^{-9}$
8-cycles	465	0.003	0.001
10-cycles	3720	0.027	0.003
$(5,3)\{2\}$	155	0.004	0.001
$(6,4)\{2\}$	930	0.023	0.001
$(7,3)\{1\}$	930	0.008	0.002
$(8,2)\{1\}$ and $(8,2)\{2\}$	465	0.008	0.001

As mentioned before, one can construct good LDPC codes using the proposed search algorithms. As an example, Figure 7 shows the frame error rate (FER) performance vs cross-over probability (α) of a quasi-cyclic code constructed so that its Tanner graph does not contain $(5,3)$ and $(8,2)$ trapping sets. It is a $(3856,3135)$ code with column weight 3, row weight 16, minimum distance $d_{\min} = 12$ and rate $R = 0.81$. The figure shows the FER performance under the sum-product algorithm with 100 iterations on the BSC. For comparison, Fig. 7 also shows the FER performance of a $(3856,3133)$ progressive edge growth (PEG) constructed code [20]. This code has girth $g = 8$. It can be seen that the quasi-cyclic code has a better performance than the PEG code.

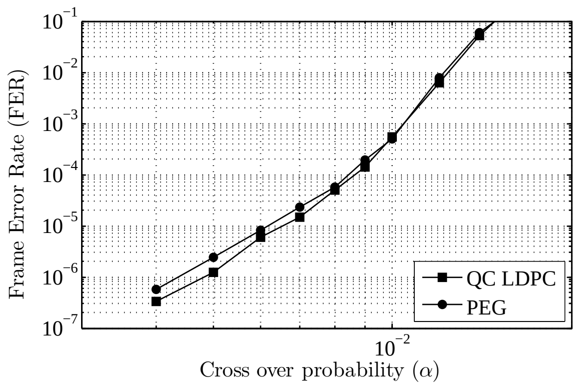

Fig. 7. FER results on the quasi-cyclic code and the PEG code.

References

[1] R. G. Gallager, *Low Density Parity Check Codes*. Cambridge, MA: M.I.T. Press, 1963.

[2] K. Krishnan and P. Shankar, "Computing the stopping distance of a Tanner graph is NP-hard," *IEEE Tran. Inf. Theory*, vol. 53, no. 6, pp. 2278–2280, Jun. 2007.

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

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

[5] 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.

[6] M. Hirotomo, Y. Konishi, and M. Morii, "Approximate examination of trapping sets of LDPC codes using the probabilistic algorithm," in *Int. Symp. Inf. Theory and Its Appl.*, Dec. 2008, pp. 1–6.

[7] S. Abu-Surra, D. DeClercq, D. Divsalar, and W. Ryan, "Trapping set enumerators for specific LDPC codes," in *Inf. Theory and Appl. Workshop*, Jan. 2010, pp. 1–5.

[8] B. Vasic, S. Chilappagari, D. Nguyen, and S. Planjery, "Trapping set ontology," in *Proc. 47th Annual Allerton Conf. on Commun., Control and Computing*, Sept. 2009, pp. 1–7.

[9] T. J. Richardson, "Error floors of LDPC codes," in *Proc. 41st Annual Allerton Conf. on Commun., Control and Computing*, Sept. 2003, pp. 1426–1435.

[10] S. K. Chilappagari, S. Sankaranarayanan, and B. Vasic, "Error floors of LDPC codes on the binary symmetric channel," in *Proc. Int. Conf. on Commun.*, vol. 3, 2006, pp. 1089–1094.

[11] S. Chilappagari and B. Vasic, "Error-correction capability of column-weight-three LDPC codes," *IEEE Trans. Inf. Theory*, vol. 55, no. 5, pp. 2055–2061, May 2009.

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

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

[14] T. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 599–618, Feb. 2001.

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

[16] "Error floors of LDPC codes - Multi-bit bit flipping algorithm." [Online]. Available: http://www2.engr.arizona.edu/~vasiclab/Projects/CodingTheory/ErrorFloorHome.html

[17] D. E. Knuth, *The art of computer programming, volume 2 (3rd ed.): seminumerical algorithms*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997.

[18] T. Halford and K. Chugg, "An algorithm for counting short cycles in bipartite graphs," *IEEE Tran. Inf. Theory*, vol. 52, no. 1, pp. 287–292, Jan. 2006.

[19] M. Karimi and A. H. Banihashemi, "A message-passing algorithm for counting short cycles in a graph," *CoRR*, vol. abs/1004.3966, 2010.

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