A PEG-like LDPC code design avoiding short trapping sets

Madiagne Diouf, David Declercq, Samuel Ouya, Bane Vasic

A PEG-like LDPC code design avoiding short Trapping Sets

Madiagne DIOUF, David DECLERCQ
ETIS, ENSEA
Univ. Cergy-Pontoise
CNRS UMR-8051
95014 Cergy-Pontoise Cedex, France
{madiagne.diouf, declercq}@ensea.fr

Samuel OUYA
LIRT, ESP
Univ. Cheikh Anta DIOP
Dakar, Senegal
samuel.ouya@gmail.com

Bane Vasic
Department of Elec.
and Comp. Eng.
University of Arizona
Tucson, AZ, 85721, USA
vasic@ece.arizona.edu

***Abstract*— In this paper, we propose a predictive method to construct regular column-weight-three LDPC codes with girth $g = 8$ so that their Tanner graphs contain a minimum number of small trapping sets. Our construction is based on improvements of the Progressive Edge-Growth (PEG) algorithm. We first show how to detect the smallest trapping sets $(5, 3)$ and $(6, 4)$ in the computation tree spread from variable nodes during the edge assignment. A precise and rigorous characterization of trapping sets $(5, 3)$ and $(6, 4)$ are given, and we then derive a modification of the Randomized Progressive Edge-Growth (RandPEG) algorithm [1] to take into account a new cost function that allows to build regular column-weight $d_v = 3$, girth 8 LDPC codes free of (5,3) and with a minimization of (6,4). We present the construction and the performance results in the context of quasi-cyclic LDPC (QC-LDPC) codes.**

***Index Terms*—Trapping sets, Error floor, Progressive Edge-Growth (PEG), Low Density Parity Check (LDPC) codes, Tanner graphs.**

I. Introduction

The design of binary LDPC codes with very low error floors is still a significant problem [2]–[4], not fully solved in the literature, and of practical importance for some applications, e.g for point-to-point satellite communication, magnetic or optical recording [5]. For such noisy channels, the error floor of LDPC decoders is due to the presence of certain topologies in the Tanner graph [6], [7] such as cycles and trapping sets (TS). In this paper, we provide a solution to build LDPC codes using a modified version of the Progressive Edge-Growth (PEG) algorithm, with the objective of avoiding the most harmful/problematic structures in the Tanner graph of the LDPC code.

In original PEG algorithm [8] is a greedy algorithm, which was proposed to maximize the local girth of the LDPC Tanner graph. Since then, improvements of the PEG algorithms have been proposed to avoid small stopping sets [9], or to minimize the number of cycles [1] with a non-greedy generalization.

So far, no attempt has been made to detect and avoid trapping sets using a PEG-like algorithm. Since LDPC decoders experience error floor problems mainly because of small trapping sets, we will focus in this paper on the smallest trapping sets that can be considered for a given girth, i.e. the trapping sets $(5, 3)$ and $(6, 4)$ for regular column-weight-three LDPC codes with girth $g = 8$ [10]. The first step of our work is to provide a characterization on how trapping sets can be detected in the computation tree that is used in the PEG algorithm. With this characterization, it is possible to predict which candidate edges will effectively create trapping sets. A cost function is then included in a Randomized-PEG algorithm [1] such that $(5, 3)$ TS are completely removed and the number of $(6, 4)$ TS is minimized.

Using this approach, we can constructively build LDPC codes with a given girth and free of small trapping sets, which generalizes the ideas presented in [1], [9]. We moreover focus on the design of quasi-cyclic LDPC (QC-LDPC) codes, which are the most largely used LDPC codes in practical applications.

The remainder of the paper is organized as follows. In Section II we introduce the basic concepts and notations. Then, the characterization of (5,3) TS and (6,4) TS is presented in Section III, and the constraints to detect those structures in the computation tree are derived. Finally, in Section IV we describe our design algorithm using Randomized PEG and we demonstrate the efficiency of our approach for small LDPC codes, which we compare to existing designs through Monte Carlo simulations. Section V concludes the paper.

II. Preliminaries and Notation

Let G denote the Tanner graph of an (N, K) binary LDPC code $\mathcal{C}$ of rate $R = K/N$, which consists of the set of N variable nodes V and the set of M check nodes C. We will use roman alphabet to denote variable nodes and Greek letters for check nodes. Two nodes are neighbors if there is an edge between them. The degree of a node in G is the number of its neighbors in G. A code $\mathcal{C}$ represented by the graph G is said to be have a regular column-weight d_v if all variable nodes in V have the same degree d_v. Equivalently, such code is said to be d_v-variable-node regular or just d_v-variable regular. The set of neighbors of a node u is denoted as $\mathcal{N}_u$ and $\mathcal{N}_U$ denotes the set of neighbors of all $u \in U$. A path in G is a finite sequence of distinct vertices (variables or checks) $u_0, \ldots, u_l$ such that u_{i-1} and u_i are neighbors for $1 \leq i \leq l$. Two paths are distinct if they differ in at least one node. A l-cycle or cycle of length l in G is a path $u_0, \ldots, u_l$ in G with $u_0 = u_l$ and if u_i is a variable node then u_{i+1} is a check node inversely. Clearly in

a bipartite graph G, l must be even. The girth g of G is the length of shortest cycle in G. $\mathcal{C}_{d_v,g}$ denotes an ensemble of d_v-variable regular codes with girth g.

We begin by providing the definition of a trapping set as originally defined by Richardson in [6].

Definition 1: Given a decoder input $\mathbf{y}$, a *trapping set* (TS) for an iterative decoder denoted by $\mathbf{T}(\mathbf{y})$ is a non-empty set of variable nodes in G that are not corrected at the end of a given number of iterations.

A common notation used to denote a TS is (a, b), where $a = |\mathbf{T}|$, and b is the number of odd-degree check nodes in the subgraph induced by the set of variable nodes $\mathbf{T}$. Let $\mathcal{T}(a, b)$ denote the bipartite graph associated with an (a, b) TS, where a is the number of variable nodes and b is the number of odd-degree check nodes present in the graph. A graph G contains an (a, b) TS of type $\mathcal{T}$ if there exists a subset of variable nodes $\mathbf{T}$ in G whose induced subgraph is isomorphic to $\mathcal{T}(a, b)$. A TS is said to be *elementary* if $\mathcal{T}$ contains only degree-one and/or degree-two check nodes. Otherwise it is non-elementary. Throughout this paper, we restrict our focus to elementary trapping sets, since they are known to be dominant in the error floor [6], [7], [11]. Henceforth, for convenience, whenever we refer to a TS, we will implicitly refer to its topological structure $\mathcal{T}$. Clearly the (a, b) notation is not sufficient to describe a topology of a trapping set as there can be many non-isomorphic graphs with a variable nodes and b odd-degree check nodes. However listing all non-isomorphic graphs is intractable when a is large, thus in this paper we use the notation which enumerates the distinct cycles in a subgraph [12]. Let $\mathcal{T}$ contains g_{2k} $(2k)$-cycles, where $k \geq 2$, then the trapping set associated with $\mathcal{T}$ is said to be of type $(a, b, \prod_{k\geq 2}(2k)^{g_{2k}})$. We also use the notation $\mathcal{T}(a, b, \prod_{k\geq 2}(2k)^{g_{2k}})$ to say that a graph G contains an $(a, b, \prod_{k\geq 2}(2k)^{g_{2k}})$ TS of type $\mathcal{T}$ if there exists a subset of variable nodes $\mathbf{T}$ in G whose induced subgraph is isomorphic to $\mathcal{T}(a, b, \prod_{k\geq 2}(2k)^{g_{2k}})$. An alternative method to describe accurately trapping sets is described in [11].

An very efficient method for constructing Tanner graphs having a large girth is by using the Progressive-Edge-Growth (PEG) algorithm [8]. In the PEG algorithm, a new edge for a variable node v has to be created, based on the expansion of a subgraph from v within depth k as shown in Fig.1, also called the neighboring tree or computation tree. It is a tree with $2(k + 1)$ levels, labeled from $[0, 2k + 1]$, where the 0^{th} level consists only of the root node v, even-numbered levels contain only variable nodes represented by $\circ$, and odd-numbered levels contain only check nodes represented by $\blacksquare$.

The depth of the tree k depends of the target girth [1], and the subgraph is called *depth-k tree*. We denote the set of all check nodes neighbors in depth-k tree by $\mathcal{M}_v^k$, and $\overline{\mathcal{M}}_v^k$ the complementary set.

We introduce in the rest of this section some useful definitions on the structure of the depth-k tree for the characterization of trapping sets.

Definition 2: Let $T_v^k(G)$ be a depth-k tree for the variable node v of a Tanner graph G. The variable nodes of $T_v^k(G)$ are copies of the variable nodes of G, and the check nodes of $T_v^k(G)$ are copies of the check nodes of G.

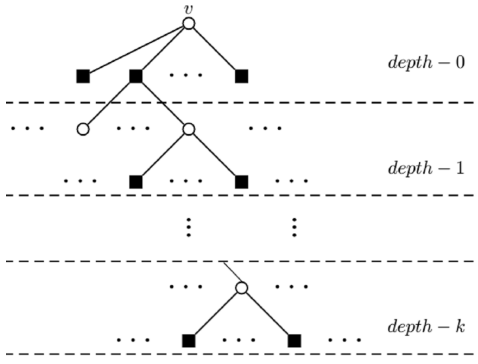


Fig. 1. A subgraph spreading from variable node v: *depth-k tree*

When there is no ambiguity we will shorten the notation and use T_v^k when the graph is specified, T_v when the depth is specified. In a systematic expansion of the depth-k tree, different copies of the same variable node can appear at different levels of T_v. Let T_v contain m copies of a node u, we will denote those copies as $u_i^{l_i}$, $1 \leq i \leq m$, where l_i is the level of the i-th copy.

Definition 3: A node $w \in T_v(G)$ is said to be an *ascendant* of a node $u \in T_v(G)$ if there exists a path starting from the node u to the root v that traverses through node w. The set of all ascendants *variable nodes* of the node u in T_v is denoted as $\mathcal{A}(u)$. For a given node set U, $\mathcal{A}_{T_v(G)}(U)$ denotes the set of ascendants variable nodes of all $u \in U$.

When there is no ambiguity, we will shorten the notation and use $\mathcal{A}(u)$ or $\mathcal{A}_{T_v}(u)$ instead of $\mathcal{A}_{T_v(G)}(u)$. We use $\mathcal{A}(U)$ in a similar fashion.

Definition 4: A node $w \in T_v$ is said to be a *parent* of a node $u \in T_v$ if w is directly connected to u along the path traversing to the root. Again, it is clear that $w \in \mathcal{N}_u$.

Definition 5: A node $w \in T_v(G)$ is said to be a *descendant* of a node $u \in T_v(G)$ if there exists a path starting from node w to the root v that traverses through node u. The set of all descendants of the node u in T_v is denoted as $\mathcal{D}(u)$. For a given node set U, $\mathcal{D}_{T_v(G)}(U)$ denotes the set of descendants of all $u \in U$. A node $w \in T_v$ is said to be a *child* of a node $u \in T_v$ if w is directly connected to u along the path traversing to the root. Clearly, $w \in \mathcal{N}_u$. A node that does not have any child nodes in T_v is called a *leaf node*.

III. Combinatorial Characterization of $(5, 3)$ and $(6, 4)$ Trapping Sets

A. Characterization of $(5, 3)$ and $(6, 4)$ Trapping Sets

In this paper, we only consider the case of $\mathcal{C}_{d_v,g}$ codes, with column weight $d_v = 3$ and girth $g = 8$. The trapping sets are formed by combination of several cycles, and (a, b) TS can be differentiated by the number of distinct cycles that form the TS. The elementary trapping sets are known to be dominant in the error floor [4], [6], [7], [11] and among the most harmful are the TS with a small number a of variable nodes.

For $\mathcal{C}_{3,8}$, the $(4,4)$ TS denote the 8-cycles, and following the ontology of trapping sets for regular column-weight-tree codes in [10] the smallest TS are $(5,3)$, and $(6,4)$. The following theorems restrict possible topologies of $(5,3)$ and $(6,4)$ trapping sets. These characteristics are used to define the criteria for detecting trapping set in the depth-k tree.

The proofs in this section are omitted because of lack of space, but will be reported in a future paper.

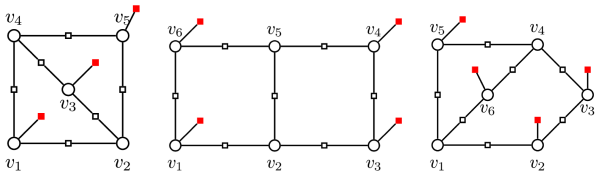
Lemma 1: There are no non-elementary trapping sets of the type $\mathcal{T}(5,3,8^3)$.

Theorem 1: Let $\mathcal{T}$ be a subgraph associated with an elementary $(5,3,8^3)$. Then any two 8-cycles in $\mathcal{T}$ have exactly three variable nodes in common.

Theorem 2: Let $\mathcal{T}$ be a subgraph associated with an elementary $(6,4,8^2,12^1)$. Then the two 8-cycles in $\mathcal{T}$ have exactly two variable nodes in common.

Theorem 3: Let $\mathcal{T}$ be a subgraph associated with an elementary $(6,4,8^1,10^2)$. Then the two 10-cycles in $\mathcal{T}$ have exactly four variable nodes in common.

The topologies of interest in this paper are characterized by the above theorems, and depicted in figures Fig. 2(a) to Fig. 2(c). Fig. 2(a) shows a $\mathcal{T}\left(5,3,8^3\right)$ where the sets of three 8-cycles are:$\{v_1,v_2,v_3,v_4\}$, $\{v_2,v_3,v_4,v_5\}$, and $\{v_1,v_2,v_4,v_5\}$, any two sets have three variables nodes in common. Similarly, Fig. 2(b) shows $\mathcal{T}\left(6,4,8^2,12^1\right)$ where the sets of two 8-cycles are:$\{v_1,v_2,v_5,v_6\}$ and $\{v_2,v_3,v_4,v_5\}$, they have v_2 and v_5 in common. For $\mathcal{T}\left(6,4,8^1,10^2\right)$ represented by Fig. 2(c), $\{v_1,v_2,v_3,v_4,v_5\}$ and $\{v_1,v_2,v_3,v_4,v_6\}$ are two 10-cycle where $\{v_1,v_2,v_3,v_4\}$ are the variable nodes in common.



(a) (b) (c)

Fig. 2. $\mathcal{T}(5,3,8^3)$; $\mathcal{T}(6,4,8^2,12)$; $\mathcal{T}(6,4,8^1,10^2)$ respectively

B. Criteria to Detect trapping set (5,3) and (6,4) in PEG

In a PEG algorithm, the cycles can be easily predicted by the level at which each check node appears in the depth-k tree. However, detecting complex structures such as TS is more tedious. In the depth-k tree, TS can be predicted by using copies of check nodes c with the knowledge of their appearance levels and of the variable nodes in common on the distinct paths $v,.....,c_i^{l_i}$ in T_v. The following theorem gives the maximum number of cycles that are created by a new edge between v and a check node c that has m copies in T_v.

Theorem 4: Let T_v contain m copies of a check node c, the maximum number of cycles that contains c is $\eta = \binom{m}{2}$. If we chose to connect v and c for the new edge then the maximum number of cycles for a new structure is $\eta + m = \frac{m(m+1)}{2}$.

Proof: Let T_v contains m copies of check node c, the maximum number of cycles that contains c is $\eta = \binom{m}{2} = \frac{m(m-1)}{2}$, if we chose to connect v and c we have m new cycles hence $\eta + m = \frac{m(m-1)}{2} + m = \frac{m(m+1)}{2}$ cycles at maximum. ■

The trapping sets $(5,3)$ and $(6,4)$ are formed by three cycles. To be able to detect those structures in the depth-k tree, we provide a characterization of the number of copies of a check node required to form a combination of α cycles.

Lemma 2: A new combination of α cycles is detected in T_v, if a check node c has m copies with $m = \lfloor \frac{-1+\sqrt{1+8\alpha}}{2} \rfloor$.

As a consequence, for a combination of three cycles, a check node c needs to have at least two copies in T_v.

Finally, using the theorems and lemmas introduced earlier in the paper, we can now state the following theorems which describe how to detect the trapping set $(5,3)$ and $(6,4)$ in T_v.

Theorem 5: Let $\mathcal{C}_{3,8}$, for each and every variable node v a new trapping-set $(5,3,8^3)$ is detected in T_v if and only if there exists at least 2 copies of check node c denoted by c_1^7, c_2^7 with the same *p*arent, for which the path $v,...,c_1^7$ and the path $v,....,c_2^7$ in T_v have three common variable nodes *(ie.* $|\mathcal{A}(c_1^7) \bigcap \mathcal{A}(c_2^7)| = 3$*)*.

Proof: See Appendix A ■

Theorem 6: Let $\mathcal{C}_{3,8}$, for each and every variable node v a trapping-set $(6,4,8^2,12^1)$ is detected in T_v if and only if we have one of two cases:

1) there exists at least 2 copies of check node c denoted by c_1^7, c_2^{11} with the same *p*arent, for which the path $v,...,c_1^7$ and the path $v,....,c_2^{11}$ in T_v have four common variable nodes *(ie.* $|\mathcal{A}(c_1^7) \bigcap \mathcal{A}(c_2^{11})| = 4$*)*.
2) there exists at least 2 copies of check node c denoted by c_1^7, c_2^7 with the same *p*arent, for which the path $v,...,c_1^7$ and the path $v,....,c_2^7$ in T_v have two common variable nodes *(ie.* $|\mathcal{A}(c_1^7) \bigcap \mathcal{A}(c_2^7)| = 2$*)*.

Proof: See Appendix B ■

Theorem 7: Let $\mathcal{C}_{3,8}$, for each and every variable node v a trapping-set $(6,4,8^1,10^2)$ is detected in T_v if and only if we have one of two cases:

1) there exists at least 2 copies of check node c denoted by c_1^7, c_2^9 with the same *p*arent, for which the path $v,...,c_1^7$ and the path $v,....,c_2^9$ in T_v have three common variable nodes *(ie.* $|\mathcal{A}(c_1^7) \bigcap \mathcal{A}(c_2^9)| = 3$*)*.
2) there exists at least 2 copies of check node c denoted by c_1^9, c_2^9 with the same *p*arent, for which the path $v,...,c_1^9$ and the path $v,....,c_2^9$ in T_v have four common variable nodes *(ie.* $|\mathcal{A}(c_1^9) \bigcap \mathcal{A}(c_2^9)| = 4$*)*.

Proof: See Appendix C ■

IV. Design Algorithm Using RandPEG

A. Description of Algorithm

In this section, we briefly review the RandPEG algorithm proposed by Venkiah and al. [1], then describe our improvement, which consists essentially in adding a new constraint in the objective function of the RandPEG. The additional constraint consists in detecting the trapping set $(5,3)$, $(6,4)$,

removing all candidate check nodes that create trapping sets (5,3) and select only the check nodes that minimize the number of trapping sets $(6,4)$.

Let us consider LDPC codes in $\mathcal{C}_{3,8}$. We seek to construct LDPC codes without trapping sets (5,3). When the depth-k tree is spread up to $k_{max} \geq 3$, the *cost function* restricts the set of check nodes candidates as follows:

- Let $\overline{\mathcal{M}}_v^{k_{max}}$ denote the set of check nodes that are possible candidates for connecting a new edge from the root of the computation tree. The first step if to remove from $\overline{\mathcal{M}}_v^{k_{max}}$ all check nodes that appear at least once in the depth-3 computation tree. This removes all check nodes that would create cycles of size < 8.
- For each remaining check node c_m in $\overline{\mathcal{M}}_v^{k_{max}}$, compute $nbtrapping_m[i]_{0\leq i\leq 1}$, the number of trapping set (5,3) and (6,4) that would be created if c_m is selected, using the characterization of theorems 5 and 6. Remove all check nodes that would create trapping sets (5,3) $i.e$ check-nodes c_m for which $nbtrapping_m[0] \neq 0$ and in order to minimize the global number of trapping sets (6,4), remove check nodes that create more than $min_m(nbtrapping_m[1])$.
- If $\overline{\mathcal{M}}_v^{k_{max}} \neq \emptyset$, randomly select a check node c_m and connect a new edge between the root variable node and c_m. If $\overline{\mathcal{M}}_v^{k_{max}} = \emptyset$, a design failure is declared.

B. Simulation Result

We focus on the design of quasi-cyclic LDPC (QC-LDPC) codes, which are the most largely used LDPC codes in practical applications. The specificity of PEG for QC-LDPC code design is to expand only the computation tree of the variable node $v = nL$, where L is the size of circulant permutation matrix, and assign L edges in one PEG step, by assigning a whole circulant. In table I, we present the statistics of several constructions for regular QC-LDPC codes with $(d_v = 3, d_c = 5)$ and $(d_v = 3, d_c = 9)$. For the $L = 31$ case, we also give the statistics of the Tanner code proposed in [13].

For the PEG algorithm with $L = 18$ the obtained girth is $g = 6$ hence we have no $(5,3)$ and $(6,4)$, but it also demonstrate the need for more efficient PEG-like constructions, as girth $g = 8$ is achievable for this rate and length. The statistics in this table clearly show that our new constraint in the design constructively avoid all TS (5,3), while it also minimize the number of TS (6,4), compared to the original RandPEG construction.

Fig.3 and Fig.4 show the performance on the BPSK-AWGN channel of several regular LDPC codes listed in table I. We can see that the PEG algorithm can greatly be improved by using the RandPEG algorithm, and even more with our new improvements. Note that for these small codeword lengths, we have not reached the error floor region in our simulation (except for the PEG, $L = 18$ code which has girth 6), but the slope improvement is significant and shows that we removed many dominant fixed point of the iterative decoder.

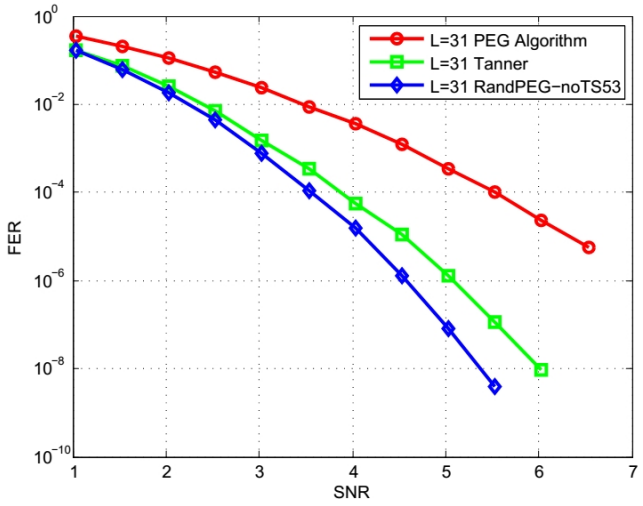


Fig. 3. Performance comparison of QC-LDPC with L=31, $d_v = 3$, $d_c = 5$

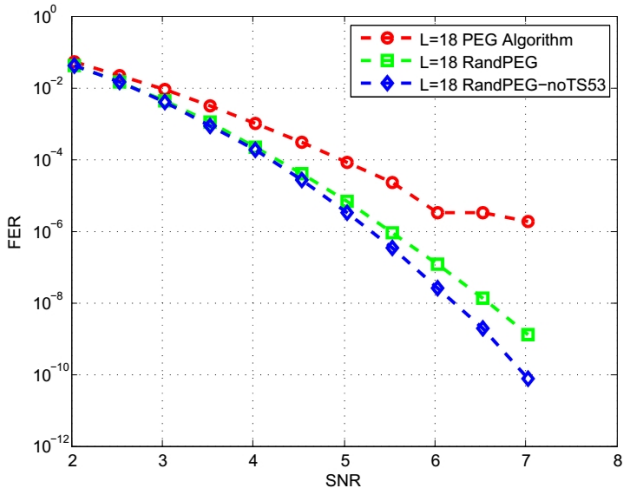


Fig. 4. Performance comparison of QC-LDPC with L=18, $d_v = 3$, $d_c = 5$

V. Conclusions

In this paper, we have proposed a predictive method to construct regular column-weight-three LDPC codes with girth $g = 8$ so that their Tanner graphs have not trapping set $(5,3)$ and a minimum number of trapping sets $(6,4)$. Our construction is based on Randomized Progressive Edge Growth (RandPEG) algorithm. Both statistics on the cycles and smallest TS and Monte Carlo simulations demonstrate the advantage of our approach for small lengths QC-LDPC codes, compared to existing designs.

Acknowledgements

This work has been partially funded by AVAGO Technologies, the NSF (grants CCF-0963726 and CCF-1314147) as well as the United States Department of State Bureau of Educational and Cultural Affairs through the Fulbright Scholar Program.

	$L=18, d_c=5$			$L=31, d_c=5$				$L=52, d_c=9$	
	PEG	RandPEG	RandPEG no (5,3)	PEG	RandPEG	RandPEG no (5,3)	Tanner	PEG	RandPEG no (5,3)
$\#cycle-6$	36	0	0	0	0	0	0	0	0
$\#cycle-8$	684	774	720	682	620	527	465	11518	9776
$\#cycle-10$	3402	3402	3582	3255	3348	3689	3720	51064	50596
$\#(5,3;8^3)$	0	144	0	62	31	0	155	1196	0
$\#(6,4;8^2,12)$	0	2286	1998	1271	930	434	0	52624	36504
$\#(6,4;8,10^2)$	0	1530	1872	620	992	620	930	50492	41184

TABLE I
COMPARISON OF THE NUMBER OF CYCLES AND THE NUMBER OF TRAPPING SETS FOR SEVERAL PEG CONSTRUCTIONS

APPENDIX A
PROOF THEOREM 6

A $(5,3,8^3)$ is composed by three cycles. Let $\mathcal{C}_{3,8}$ and T_v contain m copies of a check-node c such as two copies are denoted by $c_1^{l_1}$ and $c_2^{l_2}$. For the elementary trapping set all check nodes have two degree two neighbors at maximum hence if $c_1^{l_1}$ and $c_2^{l_2}$ have not the same *parent* the trapping set is not elementary.

Let $g_1 = l_1 + 1$, $g_2 = l_2 + 1$, and n the number of variable nodes in common between $\mathcal{A}(c_1^{l_1})$ and $\mathcal{A}(c_2^{l_2})$ ie. $n = |\mathcal{A}(c_1^{l_1}) \bigcap \mathcal{A}(c_2^{l_2})|$.

- if $l_1 \neq 7$ *(resp.* $l_2 \neq 7$*)* then $g_1 \neq 8$ *(resp.* $g_2 \neq 8$*)*. There is no only 8-cycle, hence no $(5,3,8^3)$.
- For $l_1 = 7$ and $l_2 = 7$, the length of cycle formed by two distinct paths $v, ..., c_1^7$ and $v, ..., c_2^7$ is $g = 7+7+6-4n$ *(see Theorem5)*, finally $g = 20 - 4n$, if $n \neq 3 \Rightarrow g \neq 8$ and if $n = 3 \Rightarrow g = 8$, and $|\mathcal{A}(c_1^7) \bigcup \mathcal{A}(c_2^7)| = 5$.

Hence we have a $(5,3,8^3)$, if $l_1 = l_2 = 7$, and $n = 3$.

APPENDIX B
PROOF THEOREM 7

A $(6,4,8^2,12^1)$ is composed of three cycles. Let $\mathcal{C}_{3,8}$ and T_v contain m copies of a check-node c such as two copies are denoted by $c_1^{l_1}$ and $c_2^{l_2}$. For the elementary trapping set all check nodes have two degree two neighbors at maximum hence if $c_1^{l_1}$ and $c_2^{l_2}$ have not the same *parent* the trapping set is not elementary.

Let $g_1 = l_1 + 1$, $g_2 = l_2 + 1$, and n the number of variable nodes in common between $\mathcal{A}(c_1^{l_1})$ and $\mathcal{A}(c_2^{l_2})$ ie. $n = |\mathcal{A}(c_1^{l_1}) \bigcap \mathcal{A}(c_2^{l_2})|$. $(6,4,8^2,12^1)$ contains two 8-cycles and one 12-cycle hence there is two case levels for c:

1) **Case one:** $(l_1 = 7, l_2 = 11)$ hence $(g_1 = 8, g_2 = 12)$ or $(l_1 = 11, l_2 = 7)$ hence $(g_1 = 12, g_2 = 8)$, the length of cycle formed by two distinct path $v, ..., c_1^{l_1}$ and $v, ..., c_2^{l_2}$ is: $g = 7 + 11 + 6 - 4n = 24 - 4n$. If $n \neq 4 \Rightarrow g \neq 8$ else $n = 4 \Rightarrow g = 8$ and $|\mathcal{A}(c_1^{l_1}) \bigcup \mathcal{A}(c_2^{l_2})| = 6$.
2) **Case two:** $(l_1 = l_2 = 7)$ hence $(g_1 = g_2 = 8)$, the length of cycle formed by two distinct path $v, ..., c_1^{l_1}$ and $v, ..., c_2^{l_2}$ is: $g = 7+7+6-4n = 20-4n$, if $n \neq 2 \Rightarrow g \neq 12$ else $n = 2 \Rightarrow g = 12$ and $|\mathcal{A}(c_1^{l_1}) \bigcup \mathcal{A}(c_2^{l_2})| = 6$.

The only case where there are a $(6,4,8^2,12^1)$ is $|\mathcal{A}(c_1^7) \bigcap \mathcal{A}(c_2^{11})| = 4$ or $|\mathcal{A}(c_1^7) \bigcap \mathcal{A}(c_2^7)| = 2$.

APPENDIX C
PROOF THEOREM 8

The proof of Theorem 8 is similar to the one of Theorem 7.

REFERENCES

[1] A. Venkiah, D. Declercq, and C. Poulliat, "Design of cages with a randomized progressive edge growth algorithm," in *IEEE Commun. Letters*, vol. 12, April 2008, pp. 301–303.

[2] T. Tian, C. Jones, J. D. Villasenor, and R. Wesel, "Construction of irregular ldpc codes with low error floors," in *Proc. IEEE Int. Conf. on Communications*, vol. 5, May 2003, pp. 3125–3129.

[3] X. Zheng, F. C. M. Lau, and C. K. Tse, "Constructing short-length irregular ldpc codes with low error floor," in *Proc. IEEE Commun.*, vol. 58, Oct. 2010, pp. 2823–2834.

[4] R. Asvadi, A. Banihashemi, and M. Ahmadian-Attari, "Lowering the error floor of ldpc codes using cyclic lifting," in *IEEE Trans. Inf. Theory*, vol. 57, no. 4, Apr. 2011, pp. 2213–2224.

[5] H. Xinde, L. Zongwang, B. V. K. V. Kumar, and R. Barndt, "Error floor estimation of long ldpc codes on magnetic recording channels," in *Mag. IEEE Trans.*, vol. 46, no. 6, June 2010, pp. 1836–1839.

[6] T. J. Richardson, "Error floors of LDPC codes," in *Proc. 41st Annual Allerton Conf. on Communications, Control and Computing*, 2003, pp. 1426–1435. [Online]. Available: http://www.hpl.hp.com/personal/Pascal_Vontobel/pseudocodewords/papers

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

[8] X. Y. Hu, E. Eleftheriou, and D. M. Arnold, "Regular and irregular progressif edge-growth tanner graphs," in *IEEE Trans. Inf. Theory*, vol. 51, no. 01, Jan. 2005, pp. 386–398.

[9] G. Richter and A. Hof, "On a construction method of irregular ldpc codes without small stopping sets," in *Proc. IEEE Int. Conf.*, vol. 3, June 2006, pp. 1119–1124.

[10] V. Vasic, S. K. Chilappagari, D. V. Nguyen, and S. K. Planjery, "Trapping set ontology," in *Proc. 47th Annual Allerton Conf. on Commun., Control, and Computing*, Sept. 2009.

[11] M. Karimi and A. Banihashemi, "On characterization of elementary trapping sets of variable-regular ldpc codes," in *IEEE Trans. Inf. Theory*, vol. 60, no. 09, Sept. 2014, pp. 5188–5203.

[12] D. Declercq, B. Vasic, S. K. Planjery, and E. Li, "Finite alphabet iterative decoders, Part II: Improved guaranteed error correction of LDPC codes via iterative decoder diversity," in *IEEE Trans. Commun.*, vol. 61, no. 10, Nov. 2013, pp. 4046–4057.

[13] M. Tanner, D. Sridhara, and T. Fuja, "A class of group-structured ldpc codes," 2001. [Online]. Available: citeseer.ist.psu.edu/tanner01class.html