A Low Complexity PEG-like Algorithm to Construct Quasi-Cyclic LDPC Codes

Anthony Gómez-Fonseca*, Roxana Smarandache*†, and David G. M. Mitchell‡
Departments of *Mathematics and †Electrical Engineering, University of Notre Dame, Notre Dame, IN 46556, USA
{agomezfo, rsmarand}@nd.edu
‡Klipsch School of Electrical and Computer Engineering, New Mexico State University, Las Cruces, NM 88003, USA
dgmm@nmsu.edu

2023 12th International Symposium on Topics in Coding (ISTC) | 979-8-3503-2611-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISTC57237.2023.10273481

***Abstract*—In this paper, we introduce a modified version of the Progressive Edge-Growth (PEG) algorithm that is used to construct the Tanner graphs of quasi-cyclic (QC) low-density parity-check (LDPC) codes having large girth. In our algorithms, we replace certain requirements in the original PEG algorithm, namely the expansion of subgraphs from symbol nodes and the identification of a most distant check node before placing a new edge, by the construction of certain minimal sets, called forbidden sets, that contain the problematic quasi-cyclic constraints that do not let us exceed a given girth. We propose some exponent-selection strategies that are shown to increase the chance of obtaining large girth, and reduce the number of short cycles in the Tanner graph.**

I. Introduction

Quasi-cyclic low-density parity-check (QC-LDPC) codes are now in many industry standards, including those developed by the Consultative Committee for Space Data System (CCSDS) [1], due to their structural simplicity and their capacity-approaching performance. The compact way in which QC-LDPC codes can be described is known to be the reason why they are hardware-friendly, a feature that leads to linear-time encoding [2] and efficiencies in decoder design [3]. As a consequence of having these desirable properties, significant research effort has been made to propose different construction strategies. Some of these strategies involve algebraic or combinatorial designs [4]–[6]. Other approaches are based on optimization and greedy-search algorithms, which can include graph-theoretical ideas [7], [8].

One particularly effective strategy based on the latter is the Progressive Edge-Growth (PEG) algorithm [9]. Originally proposed as a general method for constructing Tanner graphs having a large girth, the PEG algorithm works by establishing edges or connections between symbol and check nodes in an edge-by-edge manner. The placement of a new edge in the Tanner graph is done in such a way that the impact on the girth is minimized. The idea behind minimizing the impact on the girth requires the expansion of a subgraph from a symbol node and the identification of a most distant check node before placing a new edge. The complexity of the PEG algorithm scales as $\mathcal{O}(mn)$, where m is the number of check nodes and n is the number of symbol nodes in the Tanner graph. Since a Tanner graph with target girth g satisfying the input parameters might not exist, the algorithm does not guarantee a specific girth. The PEG algorithm has subsequently been modified to include quasi-cyclic (QC) constraints [10]–[13].

The most natural QC modification is given in [10], where the symbol and check nodes are divided into smaller groups having N nodes. The edges are then created in a group-by-group fashion instead of node-by-node, and the subgraphs or trees are expanded only for the first node in each group of N symbol nodes. The edges created for the remaining $N-1$ symbol nodes in the corresponding group are determined using the quasi-cyclic constraint. In [11], the concept of PEG-undetectable cycles is introduced to form part of the computation tree in a QC-PEG algorithm. PEG-undetectable cycles are cycles that could be formed by a circulant assignment during the PEG construction, while not being detected in the computation tree. In [12], [13], a modified version of the PEG algorithm is presented which can simultaneously construct large girth base graphs and determine the optimal quasi-cyclic constraints.

In this paper, we propose several low complexity PEG-like algorithms to construct QC-LDPC codes with large girth. These algorithms benefit from a low complexity since neither the expansion of subgraphs from symbol nodes, nor the identification of a most distant check node before placing a new edge, are required. The reduced complexity of these algorithms is attributed to the novel approach to construct some minimal sets, called forbidden sets, which contain the problematic quasi-cyclic constraints corresponding to the necessary and sufficient conditions that do not let us exceed a given girth.

II. Definitions, notation and background

Let $\mathcal{C}$ be a QC-LDPC code, either regular or irregular, with block length n_vN, that is based on the $n_c \times n_v$ protograph [7] described by the matrix $B = (b_{ij})_{n_c \times n_v}$, where b_{ij} is a nonnegative integer for $i \in [n_c]$ and $j \in [n_v]$, and where $[l] \triangleq \{0, 1, \ldots, l-1\}$. Then $\mathcal{C}$ can be described by a scalar parity-check matrix $H = (H_{ij})_{n_c \times n_v}$, where each H_{ij}, for $i \in [n_c]$ and $j \in [n_v]$, is a summation of b_{ij} $N \times N$ circulant permutation matrices if b_{ij} is nonzero, and the $N \times N$ all-zero matrix if $b_{ij} = 0$. Graphically, this operation is equivalent to taking an N-fold graph cover, or *lifting*, of the protograph. Here, N is called the *lifting factor* (alternatively, lifting degree, or degree of the graph cover).

Let x^r denote the $N \times N$ circulant permutation matrix obtained by circularly shifting to the left, by r positions modulo N, the entries of the $N \times N$ identity matrix I. For simplicity in the notation, let $p_{ij}(x)$ be the polynomial representation of H_{ij}, where $p_{ij}(x) = \sum_{l=0}^{N-1} a_l x^l$ and $a_l \in \{0, 1\}$ for all $l \in [N]$. Each polynomial $p_{ij}(x)$ has weight b_{ij}. Then we can rewrite the parity-check matrix H, using the polynomial representation, as $H = (p_{ij}(x))_{n_c \times n_v}$.

From the scalar parity-check matrix $H = (h_{ij})_{n_cN \times n_vN}$, we construct a bipartite graph $G = (V, E)$, called a Tanner graph [8], by considering H as its biadjacency matrix. Since the graph G is bipartite, the set of vertices V is the union of two disjoint sets $V_c = \{c_i \mid i \in [n_cN]\}$ and $V_s = \{s_j \mid j \in [n_vN]\}$. The vertices in the sets V_c and V_s are called check nodes and variable nodes, respectively. The set of edges E is the collection of all (c_i, s_j) such that $c_i \in V_c$, $s_j \in V_s$ and $h_{ij} = 1$. Collectively, the vertices in V are denoted by v_a, $a = 0, 1, \ldots, |V|-1$, and the edges by e_b, $b = 0, 1, \ldots, |E|-1$, and where $|S|$ denotes the cardinality of the set S. A (directed) walk W of length m in the graph G is an alternating sequence $W = v_0e_1v_1e_2 \cdots v_{m-1}e_mv_m$ of vertices and edges such that $e_l = (v_{l-1}, v_l) \in E$ for all $1 \leq l \leq m$. A walk W is said to be closed if $v_0 = v_m$. A cycle is a closed walk W, having distinct vertices, except for $v_0 = v_m$, and if its alternating sequence has k edges in it, then we call W a k-cycle. Finally, a *multiset* (shortened to mset) is a generalization of a set in which elements are allowed to repeat.

III. Constructing the Forbidden Sets

The algorithms that we introduce in this paper admits a significant reduction in complexity when compared to other approaches in the literature. The reason is because they depend on the construction of certain minimal sets that reduce the search space of each exponent in the polynomial parity-check matrix H. These sets are called forbidden sets and are introduced in the following definition.

Definition 1. *Let H be the polynomial parity-check matrix of a QC-LDPC code and let N be the lifting factor. The forbidden set $\mathcal{F}_a^{g,N}$ of the exponent a, corresponding to the circulant permutation matrix x^a in H, is the set of elements in $[N]$ such that if $a \in \mathcal{F}_a^{g,N}$, then girth$(H) \leq g$.* □

To illustrate how to construct these forbidden sets, consider the following example.

Example 1. Let the parity-check matrix H be given by

$$H = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & x^{i_1} & \cdots & x^{i_{n_v-1}} \\ 1 & x^{j_1} & \cdots & x^{j_{n_v-1}} \end{bmatrix}. \tag{1}$$

As discussed in [6], girth$(H) > 4$ if and only if all the elements in each one of the three msets $\{i_l \mid l \in [n_v]\}$, $\{j_l \mid l \in [n_v]\}$, and $\{i_l - j_l \mid l \in [n_v]\}$ are distinct. It is not difficult to see that

$$\mathcal{F}_{i_l}^{4,N} = \{i_{l'}, j_l + i_{l'} - j_{l'} \mid l' \in [n_v] \backslash \{l\}\},$$
$$\mathcal{F}_{j_l}^{4,N} = \{j_{l'}, i_l + j_{l'} - i_{l'} \mid l' \in [n_v] \backslash \{l\}\}.$$

Similarly, girth$(H) > 6$ if and only if, for $m \in [n_v]$, all the elements in each one of the msets $\{-i_l + i_m, -j_l + j_m \mid l \in [n_v], l \neq m\}$, $\{i_l, i_l - j_l + j_m \mid l \in [n_v], l \neq m\}$, and $\{j_l, j_l - i_l + i_m \mid l \in [n_v], l \neq m\}$ are distinct. Let,

$$\mathcal{T}_{i_l}^{6,N} = \bigcup_{m \in [n_v] \backslash \{l\}} \{i_m - j_m + j_{l'}, j_l + i_m - j_{l'} \mid l' \neq m, l\},$$
$$\mathcal{T}_{j_l}^{6,N} = \bigcup_{m \in [n_v] \backslash \{l\}} \{j_m - i_m + i_{l'}, i_l - i_m + j_{l'} \mid l' \neq m, l\}.$$

Then $\mathcal{F}_{i_l}^{6,N} = \mathcal{F}_{i_l}^{4,N} \cup \mathcal{T}_{i_l}^{6,N}$ and $\mathcal{F}_{j_l}^{6,N} = \mathcal{F}_{j_l}^{4,N} \cup \mathcal{T}_{j_l}^{6,N}$. A similar approach is used to determine $\mathcal{F}_a^{g,N}$ for any regular or irregular polynomial parity-check matrix. □

IV. A Forbidden Set Based PEG Algorithm

Let $\mathcal{C}$ be an (n_c, n_v)-regular QC-LDPC code with length $n = n_vN$ based on the $n_c \times n_v$ fully-connected (all-ones) protograph. After some row and column permutations, and some relabeling, we can write the parity-check matrix H as:

$$H = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & x^{e_{1,1}} & \cdots & x^{e_{1,n_v-1}} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x^{e_{n_c-1,1}} & \cdots & x^{e_{n_c-1,n_v-1}} \end{bmatrix}. \tag{2}$$

A. Forward selection, reducing girth where necessary

In the following example, we illustrate the idea behind our first algorithm, which tries to maximize the girth for a given code length, but reduces the target if and when the desired girth cannot be achieved. This idea imitates the original PEG algorithm, but uses the forbidden sets.

Example 2. Suppose that we want to construct a QC-LDPC code based on the 3×4 fully-connected protograph with fixed lifting factor $N = 64$. We will choose the i_l's and j_l's from the set $[N] = [64]$, selecting randomly from the available choices that permit girth 12 according to the forbidden sets, following the order i_1, i_2, i_3, j_1, j_2 and j_3, this is, row-by-row

$$H_r = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & x^{i_1} & x^{i_2} & x^{i_3} \\ 1 & x^{j_1} & x^{j_2} & x^{j_3} \end{bmatrix} \xrightarrow{\text{girth } 12} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & x^{3} & x^{22} & x^{37} \\ 1 & x^{17} & x^{26} & 0 \end{bmatrix}.$$

It turns out that our random selection of exponents satisfying the girth 12 conditions was successful for the first five exponents. To choose j_3, we ran out of choices, so the girth 10 conditions were used instead. We take $j_3 = 57$. This gives a code with parameters $[256, 67, 24]$.

If, instead of choosing the exponents row-by-row, we choose them column-by-column, this follows the order i_1, j_1, i_2, j_2, i_3 and j_3 (again randomly from the available girth 12 exponent choices), we obtain the following:

$$H_c = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & x^{i_1} & x^{i_2} & x^{i_3} \\ 1 & x^{j_1} & x^{j_2} & x^{j_3} \end{bmatrix} \xrightarrow{\text{girth } 12} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & x^{16} & x^{43} & 0 \\ 1 & x^{30} & x^{45} & 0 \end{bmatrix}.$$

In this case, our random selection of exponents satisfying the girth 12 conditions was successful for the first four exponents. The exponents $i_3 = 38$ and $j_3 = 63$ were chosen using the girth 10 conditions. This is a code with parameters $[256, 66, 24]$. □

Before describing our first algorithm, we define some functions. For a set S, the function RANDOM(S) returns a random element in S; the function LEN(V) returns the length of the list V (or the cardinality of the set V); and the function FORBIDDEN$(H, \texttt{var}, g, N)$ returns the set $\mathcal{F}_{\texttt{var}}^{g,N}$. Our version of a QC-PEG algorithm is described in Algorithm 1. Notice that Algorithm 1 constructs an $n_c \times n_v$ parity-check matrix H as in (2), following a forward search manner, maximizing the

Algorithm 1 QC-PEG algorithm with forbidden sets

Input: n_c, n_v, N.
Initialize exponents $h_{lk} = \infty$, $l \in [n_c]$, $k \in [n_v]$.
for $k = 0$ to $n_v - 1$ **do**
  for $l = 0$ to $n_c - 1$ **do**
    if $k = 0$ or $l = 0$ **then**
      $h_{lk} \leftarrow 0$
    else
      Choose largest $g \in \{4, 6, 8, 10\}$ such that $F = [N] \backslash \text{FORBIDDEN}(H, h_{lk}, g, N) \neq \varnothing$
      if there is such g **then**
        $h_{lk} \leftarrow \text{RANDOM}(F)$
      else
        $h_{lk} \leftarrow \text{RANDOM}([N])$
return H

Algorithm 2 QC-PEG algorithm with fixed target girth

Input: n_c, n_v, N, g.
Initialize exponents $h_{lk} = \infty$, $l \in [n_c]$, $k \in [n_v]$.
for $k = 0$ to $n_v - 1$ **do**
  for $l = 0$ to $n_c - 1$ **do**
    if $k = 0$ or $l = 0$ **then**
      $h_{lk} \leftarrow 0$
    else
      $F \leftarrow [N] \backslash \text{FORBIDDEN}(H, h_{lk}, g, N)$
      if $F = \varnothing$ **then**
        declare failure
      else
        $h_{lk} \leftarrow \text{RANDOM}(F)$
return H

girth where possible from previous exponent selections. The exponents are chosen column-by-column, following the idea of PEG and other quasi-cyclic versions of PEG in the literature [10]–[13]. Algorithm 1 was used to construct the matrix H_c in Example 2. (The matrix H_r in Example 2 was obtained by choosing the exponents row-by-row, and this just requires to flip the order of the two **for** loops.)

The strategy of sequentially choosing the exponents row-by-row, column-by-column, or even following a random ordering, is affected by the random selection of exponents. Using the conditions to guarantee girth 12, and after choosing the exponents $i_1 = 3$, $j_1 = 17$, $i_2 = 22$ and $j_2 = 26$, we obtain $F = [64] \backslash \text{FORBIDDEN}(H_{rS}, i_3, 10, 64) = \{1, 37\} \neq \varnothing$, where H_{rS} is the submatrix of H_r under the current setting. We set $i_3 = 37$. After this, the forbidden set for j_3, using the conditions to guarantee girth 12, is the set $[64]$, so we are forced to decrease the target girth by 2 and choose the (only remaining) value $j_3 = 57$ using the conditions to guarantee girth 10. The other possibility is to choose $i_3 = 1$, but this selection will force girth$(H) \leq 8$ for any $j_3 \in [64]$. For the matrix H_c, we choose $i_1 = 16$, $j_1 = 30$, $i_2 = 43$ and $j_2 = 45$. Considering this random selection of exponents, we obtained $F = [64] \backslash \text{FORBIDDEN}(H_{cS}, i_3, 10, 64) = \varnothing$, which is why we were unable to choose i_3, and in consequence j_3, satisfying the conditions to guarantee girth 12. Then we have to decrease the target girth by 2, and once we choose $i_3 = 38$, the only option to maintain the girth at 10 is choosing $j_3 = 63$.

B. Fixed girth

In Algorithm 1, the target girth is decreased by 2 once all the remaining exponents under the current setting have full forbidden set $[N]$. This algorithm will always successfully terminate and return H because it is not required to return a parity-check matrix with largest possible girth for the chosen lifting factor N. In fact, in the worst case scenario, the algorithm will return H with girth 4. If we force the target girth to remain fixed, we have to declare a failure once all the remaining exponents under the current setting have full forbidden set $[N]$. This modification is presented in Algorithm 2. In Example 3, we use this algorithm to construct QC-LDPC codes with girth 12.

Example 3. Let $N = 73$. We use Algorithm 2 to construct QC-LDPC codes based on the 3×4 fully-connected protograph with fixed target girth 12. We followed the same strategy as in Example 2, and constructed the matrices

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & x^{65} & x^{50} & x^{60} \\ 1 & x^{64} & x^{38} & x^{31} \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & x^{9} & x^{46} & x^{15} \\ 1 & x^{8} & x^{49} & x^{62} \end{bmatrix},$$

denoted by $H_{r,12}$ and $H_{c,12}$, respectively, by choosing the exponents row-by-row and column-by-column, respectively. The matrices $H_{r,12}$ and $H_{c,12}$ were constructed after 5017 and 3976 failed attempts of Algorithm 2, respectively. In Table I, we show the number of attempts and time taken for 15 successful outputs of Algorithm 2 for each exponent-selection strategy.[1,2] □

C. Multiples of rows

For the parity-check matrix H given as in (2), define its exponent matrix as $\mathcal{E}_H = (e_{ij})$, where $e_{ij} = 0$ if $i = 0$ or $j = 0$.

Example 4. The exponent matrices $\mathcal{E}_{H_{r,12}}$ and $\mathcal{E}_{H_{c,12}}$ of the parity-check matrices in Example 3 are given by

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 65 & 50 & 60 \\ 0 & 64 & 38 & 31 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 65 & 50 & 60 \\ 0 & 65 \cdot 65 & 65 \cdot 50 & 65 \cdot 60 \end{bmatrix} \text{ and}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 9 & 46 & 15 \\ 0 & 8 & 49 & 62 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 9 & 46 & 15 \\ 0 & 9 \cdot 9 & 9 \cdot 46 & 9 \cdot 15 \end{bmatrix},$$

respectively. From the previous equivalent matrices, modulo 73, we notice that the third row is a multiple of the second row. We have implemented this strategy where we select row-by-row only exponents that are a multiple of previous rows,

[1]The computations in Table I were done using SageMath [14] in a Lenovo NeXtScale nx360 M5 (server) with an Intel(R) Xeon(R) CPU E5-2680 v3 processor @ 2.50GHz and 263534552 kB 2133 MHz DDR4 of memory.

[2]We note that the column-by-column strategy is generally faster. This occurs as a result of the forbidden sets varying during the different stages of the exponent selection process.

TABLE I
NUMBER OF ATTEMPTS AND TIME TAKEN FOR 15 SUCCESSFUL OUTPUTS OF ALGORITHM 2.

	Exponent-selection strategy			
	row-by-row		column-by-column	
	Time	Attempts	Time	Attempts
1	4min 1s	365	0min 47s	60
2	15min 16s	1386	2min 17s	192
3	19min 53s	2029	6min 43s	555
4	20min 47s	2091	8min 37s	730
5	23min 6s	1994	9min 47s	882
6	29min 20s	2574	14min 38s	1231
7	31min 18s	2725	15min 51s	1278
8	33min 24s	3555	16min 58s	1652
9	39min 20s	3502	24min 19s	2346
10	42min 36s	4365	30min 41s	2699
11	55min 5s	5673	48min 50s	4374
12	57min 37s	5245	1hr 06min 7s	5631
13	1hr 3min 2s	6825	1hr 18min 31s	6661
14	1hr 4min 18s	6471	1hr 20min 17s	7404
15	2hr 14min 4s	13341	1hr 21min 42s	7504
avg	42min 12s	4143	32min 24s	2880

TABLE II
NUMBER OF ATTEMPTS AND TIME TAKEN FOR 15 SUCCESSFUL OUTPUTS FOR APPROACH IN EXAMPLE 4.

	Exponent-selection strategy			
	row-by-row		column-by-column	
	Time	Attempts	Time	Attempts
1	449ms	1	3.38s	8
2	1.5s	3	16.4s	36
3	12.8s	29	17.3s	38
4	16.1s	36	18.4s	41
5	16.1s	37	22.4s	49
6	22.4s	51	30.4s	68
7	24.4s	55	35.5s	79
8	28s	64	46.2s	104
9	36.4s	83	1min 2s	143
10	41.3s	97	1min 4s	147
11	46.3s	109	1min 19s	184
12	1min 3s	150	1min 22s	185
13	1min 7s	156	2min 13s	310
14	1min 9s	154	2min 43s	387
15	1min 15s	182	6min 13s	851
avg	35s	80	1min 16s	175

but the algorithm is not included for space constraints. In Table II, we show the number of attempts and time for 15 successful output of this exponent-selection strategy.[3] We note that the search time is drastically reduced. □

D. Selective exponent targeting

If the lifting factor N is not large enough for our target girth g, or if we choose a "bad" exponent, we will see that, after some point in the construction process, the forbidden set for a given exponent is the whole set $[N]$. Once this happens, we will have to either decrease the target girth or declare a construction failure. We can try to delay this point by assigning priority to those exponents with a larger-in-size forbidden set, but which is not exactly the set $[N]$. We explore this idea in Example 5 with a protograph used in the Consultative Committee for Space Data Systems (CCSDS) standards [1]. We also use this example to illustrate that the choice of initial target girth can affect the multiplicity of the g-cycles.

Example 5. Let the lifting factor be $N = 64$. Let H_1 be the parity-check matrix given by

$$\begin{bmatrix} 1+x^{13} & x^2 & x^{19} & x^{47} & 0 & x^{17} & x^{19} & 1 \\ x^{55} & 1+x^{37} & x^{20} & x^{19} & 1 & 0 & x & x^{16} \\ x^{61} & 1 & 1+x^{11} & x^{49} & x^{51} & 1 & 0 & x^{38} \\ x^{39} & x^{50} & x^{27} & 1+x^{20} & x^{53} & x^{32} & 1 & 0 \end{bmatrix},$$

This matrix was constructed starting with the conditions required to satisfy girth$(H_1) > 6$. When we found more than one candidate with the forbidden set having the same largest size, we randomly chose one of the candidates and continue the process as described. Following this strategy, we were able to choose all but one exponent satisfying the conditions to guarantee girth 8. The target girth is reduced to 6 and the last exponent selected. The code has parameters $[512, 256]$.

Let H_2 be the parity-check matrix given by

$$\begin{bmatrix} 1+x^{36} & x^{37} & x^{14} & x^{30} & 0 & x^{29} & x^{44} & 1 \\ x^{21} & 1+x^{35} & x^{19} & x^{29} & 1 & 0 & x^{52} & x^{48} \\ x^{40} & x^{38} & 1+x^{40} & x^{24} & x^{58} & 1 & 0 & x^{6} \\ x^{63} & x^{3} & x^{52} & 1+x^{49} & x^{60} & x^{50} & 1 & 0 \end{bmatrix},$$

This matrix was constructed starting with the conditions required to satisfy girth$(H_2) > 8$, which allowed us to choose the first group of 13 exponents. After this point, we had to decrease the girth conditions to satisfy girth$(H_2) > 6$ and were able to choose the second group of 9 exponents. The remaining two exponents were chosen to satisfy girth$(H_2) = 6$. The matrix H_2 gives a code with parameters $[512, 256]$.

Let H_3 be the parity-check matrix given by

$$\begin{bmatrix} 1+x^{30} & x^{9} & x^{8} & x^{62} & 0 & x & x^{24} & 1 \\ x^{48} & 1+x^{18} & x^{22} & x^{27} & 1 & 0 & x^{57} & x^{61} \\ x^{54} & x^{7} & 1+x^{22} & x^{47} & x^{4} & 1 & 0 & 1 \\ x^{52} & x^{54} & x^{14} & 1+x^{27} & x^{13} & x^{27} & 1 & 0 \end{bmatrix},$$

This matrix was constructed using the previous idea, but starting with the conditions to guarantee girth$(H_3) > 10$. Its girth is 6 and gives a code with parameters $[512, 256]$. □

Remark 2. The number of 6-cycles in the parity-check matrices H_1, H_2 and H_3 are 128, 320 and 384, respectively. All of these codes have fewer 6-cycles than the code constructed from the same protograph in [1]. We note that by initially selecting a large target girth, a larger number of exponents remain for the reduced girth. This implies that, since more exponents must be selected at the smallest girth, the multiplicity is higher. □

Let $\mathcal{E}$ be the list of exponents of the parity-check matrix H. For example, if H is the 3×4 matrix in Example 2, then $\mathcal{E} = [i_2, i_3, i_4, j_2, j_3, j_4]$ (the order of the elements in the list is irrelevant). We denote the kth element in $\mathcal{E}$ by $\mathcal{E}[k]$. In Algorithm 3, we describe the algorithm behind the construction of the matrices in Example 5.

In Algorithm 3, an exponent `var` has priority over the remaining exponents when its forbidden set $\mathcal{F}_{\texttt{var}}^{g,N}$ has the

[3]The computations in Table II were done using SageMath [14] in a MacBook Pro (13-inch, 2018, Four Thunderbolt 3 Ports) with a 2.3 GHz Quad-Core Intel Core i5 processor and 16 GB 2133 MHz LPDDR3 of memory.

Algorithm 3 QC-PEG algorithm with prioritization of exponent selection

Input: List of exponents $\mathcal{E}$, N.
Initialize exponents in $\mathcal{E}$ to ∞.
$C \leftarrow$ empty list to store chosen exponents
while $\text{LEN}(C) < \text{LEN}(\mathcal{E})$ **do**
  Choose largest $g \in \{4, 6, 8, 10\}$ such that $F := [N] \backslash \text{FORBIDDEN}(H, \texttt{var}, g, N) \neq \varnothing$ for at least one $\texttt{var} \in V \backslash C$
  $\mathcal{M} \leftarrow \{\texttt{var} \in \mathcal{E} \backslash C \mid \text{FORBIDDEN}(H, \texttt{var}, g, N)$ has largest size $< N\}$
  $\texttt{var} \leftarrow \text{RANDOM}(\mathcal{M})$
  Append var to C
  if there is such g **then**
    $\texttt{var} \leftarrow \text{RANDOM}([N] \backslash \text{FORBIDDEN}(H, \texttt{var}, g, N))$
  else
    $\texttt{var} \leftarrow \text{RANDOM}([N])$
return H

largest size less than N under the current setting. If there are at least two circulants permutation matrices with exponents satisfying this property, we choose one randomly and a random value in $F = [N] \backslash \mathcal{F}_{\texttt{var}}^{g,N}$ is assigned to the chosen exponent. Once the forbidden sets of all the remaining exponents are equal to the set $[N]$, the target girth g is decreased and we repeat the process.

Remark 3. Notice that in both Algorithms 1 and 3 we set $g = 12$ as the maximum possible value for the girth because the protographs that we are considering, the $n_c \times n_v$ fully-connected graph in Example 2 and the one used in the CCSDS standard described in Example 5, have at least one 2×3 all-ones submatrix. It is known that this substructure forces an upper bound of 12 in the girth [5]. We can use the results in [4] to check for harmful substructures in the protographs and set the maximum g accordingly. □

V. Complexity

The complexity of our proposed algorithms relies, almost entirely, in the complexity of determining the forbidden sets, as described in Section III. If H is the $3 \times n_v$ parity-check matrix given in (1), then $\text{girth}(H) > 4$ is equivalent to guarantee that all the elements in each one of the three msets $\{h_l - i_l \mid l \in [n_v]\}$, $\{h_l - j_l \mid l \in [n_v]\}$ and $\{i_l - j_l \mid l \in [n_v]\}$ are distinct. Since each one of these msets has n_v elements, a total of $2(n_v - 1)$ equations must be solved for each exponent i_l and j_l with $l \in [n_v]$. Hence, the complexity of determining the corresponding forbidden sets is $\mathcal{O}(n_v \log(N))$, where N is the lifting factor, so the complexity of returning H scales as $\mathcal{O}(n_c n_v^2 N \log(N))$ since we need to choose $(n_c - 1)(n_v - 1)$ exponents. For $\text{girth}(H) > 6$, a similar argument is used to show that the complexity of determining the corresponding forbidden sets is $\mathcal{O}(n_v^2 \log(N))$ and the complexity of returning H scales as $\mathcal{O}(n_c n_v^3 N \log(N))$. The same idea works to guarantee larger girths. We note that a direct comparison of complexity between the algorithms presented in this paper and the algorithms surveyed in the Introduction is challenging since the algorithms have different functionality and/or numerical data is unavailable. Such a comparison is the subject of ongoing work.

VI. Concluding remarks

The construction of the forbidden sets for the exponents in a polynomial parity-check matrix offers an alternative to construct QC-LDPC codes with large girth in a low complexity QC-PEG algorithm. Furthermore, the strategy of choosing a row as a multiple of another one reduces, significantly, the number of construction failures before a successful output when the target girth is fixed. Finally, assigning priority to those exponents with a larger-in-size forbidden set, but which is not exactly the set $[N]$, can reduce the number of short cycles in the Tanner graph and increase the likelihood of obtaining a larger girth.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant Nos. CCF-2145917, CNS-2148358, and HRD-1914635. A. G. F. thanks the support of the GFSD (formerly NPSC) and Kinesis-Fernández Richards fellowships.

References

[1] CCSDS, "TC Synchronization and Channel Coding. Issue 4. Recommendation for Space Data System Standards (Blue Book), CCSDS 231.0-B-4. Washington, D.C.: CCSDS," Jul. 2021.

[2] Z. Li, L. Chen, L. Zeng, S. Lin, and W. H. Fong, "Efficient encoding of quasi-cyclic low-density parity-check codes," *IEEE Trans. Comm.*, vol. 54, no. 1, pp. 71–81, Jan. 2006.

[3] Z. Wang and Z. Cui, "A memory efficient partially parallel decoder architecture for quasi-cyclic LDPC codes," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 4, pp. 483–488, Apr. 2007.

[4] 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.

[5] 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.

[6] R. Smarandache and D. G. M. Mitchell, "A unifying framework to construct QC-LDPC tanner graphs of desired girth," *IEEE Trans. Inf. Theory*, vol. 68, no. 9, pp. 5802–5822, Sept. 2022.

[7] J. Thorpe, "Low-density parity-check (LDPC) codes constructed from protographs," *Jet Propulsion Laboratory Pasadena, CA, INP Progress Report 42-154*, pp. 42–154, Aug. 2003.

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

[9] 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.

[10] Z. Li and B. V. K. V. Kumar, "A class of good quasi-cyclic low-density parity check codes based on progressive edge growth graph," *Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers*, vol. 2, pp. 1990–1994, Nov. 2004.

[11] M. Diouf, D. Declercq, M. Fossorier, S. Ouya, and B. Vasić, "Improved PEG construction of large girth QC-LDPC codes," *9th International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, pp. 146–150, Sept. 2016.

[12] X. Q. Jiang, M. H. Lee, H. M. Wang, J. Li, and M. Wen, "Modified PEG algorithm for large girth quasi-cyclic protograph LDPC codes," *2016 International Conference on Computing, Networking and Communications (ICNC)*, pp. 1–5, Feb. 2016.

[13] X. Jiang, H. Hai, H. Wang, and M. H. Lee, "Constructing large girth QC protograph LDPC codes based on PSD-PEG algorithm," *IEEE Access*, vol. 5, pp. 13 489–13 500, Aug. 2017.

[14] The Sage Developers, "SageMath, the Sage Mathematics Software System," 2018. [Online]. Available: https://www.sagemath.org