Pertanika J. Sci. & Technol. 27 (4): 1709 - 1722 (2019)

PERTANIKA JOURNALS

SCIENCE & TECHNOLOGY

Journal homepage: http://www.pertanika.upm.edu.my/

A New Method to Construct 4-Cycle-Free Parity-Check Matrices for Regular LDPC Codes

Djamel Slimani*, Adda Ali-Pacha and Naima Hadj-Said

Laboratory of Coding and Security of Information, University of Sciences and Technology of Oran Mohamed Boudiaf, Algeria USTO-MB, PoBox 1505 El M'Naouer Oran 31000 Algeria

ABSTRACT

Low-Density Parity-Check (LDPC) codes are considered among the best error-correcting codes in use today. These codes can be defined by a sparse parity-check matrix H, which has a graphical representation as a Tanner graph. Several studies have shown that the existence of 4-cycles in the Tanner graph affects the performance of LDPC codes. In this paper, we propose a method which allows the construction of 4-cycle-free parity-check matrices. The main principles behind the proposed method are as follows: First, we choose a vector V which consists of w_c ones and L-w_c zeros, in such a way that the chosen vector allows us to construct a circulant matrix H_1 without 4-cycles. Second, we pass this matrix to the proposed algorithm to obtain a set of L-vectors. When any vector taken from this set is appended as a news column in the matrix H_1, we obtain an L×(L+1) matrix without 4-cycles. Next, we select those vectors that lead to a circulant matrix H_2 without 4-cycles. Finally, we can obtain an L×2L matrix H without 4-cycles by concatenating matrices H_1 and H_2.Simulation results confirm that the structure of the matrices constructed by the proposed method significantly reduces the encoding complexity. Though the performance of these matrices at higher signal-to-noise-ratios (SNRs) is not as good as those constructed by MacKay's method, they can be applied to practical

ARTICLE INFO

Article history:
Received: 2 November 2018
Accepted: 13 May 2019
Published: 21 October 2019

E-mail addresses:
djamel.slimani@univ-usto.dz, djmslimani@gmail.com (Djamel Slimani)
a.alipacha@gmail.com (Adda Ali Pacha)
naima.hadjsaid@univ-usto.dz (Naima Hadj-Said)
**Corresponding author*

communications because of being encoded in linear time with shift registers.

Keywords: 4-cycle-free matrices, circulant matrices, LDPC codes, Tanner graph

INTRODUCTION

In the early sixties Gallager (1962) proposed a new class of block codes called low-density parity-check (LDPC) codes. Since then, these codes have been included in a variety of different standards including WiMAX (IEEE 802.16e), WiFi (IEEE 802.11n) and 10 Gb/s Ethernet (802.3an). An LDPC code is characterized by its sparse parity-check matrix H, which consists of zeros and ones with less ones than zeros. After the rediscovery of LDPC codes by MacKay and Neal (1995) in the mid-nineties, they quickly became very famous as they demonstrated an excellent performance (MacKay, 1999). The parity-check matrix H has a graphical representation known as the Tanner graph (Tanner, 1981). This graph is a bipartite one consisting of two kinds of nodes: bit (variable) nodes that correspond to the columns of H and check nodes that correspond to its rows. A bit node is connected to a check node by an edge if the value of the intersection of the column and the row corresponding to these nodes is equal to 1. A cycle in a bipartite graph is a set of edges that forms a continuous path starting with a node and returning to the same node without going through an edge more than once. The number of these edges is called the length of the cycle, and the smallest length is the girth of the code. The Tanner graph contains neither cycles of length 2 nor cycles of odd length. Therefore, the girth of this graph is at least 4. The number of cycles in the Tanner graph is one of the parameters that affect the performance of LDPC codes, particularly short cycles (Wiberg, 1996), and more precisely the cycles of length 4 (Li et al., 2017; MacKay, 1999). In the belief propagation algorithm (BPA), the presence of cycles in a Tanner graph causes a loss of independence in the messages sent by the nodes of the graph (Bandi et al., 2011).Various methods for constructing LDPC codes without short cycles have been proposed, such as constructions based on array dispersion and masking (Xu et al., 2016), based on difference sets (Esmaeili & Javedankherad, 2012) and based on sub-matrix shifting (Fan & Xiao, 2006a). The structure of the matrices constructed by these methods increases the encoding complexity.

To solve this problem, we propose a method to construct 4-cycle-free parity-check matrices, for regular LDPC codes, with low encoding complexity. These matrices having w_r ones in each row and w_c ones in each column where w_r=2w_c. In this work, only the values of w_c, which are equal to 2, 3, 4 and 5, are taken into account.

MATERIALS AND METHODS

Regular LDPC Codes

An LDPC code is characterized by its parity-check matrix H. This code is called regular if its parity-check matrix contains a fixed number w_r of ones in each row and a fixed number w_c of ones in each column (Johnson, 2010, p. 38). The matrix H represents the parity-check matrix of a regular LDPC code.

$$H = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

The Tanner graph for the parity-check matrix H is shown in the Figure 1, where the dotted lines represent a 6-cycle.Since there are no smaller cycles in this graph; its girth is 6.

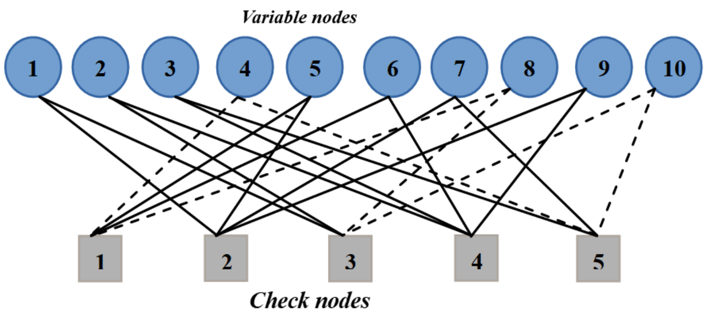

Figure 1. Tanner graph of the matrix H.

Circulant Matrices

A circulant matrix is a square matrix where each row is obtained by shifting the previous row one position to the right (Aldrovandi, 2001, p. 83), i.e,

$$H = \begin{pmatrix} h_0 & h_1 & h_2 & \cdots & h_{L-1} \\ h_{L-1} & h_0 & h_1 & \cdots & h_{L-2} \\ h_{L-2} & h_{L-1} & h_0 & \cdots & h_{L-3} \\ . & . & . & & . \\ . & . & . & & . \\ . & . & . & & . \\ h_1 & h_2 & h_3 & \cdots & h_0 \end{pmatrix}$$

In this paper, the values of h_i belong to the set $\{0,1\}$.

Construction of 4-Cycle-Free Matrices

The steps of the proposed method are as follows:

First step. In this step, we calculate all permutations P_{all} of the vector $V=[h_0\ h_1h_2\ldots h_{L-1}]$, which consists of *wc* ones and *L-wc* zeros, using the following law (Bóna, 2007, p.19):

$$P_{all} = \frac{L!}{w_c!(L-w_c)!} \qquad [1]$$

Using one of the cycle counting methods proposed by Fan and Xiao (2006b), Karimi and Banihashemi (2013), Li et al. (2015) or the permutations of the vector V which allow the construction of 4-cycle-free circulant matrices are found. We collect these permutations in a set S.

Based on simulation results, the length L is given by:

$$L \geq 2w_c\,(w_c - 1) + 1 \qquad [2]$$

For w_c equal to 2, 3 and 5, while for w_c equal to 4, L is given by:

$$L > 2w_c\,(w_c - 1) + 1 \qquad [3]$$

Second step. In this step, we took at each time a vector from the set S and constructed a circulant matrix that passed to the proposed algorithm detailed later on in the Proposed Algorithm section. The result of the algorithm is a set of all possible vectors $V_i' = [h_0' h_1' h_2' \ldots\ h_{L-1}']$ that allow us to use the matrix passed to the algorithm to construct an L×(L+1) matrix without 4-cycles as shown in the following matrix:

$$H = \begin{array}{c} \xleftrightarrow{\qquad\qquad L+1 \qquad\qquad} \\ \left(\begin{array}{ccccc|c} h_0 & h_1 & h_2 & \cdots & h_{L-1} & h'_0 \\ h_{L-1} & h_0 & h_1 & \cdots & h_{L-2} & h'_1 \\ h_{L-2} & h_{L-1} & h_0 & \cdots & h_{L-3} & h'_2 \\ . & . & . & \square & . & . \\ . & . & . & \square & . & . \\ . & . & . & \square & . & . \\ h_1 & h_2 & h_3 & \cdots & h_0 & h'_{L-1} \end{array}\right) \end{array} \updownarrow L$$

In general, not all vectors resulting from the algorithm allow the construction of circulant matrices without 4-cycles. For this reason, we chose only those that possessed the required characteristic of being able to construct matrices without 4-cycles. These vectors form a new set *S'*.

Third step. From the previous step, we noticed that each circulant matrix constructed by a vector of the set S had its own set *S'* which was found by using the proposed algorithm. In this last step, by concatenating each of these matrices with each circulant matrix, constructed by a vector of the set *S'*, we got matrices of the form:

$$H = \begin{array}{c} \xleftrightarrow{\qquad\quad L \qquad\quad}\ \xleftrightarrow{\qquad\quad L \qquad\quad} \\ \left(\begin{array}{ccccc|ccccc} h_0 & h_1 & h_2 & \cdots & h_{L-1} & h'_0 & h'_1 & h'_2 & \cdots & h'_{L-1} \\ h_{L-1} & h_0 & h_1 & \cdots & h_{L-2} & h'_{L-1} & h'_0 & h'_1 & \cdots & h'_{L-2} \\ h_{L-2} & h_{L-1} & h_0 & \cdots & h_{L-3} & h'_{L-2} & h'_{L-1} & h'_0 & \cdots & h'_{L-3} \\ . & . & . & \square & . & . & . & . & \square & . \\ . & . & . & \square & . & . & . & . & \square & . \\ . & . & . & \square & . & . & . & . & \square & . \\ h_1 & h_2 & h_3 & \cdots & h_0 & h'_1 & h'_2 & h'_3 & \cdots & h'_0 \end{array}\right) \end{array} \updownarrow L$$

The matrix H is an L×2L matrix without 4-cycles, which contains w_c ones in each column and $2w_c$ ones in each row.

$$S=\begin{Bmatrix} 000011, 000101, 000110, 001010, 001100, 010001, 010100, 011000, 100001, \\ 100010, 101000, 110000 \end{Bmatrix}.$$

Example 1. Let V be a vector of length L=6, which consists of 4 zeros and 2 ones.

Step 1. The number of permutations of V is P_{all}=15, only 12 permutations in the set S allow us to construct 4-cycle-free circulant matrices.

Step 2. We select a vector from the set S and construct a circulant matrix. This matrix is passed to the proposed algorithm. We find that each matrix passed to this algorithm has a set *S’* consisting of six vectors. For example; the circulant matrix constructed by the vector [100001] has the set:

S’={101000,100010,010100,010001,001010,000101}.

Step 3. For instance, we chose the vector [100001] with its own set *S’* (shown in the step 2) and we constructed the following 4-cycle-free circulant matrices:

$$H_1=\left(\begin{array}{cccccc|cccccc} 1&0&0&0&0&1&1&0&1&0&0&0\\ 1&1&0&0&0&0&0&1&0&1&0&0\\ 0&1&1&0&0&0&0&0&1&0&1&0\\ 0&0&1&1&0&0&0&0&0&1&0&1\\ 0&0&0&1&1&0&1&0&0&0&1&0\\ 0&0&0&0&1&1&0&1&0&0&0&1 \end{array}\right) H_2=\left(\begin{array}{cccccc|cccccc} 1&0&0&0&0&1&1&0&0&0&1&0\\ 1&1&0&0&0&0&0&1&0&0&0&1\\ 0&1&1&0&0&0&1&0&1&0&0&0\\ 0&0&1&1&0&0&0&1&0&1&0&0\\ 0&0&0&1&1&0&0&0&1&0&1&0\\ 0&0&0&0&1&1&0&0&0&1&0&1 \end{array}\right) H_3=\left(\begin{array}{cccccc|cccccc} 1&0&0&0&0&1&0&1&0&1&0&0\\ 1&1&0&0&0&0&0&0&1&0&1&0\\ 0&1&1&0&0&0&0&0&0&1&0&1\\ 0&0&1&1&0&0&1&0&0&0&1&0\\ 0&0&0&1&1&0&0&1&0&0&0&1\\ 0&0&0&0&1&1&1&0&1&0&0&0 \end{array}\right)$$

$$H_4=\left(\begin{array}{cccccc|cccccc} 1&0&0&0&0&1&0&1&0&0&0&1\\ 1&1&0&0&0&0&1&0&1&0&0&0\\ 0&1&1&0&0&0&0&1&0&1&0&0\\ 0&0&1&1&0&0&0&0&1&0&1&0\\ 0&0&0&1&1&0&0&0&0&1&0&1\\ 0&0&0&0&1&1&1&0&0&0&1&0 \end{array}\right) H_5=\left(\begin{array}{cccccc|cccccc} 1&0&0&0&0&1&0&0&1&0&1&0\\ 1&1&0&0&0&0&0&0&0&1&0&1\\ 0&1&1&0&0&0&1&0&0&0&1&0\\ 0&0&1&1&0&0&0&1&0&0&0&1\\ 0&0&0&1&1&0&1&0&1&0&0&0\\ 0&0&0&0&1&1&0&1&0&1&0&0 \end{array}\right) H_6=\left(\begin{array}{cccccc|cccccc} 1&0&0&0&0&1&0&0&0&1&0&1\\ 1&1&0&0&0&0&1&0&0&0&1&0\\ 0&1&1&0&0&0&0&1&0&0&0&1\\ 0&0&1&1&0&0&1&0&1&0&0&0\\ 0&0&0&1&1&0&0&1&0&1&0&0\\ 0&0&0&0&1&1&0&0&1&0&1&0 \end{array}\right)$$

To construct such matrices, it was enough to choose at each time a vector from the set S.Then, we passed the circulant matrix associated with this vector to the proposed algorithm so as to get a set of vectors *S’*. Matrices without 4-cycles would be constructed by concatenating the matrix passed to the proposed algorithm with each of the circulant matrices associated with the vectors of the set *S’*.

What about Large Matrices?

To construct large matrices, we simply added a number of zeros to the vectors obtained in first and second step in the Construction of 4-Cycle-Free Matrices section, in such a way that this number must be greater than or equal to L-1. These zeros had to be added, either on one side or on both sides of each vector as shown in the following example:

Example 2. The matrix H_2 of the example 1 consists of two circulant sub-matrices, the first sub-matrix is constructed by the vector [**1**0000**1**] while the second is constructed by the vector [**1**000**1**0]. To increase the size of H_2 without losing its characteristic (without

4-cycles), it is sufficient to add a number of zeros greater than or equal to 5 (because L=6) to each vector, either on one side or on both sides. To construct a 20×40 matrix, for example, we add to each vector 14 zeros. Then, the first vector becomes: [000000**100001**00000000] and the second becomes: [0000000000**100010**0000]. The matrix H_2 will be as follows:

$$H_2 = \left(\begin{array}{c|c}
0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,1\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,1\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,1\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,1\,0\,0 \\
0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,1\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,1\,0 \\
0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,1\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,1 \\
0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,1\,0\,0 & 1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,1\,0 & 0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0 \\
0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,1 & 0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0 \\
1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0 & 0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1 \\
0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0 & 1\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\
0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0 & 0\,1\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\
0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0 & 0\,0\,1\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1 & 0\,0\,0\,1\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\
1\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,1\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\
0\,1\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,1\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\
0\,0\,1\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\
0\,0\,0\,1\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,1\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0
\end{array}\right)$$

The Proposed Algorithm

Here are some notations associated with the proposed algorithm:

I: a matrix which contains the indices of ones of the matrices *H* that will be passed to the proposed algorithm. Each row of *I* represents the indices of ones in each column of *H*. The following example shows the form of the matrix*I*:

$$H = \begin{pmatrix} 1\,0\,0\,1 \\ 1\,1\,0\,0 \\ 0\,1\,1\,0 \\ 0\,0\,1\,1 \end{pmatrix}, \qquad I = \begin{pmatrix} 1\,2 \\ 2\,3 \\ 3\,4 \\ 1\,4 \end{pmatrix}$$

For the first column of *H*, the indices of ones are '1' and '2' so we will fill the first row of *I* by 1 and 2 and so on until the last row of *I*.

L: the length of the vector used to construct the circulant matrix *H*.

V': defined in second step in the Construction of 4-Cycle-Free Matrices section.

```
Algorithm: Find a set S'
Inputs: $H$, $L$ and $w_c$.
Output: Set $S'$.
 1: Construct the matrix $I$, $v = [1\ 2\ 3 \ldots L]$.
 2: for $i = 1 : L$ do
 3:    Copy all rows of $I$ that contain the value of $i$ in a vector $v_1$ without repeating the value of $i$.
 4:    Extract the components that exist in the vector $v$ and do not exist in the vector $v_1$, and replace
       the components of the vector $v_1$ by these extracted components.
```

Algorithm: Find a set S'

```
5:  Update $v_1$ by deleting all components of this vector that are ***less than*** the value of $i$.
6:  **if** $w_c = 2$ **then**
7:   **for** $j = 1$ : length of $v_1$ **do**
8:     Initialize the vector $V'$, of length $L$, to zero.
9:     $V'(i) = V'(v_1(j)) = 1$.
10:  **endfor**
11: **else**
12:  **for** $j = 1$ : length of $v_1$ **do**
13:     Copy all rows of $I$ that contain the value of $v_1(j)$ in a vector $v_2$ without repeating the value
        of $v_1(j)$.
14:     Extract the components that exist in the vector $v_1$ and do not exist in the vector $v_2$, and
        replace the components of the vector $v_2$ by these extracted components.
15:     Update $v_2$ by deleting all components of this vector that are ***greater than*** the value of $v_1(j)$.
16:     **if** $w_c = 3$ **then**
17:      **for** $k = 1$: length of $v_2$ **do**
18:       Initialize the vector $V'$, of length $L$, to zero.
19:       $V'(i) = V'(v_1(j)) = V'(v_2(k)) = 1$.
20:      **endfor**
21:     **else**
22:      **for** $k = 1$: length of $v_2$ **do**
23:         Copy all rows of $I$ that contain the value of $v_2(k)$ in a vector $v_3$ without repeating the
            value of $v_2(k)$.
24:         Extract the components that exist in the vector $v_2$ and do not exist in the vector $v_3$, and
            replace the components of the vector $v_3$ by these extracted components.
25:         Update $v_3$ by deleting all components of this vector that are ***greater than*** the value of
            $v_2(k)$.
26:         **if** $w_c = 4$ **then**
27:          **for** $l = 1$: length $v_3$ **do**
28:           Initialize the vector $V'$, of length $L$, to zero.
29:           $V'(i) = V'(v_1(j)) = V'(v_2(k)) = V'(v_3(l)) = 1$.
30:          **endfor**
31:         **elseif** $w_c = 5$ **then**
32:          **for** $l = 1$: length of $v_3$ **do**
33:           Copy all rows of $I$ that contain the value of $v_3(l)$ in a vector $v_4$, without repeating the
              value of $v_3(l)$.
34:           Extract the components that exist in the vector $v_3$ and do not exist in the vector $v_4$, and
              replace the components of the vector $v_4$ by these extracted components.
35:           Update $v_4$ by deleting all components of this vector that are ***greater than*** the value of
              $v_3(l)$.
```

Algorithm: Find a set S'

```
36:              for $m = 1$ : length $v_4$ do
37:                Initialize the vector $V'$, of length $L$, to zero.
38:                $V'(i) = V'(v_1(j)) = V'(v_2(k)) = V'(v_3(l)) = V'(v_4(m)) = 1$.
39:              endfor
40:            endfor
41:          endif
42:        endfor
43:      endif
44:    endfor
45:  endif
46: end for
```

RESULTS AND DISCUSSION

Results of the Proposed Method

In this section, we present the results of the proposed method. First, we calculated all permutations P_{all} of a vector V of length L.Then, we chose among these permutations only those that allowed the construction of matrices without 4-cycles (P_{free}). Finally, a number of 4-cycle-free matrices $H_{(L\times2L)}$ was found according to the length L. These operations would be repeated for w_c equal to 2, 3, 4 and 5.

Table 1 shows the number of matrices $H_{(L\times2L)}$ according to the length L, for w_c equal to 2. From the results in this table we note that the number of 4-cycle-free matrices increases proportionally to the length L. When L is odd, all permutations of V allow constructing matrices without 4-cycles, unlike if L is even. Tables 2, 3 and 4 show the number of matrices $H_{(L\times2L)}$ for w_c equal to 3, 4 and 5, respectively, and for some values of L. The results in these tables show that the number of 4-cycle-free matrices varies according to the value of L, but not on a regular basis.

Table 1

Number of 4-cycle-free matrices for w_c=2

L	P_{all}	P_{free}	$H_{L\times2L}$
≤4	/	/	0
5	10	10	50
6	15	12	72
7	21	21	294
8	28	24	384
9	36	36	972
10	45	40	1200
15	105	105	9450
20	190	180	28800

Table2

Number of 4-cycle-free matrices for w_c=3

L	P_{all}	P_{free}	$H_{L\times 2L}$
≤12	/	/	0
13	286	208	1352
14	364	196	0
15	455	360	12600
16	560	352	10240
17	680	544	46240
18	816	540	46656
19	969	798	138624
20	1140	800	137600

Table3

Number of 4-cycle-free matrices for w_c=4

L	P_{all}	P_{free}	$H_{L\times 2L}$
≤25	/	/	0
26	14950	4888	5408
27	17550	8748	104976
28	20475	7896	75264
29	23751	12180	518056
30	27405	11040	590400

Table4

Number of 4-cycle-free matrices for w_c=5

L	P_{all}	P_{free}	$H_{L\times 2L}$
≤40	/	/	0
41	749398	165968	26896
42	850668	140280	0
43	962598	234780	310632

Encoding Complexity Comparison

In this subsection, we compared encoding complexity of the proposed method with the methods proposed by Esmaeili and Javedankherad (2012); Fan and Xiao (2006a); MacKay (2005) and Xu et al.(2016). For that we used the encoding method proposed by Dutta and Pramanik (2015) to calculate the number of permutations needed to permute the parity-check matrix into the approximate UPPER triangular format (AUT). These permutations would be saved in a vector in order to apply the inverse permutation to each code word before it was transmitted. It is important to note that the encoding complexity increases as the number of permutations increases.

The 4-cycle-free parity-check matrices used in this comparison are as follows:

1. $V_1(9,18,26,28,32) = V_2(1,6,19,30,31) = 1$ (Size:102×204);
2. $V_1(16,17,19,25) = V_2(1,6,13,17) = 1$ (Size:124×248);
3. $V_1(12,16,18) = V_2(1,6,14) = 1$ (Size:189×378);
4. $V_1(22,31,39,41,45) = V_2(1,2,13,26,31) = 1$ (Size:408×816);
5. $V_1(16,18,19) = V_2(1,7,16) = 1$ (Size:432×864);
6. $V_1(1,6,10) = V_2(1,9,15) = 1$ (Size:765×1530);
7. $V_1(15,21,23,24) = V_2(1,6,16,20) = 1$ (Size:1020×2040);
8. $V_1(13,17,20) = V_2(1,2,19) = 1$ (Size:1080×2160);
9. $V_1(17,19,20) = V_2(1,6,20) = 1$ (Size:1320×2640);
10. $V_1(15,19,20) = V_2(1,10,18) = 1$ (Size:2000×4000);
11. $V_1(17,23,25,26) = V_2(1,5,12,17) = 1$ (Size:4000×8000);
12. $V_1(5,8,14) = V_2(1,3,16) = 1$ (Size:4320×8640);
13. $V_1(1,6,10) = V_2(1,11,14) = 1$ (Size:8000×16000);
14. $V_1(1,7,12) = V_2(1,10,13) = 1$ (Size:8640×17280).

For the proposed method:

For the method proposed by MacKay (2005):

1. 204.55.187(Size:102×204);
2. 816.3.174(Size:408×816);
3. 4000.2000.3.243 (Size:2000×4000);
4. 8000.4000.4.484 (Size:4000×8000).

For the method proposed by Xu et al. (2016):

By using the exponent matrix and masking matrix below, we constructed the following matrices by changing the identity matrix sizes:

1. Size of the identity matrix:330(Size of the resultant matrix:1320×2640);
2. Size of the identity matrix:500(Size of the resultant matrix: 2000×4000);
3. Size of the identity matrix:1000(Size of the resultant matrix: 4000×8000);
4. Size of the identity matrix:2000(Size of the resultant matrix:8000×16000).

$$P = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 23 & 67 & 83 & 117 & 142 & 158 & 206 & 249 \\ 46 & 134 & 166 & 234 & 284 & 316 & 82 & 168 \\ 69 & 201 & 249 & 21 & 96 & 144 & 288 & 87 \end{bmatrix}; \; M = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

For the method proposed by Esmaeili and Javedankherad (2012):

1. Difference set:(31,6,1),F32 : (Size of the resultant matrix:124×248);
2. Difference set:(31,6,1),F64 : (Size of the resultant matrix:189×378);
3. Difference set:(133,12,1),F256 : (Size of the resultant matrix:765×1530);
4. Difference set:(133,12,1),F256 : (Size of the resultant matrix:1020×2040);

For the method proposed by Fan and Xiao (2006a):

1. v=6,p=4 : (Size of the resultant matrix:432×864);
2. v=6,p=10 : (Size of the resultant matrix:1080×2160);
3. v=6,p=40 : (Size of the resultant matrix:4320×8640);
4. v=6,p=80 : (Size of the resultant matrix:8640×17280);

Table5

Comparison between the proposed and the first method

Methods	Matrices sizes			
	102 × 204	408 × 816	2000 × 4000	4000 × 8000
Proposed	14	07	09	09
Mackay (2005)	30	144	1900	3509

Table6

Comparison between the proposed and the second method

Methods	Matrices sizes			
	1320 × 2640	2000 × 4000	4000 × 8000	8000 × 16000
Proposed	08	09	09	10
Xu, Feng, Luo and Bai (2016)	704	1175	2321	4690

Table7

Comparison between the proposed and the third method

Methods	Matrices sizes			
	124 × 248	189 × 378	765 × 1530	1020 × 2040
Proposed	06	08	00	10
Esmaeili and Javedankherad (2012)	110	169	714	988

Table8

Comparison between the proposed and the fourth method

Methods	Matrices sizes			
	432×864	1080×2160	4320×8640	8640×17280
Proposed	06	11	09	12
Fan and Xiao (2006a)	402	1024	4213	8482

Tables 5, 6, 7 and 8 show some 4-cycle free parity-check matrices, constructed by the proposed method and the four methods mentioned above, and their corresponding number of permutations required to bring these matrices into the approximate upper triangular format. From the results shown in these tables, we can observe that the proposed method offered a gain of 16 permutations in the 102×204 matrix as a minimum gain, and 8470 as a maximum in the 8640×17280 matrix. For the 765×1530 matrix constructed by the proposed method, the number of permutations was zero, which meant that the associated encoder did not need to apply the inverse permutation to each code word before the transmission.

The proposed method gives the smallest number of permutations among the other methods for all matrices; this would imply that it offers the lowest encoding complexity. Furthermore, it provides more flexibility to construct matrices of different sizes than the other methods.

Performance of the Proposed Method

In this subsection we compared the performance, in terms of the bit error rate (BER), of two LDPC codes with parity-check matrices constructed by the proposed method and two LDPC codes with parity check-matrices constructed by MacKay's method (MacKay, 2005).The matrices constructed by MacKay are as follows:

1. 96.33.966 (w_c=3,w_r=6, size:48×96).
2. 96.44.443 (w_c=4,w_r=8, size:48×96).

Each of the matrices, constructed by the proposed method, is a concatenation of two matrices H_1 and H_2, where H_1 and H_2 are the circulant matrices constructed by the vectors V_1 and V_2 respectively. The matrices used in the simulation are defined by:

1. V_1(1,7,12)=V_2(1,10,13)=1 (w_c=3,w_r=6, size:48×96).

2. V_1 (17,23,25,26)=V_2 (1,5,12,17)=1 (w_c=4,w_r=8, size:48×96).

The encoding process can be realized using one of the methods proposed by Dutta and Pramanik (2015) or Richardson and Urbanke (2001). The encoded bits were modulated using binary phase shift keying ***(BPSK)*** before being sent over the additive white Gaussian noise ***(AWGN)*** channel. We used the ***sum-product*** algorithm (Johnson, 2010) for the decoding with a maximum of 10 iterations.

Figure 2 provides a performance comparison between the proposed method and MacKay's method for w_c=3 and 4. According to this figure, the proposed method slightly

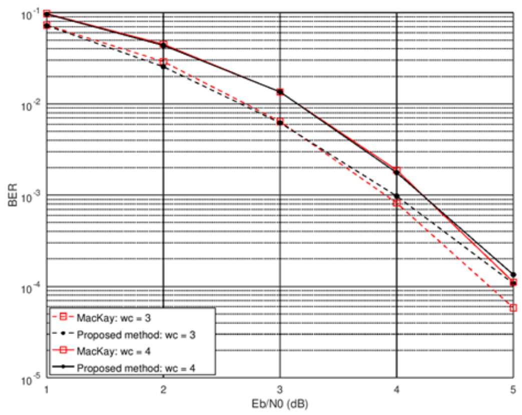

Figure 2. Performance comparison of the proposed method and that of MacKay

outperforms MacKay's method at lower SNRs, while at higher SNRs, MacKay's method performs better than the proposed method.

CONCLUSION

In this paper, we propose an original method capable of constructing 4-cycle-free parity-check matrices that can be used with regular LDPC codes. These matrices have full rank for odd-column-weight (3 and 5) and just one redundant row for even-column-weight (2 and 4). Simulation results show that the proposed method offers both an enormous reduction in encoding complexity and a very large number of 4-cycle-free parity-check matrices of different sizes. Although MacKay's method outperforms, in terms of BER, the proposed method in the high SNR region, the matrices constructed by the proposed method can be adopted in many practical applications due to their hardware implementation using simple shift registers.

ACKNOWLEDGEMENT

The first author would like to thank Dr. Abdellah Kaddai; Department of Electronics, Hassiba Ben bouali University at Chlef, Algeria; and Dr. Samir Bendoukha; Electrical Engineering Department College of Engineering at Yanbu Taibah University, Saudi Arabia; for their support and encouragement.

REFERENCES

Aldrovandi, R. (2001). *Special matrices of mathematical physics: Stochastic, circulant and bell matrices.* Singapore: World Scientific.

Bandi, S., Tralli, V., Conti, A., &Nonato, M. (2011). On girth conditioning for low-density parity-check codes. *IEEE Transactions on Communications, 59*(2), 357-362.

Bóna, M. (2007). *Introduction to enumerative combinatorics.* New York, NY: McGraw-Hill.

Dutta, A., & Pramanik, A. (2015, March 19-20). Modified approximate lower triangular encoding of LDPC codes. In *International Conference on Advances in Computer Engineering and Applications* (pp. 364-369). Ghaziabad, India.

Esmaeili, M., & Javedankherad, M. (2012). 4-Cycle free LDPC codes based on difference sets. *IEEE Transactions on Communications, 60*(12), 3579-3586.

Fan, J., & Xiao, Y. (2006a, November 6-9). A design of LDPC codes with large girth based on the sub-matrix shifting. In *IET International Conference on Wireless, Mobile and Multimedia Networks* (pp. 1-4). Hang Zhou, China.

Fan, J., & Xiao, Y. (2006b, November 16-20). A method of counting the number of cycles in LDPC codes. In *8th International Conference on Signal Processing* (pp. 2183-2186). Beijing, China.

Gallager, R. G. (1962). Low-density parity-check codes. *IRE Transactions on Information Theory, 8*(1), 21-28.

Johnson, S. J. (2010). *Iterative error correction: Turbo, low-density parity-check and repeat–accumulate codes.* New York, NY: Cambridge University Press.

Karimi, M., & Banihashemi, A. H. (2013). Message-passing algorithms for counting short cycles in a graph. I*EEE Transactions on Communications, 61*(2), 485-495.

Li, J., Lin, S., & Abdel-Ghaffar, K. (2015, June 14-19). Improved message-passing algorithm for counting short cycles in bipartite graphs. In *IEEE International Symposium on Information Theory* (pp. 416-420). Hong Kong, China.

Li, J., Lin, S., Abdel-Ghaffar, K., Ryan, W., & Costello, D. J. (2017). *LDPC code designs, constructions and unification.* New York, NY: Cambridge University Press.

MacKay, D. J. C. (1999). Good error-correcting codes based on very sparse matrices. *IEEE Transactions on Information Theory, 45*(2), 399-431.

MacKay, D. J. C. (2005). *Encyclopedia of sparse graph codes.* Retrieved April 11, 2002, from http://www.inference.phy.cam.ac.uk/mackay/codes/data.html.

MacKay, D. J. C., & Neal, R. M. (1995). Good codes based on very sparse matrices. In 5*th IMA Conference on Cryptography and Coding* (pp. 100-111). Berlin, Germany: Springer.

Richardson, T. J., & Urbanke, R. L. (2001). Efficient encoding of low-density parity-check codes. *IEEE Transactions on Information Theory, 47*(2), 638-656.

Tanner, R. M. (1981). A recursive approach to low complexity codes. I*EEE Transactions on Information Theory, 27*(5), 533-547.

Wiberg, N. (1996). *Codes and Decoding on General Graphs* (PhD dissertation). University of Linköping, Sweden.

Xu, H., Feng, D., Luo, R., &Bai, B. (2016). Construction of quasi-cyclic LDPC Codes via masking with successive cycle elimination. *IEEE Communications Letters, 20*(12), 2370-2373.