A Novel Algorithm for Removing Cycles in Quasi-Cyclic LDPC Codes

Keke Liu, Zesong Fei, Jingming Kuang, Xiangming Li
School of Information and Electronics
Beijing Institute of Technology
Beijing, P. R. China
liuke498@126.com, feizesong@bit.edu.cn

Abstract—In this paper, an algorithm for removing cycles in quasi-cyclic (QC) LDPC codes is presented. This algorithm can ensure that the code after cycle removal process preserves the quasi-cyclic structure and significantly improves the flexibility in parameter selection (such as the length of the code) of algebraic constructions of QC-LDPC codes. Besides, it has far lower computational complexity than the existing cycle removal algorithm. Experimental results show that this algorithm is very effective in improving the performance of the QC-LDPC codes and can construct code which has better performance than the corresponding binary LDPC code based on IEEE 802.16e standard.

Keywords—Quasi-Cyclic LDPC codes; cycles; large girth; combination

I. INTRODUCTION

LDPC codes, first discovered by Gallager [1] in 1962 and rediscovered by Mackay in 1990’s [2,3], have caught the attention of Channel Coding Field due to its near-capacity performance, and the constructions of LDPC codes has been a focal point of research in LDPC codes. Recently, constructions of Quasi-Cyclic(QC) LDPC codes has caused wide concern because QC-LDPC codes enable linear encoding complexity[4] as opposed to the quadratic complexity in computer-generated random LDPC codes [5].

The girth is one of key factors affecting the performance of an LDPC code. Although recent research indicates that a large girth might result in a relatively higher error floor, but more simulation results[6] has demonstrated that a relatively larger girth helps to improve the bit-error rate(BER) performance within the range of $10^{-2}$ to $10^{-6}$, and the block-error rate(BLER) within the range of $10^{-4}$ to $10^{-2}$, which is sufficient for practical applications of an LDPC code in many situations. Consequently the research into construction of an LDPC codes with relatively large girth(larger than 6) is still very valuable for practical application.

There have been several approaches, including direct construction approaches[7,15] and cycle removal based approaches[9], to remove short cycles in tanner graph and construct LDPC codes with large girth. However, these approaches are either not sufficiently flexible in the code parameter selection due to the constraint on girth, or unable to preserve the quasi-cyclic structure thereby increasing the encoding complexity.

In this paper, we present a novel algorithm aimed at cycle removal of quasi-cyclic LDPC(QC-LDPC) code. Compared to the existing methods for cycle removal or constructing LDPC codes with large girth, our new algorithm has the following advantages: 1) The proposed algorithm ensures that the code after cycle removal preserves the quasi-cyclic structure, which is essential for low-complexity encoding, but not taken into consideration in some other cycle removal methods, such as the algorithms proposed in [9]. 2) The proposed algorithm significantly enhances the flexibility in parameter selection of algebraic constructions of QC-LDPC codes, especially non-binary QC-LDPC codes constructions. 3) The proposed algorithm makes good use of the structure of the parity-check matrix of QC-LDPC code so that it has far lower computational complexity than the existing cycle removal method with respect to QC-LDPC codes.

II. QUASI-CYCLIC LDPC CODES BASED ON CIRCULANT PERMUTATION MATRICES

The parity-check matrix of a QC-LDPC code from circulant permutation matrices[10] can be represented by

$$H = \begin{bmatrix}
I(p_{1,1}) & I(p_{1,2}) & \cdots & I(p_{1,L}) \\
I(p_{2,1}) & I(p_{2,2}) & \cdots & I(p_{2,L}) \\
\vdots & \vdots & \ddots & \vdots \\
I(p_{J,1}) & I(p_{J,2}) & \cdots & I(p_{J,L})
\end{bmatrix}$$

where for $1 \leq j \leq J - 1$, $1 \leq \ell \leq L - 1$, $I(p_{j,\ell})$ represents the circulant permutation matrix obtained by shifting the $p \times p$ identity matrix $I$ to the right by $p_{j,\ell}$ places where $0 \leq p_{j,\ell} < p$. If we replace some circulant permutation matrices by $p \times p$ zero matrices, we obtain a family of QC-LDPC codes with various degree distributions. For the sake of convenience, we define $p \times p$ zero matrix as $I(-\infty)$. It is obvious that for given $J, L$ and $p$, the parity-check matrix $H$ is characterized by the following matrix called base matrix:

$$P = \begin{bmatrix}
p_{1,1} & p_{1,2} & \cdots & p_{1,L} \\
p_{2,1} & p_{2,2} & \cdots & p_{2,L} \\
\vdots & \vdots & \ddots & \vdots \\
p_{J,1} & p_{J,2} & \cdots & p_{J,L}
\end{bmatrix}$$
For a given nonzero element in a circulant permutation matrices, we define its adjacent nonzero element as the nonzero element in the row below it if the given nonzero element is in the last row of the circulant permutation matrix, its adjacent nonzero element is the one in the first row. For QC-LDPC codes, if several nonzero elements in the parity-check matrix constitute a cycle of length $2l$ (called cycle 1), we readily know that their respective adjacent nonzero element also constitute a cycle of length $2l$ (called cycle 2). We called cycle 2 the adjacent cycle of cycle 1 and define the mapping from a cycle to its adjacent cycle as adjacent cycle mapping. Based on the fact above, we conclude that in QC-LDPC code, the cycles exist in the form of group, called cycle-group. Given a cycle in a cycle-group, we can obtain any other cycle in this cycle-group by exerting adjacent cycle mapping on this cycle several times. That is to say, in a cycle-group, one cycle can completely characterize all the cycles in this cycle-group. This conclusion is peculiar to QC-LDPC code and lays the theoretical foundation for our new low-complexity cycle removal algorithm. Additionally, if one or more elements in a given cycle is included in a sub-matrix $A$, we say that this cycle passes sub-matrix $A$.

III. THE LOW-COMPLEXITY CYCLE REMOVAL ALGORITHM FOR QC-LDPC CODES

A. The Low-Complexity Cycle Removal Algorithm

Our proposed cycle removal algorithm can be divided into two basic steps: the step of searching cycles and the step of removing cycles. We can divide the parity-check matrix $H$ into several sections and each section is a row of circulant permutation matrices in $H$. For $1 \leq i \leq J$, we construct a cycle-set $C(i)$ formed by all the cycles that pass the first row of the $i$th section of $H$ but don’t pass any row above the $i$th section of $H$, in other words, $C(i)$ is formed by cycles which pass the first row of the $i$th section and are included in the last $J-i-1$ sections of $H$. Then we get a series of cycle-sets, denoted by $C(1), C(2), \ldots, C(J)$, which have the following properties: 1) any two cycle-sets don’t have any cycles in common; 2) any two cycles, whether belong to the same cycle-set or not, belong to the different cycle-groups; 3) For any given cycle-group, there exists a cycle-set in which we can find a cycle in the given cycle-group. These three properties indicate that each cycle in a cycle-set corresponds to a cycle-group. Let $C$ be the union of $C(1), C(2), \ldots, C(J)$. The elements in $C$ are all the cycles we need to search.

For a given check node $r$ in the Tanner graph, we construct a tree which has the following properties: 1) $r$ is the root node of the tree; 2) For an arbitrary node of the tree, its child nodes are all the nodes connected with it in the Tanner graph except its parent node. We call such a tree the check node $r$’s neighborhood tree, as shown in Fig.1(b). If the check node $r$ is included in a cycle of length $2l$ (the cycle consists of the red edges in Fig.1(a) and we take $l = 2$ for example), the two nodes in this cycle connected with $r$ in the Tanner graph can be both regarded as the node at the 1$\text{st}$ level and the node at the $(2l-1)$th level of $r$’s neighborhood tree, hence $r$ must appear in the $2l$th level of its neighborhood tree for two times, and there must exists two path of length $2l$ in the neighborhood tree of $r$ corresponding to the cycle, which start from the root node (the two paths are shown in Fig.1(b) by red edges). These two paths represent the same cycle, so we only need to select one path between them, called cycle-path, such that in this path the variable node at the 1$\text{st}$ level has a smaller label number than the node at the $(2l-1)$th level of $r$’s neighborhood tree (the thicker one in the two paths). Now we propose the algorithm for searching cycles in the set $C$.

The Proposed Algorithm for Searching Cycles

Step 1. Set a counter $t$ and let $t = 0$.

Step 2. Construct a tanner graph corresponding to sub-matrix consisting of the last $J-t$ sections of $H$ (call it $G(t)$).

Step 3. Select the check node corresponds to the first row of the last $J-t$ sections of $H$, and construct its neighborhood tree corresponding to $G(t)$ (call it $T(t)$).

Step 4. Identify all the cycle-paths in $T(t)$, and the cycles corresponding to the cycle-paths form the set $C(t+1)$.

Step 5. If $t \neq J-1$, $t = t + 1$ and go to Step 2; else STOP.
In the above algorithm, Step 2 and 3 ensures that any two sets of $C(1), C(2), \ldots, C(J)$ don’t have any cycles in common and that there is a one-to-one correspondence between all the found cycle-paths and the cycles in $C$.

Before we propose our algorithm for removing cycles, we first label all the cycles $C$ and all the $s = J \cdot L$ circulant permutation matrices (including all-zero matrices) in $H$, then we introduce an auxiliary binary matrix $M$ with size of $c \times s$, whose rows correspond to all the $c$ cycles in $C$ and whose columns correspond to all the $s$ circulant permutation matrices in $H$. The element at the intersection of the $i$th row and the $j$th column of $L$ is equal to ‘1’ if and only if the $j$th column of $L$ passes the $j$th circulant permutation matrix in $H$. Now we discuss the algorithm for removing cycles in QC-LDPC codes.

**The Proposed Algorithm for Removing Cycles**

**Step 1.** Construct the auxiliary binary matrix $M$.

**Step 2.** Find the number $i$, such that $i$th column of $M$ is the column with the largest column weight of $M$.

**Step 3.** Find the circulant permutation matrix in $H$ corresponding to $i$th column of $M$, and replace it by zero matrix with the same size. (We call this step ‘delete the circulant permutation matrix’)

**Step 4.** For each $j$ such that $M_{j,i} = 1$, replace all the elements ‘1’ in $j$th column of $M$ by zeros.

**Step 5.** If $M$ is not an all-zero matrix, go to Step 2; else STOP.

Now we explain this cycle removal algorithm in details. Step 1 is the initialization of our algorithm and the binary matrix $M$ characterize the cycle distribution of the quasi-cyclic parity-check matrix $H$. The purpose of Step 2 and Step 3 is to find out and delete the circulant permutation matrix that is passed by the most cycles in $C$. In fact, cycle removal process will inevitably reduce the column weight of the parity-check matrix, thereby reducing the minimum distance of the code. If the code’s minimum distance is too small, it might have a relatively higher error floor. Consequently, in order to prevent higher error floor, we need to achieve the girth we need by deleting as few circulant permutation matrices as possible. While step 2 can ensure that the number of deleted circulant permutation matrices is minimized. Since the circulant permutation matrix selected in Step 3 is deleted, the cycles in $C$ passing it are broken. Step 4 is to modify the auxiliary binary matrix $M$ in order to make it correspond to the parity-check matrix $H$ after Step 3. If all the cycles in $C$ are removed, we terminate the cycle removal process, otherwise we continue the cycle removal process by going back to step 2 until all the cycles are removed.

**B. The Analysis of Complexity of Our Cycle Removal Algorithm**

From the detail of our algorithm, we know that there are two reasons why our algorithm has low complexity compared to the conventional cycle removal algorithm. 1) our algorithm don’t need to identify all the cycles in $H$, instead we only need to identify one cycle in each cycle-group, which can totally characterize all the cycles. 2) The construction of the series of sets $C(1), C(2), \ldots, C(J)$ successfully avoid repetitive search of one cycle.

The computational complexity is mainly decided by the process of searching cycles because in this process we have to traverse almost all the path of length $2l$ in the neighborhood tree. According to the process of searching paths in the tree, we know that the computational complexity of searching paths in the neighborhood tree is linearly proportional to the number of branches of the neighborhood tree from the root to the $2l$th level. We assume that the column and row weight of $H$ are $J$ and $L$, respectively. In other words, none of the circulant permutation matrices $H$ is all-zero matrix. For a regular tanner graph with check node degree $\rho$ and variable node degree $\gamma$, although there are $\rho$ branches connecting the root check node to the variable nodes of the 1st level, we don’t need to search in the paths including the last branch connecting to the root node because in the cycle-path the 1st variable node can’t have the largest label number in the path. Hence the total number of branches of the neighborhood tree from the root to the $2l$th level can be calculated as below:

$$N(\rho, \gamma, 2l) = \sum_{i=0}^{l} (\rho - 1)^i (\gamma - 1)^{l-i} + \sum_{i=0}^{l} (\rho - 1)^i (\gamma - 1)^l - \sum_{i=0}^{l} (\rho - 1)^i (\gamma - 1)^l$$

$$= \gamma \sum_{i=0}^{l} (\rho - 1)^i (\gamma - 1)^{l-i} - \gamma (\rho - 1)^{l+1} (\gamma - 1)^l (\rho - 1)(\gamma - 1)^{-1}$$

\[ (3) \]

In (3), the first term corresponds to the branches from $2i$th level to $(2i+1)$th level $(i = 0, 1, \ldots, l)$ of the neighborhood tree, and the second term corresponds to the branches from $(2i-1)$th level to $2i$th level $(i = 1, 2, \ldots, l)$ of the neighborhood tree. In our algorithm for searching cycles, the degree of both the check node and variable node in the Tanner graph $G(t)$ decrease as $t$ increases. In fact, for the tanner graph $G(t)$, the total number of branches of the corresponding neighbor tree from the root to the $2l$th level is $N(L - t, J - t, 2l)$ for the reason that as the counter $t$ increases by 1, both the degree of check node and variable node decrease by 1. Hence in the whole process of searching cycles in our algorithm, the number of the branches in all the $t$ tanner graphs $G(0), \ldots, G(J - 1)$, denoted by $Q(2l)$ can be calculated as below:

$$Q(2l) = \sum_{i=0}^{l} (L - t)^i (J - t)^{l-i}$$

\[ (4) \]

For comparison, we also provide complexity analysis of the cycle removal approach proposed in [10]. In this approach, the process of identifying all the ‘elementary paths’ is equivalent to the process of constructing the neighborhood tree of each check node in the Tanner graph corresponding to $H$ and identifying all the cycle-paths. For the reason that this approach need to
identify all the cycles in $H$, the total number of the branches in the process of searching cycles denoted by $Q'(2l)$ amounts to:

$$Q'(2l) = p^J \cdot J \sum_{i=1}^{l} (L-1)^{i-1} (J-1)^{i-1}$$

(5)

where $p^J$ is the number of rows of the parity-check matrix $H$. Through comparison between (4) and (5) we know that $Q'(2l) \gg Q(2l)$ as long as $p$ is not too small. For example, let $p = 63, J = 4, L = 8$ and $l = 2$, then we get $Q(2l) = 914$, while $Q'(2l) = 155232$, almost 165 time as large as $Q(2l)$. Consequently, we see that our algorithm can greatly reduce the complexity of cycle removal. Table I shows the complexity comparison between our algorithm and the algorithm proposed in [9] with some specific pairs of column and row weights of $H$ when removing cycles of length 6.

<table>
<thead>
<tr>
<th>Size of circulant matrices $p$</th>
<th>Column weight $J$</th>
<th>Row weight $L$</th>
<th>Complexity of our algorithm</th>
<th>Complexity of algorithms in [9]</th>
</tr>
</thead>
<tbody>
<tr>
<td>127</td>
<td>4</td>
<td>12</td>
<td>63688</td>
<td>25101296</td>
</tr>
<tr>
<td>63</td>
<td>3</td>
<td>6</td>
<td>1836</td>
<td>314685</td>
</tr>
<tr>
<td>31</td>
<td>4</td>
<td>16</td>
<td>163176</td>
<td>15408240</td>
</tr>
</tbody>
</table>

C. Simulation Results

In all the examples to be given in this paper, we assume BPSK transmission over AWGN channel. We adopt SPA decoding for binary LDPC codes and FFT-QSPA decoding for non-binary LDPC codes with the maximum number of iterations set to 50.

Example 1: We choose two QC-LDPC codes, one code (called Code C1) is a (4113, 2288) code with rate 0.556, constructed in [12], the other code (called code C2) is a 64-ary (756, 519) QC-LDPC codes with rate 0.555, and a new (756, 504) 64-ary QC-LDPC code with rate 0.556, and a new (756, 504) 64-ary QC-LDPC code with rate 0.556, and a new (756, 504) 64-ary QC-LDPC code with rate 0.556. Our algorithm respectively in these two codes, we obtain a new binary (4113, 2285) QC-LDPC code with rate 0.555 and girth at least 10 (called code C1'), and a new (756, 504) 64-ary QC-LDPC code with rate 0.667 and girth at least 8 (called code C2'). The performances all the codes are shown in Fig.2, from which we see that at BER of $10^{-4}$, C1’ with girth 10 outperforms C1 with girth-8 by around 0.4dB, and that taking off the 0.1dB Shannon limit gap between C2 and C2’, C2’ with girth 8 outperforms C2 with girth 6 by around 0.4dB.

IV. CONSTRUCTION OF NON-BINARY QUASI-CYCLIC LDPC CODES BY COMBINATION OF PARITY-CHECK MATRICES USING CYCLE REMOVAL ALGORITHM

Algebraic tools, such as finite fields and combinatorial design have shown their effectiveness in constructing Quasi-Cyclic LDPC Codes. Taking finite fields for example, many classes of finite-field based QC-LDPC codes have been proposed, such as codes based on additive subgroups, multiplicative subgroups, primitive elements, affine mappings [11-13], multiplicative inverses [14] and so forth.

![Figure 2. The performances of codes in Example 1](image-url)
algorithm to remove the short cycles to obtain desirable performance.

The combination of several quasi-cyclic parity-check matrices enables constructing longer non-binary QC-LDPC codes over fields of small size.

Example 2: In this example we construct a QC-LDPC code over \(GF(2^4)\). First we construct two parity-check matrix \(H_1, H_2\), respectively by multiplicative subgroups[13] and multiplicative inverses[14] in \(GF(2^4)\). Both \(H_1\) and \(H_2\) are \(15\times 15\) arrays of \(15\times 15\) circulant permutation matrices. We choose the \(8\times 12\) sub-array at the upper right corner of \(H_1\), denoted by \(H_1'\), and the \(8\times 12\) sub-array at the upper left corner of \(H_2\), denoted by \(H_2'\). Arranging \(H_1', H_2'\) in a row, we obtain an \(8\times 24\) array of \(15\times 15\) circulant permutation matrices denoted by \(H(0)\) containing cycles of length 4. Applying our cycle removal algorithms in \(H(0)\) for two times, we obtain two new parity-check matrix \(H(1)\) and \(H(2)\). The null space of \(H(1)\) gives a 16-ary (360,240) QC-LDPC code with girth 6, and the null space of \(H(2)\) gives a 16-ary (360,240) QC-LDPC code with girth 8. The performance of the two (360,240) QC-LDPC codes are shown in Fig.4. We see that at BER of \(10^{-6}\), the girth-8 (360,240) code outperforms girth-6 (360,240) code by 1.25dB. To prove the excellent performance of the constructed girth-8 code, we also include the performance of a binary (1440, 960) LDPC code constructed based on IEEE 802.16e standard, which has the same rate and equivalent binary code length as our girth-8 code. We see that at BER of \(10^{-6}\), our girth-8 (360,240) 16-ary code outperforms the binary (1440, 960) LDPC code based on IEEE 802.16e standard by 0.15dB, and that at FER of \(10^{-4}\), our girth-8 code outperforms the corresponding IEEE 802.16e code by 0.2dB.

Figure 3. The performances of codes in Example 2

V. Conclusions

In this paper, we have presented an algorithm for removing cycles in QC-LDPC codes. Taking full advantage of the structure of the parity-check matrix of QC-LDPC code, our algorithm has far lower computational complexity than the existing algorithm and preserves the quasi-cyclic structure after cycle removal process, which is very important for low-complexity encoding. Simulation results shows that this algorithm is very effective in improving the performance of the QC-LDPC codes and allows for constructing longer QC-LDPC codes with large girth and far more flexible parameter selection.

REFERENCES


