Abstract—In this paper, we consider how to partition the parity-check matrices (PCMs) to reduce the hardware complexity and increase decoding throughput for the row layered decoding of quasi-cyclic low-density parity-check (QC-LDPC) codes. First, we formulate the PCM partitioning as an optimization problem, which targets to minimize the maximum column weight of each layer while maintaining a block cyclic shift property among different layers. As a result, we derive all the feasible solutions for the problem and propose a tight lower bound $\omega_{LB}$ on the minimum possible maximum column weight to evaluate a solution. Second, we define a metric called layer distance to measure the data dependency between consecutive layers and further illustrate how to identify the solutions with desired layer distance from those achieving the minimum value of $\omega_{LB} = 1$, which is preferred to reduce computation delay. Next, we demonstrate that up-to-now, finding an optimal solution for the optimization problem with polynomial time complexity is unachievable. Therefore, both enumerative and greedy partition algorithms are proposed instead. After that, we modify the quasi-cyclic progressive edge-growth (QC-PEG) algorithm to directly construct PCMs that have a straightforward partition scheme, which targets to minimize the maximum column weight of each layer. In general, there are two types of partition schemes in the literature [13], [14], [15], [16], [17], [18], [19], and [20], which update the messages in a sequential order by using the latest information. In particular, the row layered schedules in [8], [9], and [10] perform variable node (VN) updates while the column layered schedules in [11] and [12] perform variable node (VN) updates sequentially. As shown in [9], the layered schedules can achieve faster convergence speed with significantly reduced memory requirement compared to the flooding schedule. However, the conventional layered schedules, e.g., [8], [9], and [11], are initially developed to process only one row/column of the parity-check matrix (PCM) at each layer, which causes a reduction of decoding throughput.

To improve the throughput, partial parallelism has been introduced in [13], [14], [15], [16], [17], [18], [19], and [20] for quasi-cyclic LDPC (QC-LDPC) codes [21]. The idea is to process more rows/columns of the PCM in parallel at each layer. More specifically, to decode a QC-LDPC code with partial parallelism, the rows or columns of its PCM are partitioned into several groups/submatrices corresponding to different layers. At each layer, all rows or columns are processed in parallel, and the decoding conducts layer by layer. In general, there are two types of partition schemes in the literature. The first one, referred to by Type-1 partition scheme, is to set each natural QC row/column block of the PCM as a layer [13], [14], [15]. The second one, referred to by Type-2 partition scheme, is to select rows (resp. columns) from different QC row (resp. column) blocks to form a layer while maintaining a block cyclic shift property among different layers [16], [17], [18], [19], [20]. It was shown in [16], [17], [18], [19], and [20] that compared to the Type-1 partition scheme, the Type-2 partition scheme can make the hardware more efficiently reused between consecutive layers during the decoding process, leading to potentially higher communication and data storage systems [3], [4], for their capacity approaching performance with iterative message passing decoding [5]. To implement the decoding of LDPC codes, a schedule is required to determine the order of updating the messages. One well-known message-passing schedule is the flooding schedule [6], where all the variable-to-check (V2C) messages and the check-to-variable (C2V) messages are updated simultaneously and propagated along the edges in the Tanner graph [7]. However, it requires a large-scale parallelism, which leads to a high decoding complexity and memory requirement for hardware implementation.

To simplify hardware implementation and reduce the storage memories, the layered schedules were proposed in [8], [9], [10], [11], and [12], which update the messages in a sequential order by using the latest information. In particular, the row layered schedules in [8], [9], and [10] operate as a sequence of check node (CN) updates while the column layered schedules in [11] and [12] perform variable node (VN) updates sequentially. As shown in [9], the layered schedules can achieve faster convergence speed with significantly reduced memory requirement compared to the flooding schedule. However, the conventional layered schedules, e.g., [8], [9], and [11], are initially developed to process only one row/column of the parity-check matrix (PCM) at each layer, which causes a reduction of decoding throughput.

To improve the throughput, partial parallelism has been introduced in [13], [14], [15], [16], [17], [18], [19], and [20] for quasi-cyclic LDPC (QC-LDPC) codes [21]. The idea is to process more rows/columns of the PCM in parallel at each layer. More specifically, to decode a QC-LDPC code with partial parallelism, the rows or columns of its PCM are partitioned into several groups/submatrices corresponding to different layers. At each layer, all rows or columns are processed in parallel, and the decoding conducts layer by layer. In general, there are two types of partition schemes in the literature. The first one, referred to by Type-1 partition scheme, is to set each natural QC row/column block of the PCM as a layer [13], [14], [15]. The second one, referred to by Type-2 partition scheme, is to select rows (resp. columns) from different QC row (resp. column) blocks to form a layer while maintaining a block cyclic shift property among different layers [16], [17], [18], [19], [20]. It was shown in [16], [17], [18], [19], and [20] that compared to the Type-1 partition scheme, the Type-2 partition scheme can make the hardware more efficiently reused between consecutive layers during the decoding process, leading to potentially higher
decoding throughput, less memory consumption, and/or lower hardware implementation complexity. For example, as shown in [16], the Type-2 partition scheme results in a reduction in the memory and total clock cycles for convergence by 50%. Moreover, for the Type-2 partition scheme, the maximum column/row weight of each layer can be minimized to further reduce the hardware implementation complexity and increase decoding throughput. However, our partition codes, aiming to reduce hardware implementation complexity and increase decoding throughput, have not been systematically investigated, e.g., the optimal partition scheme to minimize the maximum column/row weight of each layer is not clear. With respect to this, our paper focuses on the Type-2 partition scheme.

More specifically, we systematically investigate the Type-2 partition scheme for the row layered decoding of QC-LDPC codes, aiming to reduce hardware implementation complexity and increase decoding throughput. However, our partition methods can be also applied to the column layered decoding. The main contributions of this paper are summarized below:

- We formulate the PCM partitioning as an optimization problem and propose the PCM partitioning principle to minimize the maximum column weight of each layer while maintaining a block cyclic shift property among different layers. Notably, we derive all feasible solutions (partition schemes) for the problems and propose a tight lower bound \( \omega_{LB} \) for the minimum possible maximum column weight to evaluate the quality of a solution.

- Among the solutions which achieve the minimum possible value of the lower bound \( \omega_{LB} = 1 \), we observe that, they may have different computation delay due to the data dependency issue between consecutive layers. We define a metric, called layer distance, to quantify the above issue. We further illustrate how to identify the solutions with desired layer distance from those solutions that achieve \( \omega_{LB} = 1 \).

- We prove that finding an optimal solution to the optimization problem is not easier than solving the classic graph theory problem of finding k-cliques in k-partite graphs [22], [23]. Since the latter problem does not have polynomial time complexity solutions up-to-now, we propose an enumerative algorithm and a greedy algorithm for solving the optimization problem as alternatives.

- For some cases, it may be too time-consuming to find a solution or there are no solutions achieving \( \omega_{LB} \) or the desired layer distance. Thus, we modify the quasi-cyclic progressive edge-growth (QC-PEG) algorithm [24], [25] to directly construct PCMs that have a straightforward solution to achieve \( \omega_{LB} \) or the desired layer distance.

- We evaluate the performance of the proposed enumerative and greedy algorithms for partitioning the 5G LDPC codes. There exist cases that \( \omega_{LB} \) or the desired layer distance is not achieved. Then, under the same code parameters as two 5G LDPC codes, we use the modified QC-PEG algorithm to construct QC-LDPC codes to achieve \( \omega_{LB} \) and/or a desired layer distance. Simulation results show that the constructed codes have better error correction performance and achieve less average number of iterations than the 5G LDPC codes.

The remainder of this paper is organized as follows. Section II introduces the preliminaries of QC-LDPC codes and the row layered schedule based on the sum-product algorithm (SPA) [8]. Section III formulates the PCM partitioning problem and then Section IV illustrates the characterization of it. Section V develops two algorithms to solve the PCM partitioning problem. The proposed greedy and enumerative algorithms for solving the PCM partitioning problem are presented in Section VI. Section VI develops a modified QC-PEG algorithm to design QC-LDPC codes that have a straightforward partition scheme to achieve \( \omega_{LB} \) or a desired layer distance. Section VII investigates the partition schemes obtained by the proposed greedy and enumerative algorithms for the 5G LDPC codes and also presents the error correction performance and convergence behavior of the QC-LDPC codes constructed by the modified QC-PEG algorithm. Finally, Section VIII concludes our work.

Notations: We use normal letters to represent scalars, e.g., \( M \) and \( m \). Use bold face uppercase and lowercase letters to respectively represent matrices and vectors, e.g., \( \mathbf{H} \) and \( \mathbf{y} \). Use calligraphic letters to represent sets, e.g., \( \mathcal{G} \). Use Greek letters to represent functions, e.g., \( \phi \). For nonnegative integers \( m \) and \( n \), use \( \{m, n\} \) to represent the integer set \( \{m, m+1, \ldots, n-1\} \). Moreover, if \( m = 0 \), we simplify it to \( \{n\} \). Use \( \mathbb{Z} \) to represent the set of all integers. For positive integers \( a \) and \( b \), use \( a|b \) to represent that \( b \) is divisible by \( a \).

II. PRELIMINARIES

A. QC-LDPC Codes

Let \( \mathbf{x} = (x_0, \ldots, x_{z-1}) \) be a vector of length \( z \). For any integer \( s \), the \( s \)-cyclic (right) shift of \( \mathbf{x} \) is defined as the vector \( \lambda^s(\mathbf{x}) = (x_{(s-1) \mod z}, x_{(s-2) \mod z}, \ldots, x_{(z-1) \mod z}) \).

Assume that \( \mathbf{C} \) is a matrix containing \( z \) rows. For \( i \in [z] \), denote \( \mathbf{C}[i] \) as the \( i \)-th row of \( \mathbf{C} \). Then \( \mathbf{C} \) is said to be a circulant if \( \mathbf{C} \) is a \( z \times z \) square matrix and \( \mathbf{C}[i] = \lambda^i(\mathbf{C}[0]), \forall i \in [z] \).

A QC-LDPC code belongs to the class of structured LDPC codes, with parity-check matrix consisting of \( M \times N \) circulants as follows:

\[
\mathbf{H} = \begin{bmatrix}
H_{0,0} & \cdots & H_{0,N-1} \\
\vdots & \ddots & \vdots \\
H_{M-1,0} & \cdots & H_{M-1,N-1}
\end{bmatrix},
\]

where \( \mathbf{H}_{m,n} \) is a \( Z \times Z \) circulant for \( m \in [M] \) and \( n \in [N] \). As a result, \( \mathbf{H} \) can be equivalently described by a corresponding base matrix \( \mathbf{B} \):

\[
\mathbf{B} = \begin{bmatrix}
b_{0,0} & \cdots & b_{0,N-1} \\
\vdots & \ddots & \vdots \\
b_{M-1,0} & \cdots & b_{M-1,N-1}
\end{bmatrix}.
\]
it corresponds to a zero matrix or a circulant permutation matrix. We hereafter remove the parentheses for simplicity, e.g., $b_{m,n} = -1$.

For example, consider $Z = 4$ and the base matrix

$$
B = \begin{bmatrix} 1 & 3 & -1 \\ 0 & 2 & 0 \\ \end{bmatrix}.
$$

(3)

The PCM $H$ corresponding to $B$ is given below:

$$
H = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \end{bmatrix}.
$$

(4)

Considering the Tanner graph [7] representation for $H$, we denote

$$
G = (\mathcal{V}, \mathcal{E}) = (\mathcal{V}_v \cup \mathcal{V}_c, \mathcal{E})
$$
as the Tanner Graph of the LDPC code, where $\mathcal{V}_c = \{c_m : m \in [MZ]\}$ denotes the CN set, $\mathcal{V}_v = \{v_n : n \in [NZ]\}$ denotes the VN set and $\mathcal{E} \subseteq \mathcal{V}_v \times \mathcal{V}_v$ denotes the edge set.

In the rest of this paper, we consider the problem of partitioning the $H$ given in (1). In the following, we further introduce some concepts related to $H$.

- **Row matrix**: For an ordered subset $A \subseteq [MZ]$, define $H_A$ as the matrix orderly formed by the rows of $H$, with indices in $A$.

  Taking the $H$ in (4) as an example, we have

  $$
  H_{[5,7]} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ \end{bmatrix}.
  $$

(5)

- **Block cyclic shift**: For $m \in [M], i \in [Z]$ and an arbitrary integer $s$, the block $s$-cyclic shift of $H_m[i]$ is defined as the vector $\phi^s[H_m[i]] = (\lambda^s[H_m[0][i]], \ldots, \lambda^s[H_m[N-Z-1][i]])$. Moreover, for a matrix $H_A$, $\phi^s(H_A)$ is to apply the block cyclic shift operation on each row of $H_A$.

It is easy to see that $H$ satisfies the block cyclic shift property: For each $m \in [M]$, $H_m[i] = \phi^s(H_m[0])$ for any $i \in [Z]$.

- **Maximum column weight**: For $i \in [NZ]$, denote $\omega_i(H)$ as the Hamming weight of the $i$-th column of $H$. Denote $\omega(H)$ as the maximum column weight of matrix $H$, i.e., $\omega(H) = \max_{i \in [NZ]} \omega_i(H)$. For example, $\omega(0)(H_{[5,7]}) = 0$ and $\omega(H_{[5,7]}) = 1$ in (5).

### B. Layered Decoding Algorithm

The rows of $H$ are partitioned into $L$ groups/submatrices, where the sets of the row indices in each group are respectively denoted by $T_0, \ldots, T_{L-1}$, and satisfy the following constraint:

**Constraint 1**: $T_0, \ldots, T_{L-1}$ satisfies:

$$
T_0 \cup \cdots \cup T_{L-1} = [MZ],
$$

$$
T_i \cap T_j = \emptyset, \quad \forall i, j \in [L], \quad i \neq j.
$$

For each $l \in [L]$, $H_{T_l} = [H_{T_l,0} \cdots H_{T_l,N-1}]$ is called the $l$-th layer. The decoder operates layer after layer by processing all rows of each layer in parallel based on certain message passing algorithm, e.g., SPA [8] or min-sum algorithm (MSA) [10].

#### Algorithm 1  Row Layered Sum-Product Algorithm

**Input**: Received LLR values $r = (r_0, r_1, \ldots, r_{NZ-1})$.

**Output**: Hard decision estimates $\hat{u} = (\hat{u}_0, \hat{u}_1, \ldots, \hat{u}_{NZ-1})$.

1. **Initialization**:
2. Compute $\delta_{0,m}^l = 0, \forall m \in [MZ], n \in N(c_m)$.
3. $\Lambda_n = r_n, \forall n \in [NZ]$.
4. for $e = 1$ to $I_{\text{max}}$ do
5.   for $l = 0$ to $L - 1$ do
6.     Compute $\mu_{n,m} = \Lambda_n - \delta_{e-1,m}^l$ for each $m \in T_l$ and each $n \in N(c_m)$ in parallel.
7.     Compute $\delta_{e,m}^l = 2 \tanh^{-1} \left( \prod_{n' \in N(c_m) \setminus n} \tanh \left( \frac{\mu_{n',m}^l}{2} \right) \right)$ for each $m \in T_l$ and each $n \in N(c_m)$ in parallel.
8.     Update $\Lambda_n = \Lambda_n + \sum_{m \in N(c_n) \cap T_l} (\delta_{e,m}^l - \delta_{e-1,m}^l)$ for each $n \in [NZ]$. Assume that a codeword $u = (u_0, u_1, \ldots, u_{NZ-1})$ is transmitted and $r = (r_0, r_1, \ldots, r_{NZ-1})$ is the received LLR values from the channel output with $r_n$ for node $v_n$.

   The row layered decoding based on the SPA is summarized in Algorithm 1. (We remark that the partition scheme only changes the processing order of CN update and is independent of the specific CN update algorithm. Thus, the following discussions automatically apply to the MSA or other message passing algorithm.) The following is an example of layered decoding based on the Type-1 and Type-2 partition schemes, respectively.

**Example 1**: Consider $H$ in (4). In Type-1 partition scheme, $H$ is partitioned into two layers, i.e., $H_{T_0} = H_{[0,1,2,3]}$ and $H_{T_1} = H_{[4,5,6,7]}$. In Type-2 partition scheme, $H$ can be also partitioned into two layers, i.e., $H_{T'_0} = H_{[0,2,4,6]}$ and $H_{T'_1} = H_{[1,3,5,7]}$.

$$
H_{T_0} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \end{bmatrix}.
$$

(6)
have the block $S$-cyclic shift property if $H_T^i = \phi^S(H_{T0})$ for all $i \in [L]$.

Note that only the block 1-cyclic shift property (i.e., $S = 1$) was considered in [16], [17], [18], [19], and [20]. If the $L$ layers have the block cyclic shift property, a layer can reuse the hardware by a fixed shift operator transforming it into the previous one. Consequently, the hardware reuse can be greatly simplified, compared to a partition scheme without the block cyclic shift property [16], [17], [18], [19], [20].

As shown in Example 1, when using the Type-2 scheme, each row in $H_{T1}$ is a block cyclic shift of the corresponding row in $H_{T0}$ by 1 to the right. For example, $(0010, 1000, 0000) = \phi^1((0100, 0001, 0000))$. Therefore, when computing the 1-st layer, we can use block cyclically shift the VN’s messages to the left by a constant factor (i.e., 1) such that the hardware in the 0-th layer can be reused. In the Type-1 partition scheme, we can also use shift operators and switch to do this, but the shift factors are not necessarily the same in general. This requires multiplexers to control the selection of shift factors. Therefore, compared to the Type-2 partition scheme, the Type-1 partition scheme requires more hardware cost to implement hardware reuse.

3) Power Efficiency: The power consumption of decoding a layer is proportional to the number of its non-zero entries. Accordingly, the peak (resp. average) power consumption corresponds to the largest (resp. average) number of non-zero entries of each layer. Thus, the Type-2 partition scheme can achieve the highest power efficiency because its peak-to-average power ratio (i.e., the ratio of peak power to average power) can be minimized to 1 for the decoder.

In the following, we analyze the peak-to-average power ratio of the two partition schemes in Example 1. In the Type-1 partition scheme, the 0-th layer requires 8 computation nodes and the 1-st layer requires 12 computation nodes. Noting that all of these computation nodes are the same, the peak-to-average power ratio of the Type-1 partition scheme is $12/((12 + 8)/2) = 1.2$. In the Type-2 partition scheme, both the two layers require 10 computation nodes. Therefore, the peak-to-average power ratio of the Type-2 partition scheme is $10/((10 + 10)/2) = 1$.

III. PROBLEM FORMULATION OF PCM PARTITIONING

A. Basic Optimization Problem

Let $H$ be the PCM of a QC-LDPC code with size $MZ \times NZ$ given in (1). Suppose that $H$ satisfies the following two properties:

- There are no zero rows otherwise we can remove them;
- $H$ does not contain two identical rows otherwise there are a large amount of 4-cycles greatly reducing performance.

Given integer $L > 1$, our goal is to partition the rows of $H$ into $L$ layers such that (i) Constraint 1 is satisfied, (ii) block cyclic shift property holds, and (iii) the maximum column weight of each layer is minimized. This can benefit the row layered decoding by reducing the decoding latency within a layer and simplifying the hardware reusability between consecutive layers. Specifically, we formulate the partitioning...
problem as the following optimization problem.

**Problem 1:** 
\[
\min_{S > 0, T_0, T_1, \ldots, T_{L-1}} \omega(H_{T_0}) \\
\text{s.t.} \text{ Constraint 1 is satisfied, and } \\
H_{T_l} = \phi^S(H_{T_0}), \quad \forall l \in [L]. \tag{10}
\]

Constraint 1 guarantees that the \( L \) layers form a partition of \( H \), and (10) ensures its block \( S \)-cyclic shift property. Note from (10) that \( T_1, \ldots, T_L \) are determined by \( S \) and \( T_0 \), and the layers have the same maximum column weight. Therefore, we can use \((S, T_0)\) to represent a partition scheme uniquely and just focus on the maximum column weight of \( H_{T_0} \), i.e. \( \omega(H_{T_0}) \). Any \((S, T_0)\) satisfying Constraint 1 and (10) is called a feasible solution to Problem 1. A feasible solution \((S^*, T_0^*)\) is called the optimal solution if \( \omega(H_{T_0^*}) = \min_{S > 0, T_0, T_1, \ldots, T_{L-1}} \omega(H_{T_0}) \).

**B. Data Dependency**

The optimal solution to Problem 1 provides an efficient PCM partition scheme by configuring the block cyclic shift value \( S \) and the set of row indices \( T_0 \), which can well support the parallelism, reusability, and power efficiency of a layered LDPC decoder. However, it is possible to further optimize the PCM partition scheme from data dependency aspect for achieving shorter computation delay [10]. Data dependency occurs in the message update process when two CNs have the same neighbour VN. The following is an illustrative example.

**Example 2:** Consider \( S = 1, L = 4 \) and \( H \) in (4). Note that \( \omega(H_{T_0}) \geq 1 \), otherwise \( H_{T_l} \) will be a zero matrix. It is easy to verify that \((S, T_0) = (1, \{0, 4\})\) and \((S', T_0') = (1, \{0, 7\})\) are two optimal solutions to Problem 1. whose partitioned matrices are as follows.

\[
H_{T_0} = H_{\{0, 4\}} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix},
\]

\[
H_{T_1} = H_{\{1, 5\}} = \begin{bmatrix}
0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix},
\]

\[
H_{T_2} = H_{\{2, 6\}} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix},
\]

\[
H_{T_3} = H_{\{3, 7\}} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix},
\]

\[
H_{T_0'} = H_{\{0, 7\}} = \begin{bmatrix}
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix},
\]

\[
H_{T_1'} = H_{\{1, 4\}} = \begin{bmatrix}
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 \\
\end{bmatrix},
\]

\[
H_{T_2'} = H_{\{2, 5\}} = \begin{bmatrix}
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
\end{bmatrix},
\]

\[
H_{T_3'} = H_{\{3, 6\}} = \begin{bmatrix}
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 \\
\end{bmatrix}.
\]

In partition scheme 1, the data dependency occurs during the CN update of the nodes \( c_0 \) and \( c_5 \) located in layers 0 and 1. At layer 1, the \( v_1 \)-to-\( c_0 \)-message needs to be computed (Line 6 in Algorithm 1) based on the posterior LLR \( \Lambda_1 \) updated at layer 0 (Line 8 in Algorithm 1). However, due to the delay caused by the pipelining stage [17], [18], we have to pay extra waiting time to get the latest value of \( \Lambda_1 \).

Whereas partition scheme 2 can avoid data dependency between two consecutive layers, where all the \( \Lambda_n \)'s (\( n \in [12] \)) are updated at most once in any two consecutive layers. As a consequence, the computation delay can be reduced for obtaining the latest posterior LLRs.

That is, the second partition scheme leads to a shorter computation delay compared to the first one, though they have the same maximum column weight. In summary, to reduce the computation delay, the column weight for the consecutive layers should be as small as possible.

**C. Further Optimization Problem**

In practice, there is often more than one optimal solution to Problem 1, where the data dependency of each solution may vary, as illustrated in Example 2. To minimize the computation delay, we intend to find out the optimal solution that can decrease the data dependency among consecutive layers. The general idea is to separate any two non-zero entries in a column as far as possible, with respect to layer distance.

**Definition 2 (Layer Distance):** Given \( H \) and \( L \), let \((S, T_0)\) be a feasible solution to Problem 1. The layer distance of \((S, T_0)\), denoted by \( d(S, T_0) \), is defined as the maximum integer \( l \in [L] \) such that

\[
\omega(H_{T_l}) \leq 1. \tag{11}
\]

**Remark 1:** 1) In general, the layer distance can be defined as the minimum maximum integer \( l \in [L] \) such that (11) still holds when \( T_0, T_1, \ldots, T_{L-1} \) are respectively replaced by \( T_i, T_{i+1}, \ldots, T_{i+l-1} \) where \( i \) ranges over \([L]\) and the subscript is modulo \( L \). Actually, \( i = 0 \) is enough for our definition because the resultant equation is the same as (11) by means of block cyclic shift property of \( H \).

2) \( d(S, T_0) = 2 \) is generally sufficient for practical applications [10].

3) Note that, for \( l = 0 \), we define \( \omega(\emptyset) = 0 \). Therefore, we always have \( d(S, T_0) \geq 0 \). In particular, we have \( d(S, T_0) = 0 \) if \( \omega(H_{T_0}) > 1 \). Accordingly, the data dependency is the strongest, since it occurs within a layer.

According to Definition 2, a partition scheme with layer distance \( k \) can avoid data dependency within any \( k \) consecutive layers. Adding the layer distance into the objective functions, we have the following multi-objective optimization problem.

**Problem 2:**
\[
\min_{S > 0, T_0} \omega(H_{T_0}), -d(S, T_0) \\
\text{s.t.} \text{ Constraint 1 is satisfied, and } \\
H_{T_l} = \phi^S(H_{T_0}), \quad \forall l \in [L],
\]

In Problem 2, we mean to both minimize \( \omega(H_{T_0}) \) and maximize the layer distance \( d(S, T_0) \). Note that \( d(S, T_0) \geq 1 \) only if \( \omega(H_{T_0}) = 1 \). Therefore, we should first guarantee \( \omega(H_{T_0}) = 1 \), and then further optimize \( d(S, T_0) \). In this case, the solution \((S, T_0)\) is also one of optimal solutions to Problem 1.
D. Relationship Between Problems 1 and 2

The following theorem establishes the connection between Problem 1 and Problem 2.

Theorem 1: Given $H, L$, and a feasible solution $(S, T_0)$, $d(S, T_0) \geq k$ if and only if (iff) $H_{T_0}^{(S,k)}$ is a binary matrix and $\omega(H_{T_0}^{(S,k)}) = 1$, where $H^{(S,k)} = H + \phi_S(H) + \phi^{2S}(H) + \ldots + \phi^{(k-1)S}(H)$ and the matrix addition is the decimal addition.

Proof: The necessity is obvious because $H$ is a binary matrix of form (1). Since

$$\omega(H_{T_0}^{(S,k)}) = \omega\left(\sum_{i \in [k]} \phi^{iS}(H_{T_0})\right) = \omega\left(\begin{bmatrix} H_{T_0} \\ H_{T_1} \\ \vdots \\ H_{T_{k-1}} \end{bmatrix}\right)$$

for binary $H^{(S,k)}$, the sufficiency then follows from Definition 2.

According to Theorem 1, we can get a partition scheme with desired layer distance $k$, by solving Problem 1 for $H^{(S,k)}$. Specifically, we first construct $H^{(S,k)}$, and then find an optimal solution $(S^*, T_0^*)$ to Problem 2 for $H^{(S,k)}$. If $\omega(H_{T_0}^{(S,k)}) = 1$, we have $d(S^*, T_0^*) \geq k$; otherwise it implies that there does not exist feasible solution with layer distance at least $k$.

Therefore, we mainly consider Problem 1 in what follows.

IV. CHARACTERIZATION OF PROBLEM 1

A. Problem Transformation

Definition 3: For an integer $i \in [MZ]$ and an integer $s$, define $\pi^s(i) = Z \cdot [i/Z] + ((i + s) \mod Z)$. Moreover, for an index set $T \subseteq [MZ]$, define $\pi^s(T)$ as $\{\pi^s(x) : x \in T\}$.

Lemma 1: For $i, j \in [MZ]$ and an integer $s$, $H[j] = \phi^s(H[i])$ iff $j = \pi^s(i)$.

Proof: According to block cyclic shift property of $H$, we have $\phi^s(H_m[k]) = H_{m+1}[k + s] \mod Z$. Then, sufficiency is clear. Whereas, the necessity also follows otherwise it contradicts the assumption that there are no two identical rows in $H$.

Then, it is more convenient to describe Problem 1 in terms of the row indices of $H$.

Problem 3: $\min_{S > 0, T_0} \omega(H_{T_0})$

\[ s.t. \] Constraint 1 is satisfied, and $T_l = \pi^s(T_0), \quad \forall l \in [L]. \quad (12)$

To solve this problem, a general approach is to firstly enumerate all the values of $(S, T_0)$, next check whether it is a feasible solution, and finally obtain an optimal solution. It is clear that there are $M Z / L$ options of $T_0$, i.e., the search space is prohibitively large. Accomplishing this task is impossible for a large size of $H$. Instead, we consider to directly characterize all the feasible solutions to Problem 3.

B. Feasible Solutions to Problem 3

To solve Problem 3, it suffices to determine $T_0$ satisfying (12) and Constraint 1. In this subsection, we characterize all the feasible solutions to Problem 3. For convenience, denote $T_l = T_0 \cap [mZ, (m + 1)Z)$ for each $m \in [M]$. According to Lemma 1, $T_l = \pi^s(T_0) \cap [mZ, (m + 1)Z)$, which results in $|T_l| = z/L$, i.e., it is further required that $L$ should be a factor of $Z$.

Definition 4 (s-Cyclic Shift Class): For $i \in [MZ]$ and $s \in [1, Z)$, define the $s$-class (short for s-cyclic shift class) generated by $i$ as the set $\{s \cdot r \mod Z : r \in Z\}$.

Lemma 2: Let $(S, T_0)$ be a feasible solution to Problem 3. For any $l \in [L]$, $i \in T_l$ iff $\pi^s(i) \in T_l$.

Proof: According to Definition 3, $\pi^s(\cdot)$ is a bijection. Then, the lemma follows (12).

Obviously, $\{i \in [MZ] \mid \pi^s(i) \in T_l\}$ for all $i \in [MZ]$ with $S' = \gcd(Z, S)$. Thus, hereafter we always assume that $S$ is a factor of $Z$, i.e., $S = Z$.

First of all, we focus on the characterization of $T_0$. Recall that $T_0, \ldots, T_{L-1}$ form a partition of $[Z]$, each of size $Z/L$. On the other hand, $S$-classes $\{0\}, \ldots, \{S - 1\}$ form a partition of $[Z]$, each of size $Z/S$. Thus, there exists at least an $i \in S$, satisfying that $T_0 \cap \{i\} \neq \emptyset$, otherwise we have $T_0 \cap [Z] = \emptyset$, i.e., $T_0 = \emptyset$. Accordingly, we assume $j$ is an element in $\{i\} \cap T_0$ and analyze the allocation of the rest elements in $\{i\}$ in the following.

For any $l \in [1, L]$, $\pi^s(j) \in T_0$ by (12), which implies $\pi^{(L-1)s}(j) \notin T_0$ since $T_0 \cap T_{L-1} = \emptyset$. Then, according to Lemma 2, we have $\pi^s(j) \in T_0$ since $\pi^s(j) = \pi^{(L-1)s}(j) + j$, for any $l \in [1, L]$.

Analogously, as shown in Table I, $\pi^{SL}(j), \pi^{3SL}(j), \ldots$ are in $\{T_0, T_1, T_2, \ldots\}$, $\pi^{2SL}(j), \pi^{3SL}(j), \ldots$ are in $\{T_1, T_2, \ldots\}$, and so on. That is, we have $\{i\} \subseteq T_0, \{\pi^s(j) \subseteq T_0, \ldots, \{\pi^{(L-1)s}(j) \subseteq T_{L-1}$. Moreover, since $T_0, T_1, \ldots, T_{L-1}$ are pairwise disjoint, we must have $(LS) \subseteq S$, i.e., $S$ must be a factor of $Z/L$.

To conclude, for any $i \in [S]$, once an element $j \in \{i\}$ is selected into $T_0$, the remaining elements $\pi^s(j), \ldots, \pi^{ZS}(j)$ in $\{i\}$ are automatically divided into the sets $T_0, \ldots, T_{L-1}$, as shown in Table I.

Clearly, for each $m \in [M]$, $mZ, (m + 1)Z$ can be partitioned into $S$ different $S$-classes: $C_{0m} \subseteq mZ + \{0\}, C_{0m} \subseteq mZ + \{S - 1\}$. So, the above selection method can be similarly carried on $mZ, (m + 1)Z$ to obtain $T_0, \ldots, T_{L-1}$. More specifically, for each $s \in S$, the $s$-class $C_{ms}$ can be further partitioned into $L$ different $LS$-classes: $C_{ms} \subseteq mZ + s \subseteq (LS) \subseteq (LS)$. Then, the set $T_0$ consists of $M S$ different $LS$-classes, each of which is from a different $S$-class, as characterized by the following theorem.

| $T_{0,0}$ | $T_{1,0}$ | $\ldots$ | $T_{L-1,0}$ |
|---|---|---|---|
| $\pi^s(j)$ | $\pi^{(L+1)s}(j)$ | $\ldots$ | $\pi^{(L-1)s}(j)$ |
| $\pi^{Zs}(j)$ | $\pi^{Z-(L-1)s}(j)$ | $\ldots$ | $\pi^{Zs}(j)$ |
Theorem 2: \((S, T_0)\) is a feasible solution to Problem 3 iff
\[
T_0 \in \{ \cup_{m \in [M], s \in [S]} C_{m,s,s} : \ s \text{ ranges over } [L] \}.
\]

Proof: Assume \((S, T_0)\) is a feasible solution, \(T_0 = \cup_{m \in [M]} T_{0,m} = \cup_{m \in [M], s \in [S]} (T_{0,m} \cap C_{m,s})\). As investigated above, \(T_{0,m} \cap C_{m,s}\) is an \(L\)-class, i.e., \(T_{0,m} \cap C_{m,s} \in \{ C_{m,s,s} : s \in [L] \}\). Therefore, \(T_0 \in \{ \cup_{m \in [M], s \in [S]} C_{m,s,s} : s \in [L] \}\). That is, the necessity is clear. Whereas, the sufficiency can be easily verified.

Example 3: Let \(H\) be given by (4) and \((L, S) = (2, 2)\). The rows of \(H\) can be partitioned into four \(S\)-classes (2-classes) and eight \(L\)-classes (4-classes) as shown in (13) and (14), respectively.

\[
C_{0,0} = \{0, 2\}, \ C_{0,1} = \{1, 3\}, \ C_{1,0} = \{4, 6\},
C_{1,1} = \{5, 7\}, \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \ quad
according to Theorem 2; then, we go through all the possible $S$ to obtain all the feasible solutions to Problem 3; finally, among them, the one with the minimal $\omega(H_{T_0})$ is the optimal solution.

For a given $S$, we have $|F_S| = L^{MS}$, i.e., the $\omega(\cdot)$ function is computed for $L^{MS}$ times. Computing $\omega(\cdot)$ has complexity $O(MNZ^2/L)$ since $H_{T_0}$ has NZ columns, each of which has $MZ/L$ elements. Thus, Algorithm 2 has complexity $O(MNZ^2L^{MS-1})$ for a given $S$.

Algorithm 2 Enumerative Method for Solving Problem 3

Input: $H, L$.
Output: $(S^*, T_0^*)$.

1: Initialization: $F = \emptyset$.
2: for each factor $S$ of $Z/L$ do
3: $F_S = \{(S, \cup_{m \in [M], s \in [S]} C_{m,s,:}) : s \text{ ranges over } [L]\}$.
4: $F = F \cup F_S$.
5: end for
6: Return $(S^*, T_0^*) = \arg \min_{(S,T_0) \in F} \omega(H_{T_0})$.

The above enumerative algorithm may not work for a large scale case. Thus, we present a greedy method (similar to that in [17]) as an alternative.

Greedy Algorithm: As shown in Algorithm 3, for a given $S$, the idea is to handle $C_{0,0}, \ldots, C_{M-1, S-1}$ one-by-one, corresponding to Line 4. Then, for each $C_{m,s}$, select a locally best $LS$-class $C_{m,s,t}$ to add into $T_0$, corresponding to Line 11. The $\omega(\cdot)$ function is computed for $O(MLS)$ times. As a result, Algorithm 3 has complexity $O(M^2SNZ^2)$ for a given $S$.

Algorithm 3 Greedy Method for Solving Problem 3

Input: $H, L$.
Output: $(S^*, T_0^*)$.

1: Initialization: $\omega^* = \infty$.
2: for each factor $S$ of $Z/L$ do
3: Initialization: $T_0 = \emptyset$.
4: for each $m \in [M]$ and $s \in [S]$ do
5: Initialization: $\omega_{m,s}^* = \infty$, $l^* = 0$.
6: for each $l \in [L]$ do
7: if $\omega(H_{T_0 \cup C_{m,s,:}}) < \omega_{m,s}^*$ then
8: Let $(\omega_{m,s}^*, l^*) = (\omega(H_{T_0 \cup C_{m,s,:}}), l)$.
9: end if
10: end for
11: $T_0 = T_0 \cup C_{m,s,l^*}$.
12: end for
13: If $\omega(T_0) < \omega^*$, let $(\omega^*, S^*, T_0^*) = (\omega(H_{T_0}), S, T_0)$.
14: end for

The $\omega(\cdot)$ function is computed frequently in Algorithms 2 and 3. Reducing the complexity of computing $\omega(\cdot)$ can benefit the efficiency of both algorithms. Specifically, since $\omega(H_{T_0})$ is of interest, it suffices to compute the weight of certain columns of $H_{T_0}$ instead of all its columns, as shown in the following theorem.

Theorem 4: Assume that $(S, T_0)$ is a solution to Problem 3. Then,

$$\omega(H_{T_0}) = \max_{j \in [NZ], j \mod Z = 1} \omega_j(H_{T_0}).$$

Proof: Take an arbitrary row vector $x$ from $H_{T_0}$, whose $j$-th entry is 1. According to Algorithm 2, $T_0$ consists of the $LS$-classes, we then have $\phi^{LS}(x)$ is also a row vector of $H_{T_0}$ with $\pi^{LS}(j)$-th entry being 1. Analogously, there exist row vectors in $H_{T_0}$, whose $\pi^{2LS}(j)$-th, $\pi^{3LS}(j)$-th, $\ldots$, $\pi^{Z-LS}(j)$-th entries are 1, respectively. That is, a 1-entry in $j$-th column of $H_{T_0}$ results in a 1-entry in $j$-th column of $H_{T_0}$, respectively, for each $j' \in \{j\}_{LS}$. Therefore, $\omega_j(H_{T_0}) = \omega_j'(H_{T_0})$ for each $j \in [NZ]$ and $j' \in \{j\}_{LS}$. Noting that for any $j \in [NZ]$, there exists a $j' \in \{j\}_{LS}$ such that $j' \mod Z < LS$, we have

$$\omega(H_{T_0}) = \max_{j \in [NZ]} \omega_j(H_{T_0}) = \max_{j \in [NZ], j \mod Z = 1} \omega_j(H_{T_0}).$$

The proof is completed.

By Theorem 4, the complexity of computing $\omega(\cdot)$ can be reduced by a factor of $\frac{L}{Z}$. The reduction is significant since generally $L$ and $S$ are small while $Z$ is large for practical scenarios.

The following lower bound on $\omega(H_{T_0})$ is obvious, which would be greatly helpful for evaluating the quality of a feasible solution.

Theorem 5: For any solution $(S, T_0)$ to Problem 3,

$$\omega_{LB} \triangleq \left\lceil \frac{\omega(H)}{L} \right\rceil \leq \omega(H_{T_0}).$$

Similar to Theorem 5, we have the following upper bound on layer distance.

Theorem 6: Given $H$ and $L$, let $(S, T_0)$ be an arbitrary feasible solution to Problem 3. Then,

$$d_{UB} \triangleq \left\lceil \frac{L}{\omega(H)} \right\rceil \geq d(S, T_0).$$

Proof: Recall that $H^{S,d(S,T_0)} = H + \phi^{S}(H) + \phi^{2S}(H) + \cdots + \phi^{(d(S,T_0)-1)S}(H)$. According to Theorem 5, we have

$$1 = \omega\left(H^{S,d(S,T_0)}\right) \geq \omega\left(H^{S,d(S,T_0)}\right)/L = d(S, T_0) \omega(H)/L, \quad (15)$$

since $\omega(H) = \omega(\phi^{iS}(H)), 1 \leq i < d(S, T_0)$, which leads to the claimed bound.

Further, when considering a desired layer distance and finding a proper $L$, we have the following corollary.

Corollary 1: Given $H$, to have a partition scheme with layer distance at least $k$, then $L \geq L_{LB}$ where the lower bound $L_{LB}$ is the smallest factor of $Z$ and satisfies $L_{LB} \geq k \cdot \omega(H)$.

Proof: By replacing $d(S, T_0)$ in (15) with $k \cdot \omega(H)$.

Theorem 5.

Then, the corollary follows from the fact $L \mid Z$. 


VI. PCM DESIGN FOR PARTITIONING

According to the algorithms presented in Section V, we can get a partition scheme with a maximum column weight or a large layer distance. However, for some cases, finding such solution may be too time-consuming or even there is no solution achieving \( \omega_{LB} \) or the desired layer distance.

In this section, we design such QC-LDPC codes based on QC-PEG algorithms \([24],[25]\). In particular, we first introduce the framework of QC-PEG algorithms. Then, we derive a condition for PCM to guarantee that the PCM has a partition scheme attaining \( \omega_{LB} \) or the desired layer distance. Finally, we add this condition into the CN selection strategy of a QC-PEG algorithm to construct the desired QC-LDPC codes.

A. Framework of QC-PEG

QC-PEG algorithms are widely used to design QC-LDPC codes by avoiding short cycles. To illustrate the QC-PEG algorithms, we give two notations in the following.

- Denote \( (c_i, v_j)_Z = \{(c_{v(i)}, v_{v(j)}): t \in [Z]\}\) for each \( i \in [MZ], j \in [NZ]\), as the cyclic edge set (CES) which \((c_i, v_j)\) belongs to. For a QC-LDPC PCM, we have \((c_i, v_j) \in \mathcal{E} \iff (c_j, v_j)_Z \subseteq \mathcal{E}\).

- Denote \( \mathbf{d} = (d_0, d_1, \ldots, d_{NZ-1}) \) the VN degree sequence, where \( d_j \) is the degree of the \( j \)-th VN. Clearly, \( d_j = d_j' \) for \( j, j' \in [NZ] \) with \( \lfloor j/Z \rfloor = \lfloor j'/Z \rfloor \).

The framework of QC-PEG algorithms is shown in Algorithm 4, where the key point is to design an efficient strategy for selecting a CN and connecting it to \( v_j \), corresponding to Line 4. A CN selection strategy consists of several criteria, which are carried out in order. For instance, a QC-PEG algorithm was illustrated in \([25]\), whose CN selection strategy is outlined in Strategy 1.

Strategy 1:
1) Select the CN set \( A_1 \) as the maximum subset of \( \mathcal{V}_c \) such that for any \( c_i \in A_1 \), we have \((c_i, v_j) \notin \mathcal{E}\) and \(\lfloor x \in N_c(v_{j+z}): x \equiv 0 \mod L \rfloor \leq \omega_{LB}, \forall z \in [Z]\), (17)

2) The same as Criteria 2 to 4 in Strategy 1.

B. PCM Design for Achieving \( \omega_{LB} \)

In this subsection, for given \( M, N, Z, L \) and \( \mathbf{d} \), we employ Algorithm 4 to construct \( \mathbf{H} \) achieving \( \omega_{LB} = [\omega(\mathbf{H})]/L = \max_{j \in [NZ]} \lfloor d_j/L \rfloor \) with a straightforward partition scheme \( (S, T_0) = (1, \cup_{m \in [M]} C_{m,0,0}) \). We consider \( S = 1 \) since it is the simplest and further it is always sufficient for the construction.

**Lemma 3:** \( (S, T_0) = (1, \cup_{m \in [M]} C_{m,0,0}) \) is an optimal solution with \( \omega(\mathbf{H}_{T_0}) = \omega_{LB} \) iff

\[
|\{x \in N(v_j): x \equiv 0 \mod L\}| \leq \omega_{LB}, \forall j \in [NZ],
\]

where the equality holds for some \( j \)’s.

**Proof:** Noting that \( \cup_{m \in [M]} C_{m,0,0} = \{0, L, 2L, \ldots, MZ - L\} \), for each \( j \in [NZ] \), we have \( \omega_j(\mathbf{H}_{T_0}) = |N(v_j) \cap T_0| = |N(v_j) \cap (\cup_{m \in [M]} C_{m,0,0})| = |\{x \in N(v_j): x \equiv 0 \mod L\}| \). Then, the lemma follows from the fact that \( \omega(\mathbf{H}_{T_0}) = \max_{j \in [NZ]} \omega_j(\mathbf{H}_{T_0}) \).

Based on Lemma 3, we modify the selection strategy of the QC-PEG algorithm \([25]\) to obtain a PCM \( \mathbf{H} \) with a straightforward partition scheme as follows.

**Strategy 2:**
1) Select the CN set \( A_1 \) as the maximum subset of \( \mathcal{V}_c \) such that for any \( c_i \in A_1 \), we have \((c_i, v_j) \notin \mathcal{E}\) and \(\lfloor x \in N_c(v_{j+z}): x \equiv 0 \mod L \rfloor \leq \omega_{LB}, \forall z \in [Z]\), (17)

2) The same as Criteria 2 to 4 in Strategy 1.

According to Criterion 1, with \( j \) going through \( 0, Z, \ldots, (N - 1)Z \), the PCM constructed by Strategy 2 satisfies (16) and has a partition scheme achieving \( \omega_{LB} \) consequently.

**Theorem 7:** \( \mathbf{H} \) has a partition with \( \omega(\mathbf{H}_{T_0}) = \omega_{LB} \) iff there exists a QC-LDPC PCM

\[
\mathbf{H}' = \begin{bmatrix}
\phi^{j_0}(H_0) \\
\phi^{j_1}(H_1) \\
\vdots \\
\phi^{j_{M-1}}(H_{M-1})
\end{bmatrix}, \quad j_0, j_1, \ldots, j_{M-1} \in [L],
\]

satisfying (16).

**Proof:** Let \( T_0 = \cup_{m \in [M]} C_{m,0,j_m} \) and \( T'_0 = \cup_{m \in [M]} C_{m,0,0} \). The proof is based on the fact \( \mathbf{H}_{T_0} = \mathbf{H}'_{T'_0} \).

---

Algorithm 4 Framework of QC-PEG Algorithms

**Input:** \( M, N, L, Z, \mathbf{d} \).

**Output:** \( \mathcal{G} \).

1: **Initialization:** \( \mathcal{G} = (\mathcal{V}, \emptyset) \).
2: **for** \( j = 0, Z, \ldots, (N - 1)Z \) **do**
3: **for** each \( t = 0, 1, \ldots, d_j - 1 \) **do**
4: **Select** a CN \( c_i \) based on a certain strategy.
5: \( \mathcal{E} = \mathcal{E} \cup (c_i, v_j)_Z \).
6: **end for**
7: **end for**
8: **return** \( \mathcal{G} \).

---

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
Sufficiency: Assume that \( H' \) satisfies (16). According to Lemma 3, \((1, T'_0)\) is an optimal solution with \( \omega(H'_T) = \omega_LB \). Then, we have \( \omega(H_{T_0}) = \omega(H'_T) = \omega_LB \).

Necessity: Suppose that \( H \) has a partition \((1, T_0)\) with \( \omega(H_{T_0}) = \omega_LB \). That is, \((1, T'_0)\) is an optimal solution with \( \omega(H'_T) = \omega_LB \). According to Lemma 3, (16) is satisfied for \( H' \).

Moreover, we can design a PCM achieving the lower bound for arbitrary VN degree sequence \( d \).

**Theorem 8:** Given an arbitrary \( d \), there always exists an \( H \) constructed by Strategy 2, achieving \( \omega_LB \).

**Proof:** For each \( j \in \{0, Z, \ldots, (N-1)Z\} \) and \( z \in [Z] \), we have \( x \in N(v_1) \) iff \( \pi^x(x) \in N(v_1+z) \) since \( H \) is a QC-LDPC PCM. Therefore, we have \( \{x \in N(v_{j+z}) : x \equiv 0 \mod L\} = \{x \in N(v_j) : x \equiv z \mod L\} \). Accordingly, to make sure that (17) is satisfied, we just need \( |\{x \in N(v_j) : x \equiv z \mod L\}| \leq w_LB, \forall z \in [Z]. \) (18)

On the other hand, it is easy to verify that \( N(v_j) = \{x \in \mathbb{N} \mid x \equiv z \mod L\} \) satisfies (18) with \( L \cdot w_LB \) elements. According to Theorem 5, \( L \cdot w_LB \geq \omega(H) \geq d_I \), for each \( j \in [NZ] \). Therefore, there always exist candidate CNs after performing Criterion 1 in Strategy 2.

C. PCM Design for Achieving Desired Layer Distance

In this subsection, we construct a PCM with desired layer distance \( k \). We focus on \( k > 0 \), i.e. \( \omega(H_{T_0}) = 1 \).

**Lemma 4:** \((S, T_0) = (1, \cup_{m \in [M]} C_m, 0, 0)\) is a feasible solution with layer distance at least \( k > 0 \) iff for any \( [NZ] \),

\[
\begin{align*}
&\{N(v_j) \cap N(v_{j+z}) = \emptyset, \forall z \in [1, k], \\
&\{\{x \in \cup_{t \in [k]} N(v_{j+t}) : x \equiv 0 \mod L\} \leq 1\}. \quad (19)
\end{align*}
\]

**Proof:** Sufficiency: Assume that (19) is satisfied. Recall that \( H^{(1,k)} = H + \phi^1(H) + \phi^2(H) + \cdots + \phi^{k-1}(H) \). Since \( N(v_j) \cap N(v_{j+z}) = \emptyset, \forall z \in [1, k], \) \( H^{(1,k)} \) is binary. For each \( j \in [NZ] \), we then have \( \omega_H^{(1,k)} = |\{x \in \cup_{t \in [k]} N(v_{j+t}) : x \equiv 0 \mod L| \leq 1, \) implying \( \omega(H^{(1,k)}_{T_0}) = 1 \). Thus, \( d(1, T_0) \geq k \) by \( \text{Lemma 4} \).

Necessity: Assume that \((S, T_0) = (1, \cup_{m \in [M]} C_m, 0, 0)\) is a feasible solution with layer distance at least \( k \). According to Theorem 1, we have \( H^{(1,k)} \) is binary and \( \omega(H^{(1,k)}_{T_0}) = 1 \). Therefore, we have \( N(v_j) \cap N(v_{j+z}) = \emptyset, \forall z \in [1, k] \) and for each \( j \in [NZ] \), \( |\{x \in \cup_{t \in [k]} N(v_{j+t}) : x \equiv 0 \mod L| = \omega_H^{(1,k)} \leq 1 \).

By means of Lemma 4, we can obtain a PCM with desired layer distance \( k \) by the following Strategy:

**Strategy 3:**
1) Select the CN set \( A_i \) as the maximum subset of \( V_c \) such that for any \( c_i \in A_i \), we have \( (c_i, v_j) \notin E \) and

\[
\begin{align*}
&\{N_i(v_{j+z}) \cap N_i(v_{j+z+1}) = \emptyset, \forall z \in [1, k], \\
&\{\{x \in \cup_{t \in [k]} N_i(v_{j+t}) : x \equiv 0 \mod L\} \leq 1, \quad (20)
\end{align*}
\]

for each \( z \in [Z] \), where \( N_i(v_{j+z}) \equiv N(v_{j+z}) \cup \{\pi^z(i)\} \) and \( N_i(v_{j+z+1}) \equiv N(v_{j+z+1}) \cup \{\pi^{z+1}(i)\} \) denote the index sets of CNs connected to \( v_{j+z} \) and \( v_{j+z+1} \) after the CES \((c_i, v_j)\). is added in \( E \), respectively.
TABLE III

| PCM | L | $\omega_{LB}$ | $\omega(H_{T_C}^T), S^*$ |
|-----|---|--------------|-----------------|
| 1   | 2 | 3            | 3, 1            |
|     | 3 | 2            | 3, 1            |
|     | 4 | 2            | 2, 1            |
|     | 6 | 1            | 2, 1            |
|     | 8 | 1            | 2, 1            |
|     | 12| 1            | 1, 4            |
|     | 16| 1            | 1, 1            |
|     | 24| 1            | 1, 1            |
|     | 32| 1            | 1, 1            |
|     | 48| 1            | 1, 1            |
|     | 64| 1            | 1, 1            |
|     | 96| 1            | 1, 1            |
|     | 128| 1           | 1, 1            |
|     | 192| 1           | 1, 1            |
|     | 384| 1           | 1, 1            |
| 2   | 3 | 3            | 3, 1            |
|     | 4 | 1            | 2, 1            |
|     | 6 | 1            | 1, 1            |
|     | 12| 1            | 1, 1            |
|     | 16| 1            | 1, 1            |
|     | 24| 1            | 1, 1            |
|     | 32| 1            | 1, 1            |
|     | 48| 1            | 1, 1            |
|     | 64| 1            | 1, 1            |
|     | 96| 1            | 1, 1            |
|     | 128| 1          | 1, 1            |
|     | 192| 1          | 1, 1            |
|     | 384| 1          | 1, 1            |
| 3   | 2 | 3            | 3, 1            |
|     | 4 | 2            | 2, 1            |
|     | 7 | 1            | 1, 4            |
|     | 8 | 1            | 1, 1            |
|     | 14| 1            | 1, 1            |
|     | 16| 1            | 1, 1            |
|     | 28| 1            | 1, 1            |
|     | 56| 1            | 1, 1            |
|     | 112| 1          | 1, 1            |
| 4   | 2 | 3            | 3, 1            |
|     | 4 | 4            | 4, 1            |
|     | 7 | 2            | 2, 1            |
|     | 8 | 2            | 2, 1            |
|     | 14| 1            | 1, 1            |
|     | 16| 1            | 1, 1            |
|     | 28| 1            | 1, 1            |
|     | 56| 1            | 1, 1            |
|     | 112| 1         | 1, 1            |
| 5   | 2 | 3            | 3, 1            |
|     | 4 | 2            | 2, 1            |
|     | 6 | 3            | 3, 1            |
|     | 7 | 4            | 4, 1            |
|     | 8 | 3            | 3, 1            |
|     | 14| 2            | 2, 1            |
|     | 16| 2            | 2, 1            |
|     | 28| 1            | 1, 1            |
|     | 56| 1            | 1, 1            |
|     | 112| 1         | 1, 1            |

In Table III, we use the bold face to highlight the $\omega(H_{T_C}^T)$ which is greater than $\omega_{LB}$. We use ‘−’ to indicate the case where Algorithm 2 cannot achieve the lower bound $\omega_{LB}$ within a preset running time limit. We can see that both algorithms can generally achieve $\omega_{LB}$ or differ by one. There exist cases (e.g., $L = 12$ for PCM 1) where Algorithm 2 may exceed the preset time limit due to large search space.

Table IV shows the partition results with desired layer distance for the 5G LDPC codes. We use $k$ to represent the desired layer distance and $L^*$ to represent the minimum number of layers that may have a scheme with desired layer distance. We use the bold face to highlight the $L^*$ which is greater than $L_{LB}$. We use ‘−’ to indicate the case where the algorithm cannot find a feasible solution with desired $k$ within a preset running time limit. Accordingly, we also list $L_{LB}$, which is the lower bound of $L^*$ given in Corollary 1.

We can see that in most cases, both algorithms can find a partition scheme with desired layer distance, where the minimum number of layers $L^*$ is slightly greater than or equal to $L_{LB}$. Since they work in an enumerative and greedy fashion, respectively, there exist cases (e.g., $k = 2$ for PCM 1) where Algorithm 2 can find a desired solution under a lower $L^*$ than that in Algorithm 3. In addition, there exists a case ($k = 4$ for PCM 5) where both algorithms cannot find a partition scheme with desired layer distance.

B. Decoding Performance

As can be seen from Tables III and IV, there exist cases that $\omega_{LB}$ or the desired layer distance is not achieved. For these cases, we can use the modified QC-PEG algorithm to construct QC-LDPC codes with similar parameters to achieve $\omega_{LB}$ and the desired layer distance respectively. We take PCM 1 and PCM 2 as an example.

For PCM 1, $\omega_{LB}$ is not achieved at $L = 6$, and then we construct a PCM to achieve $\omega_{LB}$ based on Strategy 2; the desired layer distance $k = 2$ is not achieved at $L = L_{LB} = 12$, and then we construct a PCM to achieve $k$ based on Strategy 3. As a comparison, we construct a PCM based on Strategy 1. The base matrices of PCM 1 and PCMs constructed by Strategies 1–3 are denoted by $B_1$–$B_4$ with details given in (22)–(25), as shown at the bottom of the next page, respectively. All these base matrices have the following same parameters: $M = 5$, $N = 27$, $Z = 384$ and VN degree

---

In Table III, we use the bold face to highlight the $\omega(H_{T_C}^T)$ which is greater than $\omega_{LB}$. We use ‘−’ to indicate the case where Algorithm 2 cannot achieve the lower bound $\omega_{LB}$ within a preset running time limit. We can see that both algorithms can generally achieve $\omega_{LB}$ or differ by one. There exist cases (e.g., $L = 12$ for PCM 1) where $\omega_{LB}$ is achievable only for $S^* > 1$. This situation has never been considered in the literature. Moreover, there exist cases (e.g., $L = 8$ for PCM 3) where Algorithm 3 cannot achieve $\omega_{LB}$ while Algorithm 2 can, as they work in a greedy and enumerative fashion, respectively. However, there also exist cases (e.g., $L = 6$ for PCM 1) where Algorithm 2 may exceed the preset time limit due to large search space.

Table IV shows the partition results with desired layer distance for the 5G LDPC codes. We use $k$ to represent the desired layer distance and $L^*$ to represent the minimum number of layers that may have a scheme with desired layer distance. We use the bold face to highlight the $L^*$ which is greater than $L_{LB}$. We use ‘−’ to indicate the case where the algorithm cannot find a feasible solution with desired $k$ within a preset running time limit. Accordingly, we also list $L_{LB}$, which is the lower bound of $L^*$ given in Corollary 1.

We can see that in most cases, both algorithms can find a partition scheme with desired layer distance, where the minimum number of layers $L^*$ is slightly greater than or equal to $L_{LB}$. Since they work in an enumerative and greedy fashion, respectively, there exist cases (e.g., $k = 2$ for PCM 1) where Algorithm 2 can find a desired solution under a lower $L^*$ than that in Algorithm 3. In addition, there exists a case ($k = 4$ for PCM 5) where both algorithms cannot find a partition scheme with desired layer distance.

B. Decoding Performance

As can be seen from Tables III and IV, there exist cases that $\omega_{LB}$ or the desired layer distance is not achieved. For these cases, we can use the modified QC-PEG algorithm to construct QC-LDPC codes with similar parameters to achieve $\omega_{LB}$ and the desired layer distance respectively. We take PCM 1 and PCM 2 as an example.

For PCM 1, $\omega_{LB}$ is not achieved at $L = 6$, and then we construct a PCM to achieve $\omega_{LB}$ based on Strategy 2; the desired layer distance $k = 2$ is not achieved at $L = L_{LB} = 12$, and then we construct a PCM to achieve $k$ based on Strategy 3. As a comparison, we construct a PCM based on Strategy 1. The base matrices of PCM 1 and PCMs constructed by Strategies 1–3 are denoted by $B_1$–$B_4$ with details given in (22)–(25), as shown at the bottom of the next page, respectively. All these base matrices have the following same parameters: $M = 5$, $N = 27$, $Z = 384$ and VN degree...
distribution $\lambda(x) = \frac{1}{27}x + \frac{3}{27}x^2 + \frac{21}{27}x^3 + \frac{1}{27}x^4 + \frac{1}{27}x^5$,
where $\lambda_i x^i$ indicates that the fraction of degree-$i$ VNs is $\lambda_i$. Moreover, the rightmost $5 \times 4$ submatrices of $B_1$–$B_4$ have the same structure (the positions of “–1” are the same).

For PCM 2, $\omega_{LB}$ is not achieved at $L = 4$, and then we construct a PCM to achieve $\omega_{LB}$ based on Strategy 2; the desired layer distance $k = 2$ is not achieved at $L = L_{LB} = 8$, and then we construct a PCM to achieve $k$ based on Strategy 3. As a comparison, we construct a PCM based on Strategy 1. The base matrices of PCM 2 and PCMs constructed by Strategies 1–3 are denoted by $B_5$–$B_8$ with details given in (26)–(29), as shown at the bottom of the page, respectively. All these base matrices have the following same parameters: $M = 4, N = 8, Z = 384$ and VN degree distribution $\lambda(x) = \frac{3}{8}x^2 + \frac{5}{8}x^3$. Moreover, the rightmost $4 \times 3$ submatrices of $B_5$–$B_8$ also have the same structure.

We investigate the performance of $B_1$–$B_8$ via Monte-Carlo simulation. We consider binary phase-shift keying (BPSK) transmission over the additive white Gaussian noise (AWGN) channels. Both the layered SPA [8] and the layered MSA [10] are adopted at the receiver. For a given PCM and a given number of layers (assume each layer has the same number of rows), different partition schemes only differ at the processing order of rows. Note that the connections between CNs and VNs are unchanged. Therefore, the error correction performance and average number of decoding iterations should

\[
B_1 = \begin{bmatrix}
307 & 19 & 50 & 369 & -1 & 181 & 216 & -1 & -1 & 317 & 288 & 109 & 17 & 357 & -1 & 215 & 106 & -1 & 242 & 180 & 330 & 446 & 1 & 0 & -1 & -1 & -1
\end{bmatrix}
\]

\[
B_2 = \begin{bmatrix}
321 & 87 & 0 & 325 & 199 & 153 & 56 & -1 & 132 & 305 & 231 & 341 & 212 & -1 & 304 & 300 & 271 & -1 & 39 & 357 & -1 & -1 & 0 & -1
\end{bmatrix}
\]

\[
B_3 = \begin{bmatrix}
332 & 181 & -1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0
\end{bmatrix}
\]

\[
B_4 = \begin{bmatrix}
126 & -1 & -1 & 205 & 275 & -1 & 44 & -1 & 120 & -1 & 148 & 263 & -1 & 131 & 200 & 371 & -1 & 176 & 255 & -1 & 183 & -1 & 379 & 380 & -1 & -1 & -1
\end{bmatrix}
\]

\[
B_5 = \begin{bmatrix}
205 & 302 & 328 & -1 & 332 & 256 & 161 & 267 & 160 & 63 & 129 & -1 & -1 & 200 & 88 & 53 & -1 & 131 & 240 & 263 & 13 & -1 & -1 & 0 & 0 & -1
\end{bmatrix}
\]

\[
B_6 = \begin{bmatrix}
276 & 87 & -1 & 0 & 275 & -1 & 199 & 153 & 56 & -1 & 132 & 305 & 231 & 341 & 212 & -1 & 304 & 300 & 271 & -1 & 39 & 357 & -1 & -1 & 0 & -1
\end{bmatrix}
\]

\[
B_7 = \begin{bmatrix}
126 & -1 & -1 & 205 & 275 & -1 & 44 & -1 & 120 & -1 & 148 & 263 & -1 & 131 & 200 & 371 & -1 & 176 & 255 & -1 & 183 & -1 & 379 & 380 & -1 & -1 & -1
\end{bmatrix}
\]

\[
B_8 = \begin{bmatrix}
332 & 181 & -1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0
\end{bmatrix}
\]
keep (almost) the same. Due to this reason, for the sake of simplicity, we perform layered decoding by taking each row of the PCMs as a layer and do not permute the PCMs’ rows. The maximum number of decoding iterations is set to 10. At least 100 frame errors are collected for each simulated signal-to-noise ratio point.

Fig. 2 and Fig. 3 show the frame error rate (FER) and the average number of decoding iterations of different codes. It can be seen that the codes constructed by Strategies 1-3 all outperform 5G LDPC codes under the same decoding algorithm. Meanwhile, codes constructed by Strategies 1–3 have comparable performance, even if Strategies 1–3 result in decreasing number of candidate codes. This implies that applying Criterion 1 in Strategy 2 or Strategy 3 to other construction methods may not affect the decoding performance of the constructed codes as well.

VIII. CONCLUSION

In this paper, we first formulated the PCM partitioning as an optimization problem for reducing the hardware complexity, which aims to minimize the maximum column weight of each layer while maintaining a block cyclic shift property among different layers. In particular, we derived all the feasible solutions and proposed a tight lower bound \( \omega_{LB} \) for the minimum possible maximum column weight to evaluate the quality of a solution. Second, we reduced the computation delay of the layered decoding by considering the data dependency issue between consecutive layers. More specifically, we illustrated how to obtain the optimal solutions with desired layer distance from those achieving the minimum value of the lower bound \( \omega_{LB} = 1 \). Third, we demonstrated that up-to-now, there exist no algorithms to find an optimal solution with polynomial time complexity and alternatively proposed both greedy and enumerative partition algorithms. Fourth, we modified the QC-PEG algorithm to directly construct PCMs that have a straightforward partition scheme to achieve \( \omega_{LB} = 1 \) or the desired layer distance. Finally, we evaluated the performance of the proposed enumerative and greedy algorithms for partitioning the PCMs of the 5G LDPC codes. For the cases where \( \omega_{LB} \) or the desired layer distance is not achievable, we used the modified QC-PEG algorithm to construct two QC-LDPC codes with the same code parameters as that of two 5G LDPC codes to achieve \( \omega_{LB} \) and desired layer distance, respectively. Simulation results show that the constructed codes have better error correction performance and achieve less average number of iterations than the underlying 5G LDPC codes.

REFERENCES

[1] T. Lu, X. He, P. Kang, and X. Tang, “Parity check matrix partitioning for layered decoding of QC-LDPC codes,” in Proc. IEEE Globecom Workshops (GC Wkshps), Dec. 2021, pp. 1–6.
[2] R. G. Gallager, “Low-density parity-check codes,” *IRE Trans. Inf. Theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962.

[3] NR: Multiplexing and Channel Coding, document TS 38.212, Release 15, 3GPP, 2017.

[4] P. V. Bhuvaneshwari and C. Tharini, “Review on LDPC codes for big data storage,” *Wireless Pers. Commun.*, vol. 117, no. 2, pp. 1601–1625, Mar. 2021.

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

[6] F. R. Kschischang and B. J. Frey, “Iterative decoding of compound codes by probability propagation in graphical models,” *IEEE J. Sel. Areas Commun.*, vol. 16, no. 2, pp. 219–230, Feb. 1998.

[7] R. M. Tanner, “A recursive approach to low complexity codes,” *IEEE Trans. Inf. Theory*, vol. IT-27, no. 5, pp. 533–547, Sep. 1981.

[8] M. M. Mansour and N. R. Shanbhag, “High-throughput LDPC decoders,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no. 6, pp. 976–996, Dec. 2003.

[9] D. E. Hocevar, “A reduced complexity decoder architecture via layered decoding of LDPC codes,” in *Proc. IEEE Workshop Signal Process. Syst.*, Oct. 2004, pp. 107–112.

[10] K. Zhang, X. Huang, and Z. Wang, “High-throughput layered decoder implementation for quasi-cyclic LDPC codes,” *IEEE J. Sel. Areas Commun.*, vol. 27, no. 6, pp. 985–994, Aug. 2009.

[11] J. Zhang and M. P. C. Fossorier, “Shuffled iterative decoding,” *IEEE Trans. Commun.*, vol. 53, no. 2, pp. 209–213, Feb. 2005.

[12] C. A. Aslam, Y. L. Guan, K. Cai, and G. Han, “Informed fixed scheduling for faster convergence of shuffled belief-propagation decoding,” *IEEE Commun. Lett.*, vol. 21, no. 1, pp. 32–35, Jan. 2017.

[13] S.-H. Kang and I.-C. Park, “Loosely coupled memory-based decoding architecture for low density parity check codes,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 53, no. 5, pp. 1045–1056, May 2006.

[14] K. K. Gunnam, G. S. Choi, M. B. Yeary, and M. Atiquzzaman, “VLSI architectures for layered decoding for irregular LDPC codes of WiMax,” in *Proc. IEEE Int. Conf. Commun.*, Jun. 2007, pp. 4542–4547.

[15] C. Studer, N. Preys, C. Roth, and A. Burg, “Configurable high-throughput decoder architecture for quasi-cyclic LDPC codes,” in *Proc. 42nd Asilomar Conf. Signals, Syst. Comput.*, Oct. 2008, pp. 1137–1142.

[16] Y.-L. Ueng and C.-C. Cheng, “A fast-convergence decoding method and memory-efficient VLSI decoder architecture for irregular LDPC codes in the IEEE 802.16e standards,” in *Proc. IEEE 66th Veh. Technol. Conf.*, Sep. 2007, pp. 1255–1259.

[17] J. J. Zhang, Z. Wang, and Y. Liu, “High-throughput layered LDPC decoding architecture,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 4, pp. 582–587, Apr. 2009.

[18] Z. Cui, Z. Wang, X. Zhang, and Q. Jia, “Efficient decoder design for high-throughput LDPC decoding,” in *Proc. IEEE Asia Pacific Conf. Circuits Syst.*, Nov. 2008, pp. 1640–1643.

[19] A. Hasani, L. Lopacinski, S. Büchner, J. Nolte, and R. Kraemer, “A modified shuffling method to split the critical path delay in layered decoding of QC-LDPC codes,” in *Proc. IEEE Int. Conf. Commun.*, Jun. 2009, pp. 1–6.

[20] A. Hasani, L. Lopacinski, and R. Kraemer, “Reduced-complexity decoding implementation of QC-LDPC codes with modified shuffling,” *EURASIP J. Wireless Commun. Netw.*, vol. 2021, no. 1, pp. 1–14, Dec. 2021.

[21] M. P. C. Fossorier, “Quasi-cyclic low-density parity-check codes from circulant permutation matrices,” *IEEE Trans. Inf. Theory*, vol. 50, no. 8, pp. 1788–1793, Aug. 2004.

[22] T. Grünert, S. Irnich, H.-J. Zimmermann, M. Schneider, and B. Wulfhorst, “Finding all k-cliques in k-partite graphs, an application in textile engineering,” *Comput. Oper. Res.*, vol. 29, no. 1, pp. 13–31, Dec. 2002.

[23] M. Mirghorbani and P. Krokhmal, “On finding k-cliques in k-partite graphs,” *Optim. Lett.*, vol. 7, no. 6, pp. 1155–1165, Aug. 2013.

[24] P. V. Bhuvaneshwari and C. Tharini, “Review on LDPC codes for big data storage,” *Wireless Pers. Commun.*, vol. 117, no. 2, pp. 1601–1625, Mar. 2021.

[25] 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,” in *Proc. Asilomar Conf. Signals, Syst., Comput.*, Pacific Grove, CA, USA, Nov. 2004, pp. 1990–1994.

[26] X. He, L. Zhou, and J. Du, “PEG-like design of binary QC-LDPC codes based on detecting and avoiding generating small cycles,” *IEEE Trans. Commun.*, vol. 66, no. 5, pp. 1845–1858, May 2018.