A Weighted Block-by-Block Decoding Algorithm for CPM-QC-LDPC Code Using Neural Network

Zuohong Xu[1], Jiang Zhu[1], Zixuan Zhang[2] and Qian Cheng[1]
[1] College of Electronic Science and Engineering, National University of Defense Technology
Changsha, 410073 - China
[e-mail: zuohong.xu@outlook.com, jangzhu@nudt.edu.cn, chengqian14a@nudt.edu.cn]
[2] College of Computer, National University of Defense Technology
Changsha, 410073 - China
[e-mail:sssss976@outlook.com]
*Corresponding author: Jiang Zhu

Received November 15, 2017; revised February 2, 2018; accepted February 28, 2018; published August 31, 2018

Abstract

As one of the most potential types of low-density parity-check (LDPC) codes, CPM-QC-LDPC code has considerable advantages but there still exist some limitations in practical application, for example, the existing decoding algorithm has a low convergence rate and a high decoding complexity. According to the structural property of this code, we propose a new method based on a CPM-RID decoding algorithm that decodes block-by-block with weights, which are obtained by neural network training. From the simulation results, we can conclude that our proposed method not only improves the bit error rate and frame error rate performance but also increases the convergence rate, when compared with the original CPM-RID decoding algorithm and scaled MSA algorithm.

Keywords: CPM-QC-LDPC code; CPM-RID decoding algorithm; neural network; weights; bit error rate; frame error rate; convergence rate

1. Introduction

Among many various types of channel codes, low-density parity-check (LDPC) code [1] is currently one of the most promising one. It has been adopted in many applications as the standard code for better performance, for example, the IEEE 802.11n, IEEE 802.16e, IEEE 802.20 standards and satellite, wireless and optical communication systems, hard disk drives and flash memories [2–7]. Therefore, designing the most efficient type of LDPC code is an essential issue.

According to much of the existing literature [8–13], quasi-cyclic (QC) LDPC code is considered the most preferred type for designing LDPC code. QC-LDPC code is given by the null space of an array ($\mathbf{H}$) of sparse circulant matrices of the same size over a finite field (binary or non-binary) [2]. In some constructions of QC-LDPC code, the sparse circulant matrices in $\mathbf{H}$ are all circulant permutation matrices (CPMs). Such types of LDPC code are abbreviated as CPM-QC-LDPC code.

In general, CPM-QC-LDPC code has many advantages over any other existing types of LDPC code, especially in terms of encoding and decoding complexity of hardware implementation. In the process of decoding, the CPM-structure of $\mathbf{H}$ allows some novel techniques to reduce the size of hardware resources and the complexity of hardware implementation. However, there are some restrictions for the existing decoding algorithm of CPM-QC-LDPC code, such as low convergence rate, relatively high bit error rate (BER) and high decoding complexity, which have aroused our great interest in improving these aspects.

The classical iterative decoding algorithm of LDPC code is sum-product algorithm (SPA), which can be represented based on a Tanner graph. To reduce its high hardware implementation complexity, many approximations to SPA algorithm have been proposed. One well-known method is the scaled min-sum algorithm (scaled MSA), but the hardware implementation complexity is still too high. In reference [11], authors presented a decoding scheme called revolving iterative decoding (RID) for CPM-QC-LDPC code, which can efficiently reduce the hardware implementation complexity. However, it needs to perform specific column and row permutations to transform the parity check matrix, which changes the structure of CPMs. Another efficient algorithm is proposed in [14–16], in which authors take the advantages of the structural properties of CPM, and improve the revolving iterative decoding (CPM-RID) algorithm, which can also reduce the decoding complexity in an efficient way. As far as we know, it is rather rare to find the applications of a neural network (NN) in decoding CPM-QC-LDPC code. Thus, we are the first to propose combining the CPM-RID decoding algorithm with a NN to decode the CPM-QC-LDPC code.

The traditional approach of a NN [17,18] is to train the neural network with a large dataset containing all codewords, and the output is expected to get the correct codeword from the noisy channel. Unfortunately, the number of codewords tends to be massive. For example, for a linear block code of length 100 with rate 0.5, there are 2^{50} different codewords; therefore, it is impossible to fully train a NN in practical implementation. To overcome this issue, recent literature [19,20] established a NN based on a Tanner graph and an iterative decoding algorithm, and uses an all-zero codeword to train and assign weights to the edges of the Tanner graph for linear codes. The simulation results prove its reasonability but the training procedure is too complicated and of high complexity. Furthermore, literature [21] has constructed a new multi-layer model to iteratively decode LDPC code, but this only applies to short code.

In this paper, we use NN to train the weights in a Tanner graph, make some modifications to the original CPM-RID algorithm and combine them together as our proposed method. Firstly, we construct a simple activation function stimulated by [21] for NN to train and assign weights to the edges of the Tanner graph. The training procedure of our model is much simpler than that in Reference [19,20] and it does not need a large training dataset. Then, we modify the original CPM-RID algorithm by directly dividing the parity check matrix into several submatrices, which can reduce the decoding delay and increase the convergence rate by adapting a certain decoding scheme, like serial mechanism.

The rest of the paper is arranged as follows. Section 2 gives an overview of the general procedure of constructing the CPM-QC-LDPC code and the original CPM-RID decoding algorithm. Our proposed decoding algorithm is introduced and explained in Section 3. Then, Section 4 conducts some necessary experiments to verify its improved decoding performance and faster convergence rate than that of the original CPM-RID algorithm and scaled MSA algorithm. Then we also give some analysis about the decoder complexity and computational complexity. In the final section, we make a conclusion about our proposed algorithm.

2. Review on CPM-QC-LDPC Code and CPM-RID Decoding Algorithm

2.1 A General Construction of CPM-QC-LDPC Code

In this subsection, we overview the general method of how to construct the CPM-QC-LDPC code [11,13,14].

Let GF(q) be a Galois field and α be a primitive element. All the elements of GF(q) can be represented by using the powers of $\alpha : \{\alpha^{-\infty} = 0, \alpha^0 = 1, \alpha^1, ..., \alpha^{q-2}\}$. We construct two arbitrary sets $S_1 = \{\alpha^{i_0}, \alpha^{i_1}, ..., \alpha^{i_m - 1}\}$ and $S_2 = \{\alpha^{j_0}, \alpha^{j_1}, ..., \alpha^{j_{n-1}}\}$ with no intersection and make sure all the elements are contained in these two sets. Next, we use S_1 and S_2 to form a base matrix $\mathbf{B} = [\alpha^{i_k} + \alpha^{j_l}]_{0 \le k < m, 0 \le l < n}$. Notice that the base matrix $\mathbf{B}$ has good structural properties in making the $\mathbf{H}$ matrix of LDPC, which has no length-4 cycle. By using the elements $\alpha^j (0 \le j < q-1)$ of $\mathbf{B}$, we can produce a vector $(0,0,\dots,1,\dots,0)$ with length of $(q-1)$, where its j-th entry equals to 1. After cyclically shifting all the entries right by one place each time, we obtain a new vector for each shift. Thereby, we will have $(q-1)$ vectors which form a $(q-1)\times(q-1)$ circulant permutation matrix $\mathbf{A}$ (CPM), denoted by $\mathbf{A}(\alpha^j)$. Noticeably, the j-th entry of the top row is 1, and all rows and columns are cyclical shifts of the previous row and column, respectively.

If we replace each entry of $\mathbf{B}$ by its corresponding CPM $\mathbf{A}(\alpha^j)$, and let $\mathbf{H}$ be an $m \times n$ array of CPMs with the size of $(q-1)\times(q-1)$, that is, $\mathbf{H}$ is an $m(q-1)\times n(q-1)$ matrix over GF(2). Then $\mathbf{H}$ is an $M \times N$ matrix which can be expressed as

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_0 \\ \mathbf{H}_1 \\ \vdots \\ \mathbf{H}_{m-1} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{0,0} & \mathbf{A}_{0,1} & \cdots & \mathbf{A}_{0,n-1} \\ \mathbf{A}_{1,0} & \mathbf{A}_{1,1} & \cdots & \mathbf{A}_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{m-1,0} & \mathbf{A}_{m-1,1} & \cdots & \mathbf{A}_{m-1,n-1} \end{bmatrix}, \tag{1}$$

where the submatrix $\mathbf{H}_i$ is a $(q-1)\times n(q-1)$ matrix containing n CPMs and each $\mathbf{A}_{i,j}$ represents a $(q-1)\times(q-1)$ CPM. It is easy to conclude that $\mathbf{H}_i$ also possesses the property of cyclical shifts like CPMs but not the same, since $\mathbf{H}_i$ has more than one CPM (each $\mathbf{H}_i$ has n CPMs). This structure allows it to cyclically shift all entries of one row only within their corresponding CPM for one time (we call this process cyclical shift within the sections).

2.2 The CPM-QC-LDPC Decoding Algorithm

Let $\mathbf{H}$ be an $m\times n$ array of CPMs with the size of $(q-1)\times(q-1)$ and $\mathbf{H}_i$, which contains n CPMs with the size of $(q-1)\times(q-1)$, be the i-th row-block of $\mathbf{H}$. Let $\mathbf{h}_{k,s}=(\mathbf{h}_{k,s,0},\mathbf{h}_{k,s,1},...,\mathbf{h}_{k,s,n-1})$ represent the s-th row of $\mathbf{H}_k$ with n sections. Each section $\mathbf{h}_{k,s,i},\ _{i\in\{0,1,...,n-1\}}$ contains $(q-1)$ elements (the s-th row of the i-th CPM). Thus, the expression of $\mathbf{H}_\text{k}$ can be represented as

$$\begin{aligned}\mathbf{H}_k &= \begin{bmatrix}\mathbf{A}_{k,0} & \mathbf{A}_{k,1} & \cdots & \mathbf{A}_{k,n-1}\end{bmatrix} \\ &= \begin{bmatrix}\mathbf{h}_{k,0,0}, & \mathbf{h}_{k,0,2} & \cdots & \mathbf{h}_{k,0,n-1} \\ \mathbf{h}_{k,1,0}, & \mathbf{h}_{k,1,2} & \cdots & \mathbf{h}_{k,1,n-1} \\ & & \vdots & \\ \mathbf{h}_{k,q-2,0}, & \mathbf{h}_{k,q-2,2} & \cdots & \mathbf{h}_{k,q-2,n-1}\end{bmatrix}.\end{aligned} \tag{2}$$

It is easy to see that if we cyclically shift all the n sections of $\mathbf{h}_{k,s}$ one place to the right within the sections, we will get the $(s+1)$-th row $\mathbf{h}_{k,s+1}$ of $\mathbf{H}_k$. Thus, all rows of $\mathbf{H}_k$ can be obtained by only cyclically shifting the first row of $\mathbf{H}_k$ within the sections. Similarly, all submatrices $\mathbf{H}_0,\mathbf{H}_1,...,\mathbf{H}_{m-1}$ can also be obtained by only cyclically shifting the first rows of all submatrices within the sections.

Let $\mathbf{H}_0^*$ be the $m\times n(q-1)$ matrix consisting of all first rows $\mathbf{h}_{0,0},\mathbf{h}_{1,0},...,\mathbf{h}_{m-1,0}$ of arrays $\mathbf{H}_0,\mathbf{H}_1,...,\mathbf{H}_{m-1}$. By cyclically shifting the $\mathbf{H}_0^*$ within the sections $q-2$ times, we can obtain the whole parity check matrix $\mathbf{H}$.

This structure property is of great benefit to the practical applications of CPM-QC-LDPC codes [14] in the sense that we can decode based on the submatrix $\mathbf{H}_0^*$ alone. Every time we decode the matrix $\mathbf{H}_0^*$, the reliabilities of the received symbols are updated with a chosen reliability updating algorithm. Then, the reliability vector and the received sequence are cyclically shifted and used as the input information to carry out the next decoding. After decoding the whole $\mathbf{H}$ matrix, a hard-decision vector $\mathbf{z}$ is formed based on the reliabilities of the decoded symbols. By computing the syndrome $\mathbf{s}=\mathrm{mod}(\mathbf{z}\cdot\mathbf{H}^T,2)$, we can decide whether vector $\mathbf{z}$ is the codeword. If it equals to $\mathbf{0}$, the decoding process stops and $\mathbf{z}$ is the codeword, otherwise the decoding process continues until the codeword is found or it exceeds the pre-set iteration number.

From the structure of $\mathbf{H}_0^*$, we can conclude that the number of rows in $\mathbf{H}_0^*$ is only $1/(q-1)$ of $\mathbf{H}$, so in the procedure of implementing the hardware decoder, the size of message processing units for check nodes as well as the number of wires that connect to the check nodes and variable nodes are reduced to $1/(q-1)$ of those in $\mathbf{H}$.

3. The Proposed Decoding Algorithm

3.1 The Establishment of Training Model

At first, the transmitted information vector is entered into variable nodes, and all weights of edges in the Tanner graph are set to 1. Our data is created by transmitting the all-zero codeword on the additive white Gaussian noise (AWGN) channel with binary phase shift keying (BPSK) modulation.

To start the NN training, we also need to set various E_b / N_0, where E_b represents the average energy per information bit and N_0 represents the one-sided power spectral density. There are two extreme cases in NN training: when $E_b / N_0 \to \infty$, it means NN trains the transmitted information without channel noise, so it can learn the code structure; when $E_b / N_0 \to 0$, it means NN only trains the channel noise, without learning the code structure. This clearly indicates there will be some proper E_b / N_0 that we can choose between these two cases. In reference [22], authors have performed some researh about how to choose the proper E_b / N_0. In our proposed method, E_b / N_0 of 1 dB is chosen for NN training, so we can both learn the code structure and channel noise at the same time.

Then, the inputs are multiplied by the corresponding weights and propagated through a nonlinear activation function, which is stimulated by the process of computing syndrome $\mathbf{s} = (s_1, s_2, ..., s_{M-1}), s_i \in \{0,1\}$ to check whether the decoded vector $\mathbf{z} = (z_0, z_1, ..., z_{N-1}), z_i \in \{0,1\}$ is the codeword. Concisely, at the end of every iteration of the decoding, we usually compute the syndrome $\mathbf{s} = \mathrm{mod}(\mathbf{z} \cdot \mathbf{H}^T, 2)$, if $\mathbf{s} = \mathbf{0}$, we may think $\mathbf{z}$ is the codeword. For example, if we have a parity check matrix $\mathbf{H}$, which can be represented in (3),

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}, \tag{3}$$

the process of computing $\mathbf{s}$ can be described by

$$\mathbf{s} = (s_1, s_2, s_3, s_4) = (z_1, z_2, z_3, z_4, z_5, z_6) \cdot \mathbf{H}^T = (z_1, z_2, z_3, z_4, z_5, z_6) \cdot \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}. \tag{4}$$

Then, $\mathbf{s} = \mathrm{mod}(\mathbf{s}, 2)$. From the expression above we can see $s_1 = \mathrm{mod}(z_1 + z_4, 2)$. To make it clear, we list the possible values in **Table 1**,

Table 1. An illustration of possible values

z_1	z_2	s_1
0	0	0
1	1	0
0	1	1
1	0	1

From **Table 1** we can see only when z_1 equals to z_2, s_1 equals to 0; when z_1 does not equal to z_2, s_1 equals to 1. This property is very similar with XOR gate which is a digital logic gate giving a true output when the number of true inputs is odd. Therefore, we can use an expression to represent this property, which is depicted as follows,

$$s_1 = z_1 \oplus z_2 = z_1(1-z_2) + z_2(1-z_1)\,, \tag{5}$$

where $\oplus$ is the logic symbol representing the addition modulo 2 operation.

Similarly, we can also compute $s_2 = \mathrm{mod}(z_2 + z_3 + z_5, 2)$, and the possible values are shown as follows,

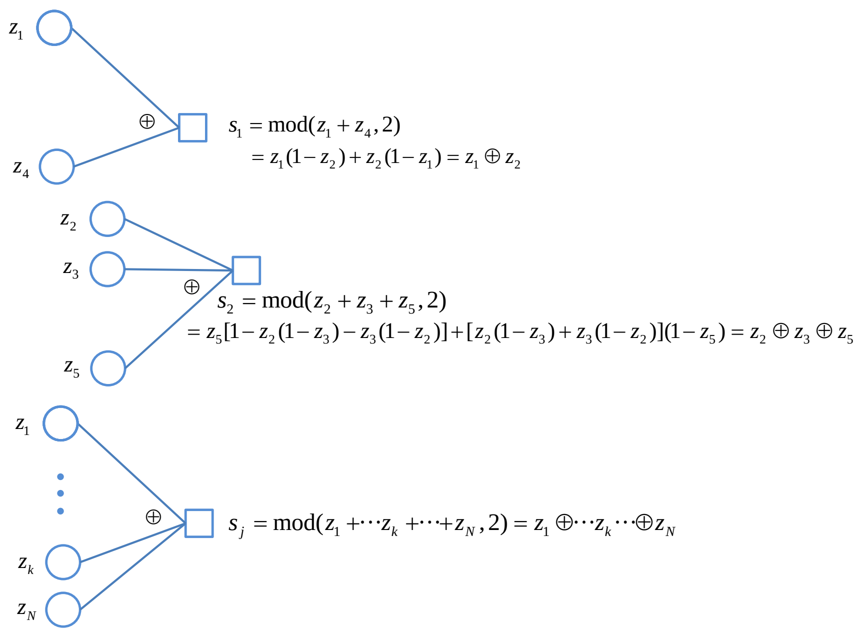
Table 2. An illustration of possible values

z_2	z_3	z_5	s_2
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

From **Table 2**, it is easy to conclude an expression for s_2 described as follows,

$$s_2 = z_2 \oplus z_3 \oplus z_5 = z_5[1 - z_2(1-z_3) - z_3(1-z_2)] + [z_2(1-z_3) + z_3(1-z_2)](1-z_5)\,. \tag{6}$$

Therefore, the process of computing the syndrome $\mathbf{s}$ can be depicted as follows,



z_1
z_4
$\oplus$
$s_1 = \mathrm{mod}(z_1 + z_4, 2)$
$= z_1(1-z_2) + z_2(1-z_1) = z_1 \oplus z_2$
z_2
z_3
z_5
$\oplus$
$s_2 = \mathrm{mod}(z_2 + z_3 + z_5, 2)$
$= z_5[1 - z_2(1-z_3) - z_3(1-z_2)] + [z_2(1-z_3) + z_3(1-z_2)](1-z_5) = z_2 \oplus z_3 \oplus z_5$
z_1
z_k
z_N
$\oplus$
$s_j = \mathrm{mod}(z_1 + \cdots z_k + \cdots + z_N, 2) = z_1 \oplus \cdots z_k \cdots \oplus z_N$

Fig. 1. An illustration of computing the syndrome

From the analysis above we can know the process of computing the syndrome is in binary domain consisting of 0 and 1. Stimulated by the analysis above, we extend the XOR function to real number field to obatin the activation function f_{output} (and we call this function XOR function). More specifically, if there are only two inputs x and y, the function is defined as follows

$$f_{output}(x,y) = x \oplus y = x(1-y) + y(1-x). \tag{7}$$

At that time, the values of x and y range from $(-\infty,+\infty)$, which is decided by the transmitted codeword, transmission energy and channel noise (signal to noise ratio). Then, we depict the f_{output} as follows,

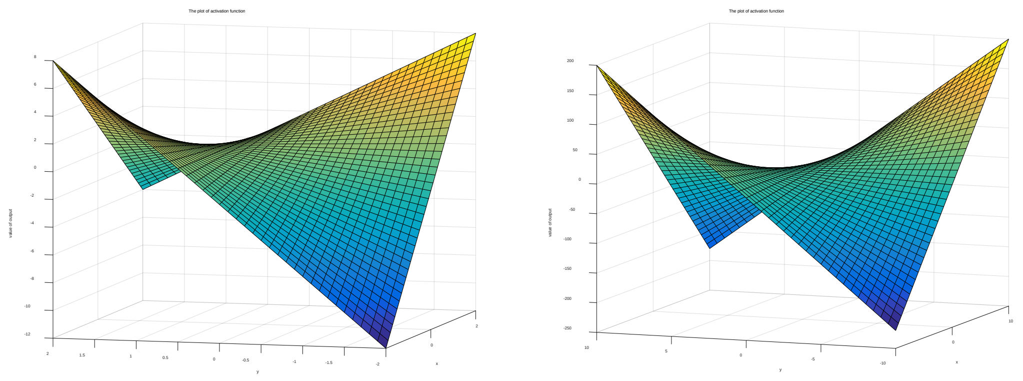


Fig. 2. The plot of XOR function

From the **Fig. 2** above we can easily see if $x, y \in [-2,2]$, the range of $f_{output}(x,y)$ is $[-12,8]$; if $x, y \in [-10,10]$, the range of $f_{output}(x,y)$ is $[-250,200]$. Theoretically, the range of $f_{output}(x,y)$ is from $-\infty$ to $+\infty$, but in practical application, it always has borders because x and y are of low probability to be large numbers (the value is related to the codeword and signal to noise ratio).

If there are three elements, activation function can be expressed as follows,

$$\begin{aligned} f_{output}(x,y,z) &= x \oplus y \oplus z \\ &= f_{output}(x,y) \oplus z \\ &= f_{output}(x,y)(1-z) + z\left[1 - f_{output}(x,y)\right] \\ &= \left[x(1-y) + y(1-x)\right](1-z) + z\left[1 - x(1-y) - y(1-x)\right]. \end{aligned} \tag{8}$$

Therefore, it is easy to conclude that if there are multiple inputs, the function can be written as

$$f_{output}(x,y,\cdots,l) = x \oplus y \oplus \cdots l = ((f_{output}(x,y) \oplus k)\cdots) \oplus l\,. \tag{9}$$

In our paper, we use induction method to obtain the expression of f_{output}. Assume the size of the input set $\{x_1, x_2, ..., x_N\}$ is N, the function f_{output} can be written in terms of polynomials as

$$\begin{aligned} f_{output}(x_1,\cdots,x_N) &= x_1 \oplus x_2 \oplus \cdots \oplus x_N \\ &= (f_{output}(x_1,x_2) \oplus x_3) \oplus \cdots \oplus x_N \\ &= (-2)^0 \sum \tilde{C}_N^1 x_i + (-2)^1 \sum \tilde{C}_N^2 x_i x_j \\ &\quad + (-2)^2 \sum \tilde{C}_N^3 x_i x_j x_k + \cdots \\ &\quad + (-2)^{N-1} \sum \tilde{C}_N^N x_1 x_2 \cdots x_N, \end{aligned} \tag{10}$$

where $\tilde{C}_N^i$ means the combination of all i elements in N elements.

We show the proof process in the Apendix.

For example, assume the input set has four elements $\{x_1,x_2,x_3,x_4\}$, then $\sum \tilde{C}_4^1 x_i$ means $x_1+x_2+x_3+x_4$, $\sum \tilde{C}_4^2 x_i x_j$ means $x_1x_2+x_1x_3+x_1x_4+x_2x_3+x_2x_4+x_3x_4$, $\sum \tilde{C}_4^3 x_i x_j x_k$ means $x_1x_2x_3+x_1x_2x_4+x_2x_3x_4+x_1x_3x_4$ and $\sum \tilde{C}_4^4 x_1x_2x_3x_4$ means $x_1x_2x_3x_4$, thus the expression of f_{output} is

$$\begin{aligned} f_{output}(x_1,x_2,x_3,x_4) = {} & (-2)^0(x_1+x_2+x_3+x_4) \\ & +(-2)^1(x_1x_2+x_1x_3+x_1x_4+x_2x_3+x_2x_4+x_3x_4) \\ & +(-2)^2(x_1x_2x_3+x_1x_3x_4+x_1x_2x_4+x_2x_3x_4) \\ & +(-2)^3 x_1x_2x_3x_4. \end{aligned} \tag{11}$$

Using the nonlinear activation function f_{output} for every check node, we can obtain the corresponding output $\mathbf{o}=[o_j]_{j=0,1..,m(q-1)-1}$, where o_j means the output of the j-th check node. For example, if the parity check matrix is the same as (3), we can obtain $\mathbf{o}$ by computing $o_0 = f_{output}(x_1,x_4)$, $o_1 = f_{outpu}(x_2,x_3,x_5)$, $o_2 = f_{output}(x_1,x_5,x_6)$ and $o_3 = f_{output}(x_3,x_6)$. After we have the output $\mathbf{o}$, we should also need to define a specific loss function E to find the optimal weights. The most common loss functions are the mean squared error function (MSE) and the binary cross-entropy function (BCE), which can be defined as

$$E_{\mathrm{MSE}} = \frac{1}{M}\sum_i \left(b_i - \hat{b}_i\right)^2, \tag{12}$$

$$E_{\mathrm{BCE}} = -\frac{1}{M}\sum_i \left[b_i \ln\left(\hat{b}_i\right) + (1-b_i)\ln\left(1-\hat{b}_i\right)\right], \tag{13}$$

where b_i is the i-th expected value and $\hat{b}_i$ is the i-th estimate value, M is the sample number. In our proposed method, we adapt the MSE function as loss function, depicted as

$$E = \frac{1}{M}\sum_{j=0}^{M-1}(e_j)^2 = \frac{1}{M}\sum_{j=0}^{M-1}(0-o_j)^2 = \frac{1}{M}\sum_{j=0}^{M-1}(o_j)^2, \tag{14}$$

where the expected output is $(0,0,\ldots,0)$, e_j is the difference between the j-th expected value and the actual value o_j.

Then, we can use the XOR function and the loss function to compute the optimal weights of the Tanner graph. From the expression of Equation (10), it is shown that function f_{output} is differential and qualified to be used in a gradient descent algorithm, which is a typical method used in NN. Therefore, we use the gradient descent algorithm to train and obtain the weights $\mathbf{W}=[w_{j,k}]_{j\in\{0,1,\ldots,m(q-1)-1\},k\in\{0,1,\ldots,n(q-1)-1\}}$ in the following way,

$$\Delta\mathbf{W}^{(l+1)} = -\mu\frac{\partial E^{(l)}}{\partial \mathbf{W}^{(l)}}, \mathbf{W}^{(l+1)} = \mathbf{W}^{(l)} + \Delta\mathbf{W}^{(l+1)}, \tag{15}$$

here, the superscript $(l+1)$ represents the $(l+1)$-th iteration, μ is the learning rate, $\Delta\mathbf{W}$ is the variance value of $\mathbf{W}$ and E is the loss function. With the help of gradient descent algorithm, we can find the optimized weights of Tanner graph which can minimize the loss function E.

Finally, when it comes to the iterations that we preset, the training process stops and we will finally obtain the final weights of Tanner graph $\mathbf{W}$.

To make our training model more understandable, an example was given as follows. The parity check matrix $\mathbf{H}$ is

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}. \tag{16}$$

Its Tanner graph is shown in **Fig. 3**, where there are six variable nodes $\{\mathrm{VN}_0, \mathrm{VN}_1, ..., \mathrm{VN}_5\}$ and four check nodes $\{\mathrm{CN}_0, \mathrm{CN}_1, \mathrm{CN}_2, \mathrm{CN}_3\}$. In the first iteration, all weights of edges are set to 1. Then inputs multiply the corresponding weights and propagate through the nonlinear activation function f_{output}, where we obtain the output $\mathbf{o} = \{o_0^{(1)}, o_1^{(1)}, o_2^{(1)}, o_3^{(1)}\}$.

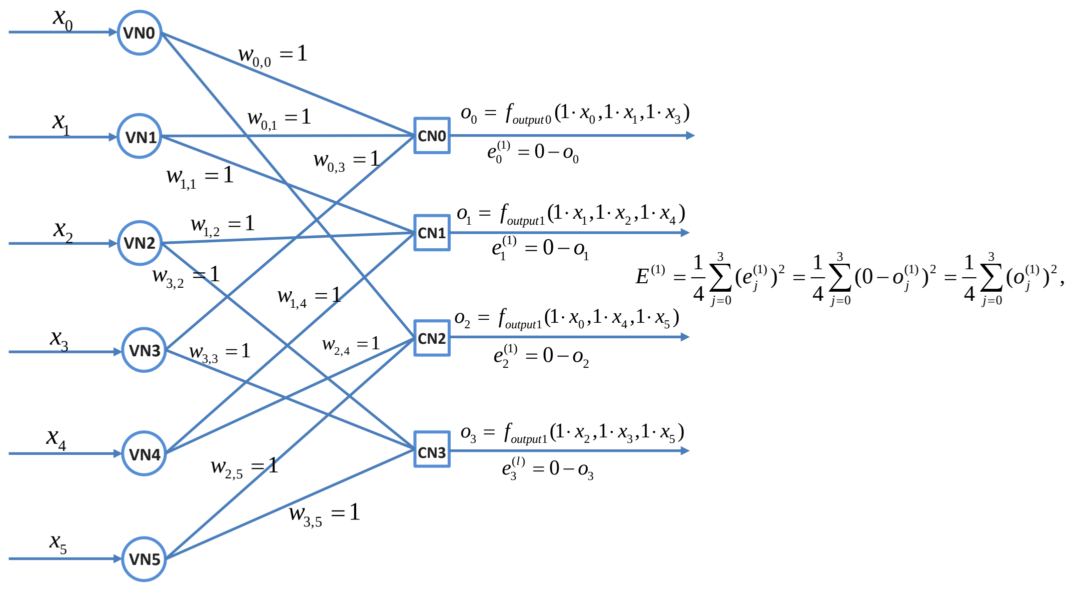

x_0
x_1
x_2
x_3
x_4
x_5
VN0
VN1
VN2
VN3
VN4
VN5
CN0
CN1
CN2
CN3
$w_{0,0} = 1$
$w_{0,1} = 1$
$w_{0,3} = 1$
$w_{1,1} = 1$
$w_{1,2} = 1$
$w_{3,2} = 1$
$w_{1,4} = 1$
$w_{2,4} = 1$
$w_{3,3} = 1$
$w_{2,5} = 1$
$w_{3,5} = 1$
$o_0 = f_{output0}(1 \cdot x_0, 1 \cdot x_1, 1 \cdot x_3)$
$e_0^{(1)} = 0 - o_0$
$o_1 = f_{output1}(1 \cdot x_1, 1 \cdot x_2, 1 \cdot x_4)$
$e_1^{(1)} = 0 - o_1$
$o_2 = f_{output1}(1 \cdot x_0, 1 \cdot x_4, 1 \cdot x_5)$
$e_2^{(1)} = 0 - o_2$
$o_3 = f_{output1}(1 \cdot x_2, 1 \cdot x_3, 1 \cdot x_5)$
$e_3^{(l)} = 0 - o_3$
$E^{(1)} = \frac{1}{4}\sum_{j=0}^{3}(e_j^{(1)})^2 = \frac{1}{4}\sum_{j=0}^{3}(0 - o_j^{(1)})^2 = \frac{1}{4}\sum_{j=0}^{3}(o_j^{(1)})^2,$

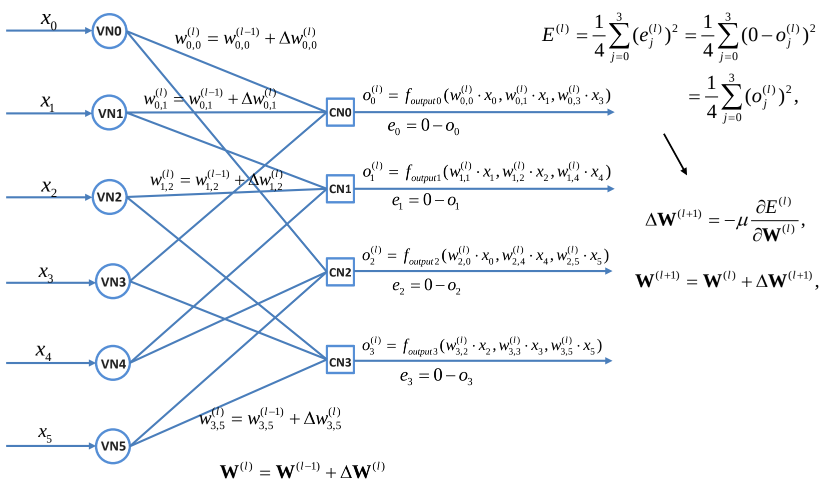
Fig. 3. An example to illustrate training model: the first iteration

Then we can obtain the MSE by computing

$$E^{(1)} = \frac{1}{4}\sum_{j=0}^{3}(e_j^{(1)})^2 = \frac{1}{4}\sum_{j=0}^{3}(0 - o_j^{(1)})^2 = \frac{1}{4}\sum_{j=0}^{3}(o_j^{(1)})^2. \tag{17}$$

According to Equation (15), after we obtain MSE, we can get $\Delta\mathbf{W}^{(2)}$ and make corrections to the weights $\mathbf{W}^{(2)} = \mathbf{W}^{(1)} + \Delta\mathbf{W}^{(2)}$. For the l-th iteration, we can see from **Fig. 4**.

In the l-th iteration, the inputs multiply the corresponding weights and propagate through the nonlinear activation function. Specifically, for the first check node, its output is $o_0^{(l)} = f_{output0}(w_{0,0}^{(l)} \cdot x_0, w_{0,1}^{(l)} \cdot x_1, w_{0,3}^{(l)} \cdot x_3)$. Based on $\mathbf{o}^{(l)} = \{o_0^{(l)}, o_1^{(l)}, o_2^{(l)}, o_3^{(l)}\}$, we can obtain the MSE $E^{(l)}$ and $\Delta\mathbf{W}^{(l+1)}$ according to Equations (14) and (15). Then we can make corrections to $\mathbf{W}^{(l+1)}$ and start the next iteration. When it comes to the iteration number that we preset, the training stops and we obtain the weights $\mathbf{W}$.



x_0
x_1
x_2
x_3
x_4
x_5
VN0
VN1
VN2
VN3
VN4
VN5
CN0
CN1
CN2
CN3
$w_{0,0}^{(l)} = w_{0,0}^{(l-1)} + \Delta w_{0,0}^{(l)}$
$w_{0,1}^{(l)} = w_{0,1}^{(l-1)} + \Delta w_{0,1}^{(l)}$
$w_{1,2}^{(l)} = w_{1,2}^{(l-1)} + \Delta w_{1,2}^{(l)}$
$w_{3,5}^{(l)} = w_{3,5}^{(l-1)} + \Delta w_{3,5}^{(l)}$
$\mathbf{W}^{(l)} = \mathbf{W}^{(l-1)} + \Delta \mathbf{W}^{(l)}$
$o_0^{(l)} = f_{output0}(w_{0,0}^{(l)} \cdot x_0, w_{0,1}^{(l)} \cdot x_1, w_{0,3}^{(l)} \cdot x_3)$
$e_0 = 0 - o_0$
$o_1^{(l)} = f_{output1}(w_{1,1}^{(l)} \cdot x_1, w_{1,2}^{(l)} \cdot x_2, w_{1,4}^{(l)} \cdot x_4)$
$e_1 = 0 - o_1$
$o_2^{(l)} = f_{output2}(w_{2,0}^{(l)} \cdot x_0, w_{2,4}^{(l)} \cdot x_4, w_{2,5}^{(l)} \cdot x_5)$
$e_2 = 0 - o_2$
$o_3^{(l)} = f_{output3}(w_{3,2}^{(l)} \cdot x_2, w_{3,3}^{(l)} \cdot x_3, w_{3,5}^{(l)} \cdot x_5)$
$e_3 = 0 - o_3$
$E^{(l)} = \frac{1}{4}\sum_{j=0}^{3}(e_j^{(l)})^2 = \frac{1}{4}\sum_{j=0}^{3}(0 - o_j^{(l)})^2 = \frac{1}{4}\sum_{j=0}^{3}(o_j^{(l)})^2,$
$\Delta \mathbf{W}^{(l+1)} = -\mu \frac{\partial E^{(l)}}{\partial \mathbf{W}^{(l)}},$
$\mathbf{W}^{(l+1)} = \mathbf{W}^{(l)} + \Delta \mathbf{W}^{(l+1)},$

Fig. 4. An example to illustrate the training model: the l-th iteration

In conclusion, it is worth mentioning that our activation function is simulated by computing the syndrome for every check node. Its aim is to utilize the output of the XOR function to alter the weights of the Tanner graph, which indirectly alters the value of the inputs by multiplying them in the process of decoding. In this way, it will reduce the influence by the channel noise and therefore, have a positive effect on the convergence rate and also the decoding performance. Concisely, if all inputs can be correctly decoded by decoding scheme, multiplying weights with inputs can make the decoding convergence faster to the codeword; if there are some abrupt errors occurring, multiplying weights with inputs can reduce the effects of errors. Consequently, the decoding performance will be improved.

3.2 Modification to the CPM-RID Decoding Algorithm

Now, we obtain the weights of the Tanner graph and assign them to the submatrix that we constructed. The construction method of the submatrix is the same as that in literature [14], where matrix $\mathbf{H}_0^*$ consists of all first rows $\mathbf{h}_{0,0}, \mathbf{h}_{1,0}, \dots, \mathbf{h}_{m-1,0}$ of m row blocks $\mathbf{H_0}, \mathbf{H_1}, \dots, \mathbf{H_{m-1}}$. In a similar manner, $\mathbf{H}_1^*$ is constructed of all second rows $\mathbf{h}_{0,1}, \mathbf{h}_{1,1}, \dots, \mathbf{h}_{m-1,1}$ and $\mathbf{H}_i^*$ is constructed by all $i+1$ rows $\mathbf{h}_{0,i}, \mathbf{h}_{1,i}, \dots, \mathbf{h}_{m-1,i}$. Therefore, we obtain $q-1$ matrix $\mathbf{H}_0^*, \mathbf{H}_1^*, \dots, \mathbf{H}_{q-2}^*$ and treat every matrix $\mathbf{H}_i^*$ as a block-layer.

During the process of decoding one submatrix $\mathbf{H}_i^*$, we choose a parallel mechanism to update the reliability information for check nodes and variable nodes, such as scaled MSA in our method. According to the conventional iterative decoding algorithm, the messages transmit along their corresponding edges to update the information, while in our proposed method, when messages transmit along their corresponding edges, they multiply the corresponding weights to update the information. For the decoding between various submatrices we adapt the serial mechanism, that is, we take the reliability vector for variable nodes as the input vector for the next submatrix $\mathbf{H}_{i+1}^{*}$ and carry on another decoding. Finally, after every m iterations, we compute the syndrome and check whether it successfully decodes.

3.3 Summary

We use $\mathbf{y}=(y_0,y_1,...,y_{n(q-1)-1})$ to represent the soft-decision received vector at the output of the receiver detector. Let $\mathbf{H}=[h_{t,j}]_{0\le t<m(q-1),0\le j<n(q-1)}$ denote the whole parity check matrix, $N(j)=\{t:0\le t<m(q-1),h_{t,j}=1\}$ denote all check nodes which are connected to the j-th variable node, and $M(t)=\{j:0\le j<n(q-1),h_{t,j}=1\}$ denote all variable nodes which are connected to the t-th check node. Let $\mathbf{Q}^{(i)}=(Q_0^{(i)},Q_1^{(i)},...,Q_{n(q-1)-1}^{(i)})$ denote the updated reliability information vector at the end of the i-th iteration for variable nodes, and $\mathbf{R}^{(i)}=(R_0^{(i)},R_1^{(i)},...,R_{m(q-1)-1}^{(i)})$ denote the updated reliability information vector at the end of the i-th iteration for check nodes. In scaled MSA-based reliability information updating algorithms, there is always an attenuation factor to reduce the error due to the approximation of the formula [15]. In our proposed algorithm, we use λ to represent the attenuation factor, which will be shown in Equation (20).

With the above notations, our proposed method is clearly explained below.

Proposed Method

Step 1: Train the weights of the training model using gradient descent algorithm and alter the weights using

$$\Delta\mathbf{W}^{(l+1)}=-\mu\frac{\partial E^{(l)}}{\partial\mathbf{W}^{(l)}},\mathbf{W}^{(l+1)}=\mathbf{W}^{(l)}+\Delta\mathbf{W}^{(l+1)}. \tag{18}$$

Step 2: When it comes to the training number we preset, the training process stops and saves the weights $\mathbf{W}=[w_{j,k}]_{j\in\{0,1,...,m(q-1)-1\},k\in\{0,1,...,n(q-1)-1\}}$. Check the weights and judge whether overfitting occurs. If it happens, go to Step 1 and adjust parameters μ to restart the training. If not, go to Step 3.

Step 3: Divide parity check matrix $\mathbf{H}$ into $q-1$ submatrices $\mathbf{H}_0^*,\mathbf{H}_1^*,...,\mathbf{H}_{q-2}^*$.

Step 4: Set $Q_j^{(0)}=y_j$ for all $0\le j<n(q-1)$ and $R_t^{(0)}=0$ for all $0\le t<m(q-1)$. Allocate the weights $\mathbf{W}=[w_{j,k}]_{j\in\{0,1,...,m(q-1)-1\},k\in\{0,1,...,n(q-1)-1\}}$ to the edges in the Tanner graph.

Step 5: Carry out the i-th iteration based on $\mathbf{H}_i^*$:

(1) If $i \bmod m\ne 0$, compute the reliability vector $\mathbf{Q}^{(i)}$ and $\mathbf{R}^{(i)}$:

$$R_t^{(i)}=\left[\prod_{j'\in M(t)\backslash j}\text{sign}(w_{i\bmod m,j'}\cdot Q_{j'}^{(i-1)})\right]\times\min_{j'\in M(t)\backslash j}\left(\left|w_{i\bmod m,j'}\cdot Q_{j'}^{(i-1)}\right|\right), \tag{19}$$

$$Q_j^{(i)}=Q_j^{(i-1)}+\sum\nolimits_{t\in N(j)}R_t^{(i)}\times\lambda, \tag{20}$$

Otherwise $i \bmod m=0$, then go to Step 6;

(2) Take reliability vector of i-th iteration as $(i+1)$-th input vector, and go to Step 5-(1).

Step 6: Compute the syndrome $\mathbf{s}=\mathbf{z}\cdot\mathbf{H}^T$. If $\mathbf{s}\ne\mathbf{0}$, $i\leftarrow i+1$, then go to Step 5; otherwise $\mathbf{s}=\mathbf{0}$, stop the process and output the codeword. Use the reliability vector to compute the hard decision vector $\mathbf{z}$.

It is worth mentioning that the parameters chosen for learning rate μ and training number are skillful in NN training. To choose proper values, we tend to perform many various experiments to choose the optimal parameters or some algorithms like greedy sequential methods. Such a kind of question is named hyperparameter optimization, which is well studied in [23]. In our paper, we do not take the optimization method into consideration because we mainly focus on the efficiency of our proposed method. Therefore, after many various experiments and comparing their results, we finally set learning rate 0.001 and training number 10000.

As for the parameter λ for computing the $Q_j^{(i)}$ (see in (20)) in our method, we also perform some experiments to find a proper value of λ. For example, we perform various experiments using scaled MSA algorithm with various values of λ on (720,360) CPM-QC-LDPC code which is constructed using the method described in [14]. The decoding iteration is 50 and the experimental results are shown as follows,

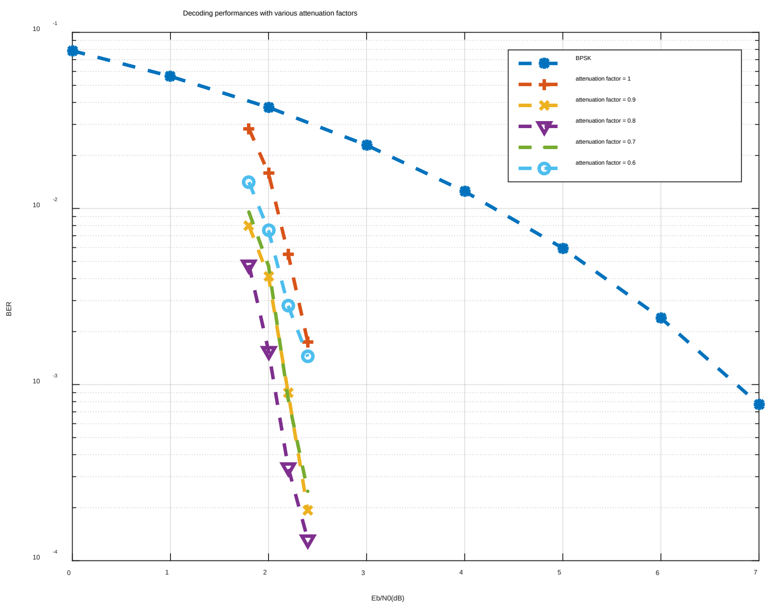


Fig. 5. The decoding performance with various attenuation factors

From the **Fig. 5** above, we can see different values of λ have different impacts on the decoding performance. When λ equals to approximate 0.8, the decoding performance is the best. Using the same method, we can finally choose the proper attenuation factor. In our paper, we choose $\lambda = 0.8$.

4. Experiments and Complexity Analysis

4.1 Experiments

In this section, we will present the MATLAB simulation results of our proposed method for CPM-QC-LDPC code with various lengths, and compare the bit error rate (BER) and frame error rate (FER) performance with that of scaled MSA and CPM-RID decoding algorithms.

Because our proposed method is performed on CPM-QC-LDPC code, we should first construct some efficient codes. The detailed method is described in Reference [14]. Briefly, we first choose the finite field GF(97) and construct two arbitrary subsets $S_1 = \{\alpha^1, \alpha^2, \alpha^3, \alpha^4\}$ and $S_2 = \{\alpha^5, \alpha^6, ..., \alpha^{12}\}$. Then, we construct a base matrix $\mathbf{B}$ by

$$\mathbf{B} = \begin{bmatrix} \alpha^1 + \alpha^5 & \alpha^1 + \alpha^6 & \cdots & \alpha^1 + \alpha^{12} \\ \alpha^2 + \alpha^5 & \alpha^2 + \alpha^6 & \cdots & \alpha^2 + \alpha^{12} \\ \alpha^3 + \alpha^5 & \alpha^3 + \alpha^6 & \cdots & \alpha^3 + \alpha^{12} \\ \alpha^4 + \alpha^5 & \alpha^4 + \alpha^6 & \cdots & \alpha^4 + \alpha^{12} \end{bmatrix}. \tag{21}$$

We can see $\mathbf{B}$ is a 4×8 matrix. Then we replace all entries in $\mathbf{B}$ with CPMs with the size of 96×96, and we can get a parity check matrix $\mathbf{H}$ with the size of 384×768. The null space of $\mathbf{H}$ represents a (768,384) CPM-QC-LDPC code C with the rate of 1/2.

Then, we perform three decoding algorithms on (768,384) CPM-QC-LDPC code C, three algorithms are performed 5 iterations and 10 iterations, respectively. For CPM-RID decoding scheme, we first construct a submatrix $\mathbf{H}_0^*$ with the size of 4×768 to decode. For our proposed method, we first use NN to train the weights of Tanner graph and then start to decode. The experimental results are shown in **Fig. 6.**

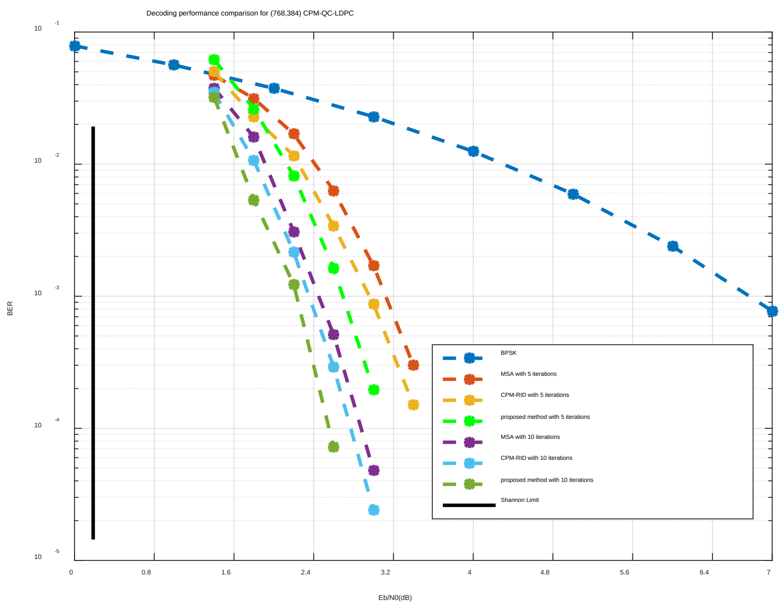


Fig. 6. Decoding performance comparison for (768,384) CPM-QC-LDPC code

From **Fig. 6**, it is easy to see our proposed method has a better and faster convergence rate than that of other two algorithms. Concisely, compared with MSA scheme decoded with 5 iterations, CPM-RID scheme with 5 iterations has a faster decoding performance (there is an approximate 0.2 dB coding gain), while our proposed method has a better performance (when BER is 10^{-3}, there is a 0.4 dB coding gain compared with that of MSA scheme and a 0.2 dB coding gain compared with that of CPM-RID scheme). When iteration equals to 10, there is also a visible coding gain using our proposed method.

Then, we compare the bit error rate performance and frame error rate performance of three algorithms, and finally show the results in **Fig. 7** and **Fig. 8**. In this experiment, we perform MSA scheme with 50 iterations, perform CPM-RID scheme with 35 iterations and proposed method with 30 iterations.

After many experiments, we find when we choose 50 iterations for MSA scheme, 35 iterations for CPM-RID scheme and 30 iterations for our proposed method, they all achieve their best decoding performance. From **Fig. 7** and **Fig. 8**, we can see our proposed method decoded with 30 times perform slightly better than that of MSA with 50 iterations and CPM-RID scheme with 35 iterations.

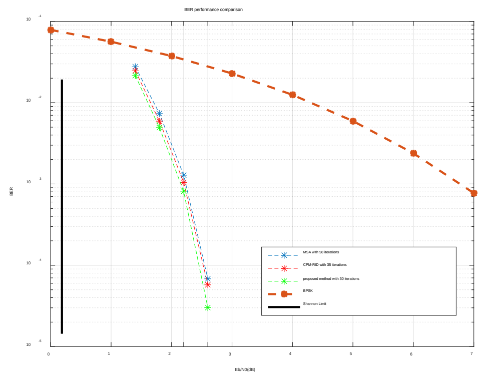


Fig. 7. BER performance comparison for (768,384) CPM-QC-LDPC code

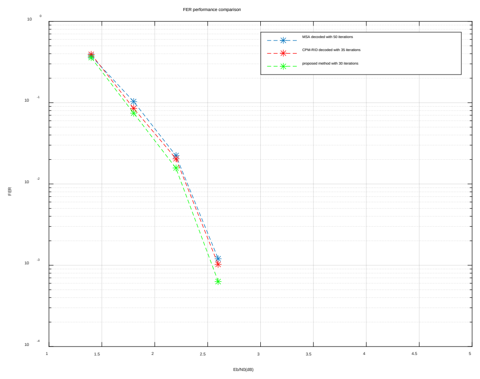


Fig. 8. FER performance comparison for (768,384) CPM-QC-LDPC code

To illustrate the efficiency of the improvement on convergence rate, we also perform another experiment to compare the iterations that we need to achieve the best performance. We choose $E_b / N_0 = 2$ dB and $E_b / N_0 = 2.4$ dB in this experiment, and depict the results in **Fig. 9**. Concisely, when $E_b / N_0 = 2$ dB, MSA scheme requires 20 iterations to achieve a stable decoding performance. CPM-RID scheme needs 15 iterations while our proposed method only needs 12 iterations. When $E_b / N_0 = 2.4$ dB, it requires 12 iterations for MSA and 10 iterations for CPM-RID to achieve a stable performance. However, our proposed method only needs 7 iterations.

4.2 Complexity Analysis

In this part we mainly analyze the implementation complexity, which is crucial to LDPC decoder. We assume we construct the code based on the finite field GF(q). Then, we construct a $m \times n$ base matrix $\mathbf{B}$. After replacing all entries in $\mathbf{B}$ with CPMs with the size of $(q-1)\times(q-1)$, we obtain a parity check matrix $\mathbf{H}_{m(q-1)\times n(q-1)}$, whose null space represents a $(n(q-1),(n-m)(q-1))$ CPM-QC-LDPC code C with the rate of $1-m/n$. The row weight of $\mathbf{H}_{m(q-1)\times n(q-1)}$ is n and the column weight is m. For CPM-RID and our proposed method, we construct the submatrix $\mathbf{H}_0^*$ with the size of $m \times n(q-1)$.

We first analyze the decoder complexity for three algorithms. We assume each real number information is quantized by b bits. For MSA scheme, it requires $n(q-1)b$ binary memory units to store $\mathbf{y} = \{y_j\}$ and $n(q-1)b$ binary memory units to store the reliability information $\mathbf{Q} = \{Q_j\}$. Because the information update Equation (19) in MSA should be

$$R_t^{(i)} = \left\{[\textstyle\prod_{j' \in M(t)\setminus j} \operatorname{sign}(Q_{j'}^{(i-1)})] \times [\min |Q_{j'}^{(i-1)}|]\right\}. \tag{22}$$

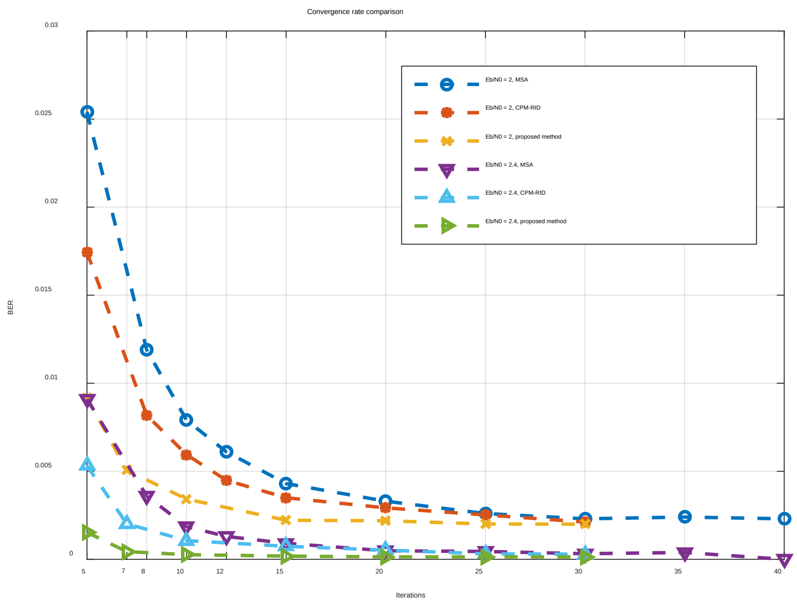


Fig. 9. Convergence rate comparison with various algorithms

(the meanings of symbols in Equation (22) are the same as those in section 3.3), it requires $2(b-1)$ binary memory units to store the two smallest magnitudes of the information of adjacent variable nodes for each check node, $\lceil \log_2 n \rceil$ binary memory units to store the location of the smallest magnitude of the information of adjacent variable nodes, and n units for signs of the information of adjacent variable nodes. Because there are $m(q-1)$ check nodes, we totally need $m(q-1)\left[2(b-1)+\lceil \log_2 n \rceil+n\right]+\ 2n(q-1)b$ binary memory units for MSA decoder.

Similarly, we can use the same method to analyze the decoder complexity for CPM-RID scheme and our proposed method. For CPM-RID scheme, we decode based on $\mathbf{H}_0^*$, which is a $m \times n(q-1)$ submatrix of $\mathbf{H}$. Therefore, we only require $m\left[2(b-1)+\lceil \log_2 n \rceil+n\right]+2n(q-1)b$ binary memory units for CPM-RID scheme. Because there are only mn non-zero elements in $\mathbf{H}_0^*$ (it means mn edges in the Tanner graph of $\mathbf{H}_0^*$), so we need extra mnb binary memory units to store the weighs. The total resources required for our proposed method is $m\left[2(b-1)+\lceil \log_2 n \rceil+n\right]+2n(q-1)b+mnb$. To make it clear, we present the information in **Table 3** as follows.

Then, we consider the computational complexity comparison among three algorithms. It is worth mentioning that we do not take the training complexity into consideration in our proposed method. That is simply because after we train the weights, we only need to save them and use them to help decode. It is also worth mentioning that in those three decoding algorithms, the process which brings out main complexity is the information update process to update the $\mathbf{Q}$. Therefore, we only consider and compare the complexity brought out by the information update process of three algorithms. To make it clear, we list three information update processes in **Table 4** (the meanings of symbols are the same as those in Section 3.3).

Table 3. A comparison of decoder complexity

Decoding Scheme	**Decoder complexity**
MSA	$m(q-1)\left[2(b-1)+\lceil \log_2 n \rceil + n\right] + 2n(q-1)b$
CPM-RID	$m\left[2(b-1)+\lceil \log_2 n \rceil + n\right] + 2n(q-1)b$
Proposed method	$m\left[2(b-1)+\lceil \log_2 n \rceil + n\right] + 2n(q-1)b + mnb$

Table 4. An illustration of three information update processes

Decoding Scheme	**Information update process**
MSA	$R_t^{(i)} = \left[\prod_{j'\in M(t)\backslash j} \operatorname{sign}(Q_{j'}^{(i-1)})\right] \times \min_{j'\in M(t)\backslash j}\left(\left\lvert Q_{j'}^{(i-1)} \right\rvert\right) \quad (23)$
CPM-RID	$R_t^{(i)} = \left[\prod_{j'\in M(t)\backslash j} \operatorname{sign}(Q_{j'}^{(i-1)})\right] \times \min_{j'\in M(t)\backslash j}\left(\left\lvert Q_{j'}^{(i-1)} \right\rvert\right) \quad (24)$
Proposed method	$R_t^{(i)} = \left[\prod_{j'\in M(t)\backslash j} \operatorname{sign}(w_{i \bmod m, j'} \cdot Q_{j'}^{(i-1)})\right] \times \min_{j'\in M(t)\backslash j}\left(\left\lvert w_{i \bmod m, j'} \cdot Q_{j'}^{(i-1)} \right\rvert\right) \quad (25)$

From Equation (23) and (24), we can see in each iteration for MSA and CPM-RID, we need to carry out $3n+\lceil \log_2(n) \rceil - 2$ real number computations ($2n-1$ computations for sign operation and $n+\lceil \log_2 n \rceil - 1$ computations for comparion operation), while in (25), we need to carry out extra n multiplication operations. However, for various decoding schemes, they have different convergence rates, so it is fair to consider the iterations required for those schemes. Assume MSA scheme requires $iter_1$ iterations to achieve a stable decoding performance, CPM-RID requires $iter_2$ iterations and our proposed method requires $iter_3$, we present the computational complexity for information update in **Table 5**,

Table 5. An illustration of computational complexity

Decoding Scheme	**Computational complexity comparison for information update**
MSA	$iter_1 \times \left[3n+\lceil \log_2(n) \rceil - 2\right]$ additions
CPM-RID	$iter_2 \times \left[3n+\lceil \log_2(n) \rceil - 2\right]$ additions
Proposed method	$iter_3 \times \left[3n+\lceil \log_2(n) \rceil - 2\right]$ additions $+\ iter_3 \times n$ multiplications

In each iteration, we can see our proposed method has a higher computational complexity than that of other two schemes. However, if we take the convergence rate into consideration, that is, our proposed method has fewer iterations to achieve a stable decoding performance, it is possible for our proposed method to have a similar computational complexity with that of other two schemes.

5. Conclusion

A combination of neural network training and CPM-RID decoding algorithm is presented in this paper.

At first, we constructed an activation function for check nodes and established a neural network model to train weights and assign them to the edges in a Tanner graph. Next, we modified the original CPM-RID algorithm by directly dividing the parity check matrix into some submatrices. In the process of decoding a submatrix we chose parallel mechanism, while between various submatrices we chose serial mechanism. Finally, the trained weights and modified CPM-RID algorithm are combined. The whole decoding process can be treated as decoding block-by-block with weights. Our simulation experiments include the comparisons among three algorithms, that is, scaled MSA algorithm, CPM-RID algorithm and our proposed method. We find that our proposed algorithm has the lowest BER and FER and the fastest convergence rate, which verifies the merits of our proposed method.

Appendix

Proof of equation (10).

In this paper, we use the induction method from Mathematics to verify the expression of f_{output}. Let n denotes the number of the input elements.

If there are only two inputs $\{x_1, x_2\}$, that is, $n = 2$, then we have

$$\begin{aligned} f_{output}(x_1, x_2) &= x_1 \oplus x_2 \\ &= x_1(1-x_2)+(1-x_1)x_2 \\ &= x_1 + x_2 - 2x_1x_2 \\ &= (-2)^0 \sum \tilde{C}_n^1 x_i + (-2)^1 \sum \tilde{C}_n^2 x_i x_j . \end{aligned}$$

In the case of $n = N$, we assume the expression of $f_{output}(x_1, x_2, \cdots, x_N)$ is

$$\begin{aligned} f_{output}(x_1, x_2, \ldots, x_N) &= (f_{output}(x_1, x_2) \oplus x_3) \oplus \cdots x_N \\ &= x_1 \oplus x_2 \oplus \cdots \oplus x_N \\ &= (-2)^0 \sum \tilde{C}_n^1 x_i + (-2)^1 \sum \tilde{C}_n^2 x_i x_j \\ &\quad + \cdots + (-2)^{N-1} \sum \tilde{C}_n^N x_1 x_2 \cdots x_N . \end{aligned}$$

Now assume that $n' = N+1$ (here, we use n' to distinguish with $n = N$), we get

$$\begin{aligned} f_{output}(x_1, x_2, \ldots, x_N, x_{N+1}) &= (f_{output}(x_1 \oplus x_2) \oplus x_3) \oplus \cdots x_N \\ &= x_1 \oplus x_2 \oplus \cdots \oplus x_N \oplus x_{N+1} \\ &= f_{output}(x_1, x_2, \ldots, x_N) \oplus x_{N+1} \\ &= [1 - f_{output}(x_1, x_2, \ldots, x_N)] \cdot x_{N+1} + f_{output}(x_1, x_2, \ldots, x_N) \cdot (1 - x_{N+1}) \\ &= [f_{output}(x_1, x_2, \ldots, x_N) + x_{N+1}] - 2 \cdot f_{output}(x_1, x_2, \ldots, x_N) \cdot x_{N+1} . \end{aligned}$$

Then we substitute the expression of $f_{output}(x_1, x_2, \cdots, x_N)$ and obtain

$$\begin{aligned} f_{output}(x_1, \cdots, x_N, x_{N+1}) &= \left[(x_1 + x_2 + \cdots + x_N + x_{N+1}) + (-2)^1 \sum \tilde{C}_{\mathrm{n}}^2 x_i x_j + \cdots + (-2)^{N-1} \sum \tilde{C}_n^N x_1 \cdots x_N \right] \\ &\quad + (-2)^1 \left[(-2)^0 \sum \tilde{C}_n^1 x_i + (-2)^1 \sum \tilde{C}_n^2 x_i x_j + \cdots + (-2)^{N-1} \sum \tilde{C}_n^N x_1 x_2 \cdots x_N \right] \cdot x_{N+1} \\ &= (-2)^0 \sum \tilde{C}_{n'}^1 x_i + (-2)^1 \sum \tilde{C}_{n'}^2 x_i x_j + \cdots + (-2)^{N-1} \sum \tilde{C}_{n'}^N x_1 x_2 \cdots x_N \\ &\quad + (-2)^N \sum \tilde{C}_{n'}^{N+1} x_1 x_2 \cdots x_{N+1} . \end{aligned}$$

Therefore, we have the expression of (3) by induction.

References

[1] R.G. Gallager, “Low density parity-check codes,” *IEEE Trans. Inf. Theory*, vol. 8, no. 1, pp. 21-28, January, 1962. Article (CrossRef Link).

[2] S. Lin, “Capacity-approaching low-density parity-check codes: recent developments and applications,” in *Proc. of 2013 Workshop on Coding and Information Theory*, January 19-20, 2013. Article (CrossRef Link).

[3] I. Tsatsaragkos and V. Paliouras, “A reconfigurable LDPC decoder optimized for 802.11n/ac applications,” *IEEE Transactions on Very Large Scale Integration Systems*, vol. PP, no. 99, pp. 1-14, September, 2017. Article (CrossRef Link).

[4] I.E. Bocharova, B.D. Kudryashov, V. Skachek and Y. Yakimenka, “Average spectra for ensembles of LDPC codes and applications,” in *Proc. of IEEE International Symposium on Information Theory*, pp. 361-365, June 25-30, 2017. Article (CrossRef Link).

[5] Keol Cho and Ki-Seok Chung, “Self-adaptive termination check of min-sum algorithm for LDPC decoders using the first two minima,” *KSII Transactions on Internet and Information Systems*, vol. 11, no. 4, pp. 1987-2001, April, 2017. Article (CrossRef Link).

[6] J. Xu and K. Zhang, “A low-complexity CLSIC-LMMSE-based multi-user detection algorithm for coded MIMO systems with high order modulation,” *KSII Transactions on Internet and Information Systems*, vol. 11, no. 4, pp. 1954-1971, April, 2017. Article (CrossRef Link).

[7] Z.X. Liu, G.X. Kang, Z.W. Si and N.B. Zhang, “Performance improvement of iterative demodulation and decoding for spatially coupling data transmission by joint sparse graph,” *KSII Transactions on Internet and Information Systems*, vol. 10, no. 12, pp. 5964-5984, December, 2016. Article (CrossRef Link).

[8] Y.Y. Tai, L. Lan, L. Zeng, S. Lin and K.A.S. Abdel-Ghaffar, “Algebraic construction of quasi-cyclic ldpc codes for the awgn and erasure channels,” *IEEE Trans. Commun.*, vol. 54, pp. 1765-1774, October, 2006. Article (CrossRef Link).

[9] J. Kang, Q. Huang, L. Zhang, B. Zhou and S. Lin, “Quasi-cyclic ldpc codes: an algebraic construction,” *IEEE Trans. Commun.*, vol. 58, pp. 1383-1396, May, 2010. Article (CrossRef Link).

[10] J. Li, K. Liu, S. Lin and K.A.S. Abdel-Ghaffar, “Quasi-cyclic LDPC codes on two arbitrary sets of a finite field,” in *Proc. of IEEE International Symposium on Information Theory*, pp. 2454-2458, June 29- July 4, 2014. Article (CrossRef Link).

[11] K. Liu, S. Lin and K.A.S. Abdel-Ghaffar, “A revolving iterative algorithm for decoding algebraic quasi-cyclic LDPC code,” *IEEE Trans. Commun.*, vol. 61, pp. 4816-4827, October, 2013. Article (CrossRef Link).

[12] D. Wang, L. Wang, X. Chen, A. Fei, C. Ju and Z. Wang, “Construction of QC-LDPC codes based on pre-masking and local optimal searching,” *IEEE Communication Letters*, vol. PP, pp. 1-1, September, 2017. Article (CrossRef Link).

[13] L. Kong, L. He, P. Chen, G. Han and F. Yang, “Protograph based quasi-cyclic LDPC coding for ultra-high density magnetic recording channels,” in *Proc. of 2015 IEEE International Magnetics Conference*, May 11-15, 2015. Article (CrossRef Link).

[14] J. Li, K. Liu, S. Lin and K.A.S. Abdel-Ghaffar, “Algebraic quasi-cyclic ldpc codes: construction, low error-floor, large girth and a reduced-complexity decoding scheme,” *IEEE Trans. Commun.*, vol.62, pp. 2626-2637, July, 2014. Article (CrossRef Link).

[15] S. Lin, K. Liu, J. Li and K.A.S. Abdel-Ghaffar, “A reduced-complexity iterative scheme for decoding quasi-cyclic low-density parity-check codes,” in *Proc. of 48th Annual Asilomar Conference on Signals, Systems and Computers*, pp. 119-125, November 2-5, 2014. Article (CrossRef Link).

[16] J. Li, K. Liu, S. Lin and K.A.S. Abdel-Ghaffar, "Decoding of quasi-cyclic LDPC codes with section-wise cyclic structure," in *Proc. of Information Theory and Applications Workshop*, pp. 1-10, February 9-14, 2014. Article (CrossRef Link).

[17] W.R. Caid and R.W. Means, "Neural network error correcting decoders for block and convolutional codes," in *Proc. of IEEE Global Telecommunications Conference*, pp. 1028–1031, December 2-5, 1990. Article (CrossRef Link).

[18] A. Hamalainen and J. Henriksson, "A recurrent neural decoder for convolutional codes," in *Proc. of IEEE International Conference on Communications*, pp. 1305–1309, June 6-10, 1999. Article (CrossRef Link).

[19] E. Nachmani, Y. Beery and D. Burshtein, "Learning to decode linear codes using deep learning," Available online: https://arxiv.org/pdf/1607.04793. Article (CrossRef Link).

[20] E. Nachmani, E. Marciano, D. Burshtein and Y. Beery, "RNN decoding of linear block codes," Available online: http://arxiv.org/pdf/1702.07560. Article (CrossRef Link).

[21] A.R. Karami, M.A. Attar and H. Tavakoli, "Multi-layer perceptron neural networks decoder for LDPC codes," in *International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 476-479, September 24-26, 2009. Article (CrossRef Link).

[22] T. Gruber, S. Cammerer, J. Hoydis and S.T. Brink, "On deep learning-based channel decoding," in *Proc. of 51st Annual Conference on Information Sciences and Systems*, pp. 1-6, March 22-24, 2017. Article (CrossRef Link).

[23] J. Bergstra, R. Bardenet, Y. Bengio and B. Kégl, "Algorithms for hyper-parameter optimization," in *Proc. of 25th Annual Conference on Neural Information Processing Systems*, pp. 2546-2554, December 17-25, 2011. Article (CrossRef Link).

Zuohong Xu received the B.S. in College of Science from National University of Defense and Technology (NUDT), Changsha, P. R. China, in June 2016. Now, he is a master of College of Electronic Science and Engineering from NUDT. During Oct. 2015 to Jul. 2016, he studied mathematics in the University of Warwick, UK. His current research field includes coding theory, information theory and wireless communication techniques.

Jiang Zhu is a professor at College of Electronic Science and Engineering, National University of Defense Technology, Changsha, P. R. China. He received his Ph.D. degree in Information and Communication Engineering from National University of Defense Technology, Changsha, P. R. China, in 2000. His current research interests include high-speed data transmission, physical layer security and satellite communications.

Zixuan Zhang received the B.E. in College of Electric Science and Engineering from National University of Defense and Technology (NUDT), Changsha, P. R. China, in June 2016. Now, he is a master of College of Computer from NUDT. His current research field includes wireless communication theory and communication network.

Qian Cheng received the B.E. and M.E. degrees in Information and Communication Engineering from Xidian University, Xi’an, P. R. China, in June 2014, and National University of Defense Technology (NUDT), Changsha, P. R. China, in December 2016, respectively. He is currently pursuing a doctoral degree at NUDT. His current research interests include physical layer security and directional modulation.