Abstract: With the application of graph neural network (GNN) in the communication physical layer, GNN-based channel decoding algorithms have become a research hotspot. Compared with traditional decoding algorithms, GNN-based channel decoding algorithms have a better performance. GNN has good stability and can handle large-scale problems; GNN has good inheritance and can generalize to different network settings. Compared with deep learning-based channel decoding algorithms, GNN-based channel decoding algorithms avoid a large number of multiplications between learning weights and messages. However, the aggregation edges and nodes for GNN require many parameters, which requires a large amount of memory storage resources. In this work, we propose GNN-based channel decoding algorithms with shared parameters, called shared graph neural network (SGNN). For BCH codes and LDPC codes, the SGNN decoding algorithm only needs a quarter or half of the parameters, while achieving a slightly degraded bit error ratio (BER) performance.

Keywords: graph neural network; channel decoding algorithm; shared graph neural network; BCH codes; LDPC codes

1. Introduction

Deep learning (DL) technology has become increasingly crucial in many disciplines, especially in image processing [1–3], wireless communication [4–6], speech recognition [7], and other fields. In particular, DL was applied in multiple-input multiple-output (MIMO) channel state information (CSI) feedback [8], intelligent reflecting surfaces [9], and the channel decoding algorithm [10–14]. Although the bit error ratio (BER) performance has been improved, it brings higher storage complexity and computational complexity compared with traditional decoder algorithm. Tanner graphs and factor graphs are commonly used in decoding algorithms for low-density parity check (LDPC), Hamming, and BCH codes. Graph theory is a very broad branch of mathematics that is applicable to real world problems. Compared with the dimensional disaster [15] caused by the increase of the block code length of the DL-based decoding algorithm, GNN [16–18] is an extension of existing neural network methods that take graph-passing messages and add aggregation networks. GNN can find a suitable model by training the graph structure data, including edge data and node data. Model complexity is independent of graph size, and the trained model is easily applicable to arbitrarily large graphs. GNN has effectively, stability, and strong generalization ability. A special heterogeneous graph with different node types and edge types is proposed in ref. [19]. A GNN-based scalable polar code is proposed in ref. [20], which is a unique graph called a reconstructed code message passing (PCCMP) diagram for polar code, which changes the successive cancellation list decoding algorithm into a heterogeneous graph, while an iterative message passing algorithm is added to the graph to find the best graph with a minimum frame ratio (FER) or BER for a suitable polar code. A GNN-based channel decoding algorithm that adds weights to Tanner graph edges for
short block codes is proposed in ref. [21]. This algorithm is a graph-theoretic extension of the neural belief propagation (NBP) algorithm. A GNN structure for BCH codes and LDPC codes is proposed in ref. [22]. This network structure is suitable for codewords of any length, and the training will not be limited by the curse of dimensionality. The BER performance of the network for eight iterations exceeds the BER performance of the weighted BP for 20 iterations. However, it uses a large number of multilayer perceptron (MLP) network aggregation messages in each iteration, which requires a lot of computing costs and storage resources. Excessive complexity is very challenging to implement on resource-constrained communication hardware, which is also the problem this paper attempts to solve.

In this work, model compression technologies are used in GNN-based channel decoding algorithms. Methods to reduce the number of correction factors of GNN-based decoder are proposed, which involve changing the directed-edge to the non-directed edge, sharing weights in factor nodes, variable nodes, factor-to-variable node messages and variable-to-factor node messages, and applying it to the BCH and LDPC decoders. The main contribution we proposed is a low-complexity GNN-based channel decoding algorithm. The novelty of this paper can be summarized as follows:

- We share the same weights in GNN-based channel decoding algorithm especially in the update of factor nodes, variable nodes, factor-to-variable node messages, and variable-to-factor node messages.
- Based on different sharing schemes, we propose two shard GNN (SGNN)-based channel decoding algorithms to balance BER performance and storage complexity.
- Furthermore, we apply the SGNN-based channel decoding algorithm to BCH and LDPC decoders, which reduces the storage resources required by GNN-based channel decoders with a slight decrease in BER performance.

The rest of this paper can be organized as follows: GNN-based channel decoding is described briefly in Section 2. In Section 3, the SGNN-based channel decoding algorithms is proposed. The performance evaluation and complexity analysis are shown in Section 4, which systematically analyzes the performance and complexity of the SGNN and GNN-based channel decoding algorithms. Section 5 discusses the issues of the paper and future research directions. Section 6 concludes the paper.

2. Preliminary

LDPC, BCH, and Hamming codes are all linear block codes, and the Tanner graph of these codes are bipartite graph. We denote these codes by \((N, K)\), where \(N\) denotes the length of codewords and \(K\) denotes the number of information bits.

Suppose that, in a communication system, binary phase shift keying (BPSK) or quadrature phase shift keying (QPSK) modulation are used after information are encoded. Assume that the message obtained is \(Y = X + N\) over the additive white Gaussian noise (AWGN) channel, where \(X \in \mathbb{C}^M\) is the transmitted complex symbol, \(N \sim \mathcal{CN}(0, \sigma^2 I_M)\) is the complex AWGN noise. \(M\) represents the number of transmitted symbols. In BPSK modulation, \(M\) measures the length of the transmitted signal as the number of bits per symbol, \(B\) is one, and \(M\) is half the signal’s length in QPSK as the number of bits per symbol, and \(B\) is 2. \(I_M\) is a matrix of size \(M\)-by-\(M\), \(\sigma^2 = N_0/2\) is the variance of the channel noise, and \(N_0\) is noise spectral density. The log-likelihood ratio (LLR) in ref. [23] of the \(v\)-th bit can be computed as

\[
    l_{v,j} = \ln \left( \frac{\sum_{X_i \in X_i,j} e^{-\frac{1}{N_0} |Y_v - X_i|^2}}{\sum_{X_i \in X_i,0} e^{-\frac{1}{N_0} |Y_v - X_i|^2}} \right), \tag{1}
\]
where \( i \in \{1, 2, \cdots, B\} \), \( X = \{x_1, \cdots, x_2^B\} \) denotes the constellation points with unit energy and \( X_{ib} \) denotes the subset of \( X \) with the \( i \)-th bit being \( b \in \{0, 1\} \). LLR in [23] for BPSK modulation can be expressed as

\[
I_v = \log\left( \frac{P(Y_v = y_v|X_v = -1)}{P(Y_v = y_v|X_v = +1)} \right) = \frac{2y_v}{\sigma^2}.
\]

We use \( E_b/N_0 = 1/(2(K/N)\sigma^2) \) as the signal-to-noise ratio, where \( E_b \) is the energy per information bit, \( y_v \) is the \( v \)-th value of the received message.

**GNN-Based Channel Decoding**

Similar to the Tanner graph, the GNN-based channel decoding also [22] has two types of nodes, namely, variable nodes (VN) and factor nodes (FN). The variable nodes correspond to the \( j \)-th column of the \( H \) matrix, and the factor nodes correspond to the \( i \)-th row of the \( H \) matrix, similar to the check node in the BP algorithm. There are \( K \) VNs, denoted by \( \mathcal{V} \), and there are \( N - K \) FNs, denoted by \( \mathcal{U} \). \( E \) denotes all of the edges in the Tanner graph, which means all the 1s in \( H \). According to graph theory, we assign value and attribute to each node and each edge in the graph, similar to the training parameters in weighted BP and min-sum algorithms [12,24]. We use \( \theta_v \) to denote each of the node attribute for VN and FN, respectively, and there are \( 2g \) attributes. \( m_{v_i \rightarrow u_j} \in R^{f_{nu}}, m_{u_j \rightarrow v_i} \in R^{f_{uv}} \) are used to assign messages to each directed edge, while different message values are assigned for different directions of each edge. In order to improve the BER performance, appropriate weights to each directional edge and each node are added, similar to the model-driven training weights. We use \( g_{v_i} \in R^{f_{nu}}, g_{u_j} \in R^{f_{uv}} \) to denote each of the node attribute for VN and FN, respectively, and there are \( 2N - K \) node attributes. \( g_{v_i \rightarrow u_j} \in R^{f_{nu}}, g_{u_j \rightarrow v_i} \in R^{f_{uv}} \) are used to assign each directed edge attributes; there is a total of \( 2E \) edge attributes.

As shown in Algorithm 1, we first initialize the FN and VN values, and then enter the decoding process until the maximum number \( T \) of decodings is reached and the iteration terminates. The edge directed message update function and the two-node FN, VN values’ update functions are trained in the GNN-based channel decoding algorithm. The GNN-based channel decoding algorithm requires the following four steps:

**Step 1:** From the VN to FN, the message \( m_{v_i \rightarrow u_j} \) is passed in the graph as

\[
m_{v_i \rightarrow u_j} \leftarrow f_{f_{uv}}([h_{v_i}, ||h_{u_j}||g_{v_i \rightarrow u_j}], \theta_{m_{v_i \rightarrow u_j}}),
\]

where \( \theta_{m_{v_i \rightarrow u_j}} \) denotes the parameters of the node, and the symbol \( || \) denotes that the two tensors are concatenation together. The \( f_{f_{uv}} \) function maps messages to high-dimensional features by multiplying weights. Simple multilayer perceptrons (MLPs) are used to expand the dimension.

**Step 2:** Update the FN value \( u_j \) as

\[
h_{u_j} \leftarrow f_{f_{nu}}([h_{u_j}, \frac{1}{|\mathcal{V}(u_j)|} \sum_{v_i \in \mathcal{V}(u_j)} m_{v_i \rightarrow u_j} || g_{u_j}], \theta_u),
\]

where \( \theta_u \) denotes the parameters in FN. We use \( \text{sum} \) aggregation similar to min-sum algorithm [24]. We may also use \( \text{mean} \), \( \text{max} \), and \( \text{min} \) operations, and the updated FN value \( u_j \) is given by \( h_{u_j} \leftarrow f_{f_{nu}}([h_{u_j}, \bigoplus_{v_i \in \mathcal{V}(u_j)} m_{v_i \rightarrow u_j} || g_{u_j}], \theta_u). \)

**Step 3:** From the FN to VN, the message \( m_{u_j \rightarrow v_i} \) is passed in the graph as

\[
m_{u_j \rightarrow v_i} \leftarrow f_{f_{uv}}([h_{u_j}, || h_{v_i} || g_{u_j \rightarrow v_i}], \theta_{m_{u_j \rightarrow v_i}}),
\]

where \( \theta_{m_{u_j \rightarrow v_i}} \) denotes the trainable parameters in another direction edge.
Step 4: Update the VN value $v_i$ as

$$h_{v_i} \leftarrow f^{F_n}(\|h_{v_i}\|_U, \sum_{u_j \in U(v_i)} m_{u_j \rightarrow v_i} ||g_{u_j}, \theta_u||, \theta_v),$$

(6)

where $\theta_v$ denotes the parameters in VN. The decoding result is also output from this step. We may also use mean, max, and min operations, and the updated VN value $v_i$ is given by

$$h_{v_i} \leftarrow f^{F_n}(\|h_{v_i}\|_U, \bigoplus_{u_j \in U(v_i)} m_{u_j \rightarrow v_i} ||g_{u_j}, \theta_u||, \theta_v).$$

Algorithm 1: GNN-based Channel Decoding Algorithm in ref. [22].

**input**: LLRs of the channel observation $l_i$

Set of trainable parameters:
- FN updates $\theta_u$
- VN updates $\theta_v$
- variable-to-factor message updates $\theta_{m_{v_i \rightarrow u_j}}$
- factor-to-variable message updates $\theta_{m_{u_j \rightarrow v_i}}$
- VN attributes $\{g_{v_i}\}$ for $v_i \in V$
- FN attributes $\{g_{u_j}\}$ for $u_j \in U$
- Message attributes $\{g_{m_{v_i \rightarrow u_j}}, g_{m_{u_j \rightarrow v_i}}\}$ for $(v_i, u_j) \in E$
- Input embedding $w$
- Output embedding $q$

**output**: Soft estimated $l$ from VN nodes

1. %Initialize graph nodes
2. for $v_i \in V$ do
3. \hspace{1em} $h_{v_i} \leftarrow l_{v_i} \cdot w$
4. end
5. $h_U \leftarrow 0$
6. Run decoding iterations
7. for $t = 0, ..., T - 1$ do
8. \hspace{1em} %Update variable-to-factor node messages
9. \hspace{2em} for $(v_i, u_j) \in E$ do
10. \hspace{3em} $m_{v_i \rightarrow u_j} \leftarrow f^{F_m}(||h_{v_i}||_U, ||g_{m_{v_i \rightarrow u_j}}||, \theta_{m_{v_i \rightarrow u_j}})$
11. \hspace{2em} end
12. \hspace{1em} %Update factor nodes
13. \hspace{2em} for $u_j \in U$ do
14. \hspace{3em} $h_{u_j} \leftarrow f^{F_n}(||h_{u_j}||_U, \sum_{v_i \in V(u_j)} m_{v_i \rightarrow u_j} ||g_{u_j}, \theta_u||, \theta_v)$
15. \hspace{2em} end
16. \hspace{1em} %Update factor node to variable node messages
17. \hspace{2em} for $(v_i, u_j) \in E$ do
18. \hspace{3em} $m_{u_j \rightarrow v_i} \leftarrow f^{F_m}(||h_{u_j}||_U, ||g_{m_{u_j \rightarrow v_i}}||, \theta_{m_{u_j \rightarrow v_i}})$
19. \hspace{2em} end
20. \hspace{1em} %Update variable nodes
21. \hspace{2em} for $v_i \in V$ do
22. \hspace{3em} $h_{v_i} \leftarrow f^{F_n}(||h_{v_i}||_U, \sum_{u_j \in U(v_i)} m_{u_j \rightarrow v_i} ||g_{v_i}, \theta_v||, \theta_v)$
23. \hspace{2em} end
24. end
25. % Return back variable node values
26. for $v_i \in V$ do
27. \hspace{1em} $l_i \leftarrow q^t h_{v_i}$
28. end
Repeat the above Steps 1, 2, 3, and 4 until the conditions of iterations are reached. The recovered binary vector $\hat{x}$ can be computed by $h_v$, as

$$l_i \leftarrow q^t h_v, \quad (7)$$

$$\hat{x}_i = \begin{cases} 1, & \text{if } l_i > 0 \\ 0, & \text{otherwise} \end{cases}, \quad (8)$$

Next, we use an example to illustrate the working principle of the GNN-based channel decoding algorithm. As shown in Figure 1, A parity-check matrix $H$ of $(N = 7, K = 4)$ is shown in Figure 1a, and the Tanner graph corresponding to this matrix is shown in Figure 1b, similar to that in ref. [25]. There are three FNs and seven VNs. Figure 2 shows the detailed process of converting the Tanner graph into the GNN-based channel decoding algorithm. After passing through the BPSK modulation and AWGN channels, we receive the LLR, whose dimension is $\text{batchsize} \times 7$, where $\text{batchsize}$ denotes the size of the batch. First, we initialize the parameters of the GNN-based channel decoding algorithm and then initialize the graph nodes values, including the values of the FN nodes and VN nodes and $h_v = 0$, whose dimension is $\text{batchsize} \times 3 \times 20$. $h_v = l_v \cdot w$, where $w$ is input embedding with a dense network, whose dimension is $\text{batchsize} \times 7 \times 20$, and the number of dimensions for vertex embeddings is 20.

![Tanner graph](image)

**Figure 1.** The Tanner graph corresponding to the parity matrix, (a) parity-check matrix; (b) Tanner graph.

The iterative link of the GNN-based channel decoding algorithm is shown in Figure 2, which includes the update of the FN nodes and the update of the VN nodes. Converting the values of the FN nodes and VN nodes into edge features, the edge $E$ of this code is 12. After adding a message attribute, we compute messages for all edges using MLPs. In this code, MLPs are implemented by a two-layer dense network; these two layers contain 40 and 20 neurons, respectively. Messages are reduced per receiving vertex by message aggregation, where we use $\text{sum}$ aggregation. After aggregation, messages dimension is $\text{batchsize} \times 3 \times 20$, and then we add a node attribute. After concatenating this information, a new embedding will be calculated by two layers of dense of MLPs, and we update the FN value $h_u \leftarrow f^{Eu}([h_u], \sum_{u \in \mathcal{U}(v)} m_{u \rightarrow v} || g_{u \rightarrow v}, \theta_u)$. The next step is to update the VN value. The update process of the VN value is similar to that of the FN value; the only difference is that the dimension of the feature information after aggregation is $\text{batchsize} \times 7 \times 20$. We update the VN value $h_v \leftarrow f^{Ev}([h_v], \sum_{v \in \mathcal{V}(u)} m_{v \rightarrow u} || g_{v \rightarrow u}, \theta_v)$, where $h_v$ is the information recovered after one iteration. Repeat the iteration until the conditions of the iterations are reached. The recovered binary vector $\hat{x}$ is computed by...
$l_i \leftarrow q^t h_{v_i}$ as a hard threshold, where $q$ is an output embedding of a dense network with one neuron and $l$ denotes the recovered soft output.

Figure 2. GNN-based channel decoding algorithm for code (7,4) with one iteration.

3. SGNN Based Channel Decoding Algorithm

In order to reduce the computational complexity and the required storage resources of the GNN-based channel decoding algorithm, the model parameters and multiplication need to be reduced. The parameters of the GNN-based channel decoding algorithm include node attributes, edge attributes, and the weights brought by the mapping of each iteration. Each node has independent properties, and each directed edge also has independent properties. In order to reduce the attributes of half of the edges, we change the attributes of the edges in different directions to the attributes of edges without directions, that is, each edge has only one attribute. Four mappings are used per iteration: an update variable-to-factor node messages, update factor nodes, update factor nodes to variable node messages, and update variable nodes. In order to reduce the parameters of the mapping, we set the weights of the mapping to be shared. Shared parameters are directly linked to performance. We propose two different sharing methods, which are introduced below.

3.1. SGNN-Based Channel Decoding Algorithm 1

In order to minimize the number of parameters of the GNN-based channel decoding algorithm, we use the same weights for all mappings $\theta$. At the same time, for directed graphs, we change them to non-directed graphs; therefore, edges in different directions in the attributes of the graph use the same assigned edge attribute $\{g_{m_{v_i u_j}}\}$, $\forall (v_i, u_j) \in E$. This sharing method is called SGNN1 and is referred to as Algorithm 2 on the next page.
Algorithm 2: The Proposed Shared GNN-based Channel Decoding Algorithm 1.

**input**: LLRs of the channel observation $l_v$

Set of trainable parameters:

- FN updates, VN updates, variable-to-factor message updates and factor-to-variable message updates operate with the same weights $\theta$

- VN attributes $\{g_{v_i}\} \forall v_i \in V$

- FN attributes $\{g_{u_j}\} \forall u_j \in U$

- Message attributes $\{g_{m_{v_i u_j}}\}, \forall (v_i, u_j) \in E$

- Input embedding $w$

- Output embedding $q$

**output**: Soft estimated $l$ from VN nodes

1. %Initialize graph nodes
2. for $v_i \in V$ do
3.   $h_{v_i} \leftarrow l_{v_i} \cdot w$
4. end
5. $h_u \leftarrow 0$
6. Run decoding iterations
7. for $t = 0, ..., T - 1$ do
8.   % Update variable-to-factor node messages
9.   for $(v_i, u_j) \in E$ do
10.     $m_{v_i \rightarrow u_j} \leftarrow f^{F_n}(|h_{v_i}| |h_{u_j}| |g_{m_{v_i u_j}}|, \theta)$
11. end
12. % Update factor nodes
13. for $u_j \in U$ do
14.     $h_{u_j} \leftarrow f^{F_n}(|h_{u_j}| \frac{1}{\sum_{v_i \in V(u_j)} m_{v_i \rightarrow u_j}} |g_{u_j}|, \theta)$
15. end
16. % Update factor node to variable node messages
17. for $(v_i, u_j) \in E$ do
18.     $m_{u_j \rightarrow v_i} \leftarrow f^{F_n}(|h_{u_j}| |h_{v_i}| |g_{m_{u_j v_i}}|, \theta)$
19. end
20. % Update variable nodes
21. for $v_i \in V$ do
22.     $h_{v_i} \leftarrow f^{F_n}(|h_{v_i}| \frac{1}{\sum_{u_j \in U(v_i)} m_{u_j \rightarrow v_i}} |g_{v_i}|, \theta)$
23. end
24. end
25. % Return back variable node values
26. for $v_i \in V$ do
27.     $l_v \leftarrow q^T h_{v_i}$
28. end

3.2. SGNN-Based Channel Decoding Algorithm 2

In order to balance the number of parameters and the BER performance of the GNN-based channel decoding algorithm, we use the same weights $\theta_1$ for the FN updates map and the variable-to-factor message updates map, the same weights $\theta_2$ for the VN updates map, and the factor-to-variable message updates map. At the same time, for directed graphs, we change them to non-directed graphs; therefore, edges in different directions in the attributes of the graph use the same assigned edge attribute $\{g_{m_{v_i u_j}}\}, \forall (v_i, u_j) \in E$. This sharing method is called SGNN2 and is referred to as Algorithm 3.
Algorithm 3: The Proposed Shared GNN-based Channel Decoding Algorithm 2.

**input**: LLRs of the channel observation \( l_v \)

Set of trainable parameters:
- FN updates and variable-to-factor message updates operate with the same weights \( \theta_1 \)
- VN updates and factor-to-variable message updates operate with the same weights \( \theta_2 \)
- VN attributes \( \{ g_{v_i} \} \) for \( v_i \in V \)
- FN attributes \( \{ g_{u_j} \} \) for \( u_j \in U \)
- Message attributes \( \{ g_{m_{v_i u_j}} \} \) for every \( (v_i, u_j) \in E \)
- Input embedding \( w \)
- Output embedding \( q \)

**output**: Soft estimated \( l \) from VN nodes

1. %Initialize graph nodes
2. for \( v_i \in V \) do
3. \( h_{v_i} \leftarrow l_{v_i} \cdot w \)
4. end
5. \( h_u \leftarrow 0 \)
6. Run decoding iterations
7. for \( t = 0, ..., T - 1 \) do
8. % Update variable-to-factor node messages
9. for \( (v_i, u_j) \in E \) do
10. \( m_{v_i \rightarrow u_j} \leftarrow f_{F_n}(|h_{v_i}||h_{u_j}||g_{m_{v_i u_j}}|, \theta_1) \)
11. end
12. % Update factor nodes
13. for \( u_j \in U \) do
14. \( h_{u_j} \leftarrow f_{F_n}(|h_{u_j}| \frac{1}{|V(u_j)|} \sum_{v_i \in V(u_j)} m_{v_i \rightarrow u_j}||g_{u_j}, \theta_1) \)
15. end
16. % Update factor node to variable node messages
17. for \( (v_i, u_j) \in E \) do
18. \( m_{u_j \rightarrow v_i} \leftarrow f_{F_n}(|h_{u_j}||h_{v_i}||g_{m_{u_j v_i}}|, \theta_2) \)
19. end
20. % Update variable nodes
21. for \( v_i \in V \) do
22. \( h_{v_i} \leftarrow f_{F_n}(|h_{v_i}| \frac{1}{|U(v_i)|} \sum_{u_j \in U(v_i)} m_{u_j \rightarrow v_i}||g_{v_i}, \theta_2) \)
23. end
24. end
25. % Return back variable node values
26. for \( v_i \in V \) do
27. \( l_i \leftarrow q \cdot h_{v_i} \)
28. end

4. Simulation Results and Discussions

We investigated the performance of the SGNN1 and SGNN2-based channel decoding algorithms for BCH1 codes and LDPC codes. The hyperparameters of the simulation are shown in Table 1. As in ref. [22], node and edge attributes are also not used. For each list-element in the training parameters, we run the SGD-updates and use a pre-defined number of training iterations.
Table 1. Parameter of simulation.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>BCH Codes</th>
<th>5G LDPC Code</th>
<th>Regular LDPC Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>activation</td>
<td>tanh</td>
<td>ReLU</td>
<td>tanh</td>
</tr>
<tr>
<td>$T$</td>
<td>8</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>$f_{tn}$</td>
<td>20</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>$f_{tn}$</td>
<td>20</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>hidden units MLP</td>
<td>40</td>
<td>48</td>
<td>64</td>
</tr>
<tr>
<td>MLP layers</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>aggregation function</td>
<td>mean</td>
<td>sum</td>
<td>mean</td>
</tr>
<tr>
<td>LLR clipping</td>
<td>$\infty$</td>
<td>20</td>
<td>$\infty$</td>
</tr>
<tr>
<td>learning rate</td>
<td>$1 \times 10^{-3}, 1 \times 10^{-4}, 1 \times 10^{-5}$</td>
<td>$1 \times 50^{-4}, 1 \times 10^{-4}, 1 \times 10^{-5}$</td>
<td>$1 \times 20^{-4}, 1 \times 10^{-4}, 1 \times 10^{-5}$</td>
</tr>
<tr>
<td>batch size</td>
<td>635,000</td>
<td>635,000</td>
<td>550,000</td>
</tr>
<tr>
<td>train iteration</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.1. BCH Codes

First, the performance of the proposed SGNN1 and SGNN2-based channel decoding algorithms [22] are obtained, in which we use eight iterations as shown in Table 1; other hyperparameters are also provided in Table 1. As shown in Figure 3, the performances of Uncoded, BP, Weighted BP, GNN, SGNN1, and SGNN2-based channel decoding algorithms are consistently improving for the BCH(63,45) code. The blue vertical line is the shannon limit of the code rate corresponding to BCH codes. A fixed number of iterations $T = 8$ is used in training; its model parameters can be used to test a variable number of iterations $T = 2, 3, 4, 6, 8, 10$, which is also a major advantage of GNN similar to hypernetwork technology [26]. However, a GNN-based channel decoder with only three iterations achieves an almost equivalent performance compared to the traditional BP decoder with twenty iterations. The SGNN1-based channel decoding algorithm with only four iterations also achieves an equivalent performance. Overall, the BER performance of the SGNN1 algorithm degrades by 0.75 dB compared to the GNN-based channel decoding algorithm with the same number of iterations, and the BER performance of the SGNN2-based channel decoding algorithm degrades by 0.1 dB. However, SGNN1 and SGNN2 only use 2440 and 4840 parameters compared to the GNN, which uses 9640 parameters; the parameters are reduced to a quarter and a half of those of the GNN, respectively, as listed in Table 2. As the number of iterations increases, the BER performance improves, but when it increases to a certain number, the BER performance degrades, so it is necessary to introduce an early stop mechanism. The number of trainable parameters in ref. [12] is 2–3 orders of magnitude larger than our decoder. However, multiplication in our decoder is much larger than that in traditional communication decoding algorithms.

As shown in Figures 4 and 5, the BER performances of the Uncoded, BP, Weighted BP, GNN, SGNN1, and SGNN2-based channel decoding algorithms are consistently improving for the BCH (7,4) code of BPSK modulation and QPSK modulation. The GNN-based channel decoding algorithm with only four iterations achieves an almost equivalent BER performance compared to the traditional BP decoder with twenty iterations. Overall, the BER performance of the SGNN1-based channel decoding algorithm degrades by 0.15 dB compared to the GNN-based channel decoding algorithm with the same number of iterations, and the BER performance of the SGNN2-based channel decoding algorithm degrades by 0.05 dB. However, SGNN1 and SGNN2 only use 2440 and 4840 parameters compared to the GNN, which uses 9640 parameters. BCH (7,4) has the same number of parameters as BCH (63,45), and the parameters are reduced to a quarter and a half, respectively, as listed in Table 2. According to our experience, the training weights of BCH (63,45) cannot achieve the desired BER performance when used in BCH (7,4).
Figure 3. BER performance of GNN [22] and SGNN for BCH (63,45) code of BPSK modulation.

Figure 4. BER performance of GNN [22] and SGNN for BCH (7,4) code of BPSK modulation.

Figure 5. BER performance of GNN [22] and SGNN for BCH (7,4) code of QPSK modulation.
4.2. Regular LDPC Code

In order to increase the applicability of the GNN, we compare the GNN and the SGNN-based channel decoding algorithms for the regular LDPC code \((v = 3, c = 6)\), when the training parameter \(T\) is 10 and the codeword length \(N\) is 100. In order to achieve fast convergence of training, MLP adds multiple neural network layers, the result is shown in Figure 6. The blue vertical line is the shannon limit of the code rate corresponding to regular LDPC codes. The SGNN1-based channel decoding algorithm performs slightly better than the twenty iterations of the weighted BP algorithm. In addition, when \(N = 100\), the BER performances of the SGNN1 and SGNN2-based channel decoding algorithms are basically the same as that of the GNN-based channel decoding algorithm. As the length of the code increases, the BER performance gap between the SGNN algorithm and the GNN-based channel decoding algorithm gradually becomes larger, with a difference of about 0.2 dB; however, the BER performance gains between the BP algorithms and the GNN-based channel decoding algorithm tend to diminish with increasing \(n\), but the weight is reduced from 28,700 to 32,016,352, as listed in Table 2.

![Figure 6. BER performance of GNN [22] and SGNN for regular LDPC (3,6) code of BPSK modulation.](image)

<table>
<thead>
<tr>
<th>Codes</th>
<th>Number of Parameters in GNN</th>
<th>Number of Parameters in SGNN1</th>
<th>Number of Parameters in SGNN2</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCH (63,45)</td>
<td>9640</td>
<td>2440</td>
<td>4840</td>
</tr>
<tr>
<td>BCH (7,4)</td>
<td>9640</td>
<td>2440</td>
<td>4840</td>
</tr>
<tr>
<td>Regular LDPC (3,6)</td>
<td>12,657</td>
<td>3201</td>
<td>6353</td>
</tr>
<tr>
<td>5G NR LDPC (60,140)</td>
<td>18,929</td>
<td>4769</td>
<td>9489</td>
</tr>
</tbody>
</table>

4.3. 5G NR LDPC Code

In order to increase the applicability of the GNN network, we compare the 5G NR LDPC code [27] GNN network with the SGNN network, when the training parameter \(T\) is 10 and the codeword length \(N\) = 140, \(K\) = 60. The blue vertical line is the shannon limit of the code rate corresponding to 5G NR LDPC codes. As the 5G NR LDPC code requires rate-matching, the codewords tested all contain the same code rate, such as \(N = 280\), \(K = 120\), \(N = 420\), \(K = 180\). As shown in Figure 7, the same weights can be used to provide parameters for the longer codeword, and flexibility is also one of the advantages of GNN. We can adapt to the training LDPC code of simple codewords, but we can test different codewords. At the same time, please note that different code rates appear in training and testing, and the weights need to be adjusted at the same time. Both algorithms...
use flooding schedules. The updates of its VN nodes and FN nodes, as well as the required number of iterations, will change with different schedules. The difference between the BER performance of SGNN1 and the GNN-based channel decoding algorithm when testing the same code of $N = 140$, $K = 60$ is about 0.5 dB, and the difference between the BER performance of the SGNN2 and GNN-based channel decoding algorithms is about 0.1 dB when testing different codes of $N = 280$, $K = 150$. A small gain can be observed between SGNN and GNN-based channel decoding algorithms. There is little room for improvement in the deep learning GNN-based channel decoding algorithm and the BP algorithm for optimal settings for 5G LDPC codes, but there is room for further improvement in the SGNN algorithm.

![Figure 7. BER performance of GNN [22] and SGNN for 5G LDPC (60,140) code of BPSK modulation.](image)

**4.4. Complexity Analysis**

The number of weights in the SGNN1 and SGNN2-based channel decoding algorithms are greatly reduced, but the BER performance is nearly the same or slightly reduced, as listed in Table 2. We can find than the SGNN1 and SGNN2-based channel decoding algorithms have low complexity compared with that of the GNN-based channel decoding algorithm. The dense network is used to initialize graph nodes and return variable node values, and the number of multiplications in this process is small and is related to the output of the dense network. In each iteration of decoding, the MLPs network is used to update factor nodes and update variable nodes, so the number of layers of the MLPs network and the output of each dense network determine the number of multiplications here. As shown in Figure 2, a total of four MLPs are $208w$ in each iteration. The reduction in the number of parameters therefore requires fewer memory storage resources, which is more beneficial for implementation on resource-constrained communication hardware. But at the same time, investigating how to reduce the number of multiplications is a potential direction for future research.

**5. Open Issues**

As mentioned above, the complexity of GNN is much higher than that of the NBP algorithm due to the multiple use of MLPs. Although the SGN-based channel decoding algorithm reduces the number of parameters to even less than that of the NBP algorithm, the number of multiplications has not been reduced. Such a large amount of calculation limits its implementation in hardware such as FPGA, let alone enabling it to replace traditional decoding algorithms in communication chips. The trained codeword weights cannot reflect the same BER performance in other codewords. Therefore, possible future research directions are as follows:
A possible way to implement a universal GNN-based channel decoder is to train GNN-based channel decoder weights for multiple forward error correction (FEC) parallel codes.

The decoding complexity of the proposed SGNN is higher compared with that of BP, and other complexity reduction methods are put together to further reduce complexity, such as pruning, quantification, etc.

Extensions to non-AWGN channels and other modulation methods are possible.

6. Conclusions

This paper proposes two weight sharing methods to reduce the complexity of channel decoders based on SGNN1 and SGNN2, which have almost the same or a slightly reduced BER performance, while SGNN1 and SGNN2 decoding algorithms have fewer parameters in comparison with the GNN-based channel decoding algorithm. We have also provided a detailed algorithm flow of SGNN1 and SGNN2 algorithms. We conducted experiments on the proposed SGNN1- and SGNN2-based BCH decoders, regular LDPC decoders, and 5G NR LDPC decoders. Numerical results show that, compared with the original decoder, the decoders based on SGNN1 and SGNN2 have a comparable performance, while the proposed algorithm requires fewer memory storage resources. The proposed low-complexity decoding algorithms have advantages when applied to resource-constrained devices such as internet of things communications, but performance requirements need to be considered.

Author Contributions: Conceptualization, Y.L. and Q.W.; Methodology, Y.L.; Validation, Q.W. and X.C.; Formal analysis, Q.W.; Investigation, Q.W.; Resources, Q.W. and Y.L.; Data curation, Q.W. and X.C.; Writing—original draft, Q.W. and Y.L.; Writing—review and editing, B.K.N., C.-T.L. and Y.M.; Visualization, Y.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by the research funding of the Macao Polytechnic University, Macao SAR, China (Project no. RP/FCA-01/2023).

Institutional Review Board Statement: The study was conducted in accordance with the Declaration of Helsinki.

Informed Consent Statement: Informed consent was obtained from all subjects involved in the study.

Data Availability Statement: For privacy reasons, the data source of this article has not been made public.

Conflicts of Interest: The authors declare no conflict of interest.

References
6. Liang, Y.; Lam, C.; Ng, B. Joint-Way Compression for LDPC Neural Decoding Algorithm With Tensor-Ring Decomposition. IEEE Access 2023, 11, 22871–22879. [CrossRef]


Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.