Decoding of Moderate Length LDPC Codes via Learned Clustered Check Node Scheduling

Salman Habib†, Allison Beemer*, and Jörg Kliewer†
†Helen and John C. Hartmann Dept. of Electrical and Computer Engineering, New Jersey Institute of Technology
*Dept. of Mathematics, University of Wisconsin-Eau Claire

2021 17th International Symposium on Wireless Communication Systems (ISWCS) | 978-1-7281-7432-7/21/$31.00 ©2021 IEEE | DOI: 10.1109/ISWCS49558.2021.9562199

***Abstract*— In this work, we consider the sequential decoding of moderate length low-density parity-check (LDPC) codes via reinforcement learning (RL). The sequential decoding scheme is modeled as a Markov decision process (MDP), and an optimized decoding policy is subsequently obtained via RL. In contrast to our previous works, where an agent learns to schedule only a single check node (CN) within a group (cluster) of CNs per iteration, in this work we train the agent to schedule all CNs in a cluster, and all clusters in every iteration. That is, in each RL step, an agent learns to schedule CN clusters sequentially depending on the reward associated with the outcome of scheduling a particular cluster. We also propose a modified MDP and a uniform sequential decoding policy, enabling the RL-based decoder to be suitable for much longer LDPC codes than the ones studied in our previous work. The proposed RL-based decoder exhibits an SNR gain of almost 0.8 dB for fixed bit error probability over the standard flooding approach.**

I. Introduction

Binary low-density parity-check (LDPC) codes are sparse graph-based channel codes. Due to their excellent error correcting performance for symmetric binary input channels [1], [2], they have recently been standardized for data communication in the 5G cellular new radio standard [3], [4]. Tanner graphs of LDPC codes are sparse bipartite graphs whose vertex sets are partitioned into check nodes (CNs) and variable nodes (VNs). Typically, iterative decoding on an LDPC Tanner graph is carried out via flooding: all CNs and VNs are updated simultaneously in each iteration [5]. In contrast, sequential LDPC decoding seeks to optimize the order of all CN (or VN) updates to improve the convergence speed and/or the decoding performance with respect to the flooding scheme [6], [7]. In this work, we study the performance of a sequential LDPC decoding scheme in which groups (clusters) of CNs are updated sequentially. In each scheduling instant, a cluster's neighbors are updated via flooding based on the latest messages propagated by its neighboring clusters.

A *node-wise scheduling* (NS) algorithm was proposed in [8], where a single CN is scheduled per decoding iteration based on its residual, given by the magnitude of the difference between two successive messages emanating from that CN. Intuitively, scheduling CNs with higher residuals is expected to lead to faster and more reliable decoding compared to the flooding scheme. Our previous work in [9], [10] proposes a reinforcement learning (RL)-based NS (RL-NS) scheme which obviates the need for computing residuals. In [9], we consider model-free RL methods by (1) computing the Gittins index of each CN, and (2) utilizing standard Q-learning [11], [12]. In [10], in addition to model-free RL, we also consider a model-based RL-NS approach based on Thompson sampling. In this work, we improve the sequential decoding performance even further by implementing an RL-based scheme that sequentially updates all clusters in each iteration, as opposed to a single CN as considered in our previous works, until a stopping condition or a maximum number of iterations is reached.

The proposed sequential LDPC decoding algorithm is modeled as a finite Markov decision process (MDP) [12], where the code's Tanner graph is viewed as an environment with $\lceil m/z \rceil$ possible actions (cluster selections), where m denotes the total number of CNs in the Tanner graph and z is the cluster size. Then, we apply RL to learn an action-value function that determines how beneficial a particular choice of cluster is for optimizing the cluster scheduling order. Specifically, we take the optimal order to be the one which yields a codeword output with the smallest number of propagated CN to VN messages using the *belief propagation* (BP) decoding algorithm. The action-value function is learned using both standard Q-learning and deep reinforcement learning (DRL) [12], [13], [14]; action-values are predicted using artificial neural networks (NNs) in the latter.

Given a cluster of CNs in the Tanner graph, let the *output* of that cluster at a particular iteration be the binary sequence resulting from hard-decisions on the posterior log-likelihood ratios (LLRs) computed by the (ordered) neighboring VNs. The state of the MDP in our RL framework is then given by the collection of all possible (cluster, cluster state) pairs. As a result, for the cluster sizes considered in this paper, the proposed RL scheme encounters a much smaller state space cardinality than the RL-NS schemes of [9], [10], in which the state space contains sequences of quantized real CN values. This modification renders the proposed RL-based decoding approach suitable for much longer block lengths of LDPC codes than those considered in our previous work. Our RL-based LDPC decoder outperforms standard BP decoding schemes in complexity by reducing the number of CN to

This work has been supported in part by U.S. NSF grant ECCS-1711056 and the U.S. Army Research Laboratory under Cooperative Agreement Number W911NF-17-2-0183.

VN message updates required for convergence. Further, for moderate-length LDPC codes, our proposed sequential decoder is able to provide an SNR gain over flooding decoding of up to 0.8 dB for fixed bit error probability.

II. Preliminaries

A. Low-density Parity-check Codes

An $[n, k]$ binary linear code is a k-dimensional subspace of $\mathbb{F}_2^n$, and may be defined as the kernel of a binary *parity-check matrix* $\mathbf{H} \in \mathbb{F}_2^{m\times n}$, where $m \geq n-k$. The code's *block length* is n, and *rate* is $(n - \text{rank}(\mathbf{H}))/n$. The *Tanner graph* of a linear code with parity-check matrix $\mathbf{H}$ is the bipartite graph $G_{\mathbf{H}} = (V \cup C, E)$, where $V = \{v_0, \ldots, v_{n-1}\}$ is a set of variable nodes (VNs) corresponding to the columns of $\mathbf{H}$, $C = \{c_0, \ldots, c_{m-1}\}$ is a set of check nodes (CNs) corresponding to the rows of $\mathbf{H}$, and edges in E correspond to 1's in $\mathbf{H}$ [15]. LDPC codes are a class of highly competitive linear codes defined via sparse parity-check matrices or, equivalently, sparse Tanner graphs [1]; they are amenable to low-complexity graph-based message-passing decoding algorithms, making them ideal for practical applications. BP iterative decoding, considered here, is one such algorithm.

In this work, we present experimental results for a standardized [384, 256]-Wireless Regional Area Network (WRAN) LDPC code [16] and a (γ, p)-regular array-based (AB-) LDPC code. In general, a (γ, k)-regular LDPC code is defined by a parity-check matrix with constant column and row weights equal to γ and k, respectively [1]. A (γ, p) AB-LDPC code, where p is prime, is a (γ, p)-regular LDPC code with additional structure in its parity-check matrix, $\mathbf{H}(\gamma, p)$ [17]. In particular,

$$\mathbf{H}(\gamma,p) = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \mathbf{I} & \cdots & \mathbf{I} \\ \mathbf{I} & \sigma & \sigma^2 & \cdots & \sigma^{p-1} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ \mathbf{I} & \sigma^{\gamma-1} & \sigma^{2(\gamma-1)} & \cdots & \sigma^{(\gamma-1)(p-1)} \end{bmatrix}, \quad (1)$$

where σ^z denotes the circulant matrix obtained by cyclically left-shifting the entries of the $p \times p$ identity matrix $\mathbf{I}$ by z (mod p) positions. Notice that $\sigma^0 = \mathbf{I}$. In this work, *lifted* LDPC codes are obtained by replacing the non-zero (resp., zero) entries of the parity-check matrix with randomly generated permutation (resp., all-zero) matrices.

B. Reinforcement Learning

In an RL problem, an agent (learner) interacts with an environment whose *state space* can be modeled as a finite MDP [12]. The agent takes *actions* that alter the state of the environment and receives a *reward* in return for each action, with the goal of maximizing the total reward in a series of actions. The optimized sequence of actions is obtained by employing a policy which utilizes an *action-value function* to determine how beneficial an action is for maximizing the long-term expected reward. In the remainder of the paper, let $[[x]] \triangleq \{0, \ldots, x-1\}$, where x is a positive integer. Suppose that an environment allows m possible actions, and let the random variable $A_\ell \in [[m]]$, with realization a, represent the index of an action taken by the agent during learning step ℓ. Let S_ℓ, with realization $s \in \mathbb{Z}$, represent the current state of the environment before taking action A_ℓ, and let $S_{\ell+1}$, with realization s', represent a new state of the MDP after executing A_ℓ. Let a state space $\mathcal{S}$ contain all possible state realizations. Also, let $R_\ell(S_\ell, A_\ell, S_{\ell+1})$ be the reward yielded at step ℓ after taking action A_ℓ in state S_ℓ.

Optimal policies for MDPs can be estimated via Monte Carlo techniques such as *Q-learning* [18], [19], [20]. The estimated action-value function $Q_\ell(S_\ell, A_\ell)$ in Q-learning represents the expected long-term reward achieved by the agent at step ℓ after taking action A_ℓ in state S_ℓ. To improve the estimation in each step, the action-value function is adjusted according to a recursion

$$Q_{\ell+1}(s,a) = (1-\alpha)Q_\ell(s,a) + \alpha\Big(R_\ell(s,a,s') + \beta \max_{a' \in [[m]]} Q_\ell(s',a')\Big), \quad (2)$$

where s' represents the new state of the MDP after taking action a in state s, $0 < \alpha < 1$ is the *learning rate*, β is the *reward discount rate*, and $Q_{\ell+1}(s,a)$ is a future action-value resulting from action a in the current state s [20]. Note that the new state is updated with each action. The optimal policy for the agent, $\pi^{(\ell)}$, in state s is given by

$$\pi^{(\ell)} = \arg\max_a Q_\ell(s,a), \quad (3)$$

where ℓ is the total number of learning steps elapsed after observing the initial state S_0.

III. RL for Sequential LDPC Decoding

An RL-based sequential decoding (RL-SD) scheme consists of a BP decoding algorithm in which the environment is given by the Tanner graph of the LDPC code, and the optimized sequence of actions, *i.e.*, the scheduling of individual clusters, is obtained using a suitable RL algorithm such as Q-learning. A single cluster scheduling step is carried out by sending messages from all CNs of a cluster to their neighboring VNs, and subsequently sending messages from these VNs to their CN neighbors. That is, a selected cluster executes one iteration of flooding in each decoding instant. Every cluster is scheduled exactly once within a single decoder iteration. Sequential cluster scheduling is carried out until a stopping condition is reached, or an iteration threshold is exceeded. The RL-SD method relies on a cluster scheduling policy based on an action-value function, which is estimated offline using the RL techniques to be discussed in Section IV.

An example of a cluster-induced subgraph for the case $z = 2$ is shown in Fig. 1. Since the full LDPC Tanner graph is connected and contains cycles, there exist dependencies between the messages propagated by the different clusters of the LDPC code. Consequently, the output of a cluster may depend on messages propagated by previously-scheduled clusters. To improve RL performance, we ensure that the clusters

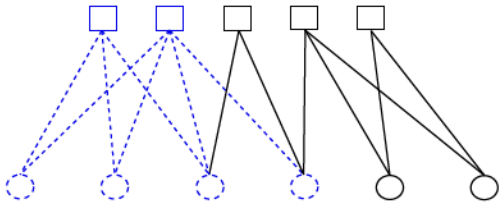

Fig. 1: Example of a cluster-induced subgraph, shown with blue squares (CNs), edges, and circles (VNs). The cluster with size $z = 2$ is given by the solid blue square CNs.

are chosen to be as independent as possible. The choice of clustering is determined prior to learning using the cycle-maximization method discussed in [9], [10]: in short, clusters are selected to maximize the number of cycles in the cluster-induced subgraph to minimize inter-cluster dependencies.

Let $\mathbf{x} = [x_0, \ldots, x_{n-1}]$ and $\mathbf{y} = [y_0, \ldots, y_{n-1}]$ represent the transmitted and the received words, respectively, where for each $v \in [[n]]$, $x_v \in \{0, 1\}$ and $y_v = (-1)^{x_v} + z$ with $z \sim \mathcal{N}(0, \sigma^2)$. The posterior log-likelihood ratio (LLR) of x_v is expressed as $L_v = \log \frac{\Pr(x_v=1|y_v)}{\Pr(x_v=0|y_v)}$. Let $\hat{L}_I^{(v)} = \sum_{c \in \mathcal{N}(v)} m_{c \to v}^{(I)} + L_v$ be the posterior LLR computed by VN v during iteration I, where $\hat{L}_0^{(v)} = L_v$ and $m_{c \to v}^{(I)}$ is the message received by VN v from neighboring CN c in iteration I. Similarly, let $\hat{L}_I^{(j,a)}$ be the posterior LLR computed during iteration I by VN j in the subgraph induced by the cluster with index $a \in [[\lceil m/z \rceil]]$. Hence, $\hat{L}_I^{(j,a)} = \hat{L}_I^{(v)}$ if VN v in the Tanner graph is also the jth VN in the subgraph induced by the cluster with index a.

After scheduling cluster a during iteration I, the output $\hat{\mathbf{x}}_a^{(I)} = [\hat{x}_{0,a}^{(I)}, \ldots, \hat{x}_{l_a-1,a}^{(I)}] \in \{0, 1\}^{l_a}$ of cluster a, where $l_a \leq k_{\max} z$, and $k_{\max}$ is the maximum CN degree of the cluster. is the number of VNs adjacent to cluster a, is obtained by taking hard decisions on the vector of posterior LLRs $\hat{\mathbf{L}}_{I,a} = [\hat{L}_I^{(0,a)} \ldots \hat{L}_I^{(l_a-1,a)}]$, computed according to

$$\hat{x}_{j,a}^{(I)} = \begin{cases} 0, & \text{if } \hat{L}_I^{(j,a)} \geq 0, \\ 1, & \text{otherwise.} \end{cases} \tag{4}$$

We call $\hat{\mathbf{x}}_a^{(I)}$ the output of cluster a: it is comprised of the bits reconstructed by the sequential decoder after scheduling cluster a during iteration I, *i.e.,* the state of the cluster is a sequence of hard-decision VN values associated with the cluster. We denote the index of a realization of $\hat{\mathbf{x}}_a^{(I)}$ in iteration I by $s_a^{(I)} \in [[2^{l_a}]]$. The signals $\hat{\mathbf{x}}_0^{(I)}, \ldots, \hat{\mathbf{x}}_{\lceil m/z \rceil - 1}^{(I)}$ at the end of decoder iteration I forms the state of the MDP associated with our RL scheme. At the end of iteration I, we may obtain the fully reconstructed signal estimate $\hat{\mathbf{x}} = [\hat{x}_0, \ldots, \hat{x}_{n-1}]$.

During the learning phase, our RL method informs the agent of the current state of the decoder and the reward obtained after performing an action (propagating a cluster). Based on these observations, the agent takes future actions, to enhance the total reward earned, which alters the state of the environment as well as the future reward. Given that the transmitted signal $\mathbf{x}$ is known during the training phase, let $\mathbf{x}_a = [x_{0,a}, \ldots, x_{l_a-1,a}]$ be a vector containing the l_a bits of $\mathbf{x}$ that are reconstructed in $\hat{\mathbf{x}}_a^{(I)}$ by cluster a. In each learning step ℓ, the reward R_a obtained by the agent after scheduling cluster a is defined as

$$R_a = \frac{1}{l_a} \sum_{j=0}^{l_a-1} \mathbb{1}(x_{j,a} = \hat{x}_{j,a}), \tag{5}$$

where $\mathbb{1}(\cdot)$ denotes the indicator function. Thus, the reward earned by the agent after scheduling cluster a is identical to the probability that the corrupted bits corresponding to the transmitted bits $x_{0,a}, \ldots, x_{l_a-1,a}$ are correctly reconstructed.

The RL-SD scheme is shown in Algorithm 1. The algorithm inputs are the soft channel information vector $\mathbf{L} = [L_0, \ldots, L_{n-1}]$ comprised of LLRs and a parity-check matrix $\mathbf{H}$ of the LDPC code, and $\hat{\mathbf{L}}_\ell = [\hat{L}_\ell^{(0)}, \ldots, \hat{L}_\ell^{(n-1)}]$ is initialized using $\mathbf{L}$. The output is the reconstructed signal $\hat{\mathbf{x}}$ obtained after executing at most $I_{\max}$ decoding iterations, or until the stopping condition shown in Step 32 is reached. The optimized scheduling order in Step 9, learned using the methods discussed in Section IV, is dynamic and depends both on the graph structure and on the received channel values.

Note that the RL-SD scheme can be viewed as a sequential generalized LDPC (GLDPC) decoder when $z > 1$, where BP decoding of a cluster-induced subgraph is analogous to decoding a sub-code of a GLDPC code. When $z = 1$, each cluster represents a single parity-check code, as is the case in a standard LDPC code.

IV. Learning the Scheduling Policy

Let $\hat{\mathbf{x}}_a^{(\ell)}$ denote the state of the MDP after scheduling a cluster with index a during learning step ℓ, and let $s_a \in [[2^{l_a}]]$ refer to the index of a realization of $\hat{\mathbf{x}}_a^{(\ell)}$. Thus, s_a also refers to the state of the MDP. Since the state-space of the clusters are pairwise disjoint, set $\mathcal{S}$ of our MDP contains $\sum_{a \in [[\lceil m/z \rceil]]} 2^{l_a}$ realizations of all the cluster outputs $\hat{\mathbf{x}}_0^{(\ell)}, \ldots, \hat{\mathbf{x}}_{\lceil m/z \rceil - 1}^{(\ell)}$, where a realization can be thought of as a (cluster, cluster state) pair. The action space is defined as $\mathcal{A} = [[\lceil m/z \rceil]]$. In the following, we discuss two distinct Q-learning-based RL approaches for solving the sequential decoding problem.

A. Using Standard Q-learning

For MDPs with moderately large state spaces, we utilize a standard Q-learning approach for determining the optimal cluster scheduling order, where the action-value for choosing cluster a in state s_a is given by

$$Q_{\ell+1}(s_a, a) = (1 - \alpha) Q_\ell(s_a, a) + \alpha \Big(R_a + \beta \max_{a' \in [[\lceil m/z \rceil]]} Q_\ell(s'_a, a') \Big). \tag{6}$$

In each learning step ℓ, cluster a is selected via an ϵ-greedy approach according to

$$a = \begin{cases} \text{selected uniformly at random w.p. } \epsilon \text{ from } \mathcal{A}, \\ \pi_Q^{(\ell)} \text{ selected w.p. } 1 - \epsilon, \end{cases} \tag{7}$$

Algorithm 1: The RL-SD Scheme

Input : $\mathbf{L}$, $\mathbf{H}$
Output: $\hat{\mathbf{x}}$

Initialization:
  $I \leftarrow 0$
  $m_{c\to v} \leftarrow 0$ // for all CN to VN messages
  $m_{v\to c} \leftarrow L_v$ // for all VN to CN messages
if $I < I_{\max}$ **then**
  foreach *cluster index* a **do**
    Determine state $s_a^{(I)}$
  end
  Obtain an optimized cluster scheduling order $\pi_i^{*(I)}$
  foreach *cluster with index* a_i **do**
    // decode cluster via flooding
    foreach *CN* c *in cluster* a_i **do**
      foreach *VN* $v \in \mathcal{N}(c)$ **do**
        compute and propagate $m_{c\to v}^{(I)}$
      end
    end
    foreach *VN* v *in the subgraph of cluster* a_i **do**
      foreach *CN* $c \in \mathcal{N}(v)$ **do**
        compute and propagate $m_{v\to c}^{(I)}$
      end
      $\hat{L}_I^{(v)} \leftarrow \sum_{c\in\mathcal{N}(v)} m_{c\to v}^{(I)} + L_v$ // update posterior LLR
    end
    // hard-decision step
    foreach *VN* v *in the subgraph of cluster* a_i **do**
      if $\hat{L}_I^{(v)} \geq 0$ **then**
        $\hat{x}_{v,a_i}^{(I)} \leftarrow 0$
      end
      else
        $\hat{x}_{v,a_i}^{(I)} \leftarrow 1$
      end
    end
    $i \leftarrow i+1$
  end
  if $\mathbf{H}\hat{\mathbf{x}} = \mathbf{0}$ **then**
    break // stopping condition reached
  end
  $I \leftarrow I+1$
end

where $\pi_Q^{(\ell)} = \arg\max_{a\in[[\lceil m/z\rceil]]} Q_\ell(s_a, a)$. For ties (as in the first iteration of Algorithm 2 for $\ell = 0$ and the first $\mathbf{L}$), we choose an action uniformly at random from all the maximizing actions. During inference, the optimized cluster scheduling policy of standard Q-learning, $\pi_i^{*(I)}$, for scheduling the ith cluster during decoder iteration I is expressed as

$$\pi_i^{*(I)} = \underset{a_i\in\mathcal{A}\setminus\{a_0,\ldots,a_{i-1}\}}{\arg\max} Q^*(s_{a_i}^{(I)}, a_i), \tag{8}$$

where $Q^*(s_{a_i}^{(I)}, a_i)$ represents the optimized action value after training has been accomplished. The policy $\pi_i^{*(I)}$ is incorporated in Step 9 of Algorithm 1 to determine the optimized cluster scheduling order.

A standard Q-learning method for sequential LDPC decoding is shown in Algorithm 2. The input to this algorithm is a set $\mathscr{L} = \{\mathbf{L}_0, \ldots, \mathbf{L}_{|\mathscr{L}|-1}\}$ containing $|\mathscr{L}|$ realizations of $\mathbf{L}$ over which Q-learning is performed, and a parity-check matrix $\mathbf{H}$. The output is $Q^*(s_{a_i}^{(I)}, a_i)$. For each $\mathbf{L} \in \mathscr{L}$, the action-value function in (6) is recursively updated $\ell_{\max}$ times.

Algorithm 2: Standard Q-learning for Sequential LDPC Decoding

Input : $\mathscr{L}$, $\mathbf{H}$
Output: optimized action value function invoked in (8)

Initialization: $Q_0(s_a, a) \leftarrow 0$ for all s_a and a
for *each* $\mathbf{L} \in \mathscr{L}$ **do**
  $\ell \leftarrow 0$
  $\hat{\mathbf{L}}_\ell \leftarrow \mathbf{L}$
  Determine initial states of all clusters using (4)
  while $\ell < \ell_{\max}$ **do**
    select cluster a according to (7)
    // decode cluster via flooding
    foreach *CN* c *in cluster* a **do**
      foreach *VN* $v \in \mathcal{N}(c)$ **do**
        compute and propagate $m_{c\to v}^{(\ell)}$
      end
    end
    foreach *VN* v *in the subgraph of cluster* a **do**
      foreach *CN* $c \in \mathcal{N}(v)$ **do**
        compute and propagate $m_{v\to c}^{(\ell)}$
      end
      $\hat{L}_\ell^{(v)} \leftarrow \sum_{c\in\mathcal{N}(v)} m_{c\to v}^{(\ell)} + L_v$ // update posterior LLR
    end
    // determine cluster output
    foreach *VN* v *in the subgraph of cluster* a **do**
      if $\hat{L}_\ell^{(v)} \geq 0$ **then**
        $\hat{x}_{v,a}^{(\ell)} \leftarrow 0$
      end
      else
        $\hat{x}_{v,a}^{(\ell)} \leftarrow 1$
      end
    end
    determine index s'_a of $\hat{\mathbf{x}}_a$
    update R_a according to (5)
    compute $Q_{\ell+1}(s_a, a)$ according to (6)
    $s_a \leftarrow s'_a$
    $\ell \leftarrow \ell+1$
  end
end

B. Using Deep RL

For MDPs with very large state spaces, the action-value function $Q_\ell(s,a)$ can be approximated as $Q_\ell(s,a;\mathbf{W})$ using a deep learning model with tensor $\mathbf{W}$ representing the weights connecting all layers in the NN [12]. We utilize a separate NN with weight $\mathbf{W}_\ell^{(a)}$ for each cluster in each learning step ℓ, since a single NN cannot distinguish between the signals $\hat{\mathbf{x}}_0^{(\ell)}, \ldots, \hat{\mathbf{x}}_{\lceil m/z\rceil-1}^{(\ell)}$, and hence the rewards $R_0, \ldots, R_{\lceil m/z\rceil-1}$ generated by the $\lceil m/z\rceil$ different clusters. The NN corresponding to each cluster learns to map the cluster output $\hat{\mathbf{x}}_a^{(\ell)}$ to a vector of $\lceil m/z\rceil$ predicted action-values $[Q_\ell(s'_a, 0; \mathbf{W}_\ell^{(a)}), \ldots, Q_\ell(s'_a, \lceil m/z\rceil - 1; \mathbf{W}_\ell^{(a)})]$.

V. Experimental Results

In this section, we test the performance of our RL-SD scheme in Algorithm 1, where the cluster selection policy of

Step 9 is learned using both DRL and standard Q-learning. As a benchmark, we compare the RL-SD scheme with flooding (*i.e.*, all clusters are updated simultaneously per iteration) and the scheme where the cluster scheduling order is randomly generated. We utilize each scheme for decoding both $[384, 256]$-WRAN irregular (see [16]) and $(3, 5)$ AB-LDPC codes. For both codes, the choice of block length (at most 500 bits) is influenced by the run-time of Algorithm 2 on our system.

Note that the LLR vectors used for training are sampled uniformly at random over a range of A equally-spaced SNR values for a given code. Hence, there are $|\mathscr{L}|/A$ LLR vectors in $\mathscr{L}$ for each SNR value considered, For both considered codes, we let the learning parameters be as follows: $\alpha = 0.1$, $\beta = 0.9$, $\epsilon = 0.6$, $\ell_{\max} = 50$, $A = 5$, and $|\mathscr{L}| = 5 \times 10^5$, where $|\mathscr{L}|$ is chosen to ensure that the training is as accurate as possible without incurring excessive run-time for Algorithm 2. Once RL is accomplished using either DRL or standard Q-learning, the corresponding cluster scheduling policy for each code is incorporated in Step 9 of Algorithm 1, resulting in RL-SD for that code. For decoding, we let $I_{\max} = 50$ (resp., 5) for the AB (resp., WRAN) code.[1] In the case of DRL, each cluster NN is based on a feed-forward architecture with an input layer of size l_a, two hidden layers of sizes 250 and 125, respectively, and an output layer of size $\lceil m/z \rceil$. The activation function used for the hidden and output layers are rectified linear unit and sigmoid, respectively.

For both training and inference, we consider the AWGN channel and transmit all-zero codewords using BPSK modulation. Training with the all-zero codeword is sufficient as, due to the symmetry of the BP decoder and the channel, the decoding error is independent of the transmitted signal [22, Lemma 4.92]. For performance measures, we consider both the bit error rate (BER), given by $\Pr[\hat{x}_v \neq x_v]$, $v \in [[n]]$, and the frame error rate (FER), given by $\Pr[\hat{\mathbf{x}} \neq \mathbf{x}]$. In the case of the WRAN LDPC code, we consider $z = 1$ only as this code has several degree-11 CNs which render both learning schemes too computationally intensive for $z > 1$. On the other hand, for the AB code, multiple cluster sizes chosen from $z \in \{1, 2, 3\}$ are used for both the random and RL-SD schemes. For $z \in \{1, 2\}$, we employ standard Q-learning to learn the cluster scheduling policy. For $z = 3$, DRL is utilized, as standard Q-learning is not feasible due to the significantly increased state space. Note that we use the same number of training examples for both standard Q-learning and DRL.

The BER vs. channel signal-to-noise ratio (SNR), in terms of E_b/N_0 in dB, for the $[384, 256]$-WRAN and $(3, 5)$ AB-LDPC codes using these decoding techniques are shown in Figs. 2 and 4, respectively. The experimental results reveal that sequential decoding of clusters outperforms the flooding scheme. Furthermore, regardless of the cluster size, the RL-SD scheme outperforms the random sequential scheduling schemes, revealing the benefit of RL. For both codes, the RL-SD scheme outperforms the other decoding schemes, including the state-of-the art hyper-network decoder of [21] (in case of the WRAN LDPC code) with a gain of around 0.5 dB for fixed BER. Note that for both codes, sequential decoding performance improves as the cluster size is reduced, mainly because the subgraphs induced by the smaller clusters are less likely to contain detrimental objects, such as cycles and absorbing sets. The FER vs. SNR performance shown in Figs. 3 and 5 show similar behavior.

In Table I, we compare the average number of CN to VN messages propagated in the considered decoding schemes to attain the results in Figs. 2-5. We note that the RL-SD scheme, on average, generates a lower number of CN to VN messages when compared to the other decoding schemes, irrespective of the cluster size. Thus, the RL-SD scheme also provides a significant reduction in message-passing complexity for moderate length LDPC codes.

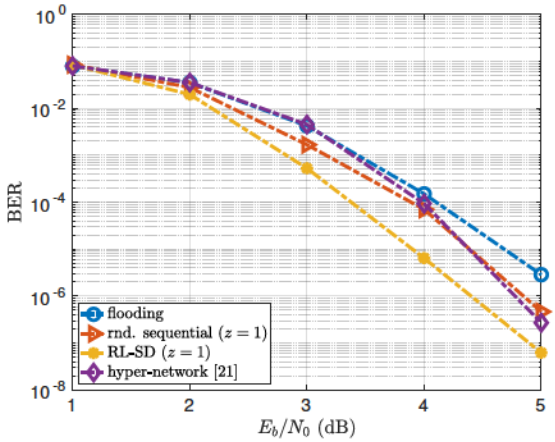

Fig. 2: BER results using different BP decoding schemes for a $[384, 256]$-WRAN LDPC code with block length $n = 384$.

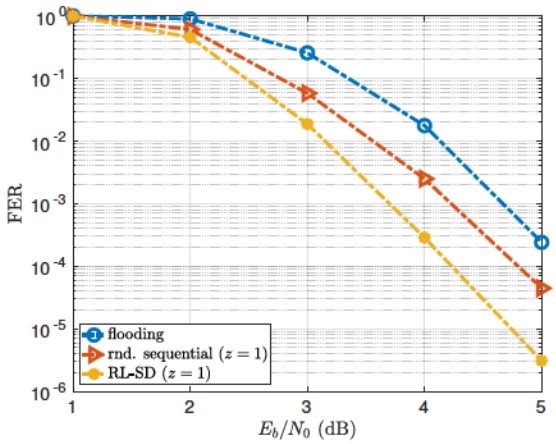

Fig. 3: FER results using different BP decoding schemes for a $[384, 256]$-WRAN LDPC code with block length $n = 384$.

VI. Conclusion

We presented novel RL-based sequential decoding schemes to optimize the scheduling of CN clusters for moderate length LDPC codes. In contrast to our previous work, the main contributions of this work include a new state space model built using the collection of possible outputs of individual

[1]We choose $I_{\max} = 5$ in case of the WRAN code for comparison with the hyper-network scheme of [21].

SNR (dB)	1	2	3
flooding	6480	6422	5171
random ($z = 1$)	6480	5827	3520
RL-SD ($z = 1$)	6467	5450	3179

SNR (dB)	1	2	3
flooding	63750	16409	8123
random ($z = 3$)	44338	11102	5005
RL-SD ($z = 3$)	40448	10694	4998
random ($z = 2$)	36328	10254	4994
RL-SD ($z = 2$)	31383	7349	4225
random ($z = 1$)	59750	10692	4812
RL-SD ($z = 1$)	51250	6240	3946

TABLE I: Average number of CN to VN messages propagated in various decoding schemes for a $[384, 256]$-WRAN (left) and $(3, 5)$ AB- (right) LDPC codes to attain the results shown in Figs. 2-5.

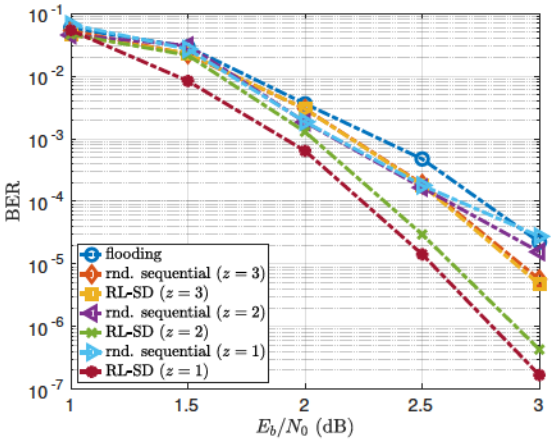

Fig. 4: BER results using different BP decoding schemes for a $(3, 5)$ AB-LDPC code with block length $n = 500$.

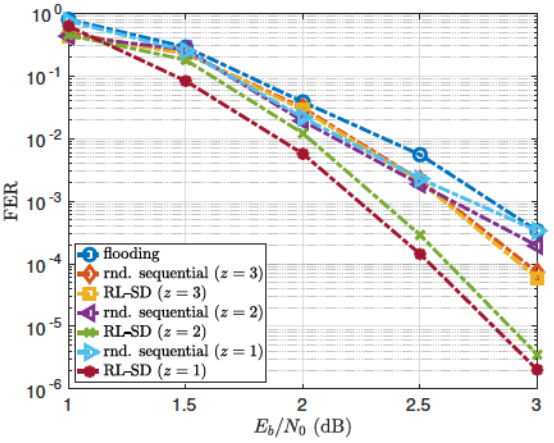

Fig. 5: FER results using different BP decoding schemes for a $(3, 5)$ AB-LDPC code with block length $n = 500$.

clusters, and a scheduling approach that updates all CN clusters sequentially within each decoder iteration. We employed DRL for cluster size 3 and standard Q-learning for smaller clusters. Experimental results show that by learning the cluster scheduling order, we can outperform a random scheduling scheme, irrespective of the cluster size. The performance gains include lowering both BER and message-passing complexity.

REFERENCES

[1] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inf. Theory*, vol. 8, no. 1, pp. 21–28, Jan 1962.

[2] D. J. Costello, Jr., L. Dolecek, T. Fuja, J. Kliewer, D. G. M. Mitchell, and R. Smarandache, "Spatially coupled sparse codes on graphs: theory and practice," *IEEE Commun. Mag.*, vol. 52, no. 7, pp. 168–176, 2014.

[3] S. Chung, G. D. Forney, T. J. Richardson, and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit," *IEEE Commun. Lett.*, vol. 5, pp. 58–60, Feb. 2001.

[4] Y. Kou, S. Lin, and M. Fossorier, "Low density parity-check codes based on finite geometries: A rediscovery and new results," *IEEE Trans. Inf. Theory*, vol. 47, pp. 2711–2736, Nov. 2001.

[5] M. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of Low Density Parity Check codes based on belief propagation," *IEEE Trans. Commun.*, vol. 47, pp. 673–680, May 1999.

[6] J. Zhang and M. Fossorier, "Shuffled belief propagation decoding," in *Proc. 36th Asilomar Conf. Signals, Syst. Comput.*, 2002, pp. 8–15.

[7] H. Kfir and I. Kanter, "Parallel versus sequential updating for belief propagation decoding," *Physica A*, vol. 330, pp. 259–270, 2003.

[8] A. V. Casado, M. Griot, and R. D. Wesel, "LDPC decoders with informed dynamic scheduling," *IEEE Trans. Commun.*, vol. 58, no. 12, pp. 3470–3479, Dec 2010.

[9] S. Habib, A. Beemer, and J. Kliewer, "Learning to decode: Reinforcement learning for decoding of sparse graph-based channel codes," *Adv. in Neural Inf. Processing Systems*, vol. 33, pp. 22 396–22 406, 2020.

[10] S. Habib, A. Beemer, and Kliewer, "Belief propagation decoding of short graph-based channel codes via reinforcement learning," *IEEE Journal on Sel. Areas in Inf. Theory*, vol. 2, no. 2, pp. 627–640, 2021.

[11] J. C. Gittins, "Bandit processes and dynamic allocation indices," *J. R. Statistics Soc. B*, vol. 41, no. 2, pp. 148–163, 1979.

[12] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction, 2nd Edition*. The MIT Press Cambridge, 2018.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, and G. O. et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, 2015.

[14] Y. Li, "Deep reinforcement learning," [Online]. Available: arXiv.org, arXiv:1810.06339v1 [cs.LG], 2018.

[15] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inf. Theory*, vol. 27, no. 5, pp. 547–553, Sep 1981.

[16] "TU Kaiserslautern channel codes database," [Online]. Available: https://www.uni-kl.de/channel-codes/channel-codes-database/more-ldpc-codes, 2015.

[17] J. L. Fan, "Array codes as low-density parity-check codes," in *Proc. of Intl. Symp. on Turbo Codes and Rel. Topics*, 2000, pp. 543–546.

[18] M. O. Duff, *Q-Learning for Bandit Problems*. CMPSCI Technical Report, 1995.

[19] F. Carpi, C. Hager, M. Martalo, R. Raheli, and H. D. Pfister, "Reinforcement learning for channel coding: Learned bit-flipping decoding," *Proc. of 57th Allerton Conf. on Commun., Control and Computing*.

[20] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, 1989.

[21] E. Nachmani and L. Wolf, "Hyper-graph-network decoders for block codes," in *Adv. in Neural Inf. Processing Systems*, 2019, pp. 2329–2339.

[22] T. J. Richardson and R. L. Urbanke, "Modern coding theory," *Cambridge University Press*, 2008.