Belief Propagation Decoding of Short Graph-Based Channel Codes via Reinforcement Learning

Salman Habib, Allison Beemer, and Jörg Kliewer, *Senior Member, IEEE*

***Abstract*—In this work, we consider the decoding of short sparse graph-based channel codes via reinforcement learning (RL). Specifically, we focus on low-density parity-check (LDPC) codes, which for example have been standardized in the context of 5G cellular communication systems due to their excellent error correcting performance. LDPC codes are typically decoded via belief propagation on the corresponding bipartite (Tanner) graph of the code via flooding, i.e., all check and variable nodes in the Tanner graph are updated at once. We model the node-wise sequential LDPC scheduling scheme as a Markov decision process (MDP), and obtain optimized check node (CN) scheduling policies via RL to improve sequential decoding performance as compared to flooding. In each RL step, an agent decides which CN to schedule next by observing a reward associated with each choice. Repeated scheduling enables the agent to discover the optimized CN scheduling policy which is later incorporated in our RL-based sequential LDPC decoder. In order to reduce RL complexity, we propose a novel graph-induced CN clustering approach to partition the state space of the MDP in such a way that dependencies between clusters are minimized. Compared to standard decoding approaches from the literature, some of our proposed RL schemes not only improve the decoding performance, but also reduce the decoding complexity dramatically once the scheduling policy is learned. By concatenating an outer Hamming code with an inner LDPC code which is decoded based on our learned policy, we demonstrate significant improvements in the decoding performance compared to other LDPC decoding policies.**

***Index Terms*—Reinforcement learning, belief propagation, LDPC codes, optimization.**

I. Introduction

BINARY low-density parity-check (LDPC) codes are sparse graph-based channel codes whose rates approach the capacity of symmetric binary input channels [3], [4]. Due to their excellent error correcting performance, they have recently been standardized for data communication in the 5G cellular new radio standard [5], [6]. LDPC codes are decoded via iterative algorithms, such as *belief propagation* (BP), which operate on the code's *Tanner graph* representation [7]. 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 a Tanner graph is carried out via flooding: all CNs and VNs are updated simultaneously [8]. The flooding schedule is computationally intensive compared to sequential scheduling, where nodes are updated serially based on the latest messages propagated by their neighbors. Sequential scheduling problems deal with finding the optimized order of node updates to improve the convergence speed and/or the decoding performance as compared to the flooding scheme.

A sequential CN scheduling scheme, so-called *node-wise scheduling* (NS) was proposed in [9], where the scheme's criterion for selecting the next CN depends on *residuals*, given by the magnitude of the difference between two successive messages emanating from each CN. In NS, all CN to VN messages corresponding to a CN with the highest residual are propagated simultaneously. NS of an iterative decoder can lead to improved performance, as shown in [9]: intuitively, the higher the residual of a CN, the further that portion of the graph is from convergence. Hence, scheduling CNs with higher residuals is expected to lead to faster and more reliable decoding compared to the flooding scheme. However, the computation of provisional messages is necessary for updating CN residuals in real-time, rendering NS more computationally intensive than the flooding scheme for the same total number of messages propagated.

To mitigate the computational complexity inherent in the approach to NS in [9], we propose an RL-based NS (RL-NS) scheme for sequential iterative decoding of short block length LDPC codes suitable for operating in the waterfall region of the bit error rate (BER) versus signal-to-noise-ratio (SNR) performance curve. Instead of computing residuals prior to scheduling, RL-NS employs a CN scheduling policy that utilizes an action-value function to determine how beneficial an action is for optimizing the scheduling order, where the optimal scheduling order is the one that yields a codeword output by propagating the smallest number of CN to VN messages. An action is defined here as selecting a single CN to convey its outgoing messages to its adjacent VNs. The scheduling algorithm is modeled as a Markov decision process (MDP) [10], where the Tanner graph is viewed as an environment with m possible actions (CN scheduling operations), and an agent learns to schedule CNs that elicit the highest reward. Repeated scheduling enables the agent to accurately estimate the action-value function.

Manuscript received October 15, 2020; revised March 18, 2021 and April 9, 2021; accepted April 10, 2021. Date of publication April 16, 2021; date of current version June 21, 2021. This work was supported in part by U.S. NSF under Grant ECCS-1711056, and in part by the Combat Capabilities Development Command of the U.S. Army Research Laboratory through Cooperative Agreement under Grant W911NF-17-2-0183. This paper was presented in part at the 2020 IEEE Information Symposium on Information Theory, Los Angeles, CA, USA, [1], and the 34th Conference on Neural Information Processing Systems (NeurIPS), Vancouver, BC, Canada, 2020 [2]. *(Corresponding author: Salman Habib.)*

Salman Habib and Jörg Kliewer are with the Helen and John C. Hartmann Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102 USA (e-mail: sh383@njit.edu).

Allison Beemer is with the Department of Mathematics, University of Wisconsin-Eau Claire, Eau Claire, WI 54701 USA.

Digital Object Identifier 10.1109/JSAIT.2021.3073834

Note that machine learning-assisted BP decoding of linear codes has been addressed in, e.g., [11]–[13], which use deep learning based on neural networks to learn the noise on the communication channel. A deep learning framework based on hyper-networks is used for decoding short block length LDPC codes in [14], where the hidden layers of the network unfold to represent Tanner graphs executing successive message passing iterations. In [15], [16], reinforcement learning (RL) is proposed for constructing polar codes. [15] focuses on BP-based polar code decoding and frames the factor graph selection problem as a multi-armed bandit (MAB) problem. On the other hand, [16] frames the construction of polar codes as a maze traversing game where a chosen path in the maze corresponds to a unique polar code construction. Further, RL was recently applied to hard decision-based iterative decoding in [17]. However, to the best of our knowledge, RL has not previously been successfully applied to soft iterative decoding of LDPC codes in the open literature. Our work also differs from the vast majority of works (including [11]–[14], [17]) in that our decoder is not based on deep learning. In this paper, we propose both model-free and model-based RL strategies for sequential decoding of short LDPC codes targeted for operating in the waterfall region. In the model-free category, we consider both computing Gittins indices (GIs) of CNs and Q-learning for estimating the corresponding action-value functions. For our model-based approach, we employ Thompson sampling (TS).

In the GI scheme, our RL problem is viewed as Markovian MAB problem where each CN (arm) is considered to be an independent bandit process (with independent rewards), leading to a learning complexity that grows linearly with the number of CNs. In the case of TS, CN to VN messages are assumed to be independent and normally distributed, allowing us to sample rewards from a known distribution in each learning step. For our model-free strategy, we first employ a TS-based NS algorithm, where the rewards are sampled from a chi-squared distribution, without learning any action values. Later on, we propose a variant of this approach by incorporating Q-learning. Specifically, to generate action values, rewards are sampled instead of being computed as in standard Q-learning.

Q-learning [18], [19] is a Monte Carlo approach for estimating an action-value function without explicit assumptions on the distribution of the bandit processes [20]. A major drawback of applying Q-learning to the sequential CN scheduling problem at hand is that the learning complexity grows exponentially with the number of CNs. Indeed, a straightforward choice of the underlying state space of the Q-learning problem is the space of vectors of quantized CN values: however, the number of CNs ranges in the hundreds for practical LDPC codes, so that even for a binary quantization of each CN value, the cardinality of the state space is not computationally manageable in the learning process. A multitude of methods for reducing the learning complexity in RL have been proposed in the literature: for example, complexity may be reduced by partitioning the state space (see, e.g., [21], [22]), imposing a state hierarchy (see, e.g., [23]), or reducing dimensionality (see, e.g., [24], [25]).

In this work, we follow an approach similar to these methods in order to reduce complexity, albeit tailored to the problem at hand. Specifically, for Q-learning we propose grouping the CNs into clusters, each with a separate state and action space. While this approach has the potential to make learning tractable, it also assumes independence of the clusters, an assumption that will not hold due to the inevitable presence of cycles in the Tanner graph. In order to mitigate the detrimental effect of clustering on the learning performance, we leverage the structure of the Tanner graph of the code by optimizing the clusters so that dependencies between clusters are minimized. To this end, we define novel graphical substructures in the Tanner graph termed cluster-connecting sets (CCSs), which capture the connectivity between CN clusters, and analyze their graph-theoretic properties. We give general bounds on the size of a CCS, as well as tighter bounds and some exact results for regular and array-based LDPC codes whose performance is simulated. Guided by the properties of CCSs, we propose a novel cluster selection scheme to optimize Q-learning performance.

This paper extends our previous work in [1], [2] in several aspects and presents these novel results in a comprehensive fashion, highlighted as follows:

- We propose a model-based version of RL based on Thompson sampling in addition to model-free methods. This approach has the salient feature that it allows us to incorporate the graph message densities into the model and thus is capable of providing a significant performance improvement.
- We provide an additional graph-theoretic result related to the size of CCSs in array-based LDPC codes, along with the complete proof of Theorem 2, superseding the results in [2].
- Based on the observation that RL scheduling schemes tend to result in isolated bit errors in the decoded codeword estimate, as compared to bursty errors for BP flooding, we also propose a novel code construction with an added high rate outer code. Our results show that with only a minor rate penalty and negligible additional complexity, a low complexity Hamming outer code can significantly improve the performance beyond the gain already observed for RL sequential decoding, but that this outer code is not able to improve on BP decoding via flooding.

The rest of the paper is organized as follows. Necessary background is given in Section II. In Section III, we discuss the RL-NS scheme in detail. In Section IV we provide the details of CN scheduling policies learned via the model-free and model-based RL methods outlined above. In Section V we introduce CCSs and show how they are related to the dependencies between clusters. A novel cluster optimization scheme based on the detection of cycles is discussed in Section VI. In Section VII we discuss the experimental setup, and analyze numerical results by comparing the proposed RL-based sequential decoding schemes to conventional decoders found in the literature. Section VIII concludes the paper.

II. Preliminaries

A. LDPC 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 (non-unique) binary *parity-check matrix* $\mathbf{H} \in \mathbb{F}_2^{m\times n}$, where $m \geq n-k$. 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 H, and edges in E correspond to the 1's in $\mathbf{H}$ [7]. That is, $\mathbf{H}$ is the (simplified) adjacency matrix of $G_{\mathbf{H}}$. For a subset X of nodes, denote by $\mathcal{N}(X)$ the set of all neighbors of X, and define $\mathcal{N}_A(X) = \mathcal{N}(X) \cap A$, where A is some subset of nodes.

LDPC codes are a class of highly competitive linear codes defined via sparse parity-check matrices or, equivalently, sparse Tanner graphs [3]. Due to this sparsity, LDPC codes 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. Note that critical substructures in the Tanner graph, such as absorbing sets (ABSs) [26], are capable of producing decoder failures. We have stated in [1] that sequential scheduling significantly reduces the impact of ABSs on the decoder performance.

In this work, we present experimental results for two particular classes of LDPC codes: (j, k)-regular and array-based (AB-) LDPC codes. A (j, k)-regular LDPC code is defined by a parity-check matrix with constant column and row weights equal to j and k, respectively [3]. 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, $\mathbf{H}(\gamma, p)$ [27]. In particular,

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

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 $\sigma^0 = \mathbf{I}$. Each row (resp., column) of sub-matrices of $\mathbf{H}(\gamma, p)$ forms a *row* (resp., *column*) *group*. Observe that there are a total of p (resp., p^2) column groups (resp., columns) and γ (resp., γp) row groups (resp., rows) in $\mathbf{H}(\gamma, p)$. A *lifted* LDPC code is obtained by replacing the non-zero (resp., zero) entries of the parity-check matrix with randomly generated permutation (resp., all-zero) matrices [28].

B. Reinforcement Learning

In an RL problem, an agent (learner) interacts with an environment by taking actions, which alter the state of the environment, and receiving a reward in return for each action. The goal of the agent is to maximize the total reward in a series of actions. Here, the environment is given by the Tanner graph of the code whose state space is modeled as a finite MDP [10], and the optimized sequence of actions, i.e., the scheduling of individual CNs, is obtained by employing an action selection policy learned either by computing GIs, via Q-learning, or via TS.

In the remainder of the paper, define $[[x]] \triangleq \{0, \ldots, x-1\}$, where x is a positive integer. In an environment with m possible actions, let $S_t^{(0)}, \ldots, S_t^{(m-1)}$ represent the m possible states resulting from taking those actions, where each random variable (r.v.) $S_t^{(j)}$, $j \in [[m]]$, can take M possible real values. Let state space $\mathcal{S}$ contain all M^m possible realizations of the sequence $S_t^{(0)}, \ldots, S_t^{(m-1)}$, and let the r.v. $S_t \in [[M^m]]$, with realization s represent the index of the realization $s_t^{(0)}, \ldots, s_t^{(m-1)}$. Since each index corresponds to a unique realization, we also refer to S_t as the state of the environment at time t. If an action (scheduling of a CN) is modeled as an independent random process, we define a r.v. $\hat{S}_t \in [[M]]$, with realization $\hat{s}$, as the realization $s_t^{(j)}$ of any CN j. Let $A_t \in [[m]]$, with realization a, represent the index of an action taken by the agent at time t, and $\mathcal{A} = [[m]]$ be an action space, where $a \in \mathcal{A}$. Let S_{t+1} represent a new state of the MDP after taking action A_t, and let s' denote its realization. Also, let a r.v. $R_t(S_t, A_t, S_{t+1})$, with realization r be the reward yielded at time t after taking action A_t in state S_t.

C. Solving the RL Problem by Computing Gittins Indices

In case of GIs, our RL problem can be viewed as a MAB problem with Markovian bandits, where the playing of an arm represents an action and is equivalent to scheduling a CN in our setup. If an m-armed bandit problem, formulated as an MDP, is solved via Markov decision theory, the state space consists of M^m possible state realizations of all the m arms. Consequently, the complexity of solving the MAB via Markov decision theory grows exponentially with the number of arms. On the other hand, if the arms are independent bandit processes, it is clear that the optimal solution to the m-armed bandit problem can be obtained by solving m 1-dimensional optimization problems, leading to an exponential complexity reduction. Hence, for a given arm with index a, one need only compute its action-value function, in this case known as the Gittins index (GI) $G(\hat{s}, a)$, given by [29]

$$G(\hat{s},a)=\max_{p_\tau\in\mathcal{P}}\frac{\mathbb{E}_{\tau,\hat{s}'}\left[\sum_{t=0}^{\tau-1}\beta^t R_t\left(\hat{S}_t,A_t,\hat{s}'\right)|\hat{S}_0=\hat{s},A_t=a\right]}{\mathbb{E}_\tau\left[\sum_{t=0}^{\tau-1}\beta^t|\hat{S}_0=\hat{s},A_t=a\right]}, \tag{2}$$

where τ is a r.v. with realizations in $\{1, 2, \ldots, \}$ that gives the number of times the agent plays arm a, p_τ is the distribution of τ, $\mathcal{P}$ represents the collection of all allowed distributions and is determined by allowed stopping time policies, and $0 < \beta < 1$ is the reward discount rate. The action-value function $G(\hat{s}, a)$ represents the long-term expected reward for taking action a in state $\hat{s}$, indicating how beneficial it would be for the agent to take that action [10], [29]. For the Gittins scheme, the optimal arm scheduling policy for an agent is given by

$$\pi_G = \arg\max_a G(\hat{s}, a). \tag{3}$$

D. Solving the RL Problem via Q-Learning

Optimal policies for MDPs can also be estimated via Monte Carlo techniques such as Q-learning [17], [18], [20]. The estimated action-value function $Q_t(S_t, A_t)$ in Q-learning represents the expected long-term reward achieved by the agent at time t after taking action A_t in state S_t. To improve the estimation in each time step, the action-value function is adjusted according to a recursion

$$Q_{t+1}(s,a) = (1-\alpha)Q_t(S_t = s, A_t = a) + \alpha\Big(R_t(s,a,S_{t+1} = f(s,a)) + \beta \max_{a' \in [[m]]} Q_t\big(f(s,a),a'\big)\Big), \quad (4)$$

where $f(s,a)$ represents the new state s' as a function of s and a, $0 < \alpha < 1$ is the learning rate, and $Q_{t+1}(s,a)$ is a future action-value resulting from action a in the current state s [18, pp. 95–96]. Note that the observed state, which is a collective state $S_t^{(0)}, \ldots, S_t^{(m-1)}$ of all m bandit processes, allows the agent to incorporate any dependencies of the arms, unlike the Gittins scheme.

In Q-learning, the optimal policy for the agent, $\pi_Q^{(\tau)}$, in state s is given by

$$\pi_Q^{(\tau)} = \arg\max_a Q_\tau(s,a), \quad (5)$$

where τ is the total number of time steps after observing an initial state S_0. Although the optimal policy is initially unknown to the agent, with the aid of Q-learning it is possible to recursively determine the policy and the action-value function together via ϵ-greedy exploration (see Section IV for details).

E. Solving the RL Problem via Thompson Sampling

TS is widely used for solving RL problems [30]. In this approach, the environment is represented using a statistical model which, in each TS step t, predicts the reward R_t and/or the new state S_t' given the current state S_t and action A_t. The rewards associated with each action are randomly sampled from a posterior distribution representing the agent's prior belief of the expected reward. This randomization allows the agent to actively explore the environment. Suppose that the modeled environment generates a new state based on a conditional probability measure $p(S_t' = s' | A_t = a)$, and let $\theta_a = \mathbb{E}_{s'}[R_t | S_t, a]$ denote the mean reward (model parameter) for playing a bandit's arm a, which is initially unknown to the agent. Also, let the agent's uncertainty about the value of θ_a be represented using the prior distribution $p_a = \mathbb{P}(\theta_a \in \mathcal{R} | S_t, a)$, where $\mathcal{R}$ denotes the set of all possible rewards.

In a typical TS scheme, an estimated mean reward $\hat{\theta}_a$ is randomly sampled from the prior p_a, and the action a with the potential of generating the highest expected long-term reward is selected according to $a = \arg\max_{a \in \mathcal{A}} \hat{\theta}_a$. The prior distribution p_a is then updated based on the knowledge of a and s', and subsequently used for sampling the reward in the next TS step. Repeated sampling allows the agent to take actions which are expected to generate high rewards in the long run.

III. RL-Based Node-Wise Scheduling

The proposed RL-NS is a serial decoding algorithm in which a single message-passing iteration is given by messages sent from a scheduled CN to all its neighboring VNs, and subsequent messages sent from these VNs to their other CN neighbors. Sequential CN scheduling is carried out until a stopping condition is reached, or an iteration threshold is exceeded. The RL-NS decoder applies a scheduling policy based on an action-value function to decide the CN to be scheduled next, avoiding the real-time calculation of residuals.

We define the optimal scheduling order to be the one that yields a codeword output by propagating the smallest number of CN to VN messages. The decoding algorithm informs the imaginary agent of the current state of the decoder, and the reward obtained after performing an action (scheduling a CN). Based on these observations, the agent takes future actions, to enhance the total reward earned, which alters the state of the environment and also the future reward. In this work, the reward R_a obtained by the agent after scheduling CN a is defined as $R_a = \max_{v \in \mathcal{N}(a)} r_{a \to v}$, where the residual $r_{a \to v}$ is computed according to

$$r_{a \to v} \triangleq |m'_{a \to v} - m_{a \to v}|. \quad (6)$$

Here, $m_{a \to v}$ is the message sent by CN a to its neighboring VN v in the previous iteration, and $m'_{a \to v}$ is the message that CN a would send to VN v in the current iteration, if scheduled.

The magnitude of the residual diminishes as the BP algorithm converges. Consequently, propagating CN to VN messages with relatively large residuals first is expected to lead to faster convergence of the BP algorithm [9]. Note that in our RL problem, residuals are computed for estimating the action-value functions only, which is done offline (see Section IV for details).

An iteration number ℓ (resp., iteration threshold $\ell_{\max}$) of the RL-NS scheme is analogous to a time step t (resp., stopping time τ) discussed in Section II. Let $\mathbf{x} = [x_0, \ldots, x_{n-1}]$ and $\mathbf{y} = [y_0, \ldots, y_{n-1}]$ represent the transmitted and the received word, respectively, where $x_i \in \{0,1\}$, $y_i = (-1)^{x_i} + z$, and $z \sim \mathcal{N}(0, \sigma^2)$. The posterior log-likelihood ratio (LLR) of x_i is expressed as $L_i = \log \frac{\Pr(x_i=1|y_i)}{\Pr(x_i=0|y_i)}$. The soft channel information input to the RL-NS algorithm is a vector of LLRs denoted as $\mathbf{L} = [L_0, \ldots, L_{n-1}]$. In the RL-NS scheme, the value (or state) of CN j at the end of iteration ℓ is defined by $\hat{s}_\ell^{(j)} = \sum_{i=0}^{n-1} H_{j,i} \hat{L}_\ell^{(i)}$, where $\hat{L}_\ell^{(i)} = \sum_{c \in \mathcal{N}(v_i)} m_{c \to v_i} + L_i$ is the posterior LLR computed by VN v_i at the end of iteration ℓ, and $m_{c \to v_i}$ is the message received by VN v_i from a neighboring CN c. The resulting vector $\hat{\mathbf{S}}_\ell = \hat{S}_\ell^{(0)}, \ldots, \hat{S}_\ell^{(m-1)}$, with realization $\hat{\mathbf{s}}_\ell = \hat{s}_\ell^{(0)}, \ldots, \hat{s}_\ell^{(m-1)}$, represents a "soft-syndrome" vector of the RL-NS scheme obtained at the end of iteration ℓ.

Since we model the NS scheme as a finite MDP, it is necessary to quantize each soft CN state. Let $g_M(\cdot)$ denote an M-level scalar quantization function that maps a real number to any of the closest M possible representation points set by the quantizer. Let $\mathbf{S}_\ell = S_\ell^{(0)}, \ldots, S_\ell^{(m-1)}$ be the quantized syndrome vector, where a realization $s_\ell^{(j)} = g_M(\hat{s}_\ell^{(j)})$. We call $\mathbf{S}_\ell$ a quantized soft syndrome for the case $M > 2$, and a binary syndrome if $M = 2$. The state space containing all possible

Algorithm 1: RL-NS for LDPC Codes

Input : $\mathbf{L}$, $\mathbf{H}$
Output: reconstructed signal
Initialization:
  $\ell \leftarrow 0$
  $m_{c\to v} \leftarrow 0$ // for all CN to VN messages
  $m_{v\to c} \leftarrow L_v$ // for all VN to CN messages
  $\hat{\mathbf{L}}_\ell \leftarrow \mathbf{L}$
  $\hat{\mathbf{S}}_\ell \leftarrow \mathbf{H}\hat{\mathbf{L}}_\ell$
foreach $a \in [[m]]$ **do**
  $s_\ell^{(a)} \leftarrow g_M(\hat{s}_\ell^{(a)})$ // M-level quantization
end
// decoding starts
if *stopping condition not satisfied or* $\ell < \ell_{\max}$ **then**
  $s \leftarrow$ index of $\mathbf{S}_\ell$
  select CN a according to an optimum scheduling policy
  foreach *VN* $v \in \mathcal{N}(a)$ **do**
    compute and propagate $m_{a\to v}$
    foreach *CN* $c \in \mathcal{N}(v) \setminus a$ **do**
      compute and propagate $m_{v\to c}$
    end
    $\hat{L}_\ell^{(v)} \leftarrow \sum_{c\in\mathcal{N}(v)} m_{c\to v} + L_v$ // update LLR of v
  end
  foreach *CN* j *that is a neighbor of* $v \in \mathcal{N}(a)$ **do**
    $\hat{s}_\ell^{(j)} \leftarrow \sum_{v'\in\mathcal{N}(j)} \hat{L}_\ell^{(v')}$
    $s_\ell^{(j)} \leftarrow g_M(\hat{s}_\ell^{(j)})$ // update syndrome $\mathbf{S}_\ell$
  end
  $\ell \leftarrow \ell + 1$ // update iteration
end

quantized syndromes is represented as $\mathcal{S}^{(M)}$. Let $S_\ell \in [[M^m]]$, with realization s, denote the index of a realization of $\mathbf{S}_\ell$, let $\hat{S}_\ell \in [[M]]$, with realization $\hat{s}$, represent a quantized CN value, and let an action $A_\ell \in \mathcal{A}$, with realization a, denote the index of a scheduled CN in iteration ℓ. The proposed RL framework for sequential decoding is shown in Fig. 1. The idea is that scheduling a CN updates the state of the environment, namely the Tanner graph of the code, which in turn provides the resulting residual as a reward to the agent. This reward is used to schedule a new CN in the next iteration.

The RL-NS based sequential LDPC decoder is shown in Algorithm 1. The input to this algorithm is a soft channel information vector $\mathbf{L}$ and a parity-check matrix $\mathbf{H}$. Note that the CN scheduling policy in Step 12 of Algorithm 1 is equivalent to scheduling the CN with the highest expected residual. Observe that the time complexity for selecting this CN grows linearly with the total number of CNs, as opposed to being zero for flooding BP.

The RL-NS algorithm is dynamic, and depends both on the graph structure and on received channel values: thus, the scheduling order may differ with subsequent transmissions, and will outperform a scheduling order which is fixed in advance. As NS follows a greedy schedule, there exists a non-zero probability that initially correct, but unreliable, bits are wrongly corrected into an error that is propagated in subsequent iterations. In contrast, our RL-NS scheme allows some room for exploration (not just exploitation, as in NS) by scheduling the CN with the highest expected *long-term* residual, mitigating such a potential error propagation.

Finally, we remark on a decoding error type encountered by the NS scheme which we expect to correct using RL-NS: namely, undetected errors. Undetected errors occur when the Hamming distance between the received and decoded codewords is greater than the distance between the transmitted and received ones. In the RL-NS scheme, the agent attempts to schedule a CN based on its *expected* residual instead of the immediate one. As a result, the RL-NS scheme employs a more global decoding approach in contrast to the pure NS scheme [9] and is more likely to overcome undetected errors.

IV. Learning CN Scheduling Policies

In our RL problem, the agent's goal is to estimate, from experience, an optimum CN scheduling policy to be used in Step 12 of Algorithm 1. This task manifests an exploration vs. exploitation trade-off which is typical of any RL framework. To maximize the total reward in the long-run, the agent must schedule CNs that are known to produce high residuals (exploitation). But to discover such CNs, the agent must select CNs that were not scheduled before (exploration). To accommodate this trade-off, we utilize the well-known ϵ-greedy exploration scheme [10], [31]. The estimate of the action-value function improves as more CNs are scheduled by the agent. In the following, we discuss several RL techniques, both model-free (in Sections IV-A and IV-B) and model-based (in Section IV-C), for learning the action-value function used in RL-NS.

A. Estimating the GIs

In this approach, the m CNs are assumed to be m independent bandit processes, implying that scheduling a particular CN does not affect the state of the remaining ones. This assumption would hold in tree-like Tanner graphs where a message propagated by a CN to a VN is independent of the messages computed by all other CNs in the graph. However, in practice, Tanner graphs contain cycles that induce dependencies between the CN to VN messages. Nonetheless, the GI approach offers a low-complexity learning task where the size of the state space observed by the agent increases linearly with the number of CNs in the underlying graph. Note that in our problem $P(S_{\ell+1}|S_\ell, A_\ell) \in \{0, 1\}$ as the Tanner graph is fixed and the messages are deterministically computed. We also apply a specific stopping time policy by selecting an integer $\ell_{\max}^* \in \{1, 2, \ldots, \}$ with the condition $\Pr(\ell_{\max} = \ell_{\max}^*) = 1$. Based on these considerations, (2) is rewritten for our MAB problem as

$$G(\hat{s}, a) = \max_{\ell_{\max}^*} \frac{\sum_{\ell=0}^{\ell_{\max}-1} \beta^\ell R_\ell(\hat{s}, a, \hat{s}')}{\sum_{\ell=0}^{\ell_{\max}-1} \beta^\ell}. \tag{7}$$

Since an agent must obtain the GIs via RL, we obtain an average GI, denoted $\tilde{G}(\hat{s}, a)$, over multiple realizations of $\mathbf{L}$. After computing $\tilde{G}(\hat{s}, a)$ for all $\hat{s}$ and a, the optimum CN scheduling policy is $\hat{\pi}_G = \arg\max_a \tilde{G}(\hat{s}, a)$.

B. Q-Learning

Q-learning is an adaptive algorithm for computing optimal policies for MDPs [18], [20]. Q-learning does not rely

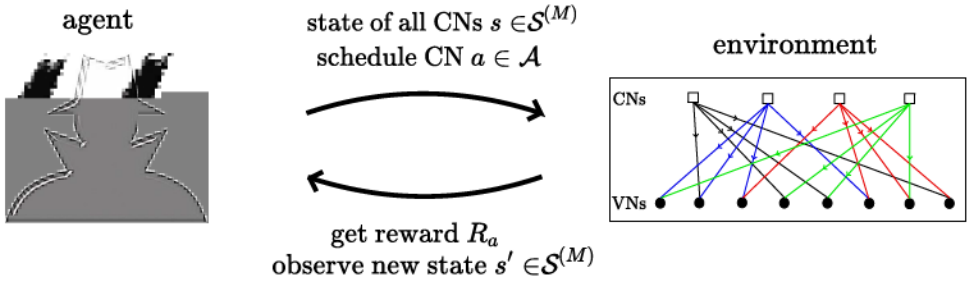

Fig. 1. Illustration of the RL setting used in this work. In every learning step, a fictitious agent schedules a CN with index a when the state of the environment is s, and receives reward R_a, which is also the maximum residual of CN a. The state of the environment changes from s to s' as the quantized soft-syndrome is updated after scheduling.

on explicit assumptions on the distribution of the bandit processes, unlike the Gittins scheme where the LLRs emanating from CNs are assumed to be independent. However, the state space observed by the agent now grows exponentially with m. As a result, the traditional Q-learning approach suffers from a much greater learning complexity. To overcome this problem, we propose a novel *clustering* strategy. A cluster is defined as a set of CNs with separate state and action spaces. Let $z \ll m$ represent the size (number of CNs) in a cluster. The state of a cluster with index $u \in [[\lceil \frac{m}{z} \rceil]]$, is a sub-syndrome $\mathbf{S}_\ell^{(u,z)} = S_\ell^{(uz)}, \ldots, S_\ell^{(uz+z-1)}$ of the syndrome $\mathbf{S}_\ell$, with a state space $\mathcal{S}_u^{(M)}$ containing all possible M^z sub-syndromes $\mathbf{S}_\ell^{(u,z)}$, where $|\mathcal{S}_u^{(M)}| \ll |\mathcal{S}^{(M)}|$. Hence, the total number of states that may be observed by the agent is upper-bounded by $\lceil \frac{m}{z} \rceil |\mathcal{S}_u^{(M)}|$. The action space of cluster u is defined as $\mathcal{A}_u = [[z]]$. We will discuss graph-theoretic properties of CN clusters and the resulting optimization approach in Sections V and VI, respectively. For now, let us assume that a set of CN clusters has been chosen as a result of this optimization. We denote the set of CNs belonging to the cluster of index u by $C_u = \{c_1, \ldots, c_z\}$.

Smaller clusters lead to reduced complexity, but the larger the size of the cluster, the greater the ability of the agent to take into account any dependencies between CN LLRs. Hence, there exists a trade-off between the ability of clustered Q-learning to take into account these dependencies, and the learning complexity: the cluster size should be large enough to accommodate the dependencies of the CN messages as much as possible, but not so large that learning is infeasible.

Apart from this trade-off, clustering does not restrict Q-learning, as the reward obtained by the agent is indifferent to the size and location of the clusters in the Tanner graph. Moreover, scheduling a CN in one cluster may affect the residuals of other CNs distributed across multiple clusters due to their connection with a common set of VNs. Also note that in traditional Q-learning, a scheduling operation alters the state corresponding to the entire syndrome $\mathbf{S}_\ell$, whereas in the clustering approach, only the states of the clusters that are connected to the neighbors of scheduled CN are affected. As a result, states of the unaffected clusters may be reused for estimating future action-values. Based on the considerations above and noting that decoder iteration ℓ is analogous to time t, (4) can be rewritten for clustered Q-learning as

$$Q_{\ell+1}(s_u, a_u) = (1-\alpha) Q_\ell(s_u, a_u) + \alpha \Big(R_\ell(s_u, a_u, f(s_u, a_u)) + \beta \max_{u', a_{u'}} Q_\ell(f(s_{u'}, a_{u'}), a_{u'}) \Big), \quad (8)$$

where $s_u \in [[M^z]]$ and $a_u \in \mathcal{A}_u$ are the state and action indices of cluster u, respectively, $f(s_u, a_u)$ represents the new state $s'_u \in [[M^z]]$ as a deterministic function of s_u and a_u, and $R_\ell(s_u, a_u, s'_u) = R_{a_u}$. In clustered Q-learning, the action in optimization step ℓ is selected via an ϵ-greedy approach according to

$$a_u = \begin{cases} \text{uniformly random over } u \text{ and } \mathcal{A}_u \text{ w.p. } \epsilon, \\ \pi_Q^{(\ell)} \text{ w.p. } 1-\epsilon, \end{cases} \quad (9)$$

where the CN scheduling policy is

$$\pi_Q^{(\ell)} = \underset{a_u \text{ s.t. } u \in [[\lceil \frac{m}{z} \rceil]]}{\arg\max} Q_\ell(s_u, a_u). \quad (10)$$

Once RL has been accomplished, $\pi_Q^{(\ell_{\max})}$ yields an optimized CN scheduling policy, where $\ell_{\max}$ is the maximum number of decoder iterations (learning steps) for a given input of channel information $\mathbf{L}$.

Algorithm 2 gives the method for clustered Q-learning. 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 clustered Q-learning is performed, a clustering C_u for all cluster indices u, and a parity-check matrix $\mathbf{H}$. This algorithm trains the agent to learn the optimum CN scheduling policy in (10) for a given $\mathbf{H}$, which is later incorporated in Step 12 of Algorithm 1. In each optimization step ℓ, the agent performs NS for a given $\mathbf{L}$. As a result, clustered Q-learning can be viewed as a recursive Monte Carlo estimation approach with $\mathbf{L}$ being the source of randomness.

C. Thompson Sampling

TS has been applied successfully for RL in finite MDPs (see e.g., [30]). In this work, we show how TS can also be used for sequential decoding of LDPC codes. To accomplish this, we consider two distinct RL schemes which incorporate TS.

In the first approach, we model the decoding environment using a density evolution approach based on Gaussian approximation of the CN to VN messages [32]. The decoding error probability, P_ℓ, in iteration ℓ, computed by this density

Algorithm 2: Clustered Q-Learning

Input : $\mathscr{L}$, $\mathbf{H}$, and a clustering C_u for all cluster indices u
Output: Estimated $Q_{\ell_{\max}}(s_u, a_u)$ for all u

Initialization: $Q_0(s_u, a_u) \leftarrow 0$ for all s_u, a_u and u
for *each* $\mathbf{L} \in \mathscr{L}$ **do**
  $\ell \leftarrow 0$
  $\hat{\mathbf{L}}_\ell \leftarrow \mathbf{L}$
  $\hat{\mathbf{S}}_\ell \leftarrow \mathbf{H}\hat{\mathbf{L}}_\ell$
  foreach $a \in [[m]]$ **do**
    $s_\ell^{(a)} \leftarrow g_M(\hat{s}_\ell^{(a)})$ `// M-level quantization`
  end
  while $\ell < \ell_{\max}$ **do**
    schedule CN a_u according to (10)
    select u as cluster index of CN a_u
    $\mathbf{S}_\ell^{(u,z)} \leftarrow s_\ell^{(j_1)}, s_\ell^{(j_2)}, \ldots, s_\ell^{(j_z)}$
    $s_u \leftarrow$ index of $\mathbf{S}_\ell^{(u,z)}$
    foreach *VN* $v \in \mathcal{N}(a_u)$ **do**
      compute and propagate $m_{a_u \to v}$
      foreach *CN* $c \in \mathcal{N}(v) \setminus a_u$ **do**
        compute and propagate $m_{v \to c}$
      end
      $\hat{L}_\ell^{(v)} \leftarrow \sum_{c \in \mathcal{N}(v)} m_{c \to v} + L_v$ `// update LLR of v`
    end
    foreach *CN* j *that is a neighbor of* $v \in \mathcal{N}(a_u)$ **do**
      $\hat{s}_\ell^{(j)} \leftarrow \sum_{v' \in \mathcal{N}(j)} \hat{L}_\ell^{(v')}$
      $s_\ell^{(j)} \leftarrow g_M(\hat{s}_\ell^{(j)})$ `// update syndrome` $\mathbf{S}_\ell$
    end
    $s'_u \leftarrow$ index of updated $\mathbf{S}_\ell^{(u,z)}$
    $R_\ell(s_u, a_u, s'_u) \leftarrow$ highest residual of CN a_u
    compute $Q_{\ell+1}(s_u, a_u)$ according to (8)
    $\ell \leftarrow \ell + 1$ `// update iteration`
  end
end

evolution scheme can be formulated exactly as

$$P_\ell = Q\left(\sqrt{\frac{\mu_L + \lambda \mu_{m_{c\to v}}^{(\ell)}}{2}}\right)$$

for regular LDPC codes, where $\mu_{m_{c\to v}}^{(\ell)}$ is the mean of the CN to VN message $m_{c\to v}^{(\ell)}$, λ is the VN degree of the LDPC code, and μ_L is the mean of the input LLRs. By incorporating Gaussian approximation, we assume that a CN to VN message ($m_{c\to v}^{(\ell)}$) in NS is independent of the other messages generated by CN c, and normally distributed according to $\mathcal{N}(\mu_{m_{c\to v}}^{(\ell)}, 2\mu_{m_{c\to v}}^{(\ell)})$, where $\mu_{m_{c\to v}}^{(\ell)}$ is the mean of $m_{c\to v}^{(\ell)}$. Based on this assumption, the difference $m'_{a\to v} - m_{a\to v}$, of the successive CN to VN messages passing along edge $c \to v$, possesses a normal difference distribution with mean $\mu_{m_{c\to v}} - \mu'_{m_{c\to v}}$ and variance $2\mu_{m_{c\to v}} + 2\mu'_{m_{c\to v}}$. This allows us to employ the MSE residual $r'_{c\to v} = (m'_{a\to v} - m_{a\to v})^2$ as a non-central chi-square distributed reward, sampled in each decoding step. We call the resulting RL approach *NS based on TS* (*NS-TS*), shown in Algorithm 3.

In contrast to the other decoding methods proposed in this paper, Algorithm 3 directly solves our RL problem without learning action-values, and hence no clustering is required. In

Algorithm 3: NS Based on TS

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

1 Initialization:
2 $\ell \leftarrow 0$
3 $m_{v\to c} \leftarrow L_v$ `// for all VNs`
4 $m_{c\to v} \leftarrow 0$ `// for all CNs`
`// decoding begins`
5 **while** $\ell < \ell_{\max}$ **do**
6 **if** $\ell = 0$ **then**
7 schedule CN a randomly
8 **end**
9 **else**
10 schedule CN a with the highest residual
11 **end**
12 **foreach** *VN* $v \in \mathcal{N}(a)$ **do**
13 compute and propagate $m_{a\to v}$
14 **foreach** *CN* $c \in \mathcal{N}(v) \setminus a$ **do**
15 compute and propagate $m_{v\to c}$
16 **foreach** *edge* $c \to v$ *such that* $v \in \mathcal{N}(a)$ **do**
17 $m'_{c\to v} \leftarrow m_{c\to v}$ `// previous mesg.`
18 compute $m_{c\to v}$ `// current mesg.`
19 update $\mu_{m_{c\to v}}$ and $\mu_{m'_{c\to v}}$
20 $\delta_{c\to v} \leftarrow \left(\frac{\mu_{m_{c\to v}} - \mu_{m'_{c\to v}}}{\sqrt{2\mu_{m_{c\to v}} + 2\mu_{m'_{c\to v}}}}\right)^2$ `// update non-centrality parameter`
21 $r'_{c\to v} \sim \chi(1, \delta_{c\to v})$ `// sample and store residual`
22 **end**
23 **end**
24 $\hat{L}_\ell^{(v)} \leftarrow \sum_{c \in \mathcal{N}(v)} m_{c\to v} + L_v$ `// update aposterior LLR`
25 **end**
  `// hard-decision step`
26 **foreach** *VN* v **do**
27 **if** $\hat{L}_\ell^{(v)} < 0$ **then**
28 $\hat{x}^{(v)} \leftarrow 1$
29 **end**
30 **else**
31 $\hat{x}^{(v)} \leftarrow 0$
32 **end**
33 **end**
34 **if** $\mathbf{H}\hat{\mathbf{x}} = \mathbf{0}$ **then**
35 break `// stopping condition reached`
36 **end**
37 $\ell \leftarrow \ell + 1$
38 **end**

Step 21 of Algorithm 3, $\chi(1, \delta_{c\to v})$ denotes the non-central chi-squared distribution of $r'_{c\to v}$ with degree of freedom 1 and non-centrality parameter $\delta_{c\to v} = (\frac{\mu_{m_{c\to v}} - \mu_{m'_{c\to v}}}{\sqrt{2\mu_{m_{c\to v}} + 2\mu_{m'_{c\to v}}}})^2$, which is updated in each TS step. The CN with the highest sampled residual $r'_{c\to v}$ is selected in Step 10. In comparison, the CN selection criteria in NS is based on a calculated $r_{c\to v}$, as defined in (6). The means computed in Step 19 of Algorithm 3 are taken over the previous ten messages (at most) passed along the edge $c \to v$. Since no Q-learning is involved, Algorithm 3 can be used to decode much longer block codes than considered in this paper, with reasonable computational complexity.

Algorithm 4: TS-Based Clustered Q-Learning

Input : $\mathscr{L}$, $\mathbf{H}$, and a clustering C_u for all cluster indices u
Output: Estimated action-value $Q_{\ell_{\max}}(s_u, a_u)$ for all s_u and a_u

1 Initialization: $Q_0(s_u, a_u) \leftarrow 0$ for all s_u and a_u
2 **for** *each* $\mathbf{L} \in \mathscr{L}$ **do**
3 $\quad \ell \leftarrow 0$
4 $\quad \mu_{m_{v\to c}} \leftarrow \mu_L$ // for all VNs
5 $\quad m_{c\to v} \leftarrow 0$ // for all CNs
6 $\quad \hat{\mathbf{L}}_\ell \leftarrow \mathbf{L}$
7 $\quad \hat{\mathbf{S}}_\ell \leftarrow \mathbf{H}\hat{\mathbf{L}}_\ell$
8 $\quad$ **foreach** $j \in [[m]]$ **do**
9 $\quad\quad s_\ell^{(j)} \leftarrow g_M(\hat{s}_\ell^{(j)})$ // M-level quantization to obtain $\mathbf{S}_0$
10 $\quad$ **end**
11 $\quad$ **while** $\ell < \ell_{\max}$ **do**
12 $\quad\quad$ schedule CN a_u according to (10)
13 $\quad\quad$ select u as cluster index of CN a_u
14 $\quad\quad \mathbf{S}_\ell^{(u,z)} \leftarrow s_\ell^{(j_1)}, s_\ell^{(j_2)}, \ldots, s_\ell^{(j_z)}$
15 $\quad\quad$ **foreach** *VN* $v \in \mathcal{N}(a_u)$ **do**
16 $\quad\quad\quad \mu_{m_{a_u\to v}} \leftarrow \phi^{-1}\left(1 - [1 - \phi(\mu_{m_{v\to a_u}})]^{\rho-1}\right)$ // update sampling parameter
17 $\quad\quad\quad$ sample and propagate $m_{a_u\to v}$
18 $\quad\quad\quad$ **foreach** *CN* $c \in \mathcal{N}(v) \setminus a_u$ **do**
19 $\quad\quad\quad\quad \mu_{m_{v\to c}} \leftarrow \mu_L + (\lambda - 1)\mu_{m_{c\to v}}$
20 $\quad\quad\quad\quad$ **foreach** *edge* $c \to v$ *such that* $v \in \mathcal{N}(a_u)$ **do**
21 $\quad\quad\quad\quad\quad \mu'_{m_{c\to v}} \leftarrow \mu_{m_{c\to v}}$
22 $\quad\quad\quad\quad\quad \mu_{m_{c\to v}} \leftarrow \phi^{-1}\left(1 - [1 - \phi(\mu_{m_{v\to c}})]^{\rho-1}\right)$
23 $\quad\quad\quad\quad\quad \delta_{c\to v} \leftarrow \left(\frac{\mu_{m_{c\to v}} - \mu'_{m_{c\to v}}}{\sqrt{2\mu_{m_{c\to v}} + 2\mu'_{m_{c\to v}}}}\right)^2$ // update non-centrality parameter
24 $\quad\quad\quad\quad\quad r'_{c\to v} \sim \chi(1, \delta_{c\to v})$ // sample and store residual
25 $\quad\quad\quad\quad$ **end**
26 $\quad\quad\quad$ **end**
27 $\quad\quad\quad \hat{L}_\ell^{(v)} \leftarrow \sum_{c\in\mathcal{N}(v)} m_{c\to v} + L_v$ // update LLR
28 $\quad\quad$ **end**
29 $\quad\quad R_\ell(s_u, a_u) \leftarrow$ highest residual of CN a_u
30 $\quad\quad$ compute $Q_{\ell_{\max}}(s_u, a_u)$ according to (8)
31 $\quad\quad$ **foreach** *CN* j *that is a neighbor of* $v \in \mathcal{N}(a_u)$ **do**
32 $\quad\quad\quad \hat{s}_\ell^{(j)} \leftarrow \sum_{v'\in\mathcal{N}(j)} \hat{L}_\ell^{(v')}$
33 $\quad\quad\quad s_\ell^{(j)} \leftarrow g_M(\hat{s}_\ell^{(j)})$ // update syndrome $\mathbf{S}_\ell$
34 $\quad\quad$ **end**
35 $\quad\quad \ell \leftarrow \ell + 1$
36 $\quad$ **end**
37 **end**

Second, we propose a novel TS-based clustered Q-learning scheme, shown in Algorithm 4. The inputs to the algorithm are $\mathscr{L}$, $\mathbf{H}$, and a clustering C_u for all cluster indices u. The algorithm outputs the estimated action-values. The mean of the input LLR, μ_L, and the CN and VN degrees in the corresponding Tanner graph of $\mathbf{H}$, ρ and λ respectively, are used to update the message means propagated in each iteration (see Steps 16, 19, 22). The rewards in this algorithm are sampled from a $\chi(1, \delta_{c\to v})$ distribution in each Q-learning iteration, as shown in Step 24. Unlike Algorithm 3, Algorithm 4 is not a direct decoding scheme. Instead, Algorithm 4 is a model-based RL approach which estimates the action-values via TS for offline learning of the CN scheduling policy in (10). The policy is then incorporated in the CN scheduling step of Algorithm 1, resulting in a unique RL-NS scheme. Consequently, Algorithm 4 is the TS counterpart of Algorithm 2.

In contrast to Algorithm 3, the means of the CN to VN messages in Algorithm 4 are computed based on the Gaussian approximation of messages (see Steps 16 and 22) by evaluating the ϕ function (whose details can be found in [32]), while the means of the CN to VN messages in Algorithm 3 are computed empirically. Indeed, the CN to VN message $m_{a_u\to v}$ in Step 17 of Algorithm 4 is sampled from $\mathcal{N}(\mu^{(\ell)}_{m_{a_u\to v}}, 2\mu^{(\ell)}_{m_{a_u\to v}})$, whereas the CN to VN message $m_{a\to v}$ in Step 13 of Algorithm 3 is computed. In other words, Algorithm 3 directly employs NS to update the non-centrality parameter $\delta_{c\to v}$, while Algorithm 4 obviates the need for NS by updating $\delta_{c\to v}$ based on a statistical model of the environment (decoder), providing a significant reduction in learning complexity with respect to a TS algorithm that would run NS. Nonetheless, Algorithm 4 is still more complex than Algorithm 2 due to the TS step involved in Step 24. Note that though Algorithm 3 invokes the NS scheme, it can still be viewed as a model-based RL approach as the agent interacts with the environment in real-time to update (learn) the non-centrality parameter $\delta_{c\to v}$ (see Step 20).

An advantage of the model-based scheme is that it is not necessary to learn action values to choose the optimum CN scheduling policy, as shown in Algorithm 3, and hence fairly long LDPC codes can be decoded. Moreover, TS allows the agent to actively explore the environment via reward sampling. However, the downside of TS is that once Q-learning is incorporated, the resulting algorithm (Algorithm 4) is even more complex than the method used for model-free learning (Algorithm 2) due to the residual sampling step.

V. Cluster-Connecting Sets

In clustered Q-learning, the total number of states observed by the agent is upper-bounded by the number of clusters times the number of states within each cluster: $\left\lceil \frac{m}{z} \right\rceil |\mathcal{S}_u^{(M)}| \ll |\mathcal{S}^{(M)}|$. Note that the learning complexity in clustered Q-learning is $\mathcal{O}(z^M)$, whereas in standard Q-learning the complexity scales as $\mathcal{O}(m^M)$, with cluster size $z \ll m$. Consequently, clustering enables a tractable RL task for sequential scheduling of short LDPC codes.

While clustered Q-learning serves as an approximation for standard Q-learning, the strategy suffers from a performance loss due to the existence of dependencies between separate clusters. As cluster size increases, the agent can more accurately take into account any dependencies between CN log-likelihood ratios (LLRs).

To better analyze this loss in performance, we introduce a critical sub-structure in the Tanner graph $G_{\mathbf{H}}$ of an LDPC code. For a cluster $C_u = \{c_1, \ldots, c_z\}$ let $\overline{C}_u = C \setminus C_u$ be the remaining CNs in the Tanner graph. Let $\mathcal{N}_V(C_u)$ be the set of all neighbors of C_u in V. For $W \subseteq \mathcal{N}_V(C_u)$, let $\mathcal{N}_C(W)$ be the set of all neighbors of W in C.

Definition 1: Fix a CN cluster C_u, and let $W \subseteq \mathcal{N}_V(C_u)$. We say that W is the *cluster-connecting set (CCS)* corresponding to C_u if and only if for all $v \in W$, both $\mathcal{N}_{C_u}(v)$ and $\mathcal{N}_{\overline{C}_u}(v)$

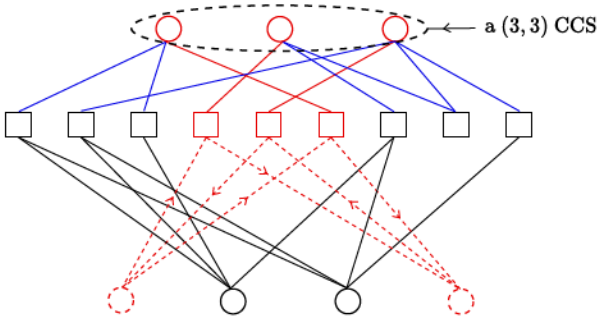

Fig. 2. Depiction of the (3, 3) CCS (shown using solid red circles) corresponding to cluster C_u (red squares). The dashed edges connect C_u to neighboring VNs (dashed red) not belonging to the CCS. Arrows highlight the cycles incident to the cluster.

are nonempty. If $|W| = A$, and $|\mathcal{N}_{C_u}(W)| = B$, we say that W is an (A, B) CCS.

CCSs induce dependencies between the messages generated by the CNs in C_u and those in $\overline{C}_u$. Roughly speaking, the number of edges in a Tanner graph that connect C_u to $\overline{C}_u$ grows with the size of C_u's CCS. That is, on average, the larger the size of the CCS corresponding to C_u, the greater the dependence between C_u and $\overline{C}_u$. This holds exactly for Tanner graphs in which all VNs have the same degree.

Fig. 2 depicts a Tanner graph $G_{\mathbf{H}}$ containing a cluster of CNs C_u whose corresponding CCS, W, is a (3, 3) CCS. Squares (resp. circles) represent CNs (resp. VNs). The three CNs shown using red squares represent C_u, whereas the black squares represent $\overline{C}_u$. The VNs in $\mathcal{N}_V(C_u)$ are represented using red circles. The three VNs of the CCS, W, are represented by solid red circles. Note that the VNs in dashed red circles, belonging to the set $\mathcal{N}_V(C_u) \setminus W$, are connected to only C_u, and hence do not belong to the CCS. Observe that there are no cycles in the subgraph induced by $W \cup \mathcal{N}_{C_u}(W)$. However, there are three distinct 4-cycles in the subgraph induced by $(\mathcal{N}_V(C_u) \setminus W) \cup C_u$.

In the remainder of the paper, we will consider Tanner graphs that are connected and contain cycles. We first show that any choice of cluster that is a proper subset of the CNs of a connected Tanner graph has a nonempty CCS.

Proposition 1: Consider a connected Tanner graph with m CNs, and choose a cluster C_u with $z < m$ CNs. Then, the CCS corresponding to C_u is nonempty.

Proof: Consider the set of VNs given by $\mathcal{N}_V(C_u)$. If every VN in this set is only adjacent to CNs in C_u, then the Tanner graph is not connected. By the contrapositive, we conclude that the CCS corresponding to C_u has at least one element. ■

In Section VI, we will be interested in optimizing clusters by minimizing the number of edges incident to the corresponding CCSs. However, because (j, k)-regular codes have regular VN degree, the size of a CCS for these classes is a constant multiple of the number of incident edges. Since our simulations in Section VII focus on (j, k)-regular and AB-LDPC codes (a subclass of regular codes), we present here specific results on the size of a CCS for each class.

Theorem 1: Let $j, k \geq 2$, and $z \geq 1$ be integers. Suppose G is a (j, k)-regular Tanner graph, and let C_u be a cluster of CNs of size $|C_u| = z$ in G. If $|\mathcal{N}_V(C_u)| = v$, then the number of variable nodes in W, the CCS corresponding to C_u, is bounded as follows.

$$v - \left\lfloor \frac{kz}{j} \right\rfloor \leq |W| \leq \min\{jv - kz, v\}.$$

Proof: It is straightforward to see that there are $jv - kz$ edges in G that are incident to $\mathcal{N}_V(C_u)$ but not to C_u. In other words, there are $jv - kz$ edges exiting the subgraph induced by $C_u \cup \mathcal{N}_V(C_u)$. Consider the CCS W, the subset of $\mathcal{N}_V(C_u)$ comprised of all VNs incident to an exiting edge. The minimum size of W corresponds to the case in which the exiting edges are concentrated at a few VNs in $\mathcal{N}_V(C_u)$. That is, $|W| \geq \lceil (jv - kz)/j \rceil = \lceil v - (kz/j) \rceil = v - \lfloor kz/j \rfloor$. On the other hand, the maximum size of W corresponds to the case where the exiting edges are spread across as many VNs as possible: $|W| \leq \min\{jv - kz, v\}$. Notice that $W = jv - kz$ when there is one edge per VN in W exiting the subgraph, but that this bound will not be tight if any VN in W has more than one incident exiting edge. ■

Remark 1: For fixed j, k, and z, the upper and lower bounds given in Theorem 1 are increasing functions of $|\mathcal{N}_V(C_u)| = v$. In other words, the more neighbors a cluster has, the larger the maximum possible size of W (and resulting $j|W|$ CCS edges), and the larger the minimum possible size of W (and $j|W|$). It is important to note that we are not claiming that a higher number of neighbors necessitates a larger CCS. Rather, the shifting window of possible CCS sizes suggests a trend, even if a strict increase does not hold. Indeed, we cannot make such a claim, since subsequent intervals may be overlapping as v increases. Take, for example, $j = 3$, $k = 6$, and $z = 5$. Then, if $v = 11$, $|W| \in \{1, 2, 3\}$, while if $v = 12$, $|W| \in \{2, 3, 4, 5, 6\}$. Thus, a cluster with 12 neighbors could have fewer CCS elements than a cluster with 11 neighbors. We also observe that v is bounded above by kz and below by kz/j, so that $|W|$ is always bounded below by 1 (in a connected graph with $z < m$) and above by kz.

Because they are $(3, p)$-regular, Theorem 1 gives bounds on the size of CCSs in $(3, p)$ AB-LDPC codes as well. In particular, for a cluster of size z, $v - \lfloor \frac{pz}{3} \rfloor \leq |W| \leq \min\{3v - pz, v\}$. However, given the added structure of an AB-LDPC code Tanner graph, we may conclude more. We restrict $z \leq p$; this is a reasonable assumption, given $m = 3p$ in $(3, p)$ AB-LDPC codes.

Theorem 2: Consider an AB-LDPC code defined by a parity-check matrix $\mathbf{H}(3, p)$, let C_u be a cluster of size $1 \leq z \leq p$, and let W be the CCS corresponding to C_u.

(i) If all CNs in C_u belong to the same row group of $\mathbf{H}(3, p)$, $|W| = zp$.

(ii) If all CNs in C_u belong to two row groups of $\mathbf{H}(3, p)$, $|W| = z(p - a) + a^2$, where a is the number of CNs in one of the row groups.

(iii) In general, $|W| \geq ((1 + 2p)z - z^2)/4$.

(iv) If $3 \leq z < p$ and every CN in C_u belongs to at least one 6-cycle such that the other two CNs in the 6-cycle also belong to C_u, $|W| \leq zp - z$.

Proof:

(i) If all the rows corresponding to the CNs of cluster C_u are in the same row group, then no two CNs in C_u will have any VNs in common. Hence, each VN in $\mathcal{N}_V(C_u)$ must also be adjacent to $\overline{C}_u$, implying that $\mathcal{N}_V(C_u)$ is a CCS with $|\mathcal{N}_V(C_u)| = |C_u|p = zp$.

(ii) If all the rows corresponding to the CNs of cluster C_u are in the same two row groups, then every VN in $\mathcal{N}_V(C_u)$ has at most two neighbors in C_u, and hence must be in W. Since any two CNs falling in different row groups share exactly one VN, the number of VN neighbors of C_u is $zp - a(z-a)$, where a is the number of CNs in one row group. Indeed, exactly $a(z-a)$ VNs are counted twice in zp.

(iii) There are p VN neighbors of each CN $c \in C_u$, no pair of which can share another CN neighbor due to the absence of 4-cycles in $(3, p)$ AB-LDPC Tanner graphs. Thus, the number of VNs in $\mathcal{N}_V(c)$ that have no neighbors outside of C_u is at most $(z-1)/2$: each such VN has two other neighbors in the remaining $z-1$ CNs of C_u, and all neighbors of these VNs must be distinct. Thus, there are at least $p - (z-1)/2$ VNs adjacent to c that have at least one neighbor outside of C_u. This is true for all z choices of $c \in C_u$. Since we may be counting each of these VNs up to two times with different CNs, there are at least $(p - [(z-1)/2])z/2 = ((1+2p)z - z^2)/4$ VNs in W.

(iv) In this scenario, at least $2z$ of the zp edges incident to C_u are incident to VNs of degree 2 in the subgraph induced by $C_u \cup \mathcal{N}_V(C_u)$. The remaining $z(p-2)$ edges may be incident to VNs of degree 1 in the subgraph. Thus, the number of elements of $\mathcal{N}_V(C_u)$ is bounded above by $z(p-2) + \frac{2z}{2} = zp - z$. Since $|W| \leq |\mathcal{N}_V(C_u)|$, the result follows. ∎

Notice that Theorem 2(iii) holds for all configurations of CNs, but is only useful if the CNs in C_u span all three row groups of $\mathbf{H}(3, p)$ since we have exact results for the other two cases. This bound, along with that in part (iv), should be compared with those of Theorem 1. Indeed, we find that Theorem 2(iii) gives a tighter lower bound for smaller values of $\mathcal{N}_V(C_u)$, and Theorem 2(iv) is a tighter upper bound for (a particular type of cluster and) larger values of $\mathcal{N}_V(C_u)$. Theorem 2(i) and (ii), as an exact results, give clear improvements on Theorem 1 for row configurations spanning fewer than all three row groups.

The CNs of the clusters in Theorem 2(i) belong to the same row group in the corresponding parity-check matrix; if the rows corresponding to the CNs are consecutive within the row group, we call them *contiguous*. Clusters as in Theorem 2(ii) may also have contiguous CNs. In comparison, the CNs of the clusters in Theorem 2(iv) cannot all be contiguous, since a 6-cycle must span three distinct row groups of a $\mathbf{H}(3, p)$ AB-LDPC code [26], [33]. By comparing the result on CCS size given in Theorem 2(i) with the bound in 2(iv), we see that choosing CNs that span three row groups and form internal 6-cycles is guaranteed to lower the bound on the size of the corresponding CCS from the case where we choose CNs all within a single group. Provided that $z > 4$, this will also be an improvement on clusters spanning two groups, as stated in Theorem 2(ii). Observing that the number of edges incident to a CCS W in a $(3, p)$-regular graph is equal to $3|W|$, we conclude that in case of $(3, p)$ AB-LDPC codes, any cluster selection scheme should ensure that clustered CNs are not contiguous, and each cluster contains as many 6-cycles as possible. This is indeed the approach used in Section VI to minimize connectivity of clusters.

VI. Cluster Optimization

In this section, we leverage the results from the previous section and propose a cluster optimization strategy, with the goal of minimizing the number of edges connected to each CCS in a Tanner graph. Indeed, these edges are responsible for the propagation of messages between clusters, and consequently for the resulting dependencies between clusters.

Let $E(C_u, W)$ be the set of edges that connect C_u to its CCS W, $E(\overline{C}_u, W)$ the set of edges connecting $\overline{C}_u$ to W, and $\zeta(C_u) \triangleq |E(\overline{C}_u, W)| + |E(C_u, W)|$ the total number of edges by which C_u is connected to $\overline{C}_u$ via W. As discussed in the previous section, minimizing $\zeta(C_u)$ is equivalent to minimizing the size of a CCS for (j, k)-regular Tanner graphs, since $\zeta(C_u) = j|W|$.

We cluster the set of CNs of a Tanner graph via a greedy iterative process: the z CNs of the first cluster, C_1^*, are chosen optimally from the set of all CNs. Each subsequent cluster is then chosen optimally from the set of remaining check nodes, with the last cluster consisting of the leftover CNs. Let $1, \ldots, u_{\lceil \frac{m}{z} \rceil}$ denote the indices of the clusters. Formally, this multi-step optimization is given by

$$C_e^* = \underset{C_e \subseteq C \setminus C_{e-1}^* \cup \cdots \cup C_1^*,\ |C_e| = z}{\arg\min} \zeta(C_e), \tag{11}$$

where $e \in 1, \ldots, \left\lceil \frac{m}{z} \right\rceil - 1$ denotes the index of the optimization step, C_e represents a cluster with index e, and C_e^* denotes the optimal cluster obtained in step e. The final cluster is obtained automatically, and hence excluded from (11).

Note that the complexity of the optimization in (11) grows exponentially with m, as we need to search over $\binom{m-z(e-1)}{z}$ possible cluster choices in each step. To overcome this, we propose a more computationally feasible cluster optimization approach based on our observation that the lower bounds in Theorems 1 and 2 correspond to a maximization of cluster-internal cycles: in each step, we cluster CNs so that the subgraph induced by the cluster and its neighbors contains as many cycles as possible. Such a cluster will in general have fewer neighbors compared to one that does not induce cycles. In turn, the maximum possible size of the corresponding CCS (and the number of exiting edges) will likely be reduced (see Remark 1). The cluster optimization approach based on cycle maximization is given by

$$\tilde{C}_e^* = \underset{C_e \subseteq C \setminus \tilde{C}_{e-1}^* \cup \ldots \cup \tilde{C}_1^*,\ |C_e| = z}{\arg\max} \eta_\kappa(C_e), \tag{12}$$

where $\tilde{C}_e^*$ denotes a cycle-maximized cluster, and $\eta_\kappa(C_e)$ denotes the number of length κ cycles in the graph induced

TABLE I
OVERVIEW OF THE DECODING SCHEMES USED FOR OBTAINING THE SIMULATION RESULTS

Scheme	NS-TS	NS	QR	QO	QR-TS	QO-TS
Name	Node-wise scheduling via TS	Node-wise scheduling	Random clus. Q-learn.	Opt. clus. Q-learn.	QR based on TS	QO based on TS
Approach	Alg. 3	Alg. 3 of [9]	Algs. 1 and 2	Algs. 1 and 2	Algs. 1 and 4	Algs. 1 and 4

by $\mathcal{N}_V(C_e) \cup C_e$. The possible values of κ depend on the girth of the considered code. Smaller choices of κ yield lower optimization complexity, as larger cycles are more difficult to enumerate. In case of LDPC codes, optimized cycle detection algorithms based on message-passing have complexity of $\mathcal{O}(gE^2)$, where g is Tanner graph's girth and E is the total number of edges [34]. In a $(3, p)$ AB-LDPC code, whose graph has girth 6, we choose $\kappa = 6$. The method for generating cycle-maximized clusters is presented in [2, Algorithm 3]. Let T be the total number of κ-cycles in $G_{\mathbf{H}}$. The optimization complexity of this algorithm depends on T, which is expected to be much smaller than $\binom{m-z(e-1)}{z}$ for sparse $\mathbf{H}$. For example, in a $(3, p)$ AB-LDPC code, there are only $T = p^2(p-1)$ 6-cycles in the Tanner graph [26].

In Section VII we consider the case where $p \geq 5$ and $z = 7$. For these parameters and for $e = 1$, the number of cluster choices using the optimization in (11) is $\binom{\gamma p}{z} = \mathcal{O}(p^7)$ as opposed to $p^2(p-1)$ using (12).

VII. SIMULATION RESULTS

A. Experimental Setup

We perform learned sequential decoding by employing cluster-based RL with random and (cycle-)optimized clusters. We then utilize each of these schemes for sequential decoding of random $(3, 6)$-regular LDPC, $(3, 7)$ AB-LDPC lifted by a factor of 4, $(96, 48)$ MacKay [35], $(63, 51)$ BCH, and a $(186, 78)$ concatenated code with rate 13/31, comprising an outer Hamming code and an inner systematic LDPC code. This is motivated by the observation that the individual bit error after RL based sequential decoding are relatively spread out across the decoded codewords and thus may be "cleaned-up" by an appropriate outer code. Specifically, we consider three $(31, 26)$ Hamming outer codes, and a $(186, 93)$ systematic (and thus irregular) LDPC inner code. In the encoding step, a message vector of length 78 is split into three length 26 vectors. The three resulting Hamming codewords, each of length 31, are then concatenated into a single vector, $\mathbf{u} = [u_0, \ldots, u_{m-1}]$, $m = 93$, and subsequently encoded using the inner systematic LDPC code encoder to generate a codeword $\mathbf{x}$ of length $n = 186$. The LDPC code is decoded via RL based sequential decoding to a length 186 binary vector $\hat{\mathbf{x}} = [\hat{x}_0, \ldots, \hat{x}_{n-1}]$, and the systematic portion of this vector is decoded via separate Hamming decoders producing the vector $\hat{\mathbf{u}} = [\hat{u}_0, \ldots, \hat{u}_{m-1}]$. Note that for this concatenated construction the observed BER is given by $\Pr[\hat{u}_j \neq u_j]$, $j \in [[m-1]]$, whereas for the other codes it is given by $\Pr[\hat{x}_i \neq x_i]$, $i \in [[n-1]]$.

We use Algorithms 2 and 4 to implement the cluster-based learning approaches mentioned above. The RL-NS schemes resulting from Algorithm 2 (resp., 4) for random and optimized clustering are denoted by QR and QO (resp., QR-TS and QO-TS), respectively. Note that in Algorithms 2 and 4 the $\mathbf{S}_\ell^{(u,z)}$ vector is updated as $\mathbf{S}_\ell^{(u,z)} = [s_\ell^{(j_1)}, s_\ell^{(j_2)}, \ldots, s_\ell^{(j_z)}]$, where CN indices $j_1, j_2, \ldots, j_z \in \{0, \ldots, m-1\}$ are randomly ordered in the case of random clustering, whereas in optimized clustering, the order depends on the underlying cluster $\tilde{C}_u^*$ obtained via cycle maximization. We compare these two schemes to our other RL-based decoders, one of which is based on computing GIs, while the other is based on Algorithm 3 (NS-TS).

Each element of the state vector $\hat{\mathbf{S}}_\ell$ is quantized using a standard scalar quantization algorithm [36], where a realization $\hat{s}_\ell^{(j)}$ represents the "source signal" to be quantized. Provided that there exists a sufficiently large dataset of source realizations, the quantization algorithm recursively optimizes the boundary and the representation points of the quantizer by minimizing the distortion $\mathbb{E}[(\hat{s}_\ell^{(j)} - g_M(\hat{s}_\ell^{(j)}))^2]$ over the entire dataset. Depending on the code, we generate a dataset comprising 10^5 realizations of $\hat{s}_\ell^{(j)}$ by randomly scheduling CNs via NS to estimate its distribution.

Although clustering reduces the state-space significantly compared to standard Q-learning, clustered Q-learning is still more complex than estimating GIs. Hence, our choice of code block lengths for the clustered Q-learning schemes are influenced by the run-time complexity of our Q-learning algorithms. We found short codes, with a block length 200 or less, to be suitable for Q-learning on our system with a reasonable cluster size z. For the considered codes, we choose the learning parameters as follows: $\alpha = 0.1$, $\beta = 0.9$, $\epsilon = 0.6$, $\ell_{\max} = 25$, and $|\mathscr{L}| = 1.25 \times 10^6$. We choose $z = 6$ for the MacKay and BCH codes, and $z = 7$ for the remaining codes, respectively. In addition, $M = 8$ is chosen for $(63, 51)$ BCH, whereas $M = 4$ is considered for the other codes. Finally, we compare the performances of our proposed RL methods with standard decoding schemes such as flooding and NS for $\ell_{\max} = 25$.

Note that M directly impacts the accuracy of our Q-learning schemes. A large number of quantization levels would naturally improve the Q-learning accuracy, but the learning complexity grows exponentially with increasing M. The magnitudes of the differences between the CN states at a given iteration diminishes as the decoder approaches convergence. Consequently, a large M ($\gg 4$) is necessary to accurately quantize the CN values at high SNR. We found Q-table[1] sizes corresponding to a small M to be the most suitable to learn on our system within a reasonable time period.

B. Numerical Results

The BER vs. channel SNR performances in dB of the considered codes using various decoding techniques are shown in Figs. 3 and 4. For clarification of the acronyms used in the

[1] A Q-table consisting of $\lceil m/z \rceil$ sub-tables, each of dimension $M^z \times z$, is used to store the action-values learned using Algorithms 2 and 4.

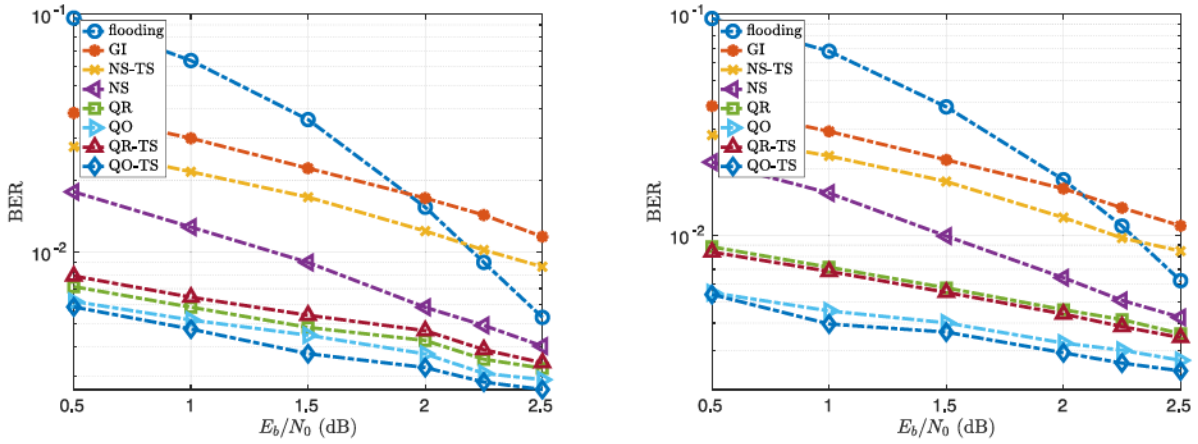

Fig. 3. BER results using different BP decoding schemes for a (3, 6)-regular LDPC (left figure) and (3, 7) AB-LDPC code (right figure, code is lifted), with block length $n = 196$.

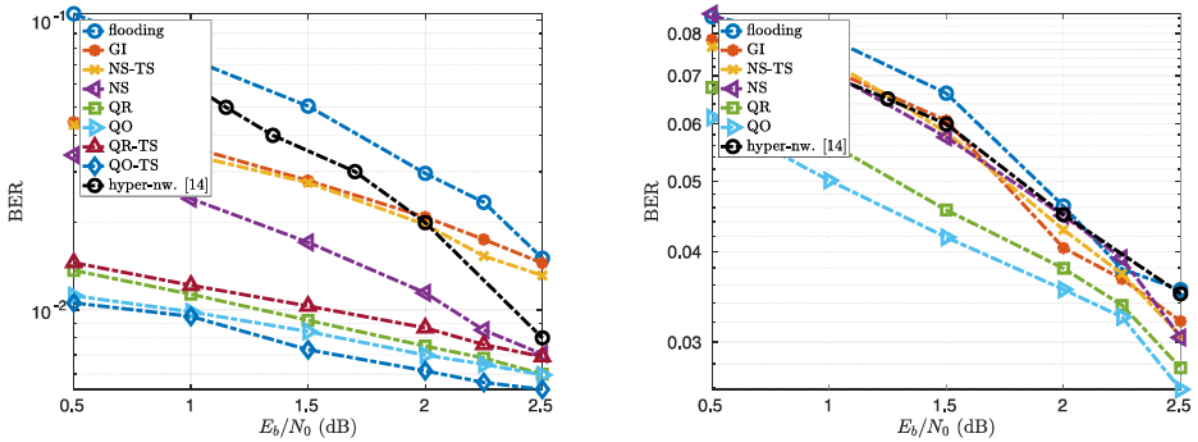

Fig. 4. BER results using different BP decoding schemes for a (96, 48) MacKay code (left figure) and (63, 51) BCH code (right figure), with block lengths 96 and 63, respectively.

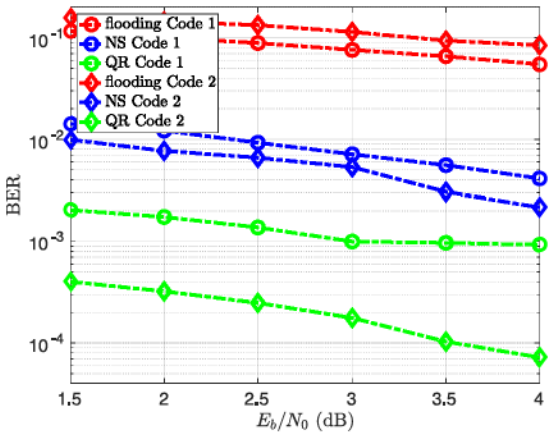

Fig. 5. Performances of a rate 1/2 systematic (186, 93) LDPC (Code 1) and a rate 13/31 concatenated (186, 78) Hamming-LDPC (Code 2) code using the flooding, NS, and QR decoding schemes.

plots, see Table I. The experimental results reveal that RL-based decoding schemes in general are superior to the non-RL decoding schemes in terms of BER performance, thus showing the effectiveness of our learning approach. In the case of the MacKay and BCH codes, we outperform the state-of-the art hyper-network decoder proposed in [14] for the chosen SNR regime and $\ell_{\max} = 25$. In Tables II and III, we compare the average number of CN to VN messages propagated in the considered decoders to attain the results in Figs. 3 and 4, respectively. In Table II, the numbers without (resp., with) parentheses correspond to the (3, 6)-regular (resp., (3, 7) AB-) LDPC code. On the other hand, in Table III, the numbers without (resp., with) parentheses correspond to the (96, 48) MacKay (resp., (63, 51) BCH) code. We note that, on average, the Q-learning based decoders generate a much lower

TABLE II
AVERAGE NUMBER OF CN TO VN MESSAGES PROPAGATED IN VARIOUS DECODING SCHEMES FOR A (3, 6)-REGULAR ((3, 7) AB-) LDPC CODE TO ATTAIN THE RESULTS SHOWN IN FIG. 3

SNR (dB)	1.5	2	2.5
flooding	13327 (16098)	6316 (8117)	2500 (3064)
GI	234 (281)	210 (247)	174 (205)
NS	229 (279)	203 (245)	173 (203)
QR	235 (286)	214 (244)	182 (214)
QO	209 (283)	181 (243)	163 (208)
QR-TS	244 (282)	209 (243)	179 (210)
QO-TS	237 (290)	216 (243)	178 (208)

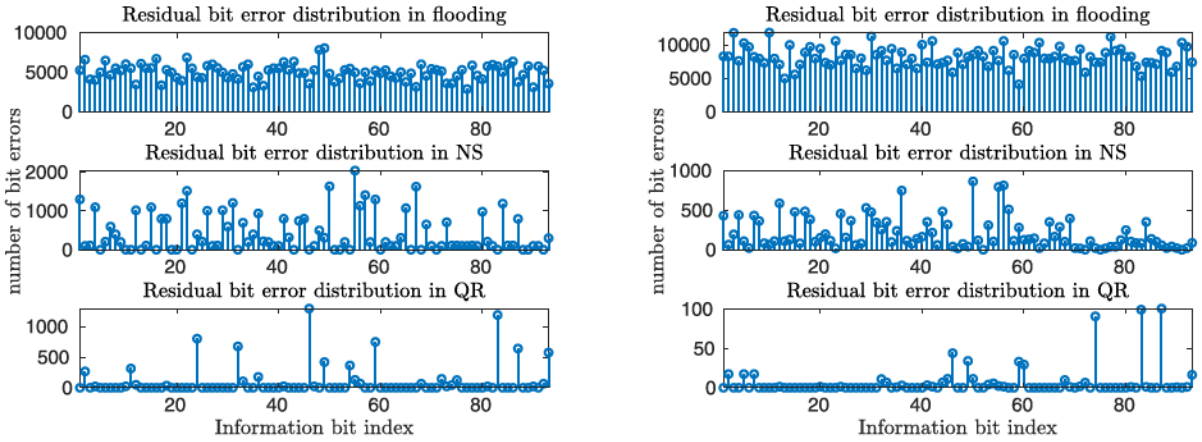

Fig. 6. Residual bit error distribution for $\hat{\mathbf{u}}$ after decoding 10^5 transmissions at $E_b/N_0 = 4$ dB using different decoding schemes for a (186, 93) systematic LDPC code (left figure), and a (186, 78) concatenated Hamming-LDPC code (right figure), respectively. These results correspond to the BER values shown in Fig. 5.

TABLE III
AVERAGE NUMBER OF CN TO VN MESSAGES PROPAGATED IN VARIOUS DECODING SCHEMES FOR A (96, 48) MACKAY ((63, 51) BCH) CODE TO ATTAIN THE RESULTS SHOWN IN FIG. 4

SNR (dB)	1.5	2	2.5
flooding	8531 (40442)	5833 (33787)	4303 (28127)
GI	254 (17003)	209 (13914)	178 (13758)
NS	220 (16593)	206 (14345)	163 (11392)
QR	244 (15816)	204 (15211)	182 (12899)
QO	239 (17422)	200 (15957)	180 (13542)
QR-TS	243	231	211
QO-TS	275	252	238

number of CN to VN messages when compared to the flooding scheme. Moreover, in contrast to NS, these schemes avoid the computation of residuals in real-time, providing a significant reduction in message-passing complexity for short LDPC codes. Note that the BCH code has a much denser Tanner than those of the other codes. Therefore, the TS results for the BCH code are excluded due to a significantly increased training complexity resulting from the code's graph density. This is also the reason that the number of CN to VN messages in Table III is much higher for the BCH code as compared to the MacKay code.

The results for the (186, 78) Hamming-LDPC concatenated code (labeled Code 2) are shown in Fig. 5. As a benchmark, we compare its performance to the (186, 93) systematic LDPC code (labeled Code 1) used as the inner code for the (186, 78) concatenated code. The systematic LDPC code has been obtained by converting the parity check matrix of a (3, 6) regular non-systematic LDPC code into reduced row echelon form. For decoding the systematic inner LDPC code, we only implement the QR scheme. The reason is that our cluster optimization and TS approaches are not suitable for systematic LDPC codes, for which the non-systematic portion of the code's Tanner graph is dense and irregular. The BER of the (186, 93) (inner) LDPC code is computed by detecting the number of erroneous bits in $\hat{\mathbf{u}}$. Fig. 5 shows the poor performance of BP flooding decoding the (186, 93) systematic LDPC code in isolation, which is due to the existence of many short cycles in the parity part of its parity check matrix. The (186, 78) concatenated construction significantly outperforms the systematic LDPC code by using RL-NS sequential scheduling with QR clustering. The reason for this surprising behavior is displayed in Fig. 6, which shows the bit error distribution in $\hat{\mathbf{u}}$ for the (186, 93) systematic LDPC (left) and the (186, 78) concatenated code (right). As anticipated, we can see that the outer Hamming code acts as a clean-up code for the inner LDPC code. By comparing the error distribution of the concatenated code to the non-concatenated one, we notice that an outer code significantly reduces the number of bit errors in the systematic portion of the signal reconstructed by the inner LDPC code. Notably, due to the nature of the error distribution seen on the left side of the figure, the concatenated coding scheme is more effective when used in conjunction with RL-based decoding. In particular, the inner LDPC code, decoded via QR, generates a large number of singe-bit errors, and the Hamming outer codes are able to correct these individual errors. On the other hand, the flooding scheme generates burst errors throughout each length-31 decoded block, rendering the Hamming code ineffective. However, these burst errors occur to a lesser extent in the case of NS, resulting in a BER reduction.

VIII. CONCLUSION

We presented several RL-based decoding schemes to optimize the scheduling of BP decoders for short LDPC codes suitable for performing in the waterfall regime. The main ingredient is a new state space clustering approach to significantly reduce the learning complexity. Experimental results show that optimizing the cluster structure by maximizing cluster-internal short cycles provides significant improvements in the decoding performance when compared with both previous scheduling schemes and clustered Q-learning with non-optimized clusters. These gains include lowering both BER and message-passing complexity. We also demonstrate that by concatenating a short outer Hamming code with an inner LDPC code, we significantly improve the error correction capability of our RL-based sequential decoder with

only a minor penalty in rate and complexity. As Bayesian inference over graphical models is at the core of many machine learning applications, ongoing work includes extending our RL approach to other applications involving BP-based message passing over a factor graph defined by an underlying probabilistic model.

REFERENCES

[1] S. Habib, A. Beemer, and J. Kliewer, "Learned scheduling of LDPC decoders based on multi-armed bandits," in *Proc. IEEE Int. Symp. Inf. Theory*, 2020, pp. 2789–2794.

[2] S. Habib, A. Beemer, and J. Kliewer, "Learning to decode: Reinforcement learning for decoding of sparse graph-based channel codes," in *Proc. 34th Conf. Neural Inf. Process. Syst. (NeurIPS)*, Vancouver, BC, Canada, 2020.

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

[4] 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, Jul. 2014.

[5] 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, no. 2, pp. 58–60, Feb. 2001.

[6] Y. Kou, S. Lin, and M. P. C. Fossorier, "Low density parity-check codes based on finite geometries: A rediscovery and new results," *IEEE Trans. Inf. Theory*, vol. 47, no. 7, pp. 2711–2736, Nov. 2001.

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

[8] 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, no. 5, pp. 673–680, May 1999.

[9] A. I. 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.

[10] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2015.

[11] E. Nachmani, E. Marciano, L. Lugosch, W. J. Gross, D. Burshtein, and Y. Béery, "Deep learning methods for improved decoding of linear codes," *IEEE J. Sel. Topics Signal Process.*, vol. 12, no. 1, pp. 119–131, Feb. 2018.

[12] I. Béery, N. Raviv, T. Raviv, and Y. Béery, "Active deep decoding of linear codes," *IEEE Trans. Commun.*, vol. 68, no. 2, pp. 728–736, Feb. 2020.

[13] Y. Jiang, H. Kim, H. Asnani, S. Kannan, S. Oh, and P. Viswanath, "Deep learning based channel codes for point-to-point communication channels," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 2758–2768.

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

[15] N. Doan, S. A. Hashemi, and W. Gross, "Decoding polar codes with reinforcement learning," 2020. [Online]. Available: arXiv:2009.06796.

[16] Y. Liao, S. A. Hashemi, J. Cioffi, and A. Goldsmith, "Construction of polar codes with reinforcement learning," 2020. [Online]. Available: arXiv:2009.09277.

[17] F. Carpi, C. Hager, M. Martalo, R. Raheli, and H. D. Pfister, "Reinforcement learning for channel coding: Learned bit-flipping decoding," in *Proc. 57th Allerton Conf. Commun. Control Comput.*, 2019, pp. 922–929.

[18] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Univ. Oxford, Oxford, U.K., 1989.

[19] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.

[20] M. O. Duff, "Q-learning for bandit problems," Dept. Comput. Sci., Univ. Massachusetts, Amherst, MA, USA, Rep. CMPSCI 95-26, 1995.

[21] Ö. Şimşek, A. P. Wolfe, and A. G. Barto, "Identifying useful subgoals in reinforcement learning by local graph partitioning," in *Proc. 22nd Int. Conf. Mach. Learn.*, 2005, pp. 816–823.

[22] S. Mannor, I. Menache, A. Hoze, and U. Klein, "Dynamic abstraction in reinforcement learning via clustering," in *Proc. 21st Int. Conf. Mach. Learn.*, 2004, pp. 71–78.

[23] R. Parr and S. J. Russell, "Reinforcement learning with hierarchies of machines," in *Proc. Adv. Neural Inf. Process. Syst.*, 1998, pp. 1043–1049.

[24] S. Bitzer, M. Howard, and S. Vijayakumar, "Using dimensionality reduction to exploit constraints in reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2010, pp. 3219–3225.

[25] D. Shah and Q. Xie, "Q-learning with nearest neighbors," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 3111–3121.

[26] L. Dolecek, Z. Zhang, V. Anantharam, M. Wainwright, and B. Nikolic, "Analysis of absorbing sets and fully absorbing sets of array-based LDPC codes," *IEEE Trans. Inf. Theory*, vol. 56, no. 1, pp. 181–201, Jan. 2010.

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

[28] A. Beemer, S. Habib, C. Kelley, and J. Kliewer, "A generalized algebraic approach to optimizing SC-LDPC codes," in *Proc. 55th Allerton Conf. Commun. Control Comput.*, Oct. 2017, pp. 672–679.

[29] J. C. Gittins, "Bandit processes and dynamic allocation indices," *J. Royal Stat. Soc. B, Methodol.*, vol. 41, no. 2, pp. 148–163, 1979.

[30] D. Russo, B. V. Roy, A. Kazerouni, and I. Osband, "A tutorial on Thompson sampling," *Found. Trends® Mach. Learn.*, vol. 11, no. 1, pp. 1–96, 2017.

[31] J. Vermorel and M. Mohri, "Multi-armed bandit algorithms and empirical evaluation," in *Proc. Eur. Conf. Mach. Learn. (ECML)*, 2005, pp. 437–448.

[32] S. Chung, T. J. Richardson, and R. L. Urbanke, "Analysis of sum-product decoding of low-density parity-check codes using a Gaussian approximation," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 657–670, Feb. 2001.

[33] R. M. Tanner, D. Sridhara, A. Sridharan, T. E. Fuja, and D. J. Costello, Jr., "LDPC block and convolutional codes based on circulant matrices," *IEEE Trans. Inf. Theory*, vol. 50, no. 12, pp. 2966–2984, Dec. 2004.

[34] J. Li, S. Lin, and K. Abdel-Ghaffar, "Improved message-passing algorithm for counting short cycles in bipartite graphs," in *Proc. IEEE Int. Symp. Inf. Theory*, 2015, pp. 416–420.

[35] D. J. C. MacKay, *Information Theory, Inference and Learning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2003.

[36] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression* (Engineering and Computer Science), vol. 159. Boston, MA, USA: Springer, 1992.