Learned Scheduling of LDPC Decoders Based on Multi-armed Bandits

Salman Habib, Allison Beemer, and Jörg Kliewer
Helen and John C. Hartmann Dept. of Electrical and Computer Engineering,
New Jersey Institute of Technology, Newark, NJ 07102

***Abstract*— The multi-armed bandit (MAB) problem refers to the dilemma encountered by a gambler when deciding which arm of a multi-armed slot machine to pull in order to maximize the total reward earned in a sequence of pulls. In this paper, we model the scheduling of a node-wise sequential LDPC decoder as a Markov decision process, where the underlying Tanner graph is viewed as a slot machine with multiple arms corresponding to the check nodes. A fictitious gambler decides which check node to pull (schedule) next by observing a reward associated with each pull. This interaction enables the gambler to discover an optimized scheduling policy that aims to reach a codeword output by propagating the fewest possible messages. Based on this policy, we contrive a novel MAB-based node-wise scheduling (MAB-NS) algorithm to perform sequential decoding of LDPC codes. Simulation results show that the MAB-NS scheme, aided by an appropriate scheduling policy, outperforms traditional scheduling schemes in terms of complexity and bit error probability.**

I. Introduction

Low-density parity-check (LDPC) codes are sparse graph-based codes whose rates approach the capacity of symmetric binary input channels [1], [2]. LDPC codes are decoded via iterative algorithms, such as *belief propagation* (BP), which operate on the code's Tanner graph [3]. Traditional BP decoders employ a flooding scheme, where at each iteration, all variables nodes (VNs) of the Tanner graph are updated simultaneously, followed by a simultaneous update of all check nodes (CNs). The flooding schedule is computationally intensive compared to sequential scheduling, where the nodes of a Tanner graph 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 with respect to the flooding scheme.

In [4], a sequential CN scheduling scheme, so-called *node-wise scheduling* (NS), is proposed, where the criterion for selecting the next CN depends on its residual, given by the magnitude of the difference between two successive messages emanating from that CN. In NS, all CN to VN messages corresponding to a CN with the highest residual are propagated simultaneously. Although the NS method converges faster than the flooding scheme, it computes provisional messages for updating the CN residuals in real-time, making NS more computationally intensive than the flooding scheme for the same total number of messages propagated.

In this paper, we mitigate the computational complexity inherent in NS by employing a *multi-armed bandit-based NS* (MAB-NS) scheme. To the best of our knowledge, such a scheme has not been proposed in conjunction with BP decoder scheduling in the open literature. Instead of computing residuals, the proposed MAB-NS algorithm evaluates an action-value function prior to scheduling, which determines how beneficial an action (a selected CN) is for maximizing convergence speed/performance of the decoder. We model the NS algorithm as a *Markov decision process* (MDP), where the Tanner graph is conceived as an m-armed slot machine with m CNs (arms), and a fictitious gambler attempts to discover the CNs that elicit the highest reward. In this paper we employ the CN residual [4] as a reward function. Repeated scheduling during the learning phase enables the gambler to estimate the action-value function, which can be viewed as a reinforcement learning (RL) task intended to optimize the CN scheduling order. We refer to this RL task as the MAB problem.

In the following, we discuss two strategies for estimating the action-value function: by computing the Gittins indices (GIs) of all the CNs, and via Q-learning. In the GI approach, each CN is considered as an independent bandit process (with independent rewards), leading to a learning complexity that grows linearly with the number of CNs. In comparison, Q-learning is a Monte Carlo approach for estimating the action-value function, without explicit assumptions on the distribution of the bandit processes. However, the learning complexity in this approach can grow exponentially with the number of CNs. To lower the Q-learning complexity, we propose a novel clustering scheme where CNs are grouped into smaller clusters with separate state and action spaces.

II. Preliminaries

A. LDPC Codes

A (J, K)-regular LDPC code is defined as the null space of a sparse parity-check matrix $\mathbf{H} \in \mathbb{F}_2^{m\times n}$, where each row (resp. column) of $\mathbf{H}$ contains K (resp. J) ones, with $K \ll m$, $J \ll n$. The code *block length* is n and its *design rate* is given by $R = 1 - \frac{m}{n}$, which is equal to the actual rate if and only if the parity-check matrix is full-rank. In a sequential BP decoding step, a *CN residual* $r_{m_{a\to v}}$ is defined as

$$r_{m_{a\to v}} \triangleq |m'_{a\to v} - m_{a\to v}|. \tag{1}$$

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.

Let $G_{\mathbf{H}} = (V \cup C, E)$ denote the Tanner graph corresponding to $\mathbf{H}$, where $V = \{v_0, \ldots, v_{n-1}\}$ is a set of n VNs, $C = \{c_0, \ldots, c_{m-1}\}$ is a set of m CNs, and there is an edge in set E connecting v_i to c_j if and only if $H_{j,i} = 1$. In $G_{\mathbf{H}}$, if $X \subseteq V$, let $\mathcal{N}(X)$ be the set of all neighbors of X, and let $O(X)$ be the set of neighbors of X with odd degree in the subgraph induced by $X \cup \mathcal{N}(X)$. For $A > 0, B \geq 0$, an (A, B) absorbing set (ABS) X is a set of VNs with $|X| = A$,

This work was supported in part by NSF grant ECCS-1711056 and by the US Army Research Office under Cooperative Agreement Number W911NF-17-2-0183.

$|O(X)| = B$, and the property that each VN in X has strictly fewer neighbors in $O(X)$ than in $C \setminus O(X)$ [5].

ABSs are known to cause BP decoder failures, exhibited by a flattening of the bit error rate (BER) performance curve in the high signal-to-noise ratio (SNR) region, also known as an error-floor [5]. In particular, $(3, 3)$ ABSs are most harmful for column weight-3 array-based (AB) LDPC codes [6], [7].

B. Multi-armed Bandits

The MAB is a special RL problem where in each time step, a gambler must decide which arm of an m-armed slot machine to pull next in order to maximize the total reward in a series of pulls. In this paper, the m-armed slot machine is modeled as a finite Markov decision process (MDP) [8], and the optimized scheduling order of arms is obtained by solving the related MAB problem.

In the remainder of the paper, let $[[x]] \triangleq \{0, \ldots, x-1\}$, where x is a positive integer. In an m-armed bandit problem, let $S_t^{(0)}, \ldots, S_t^{(m-1)}$ represent m bandit processes (arms), where each random variable (r.v.) $S_t^{(j)}$, $j \in [[m]]$, can take M possible real values. Let a state space $\mathcal{S}$ contain all M^m possible realizations of the sequence $S_t^{(0)}, \ldots, S_t^{(m-1)}$. Let the r.v. $S_t \in [[M^m]]$, with realization s represent the index of 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 a m-armed slot machine at time t. If the arms are modeled as independent bandit processes, we define a r.v. $\hat{S}_t \in [[M]]$, with realization $\hat{s}$, as the realization $s_t^{(j)}$ of arm j. Let $A_t \in [[m]]$ represent an action, with realization a, indicating the index of an arm that has been pulled by the gambler 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 pulling arm 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 playing arm A_t in state S_t.

C. Solving the MAB Problem by Computing Gittins Indices

As outlined above, 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 only needs to compute its action-value function, in this case known as the Gittins index (GI) $G(\hat{s}, a)$, given by [9]

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

where τ is a r.v. with realizations in $\{1, 2, ...\}$ that gives the number of times the gambler 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 gambler to take that action [8], [9]. For the Gittins scheme, the optimal arm scheduling policy for a gambler is given by

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

D. Solving the MAB Problem via Q-learning

Optimal policies for MDPs can also be estimated via Monte Carlo techniques such as Q-learning [11], [10], [12]. The estimated action-value function $Q_t(S_t, A_t)$ in Q-learning represents the expected long-term reward achieved by the gambler 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

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

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 [12, 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 gambler to incorporate any dependencies of the arms, unlike the Gittins scheme.

In Q-learning, the optimal policy for the gambler, $\pi_Q^{(\tau)}$, in state s is determined as

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

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 gambler, 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).

III. MAB-based Node-wise Scheduling

The proposed MAB-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 variable nodes, and subsequent messages sent from these variable nodes to their other CN neighbors. Sequential CN scheduling is carried out until a stopping condition is reached, or an iteration threshold is exceeded. The MAB-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.

Now, we consider the effect of $(3, 3)$ ABSs on the performance of sequential decoders. Suppose that the VNs $v_1, v_2, v_3 \in \{0, 1, \ldots, n-1\}$ of a $(3, 3)$ ABS are initially in error. In a sequential scheduling step, suppose that a neighboring CN $\mathcal{N}(v_{i'})$, $i' \in \{1, 2, 3\}$ corrects the erroneous belief (LLR) of $v_{i'}$. Then, in the second half of the same iteration, this VN can correct its CN neighbors, and these CNs can correct the remaining VNs in the ABS. On the other hand, for the flooding scheme, the erroneous VN will also receive messages from the other elements of $\mathcal{N}(v_{i'})$, which can potentially reinforce the error. If this is the case,

the decoder is trapped as this VN cannot correct any of its neighbors. Thus we can state the following remark.

Remark. *Sequential scheduling techniques such as the proposed MAB-NS scheme are more likely to correct errors associated with* $(3,3)$ *ABSs than a flooding-based scheme.*

We define the optimal scheduling order to be the one that yields a codeword output by propagating the least number of CN to VN messages. The decoding algorithm informs the imaginary gambler of the current state of the decoder, and the reward obtained after performing an action (scheduling a CN). Based on these observations, the gambler 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 gambler after scheduling CN a is defined as $R_a = \max_{v \in \mathcal{N}(a)} r_{m_{a\to v}}$, where $r_{m_{a\to v}}$ is computed according to (1).

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 [4]. Note that in our MAB problem, residuals are computed for estimating the action-value function only, which is done off-line (see Section IV for details).

An iteration number ℓ (resp. iteration threshold $\ell_{\max}$) of the MAB-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 MAB-NS algorithm is a vector of LLRs denoted as $\mathbf{L} = [L_0, \ldots, L_{n-1}]$. In the MAB-NS scheme, the value (or state) of CN j at the end of iteration ℓ is denoted 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. Let $\hat{\mathbf{S}}_\ell = \hat{S}_\ell^{(0)}, \ldots, \hat{S}_\ell^{(m-1)}$, with realization $\hat{\mathbf{s}}_\ell = \hat{s}_\ell^{(0)}, \ldots, \hat{s}_\ell^{(m-1)}$, represent a soft syndrome vector of the MAB-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 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 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 MAB-NS scheme 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.

Algorithm 1: MAB-NS for LDPC codes

Input : $\mathbf{L}$, $\mathbf{H}$
Output: reconstructed codeword
1 Initialization:
2 $\quad \ell \leftarrow 0$
3 $\quad m_{c\to v} \leftarrow 0$ // for all CN to VN messages
4 $\quad m_{v_i\to c} \leftarrow L_i$ // for all VN to CN messages
5 $\quad \hat{\mathbf{L}}_\ell \leftarrow \mathbf{L}$
6 $\quad \hat{\mathbf{S}}_\ell \leftarrow \mathbf{H}\hat{\mathbf{L}}_\ell$
7 **foreach** $a \in [[m]]$ **do**
8 $\quad s_\ell^{(a)} \leftarrow g_M(\hat{s}_\ell^{(a)})$ // M-level quantization
9 **end**
// decoding starts
10 **if** *stopping condition not satisfied or* $\ell < \ell_{\max}$ **then**
11 $\quad s \leftarrow$ index of $\mathbf{S}_\ell$
12 $\quad$ select CN a according to an optimum scheduling policy
13 $\quad$ **foreach** $v_k \in \mathcal{N}(a)$ **do**
14 $\quad\quad$ compute and propagate $m_{a\to v_k}$
15 $\quad\quad$ **foreach** $c_j \in \mathcal{N}(v_k) \setminus a$ **do**
16 $\quad\quad\quad$ compute and propagate $m_{v_k\to c_j}$
17 $\quad\quad$ **end**
18 $\quad\quad$ $\hat{L}_\ell^{(k)} \leftarrow \sum_{c\in\mathcal{N}(v_k)} m_{c\to v_k} + L_k$ // update LLR of v_k
19 $\quad$ **end**
20 $\quad$ **foreach** *CN* j *that is a neighbor of* $v_k \in \mathcal{N}(a)$ **do**
21 $\quad\quad$ $\hat{s}_\ell^{(j)} \leftarrow \sum_{v_i\in\mathcal{N}(j)} \hat{L}_\ell^{(i)}$
22 $\quad\quad$ $s_\ell^{(j)} \leftarrow g_M(\hat{s}_\ell^{(j)})$ // update syndrome $\mathbf{S}_\ell$
23 $\quad$ **end**
24 $\quad$ $\ell \leftarrow \ell + 1$ // update iteration
25 **end**

Now, we address a decoding error encountered by the NS scheme which we expect to correct using MAB-NS. 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 MAB-NS scheme, the gambler attempts to schedule a CN based on its expected residual instead of the immediate one. As a result, the MAB-NS scheme employs a more global decoding approach in contrast to the pure NS scheme [4] and is more likely to overcome undetected errors compared to NS.

IV. Learning CN Scheduling Policies

In our MAB problem, the gambler's goal is to estimate, from experience, the action-value function used for implementing the CN scheduling policy shown 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 gambler must schedule CNs that are known to produce high residuals (exploitation). But to discover such CNs, the gambler must select CNs that were not scheduled before (exploration). To accommodate this trade-off, we utilize the well-known ϵ-greedy exploration scheme [8], [13]. The estimate of the action-value function improves as more CNs are scheduled by the gambler. In the following, we discuss two RL techniques for learning the action-value function used in MAB-NS: by estimating the GIs, and via Q-learning.

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 holds 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 gambler increases linearly with the number of CNs in the underlying graph. Note that $P(S_{\ell+1}|S_\ell, A_\ell) \in \{0,1\}$ as the Tanner graph is fixed and the messages are deterministically computed. We also apply a specific stopping time policy by selecting an integer $\ell^*_{\max} \in \{1,2,\dots,\}$ with the condition $\Pr(\ell_{\max} = \ell^*_{\max}) = 1$. Based on these considerations, (2) is rewritten for our MAB problem as

$$G(\hat{s}, a) = \max_{\ell^*_{\max}} \frac{\sum_{\ell=0}^{\ell_{\max}-1} \beta^\ell R_\ell(\hat{s}, a, \hat{s}')}{\sum_{\ell=0}^{\ell_{\max}-1} \beta^\ell}. \tag{6}$$

Since a gambler 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 [11], [12]. Q-learning does not rely 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 gambler now grows exponentially with m. As a result, the traditional Q-learning approach suffers from a much greater learning complexity compared to the GI scheme. 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 $\mathbf{S}_\ell^{(u,z)} = S_\ell^{(uz)}, \dots, S_\ell^{(uz+z-1)}$ 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 observed by the gambler is upper-bounded by $\lceil \frac{m}{z} \rceil |\mathcal{S}_u^{(M)}|$. The action space of cluster u is defined as $\mathcal{A}_u = [[z]]$.

Note that the larger the size of the cluster, the greater the ability of the gambler to take into account any dependencies between CN LLRs. Hence, there exists a trade-off between the effectiveness of clustered Q-learning to take into account these dependencies, and the learning complexity which grows exponentially with the size of each cluster: the cluster size should be large enough to accommodate the dependencies of the CN messages as much as possible, but not so large that makes learning infeasible.

Apart from this trade-off, clustering does not restrict Q-learning, as the reward obtained by the gambler 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

$$Q_{\ell+1}(s_u, a_u) = (1-\alpha) Q_\ell(s_u, a_u) + \alpha \left(R_\ell(s_u, a_u, f(s_u, a_u)) + \beta \max_{u', a_{u'}} Q_\ell(f(s_{u'}, a_{u'}), a_{u'}) \right), \tag{7}$$

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 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

$$a_u = \begin{cases} \text{uniformly random over } u \text{ and } \mathcal{A}_u \text{ w.p. } \epsilon, \\ \pi_Q^{(\ell)} \text{ w.p. } 1-\epsilon, \end{cases} \tag{8}$$

where $\pi_Q^{(\ell)} = \arg\max_{a_u \text{ s.t. } u \in [[\lceil \frac{m}{z} \rceil]]} Q_\ell(s_u, a_u)$. After successful learning, $\pi_Q^{(\ell_{\max})}$ yields an optimized CN scheduling policy, where $\ell_{\max}$ is the maximum number of decoder iterations for a given input of channel information $\mathbf{L}$.

Algorithm 2 represents the method for clustered Q-learning. The input to this algorithm is a set $\mathscr{L} = \{\mathbf{L}_0, \dots, \mathbf{L}_{|\mathscr{L}|-1}\}$ containing $|\mathscr{L}|$ realizations of $\mathbf{L}$ over which clustered Q-learning is performed, and a parity-check matrix $\mathbf{H}$. This algorithm trains the gambler to learn the optimum CN scheduling policy for a given $\mathbf{H}$. In each optimization step ℓ, the gambler 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.

V. Decoder Implementation and Simulation Results

Each element of the state vector $\hat{\mathbf{S}}_\ell$ is quantized using a standard scalar quantization algorithm [14], where a realization $\hat{s}_\ell^{(j)}$ represents the "source signal" to be quantized. Provided that there exist 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. We generate a dataset comprising 10^5 realizations of $\hat{s}_\ell^{(j)}$ by randomly scheduling CNs via NS to estimate its distribution.

To verify the effectiveness of learned CN scheduling based on GI estimation and clustered Q-learning, we apply the scheduling policies $\hat{\pi}_G$ and $\pi_Q^{(\ell_{\max})}$, respectively, in Step 12 of Algorithm 1, resulting in schemes denoted as MAB-NS-1 and MAB-NS-2, respectively. We then utilize MAB-NS-1 and MAB-NS-2 for sequential decoding of both random $(3,6)$-regular and $(3,7)$-array-based (AB) LDPC codes [16]. Even though clustering helps to reduce the state-space significantly

Algorithm 2: Clustered Q-learning

Input : $\mathscr{L}$, $\mathbf{H}$
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 (8)
    select u as cluster index of CN a_u
    $\mathbf{S}_\ell^{(u,z)} \leftarrow s_\ell^{(uz)}, \ldots, s_\ell^{(uz+z-1)}$
    $s_u \leftarrow$ index of $\mathbf{S}_\ell^{(u,z)}$
    foreach $v_i \in \mathcal{N}(a_u)$ **do**
      compute and propagate $m_{a_u \to v_i}$
      foreach $c_j \in \mathcal{N}(v_i) \setminus a_u$ **do**
        compute and propagate $m_{v_i \to c_j}$
      end
      $\hat{L}_\ell^{(i)} \leftarrow \sum_{c \in \mathcal{N}(v_i)} m_{c \to v_i} + L_i$ // update LLR
    end
    foreach *CN* j *that is a neighbor of* $v_k \in \mathcal{N}(a_u)$ **do**
      $\hat{s}_\ell^{(j)} \leftarrow \sum_{v_i \in \mathcal{N}(j)} \hat{L}_\ell^{(i)}$
      $s_\ell^{(j)} \leftarrow g_M(\hat{s}_\ell^{(j)})$ // 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 (7)
    $\ell \leftarrow \ell + 1$ // update iteration
  end
end

compared to standard Q-learning, it is still more complex than estimating GIs. Hence, our choice of code block length for MAB-NS-1 and MAB-NS-2 is influenced by the run-time complexity of Algorithm 2. Since a large $|\mathscr{L}|$ is needed to accurately estimate the action-value, we found short codes, with a block length of approximately 200, to be suitable for implementing Algorithm 2 on our system with a reasonable cluster size z.

For learning, we consider $\alpha = 0.1$, $\beta = 0.9$, $\epsilon = 0.6$, $z = 7$, $M = 4$, $\ell_{\max} = 25$, for both codes, and $|\mathscr{L}| = 2.5 \times 10^7$ (resp. 5×10^7) for the $(3, 6)$-regular (resp. $(3, 7)$-AB) code. For GI estimation and clustered Q-learning, training is done according to Section IV-A and Algorithm 2, respectively, for each code, and the corresponding action-value functions are used to perform the MAB-NS-1 and MAB-NS-2 schemes, respectively. We compare the performances of MAB-NS-1 and MAB-NS-2 with existing BP decoding schemes of flooding and NS for $\ell_{\max} = 25$. The BER performances of $(3, 6)$-regular and $(3, 7)$-AB LDPC codes using these decoding techniques are shown in Figs. 1 and 2, respectively. These results clearly demonstrate that MAB-NS-2 is superior to the other decoding schemes in terms of BER performance, although the gain is diminished for more structured codes as AB LDPC constructions. A comparison with a $(128, 64)$ Reed Muller code under majority logic decoding reveals a threshold of around 4.5 dB and performs therefore much worse compared to all the codes shown in Figs. 2 and 3.

In Table I, we compare the average number of CN to VN messages propagated in the different decoding schemes to attain the results in Figs. 1 and 2. The numbers without (resp. with) parentheses correspond to the $(3, 6)$-regular and (resp. $(3, 7)$-AB) LDPC code. We see that MAB-NS-2 generates a lower number of CN to VN messages compared to the other schemes. Moreover, unlike NS, MAB-NS-2 avoids the computation of residuals in real-time, providing a significant reduction in message-passing complexity for short LDPC codes.

Future work will focus on extending these promising initial results to longer LDPC codes.

SNR	0	0.6	1.2	1.4	1.8
flooding	16070 (29257)	13923 (27684)	9274 (22339)	7657 (17367)	4523 (10401)
MAB-NS-1	337 (402)	299 (367)	254 (314)	253 (286)	220 (254)
NS	324 (389)	290 (358)	256 (299)	238 (278)	222 (251)
MAB-NS-2	280 (374)	236 (308)	181 (281)	162 (276)	138 (234)

TABLE I: Average number of CN to VN messages propagated in various decoding schemes for a $(3, 6)$-regular ($(3, 7)$-AB) LDPC code for attaining the results shown in Figs. 1 and 2.

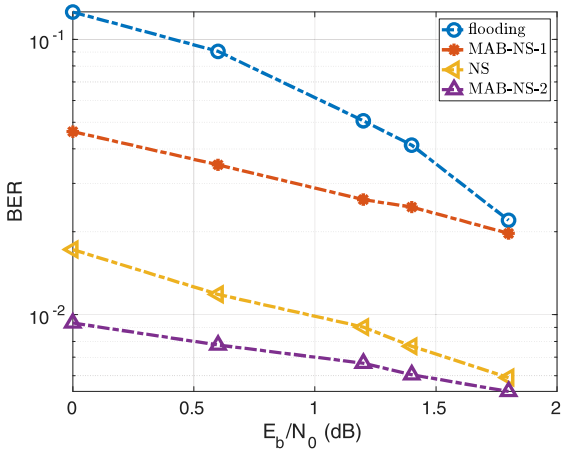

Fig. 1: BER results using different BP decoding schemes for a $(3, 6)$-regular LDPC code with block length $n = 196$.

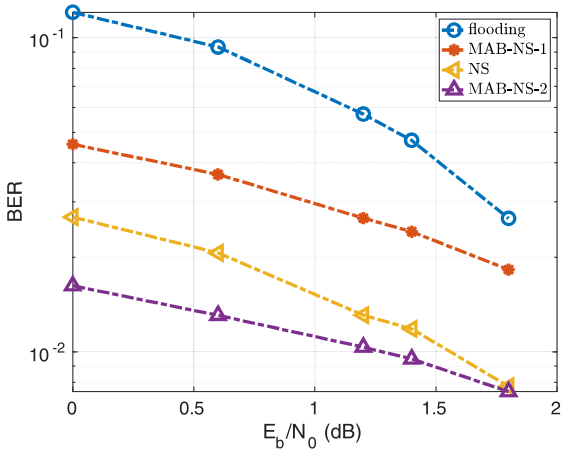

Fig. 2: BER results using different BP decoding schemes for a $(3, 7)$-AB LDPC code with block length $n = 196$.

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 Comms. Mag.*, vol. 52, no. 7, pp. 168–176, 2014.

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

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

[5] L. Dolecek, Z. Zhang, V. Anantharam, M. Wainright, and B. Nikolic, "Analysis of absorbing sets and fully absorbing sets of array-based LDPC codes," *IEEE Trans. Inf. Theory*, vol. 56, pp. 181–201, Jan. 2010.

[6] B. Amiri, A. Reisizadehmobarakeh, H. Esfahanizadeh, J. Kliewer, and L. Dolecek, "Optimized design of finite-length separable circulant-based spatially-coupled codes: An absorbing set-based analysis," *IEEE Trans. on Commun.*, vol. 64, no. 10, pp. 4029–4043, Oct 2016.

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

[8] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," *The MIT Press Cambridge*, 2015.

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

[10] 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 Communication, Control and Computing*.

[11] M. O. Duff, "Q-learning for bandit problems," *CMPSCI Technical Report 95-26*, 1995.

[12] C. J. C. H. Watkins, "Learning from delayed rewards," *PhD Thesis, King's College*, 1989.

[13] J. Vermorel and M. Mohri, "Multi-armed bandit algorithms and empirical evaluation," *European Conference on Machine Learning (ECML)*, pp. 437–448, 2005.

[14] A. Gersho and R. M. Gray, "Vector quantization and signal compression," *The Springer International Series in Engineering and Computer Science (Communications and Information Theory), vol 159. Springer, Boston, MA*, 1992.

[15] 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.

[16] J. L. Fan, "Array codes as low-density parity-check codes," *Proc. 2nd Intl. Symp. Turbo Codes and Rel. Topics*, pp. 543–546, Sep 2000.