2023 12th International Symposium on Topics in Coding (ISTC) | 979-8-3503-2611-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISTC57237.2023.10273470

Learning to Decode Linear Block Codes using Adaptive Gradient-Descent Bit-Flipping

Jovan Milojković
School of Electrical Engineering
University of Belgrade
Belgrade, Serbia
mj205018p@student.etf.bg.ac.rs

Srdan Brkic
School of Electrical Engineering
University of Belgrade
Belgrade, Serbia
srdjan.brkic@etf.rs

Predrag Ivaniš
School of Electrical Engineering
University of Belgrade
Belgrade, Serbia
predrag.ivanis.etf.rs

Bane Vasić
Department of ECE
University of Arizona
Tucson, USA
vasic@ece.arizona.edu

***Abstract*—In this paper we propose a generalization of the recently published adaptive diversity gradient-descent bit flipping (AD-GDBF) decoder, named generalized AD-GDBF (gAD-GDBF) decoder. While the original AD-GDBF decoder was designed for the binary symmetric channel and used mostly to decode regular low-density parity-check codes, the gAD-GDBF algorithm incorporates several improvements which makes it eligible for the additive white Gaussian channel and decoding of arbitrary linear block code. The gAD-GDBF decoder uses the genetic algorithm to optimize a set of learnable parameters, for a targeted linear block code. The effectiveness of the proposed method is verified on short Bose–Chaudhuri–Hocquenghem (BCH) codes, where it was shown that for the same number of decoding iterations the gAD-GDBF decoder outperforms the belief-propagation decoder in terms of bit error rate and at the same time reduces the decoding complexity significantly.**

***Index Terms*—Bose–Chaudhuri–Hocquenghem codes, bit-flipping, genetic algorithm, gradient-descent, diversity decoding, linear block codes**

I. Introduction

In the fifth-generation standard for broadband cellular networks (5GNR) ultra-reliable low-latency communication (URLLC) is recognized as one of the most important service category. However, the most powerful capacity-approaching error correction codes, like turbo or low-density parity-check (LDPC) codes, do not meet the requirements of the URLLC [1]. Recently, considerable progress was made related to the design of short precoded polar codes [2]. Nevertheless, it was shown in [1] that short Bose–Chaudhuri–Hocquenghem (BCH) codes perform closely to the theoretical bounds for a wide range of error rates, under the assumption of the maximum likelihood (ML) decoding.

Given the fact that the ML decoding of BCH codes is considered to be unacceptably complex, recent research attempts try to improve decoders traditionally applied on LDPC codes, like belief propagation (BP) and offset min-sum (OMS), in order to use them on BCH codes. The most promising methods use machine learning techniques to adjust the BP/OMS decoders to a certain BCH code. In the pioneering work [3], Nachmani *et al.* showed that the BP decoder is only a special case of a deep neural network (DNN) and created neural BP (NBP) decoder, by adding learnable weights to messages passed between nodes of the Tanner graph. Interestingly, Lugosch and Gross in [4] showed that an even bigger improvement could be achieved with neural OMS (NOMS) decoder. Their work was refined again by Nachmani *et al.* in [5], [6], by using recursive DNNs and optimization of neural network node activations. Finally, Tian *et al.* in [7] proposed an edge-weighted graph neural network decoder, which also surpasses the BP decoder on short block codes.

The majority of the aforementioned approaches use DNNs to design the decoder, while the number of hidden layers is proportional to the number of decoding iterations. This makes the optimization of the decoders for a larger number of iterations hard, and we argue that the gain of neural decoders, with respect to the BP decoder, usually reduces, if the number of iterations increases (this claim is consistent with numerical results from [7]). Furthermore, training of DNN-based decoders is conduced by minimizing cross-entropy loss function, which can produce suboptimal decoders, as the actual performance metrics are bit or frame error rates.

In this paper we take a different learning approach, which represents a generalization of our recently proposed adaptive diversity gradient-descent bit-flipping (AD-GDBF) decoder [8]. The AD-GDBF decoder is a collection of GDBF decoders with momentum (GDBF-w/M), proposed by Savin in [9], with synergistic behaviour, that is accomplished through optimization of momentum values and weights associated to the GDBF energy function. The AD-GDBF decoder from [8] was designed only for the binary symmetric channel and primarily regular LDPC codes, while here we propose several modifications that ensure its competitiveness on additive white Gaussian noise (AWGN) channels and short block codes, like BCH codes. The learnable parameters are obtained by the genetic algorithm and training on a predefined error set, while the bit error rate is directly minimized during the training.

Although some significant probabilistic bit-flipping decoders were proposed in the past, like noisy GDBF (NGBF)

This research was supported by the Science Fund of the Republic of Serbia, under grants No. 7750284 (Hybrid Integrated Satellite and Terrestrial Access Network - hi-STAR) and No. 6462951 (LIDA). This work was also supported by the Serbian Ministry of Science, Technological Development and Innovation. B. Vasić acknowledges the support of the NSF under grants CCF-1855879, CCF-2100013, CIF-2106189, CCSS-2027844, CCSS-2052751. and NASA-SURP. B. Vasić has disclosed an outside interest in Codelucida to the University of Arizona. Conflicts of interest resulting from this interest are being managed by The University of Arizona in accordance with its policies.

[10], [11] and the probabilistic GDBF-w/M [9] decoders, the bit-flipping decoders are usually considered to be inferior compared to the BP decoder on AWGN channel, when applied to LDPC codes. To the best of our knowledge, bit-flipping algorithms, that approach the BP decoding on block codes with dense parity matrices, have not been reported. In this paper we show that our decoding framework can outperform the BP decoder on short BCH codes, run with the same number of iterations, while its complexity is significantly lower than the BP decoding complexity.

II. Learnable diversity decoding framework

A. Preliminaries

Consider a binary linear block code (N, K), with code length N and K information bits per codeword, described by a parity check matrix $\mathbf{H} = [h_{ji}]_{M\times N}$. Each column of the parity matrix is associated to a *variable* v_i, $1 \le i \le N$ and a set indices $\mathcal{P}(v_i) = \{j|h_{ji} = 1\}$, where cardinality $|\mathcal{P}(v_i)|$ represents a degree of the variable v_i. Similarly, the j-th row of $\mathbf{H}$ is coupled with the parity check equation c_j, $1 \le j \le M$, defined by the following set of indices $\mathcal{Q}(c_j) = \{i|h_{ji} = 1\}$. The total number of ones in the parity matrix we denote by $E = \sum_{j,i} h_{j,i}$. Let us define the bipolar codeword of a code as $\mathbf{x} = (x_1, x_2, \dots, x_N)$, where $x_i \in \{\pm 1\}$ satisfies $\prod_{i\in\mathcal{Q}(c_j)} x_i = 1$, for all $1 \le j \le M$. We consider transmission through the binary-input AWGN channel, which corrupts sent codeword $\mathbf{x}$ and outputs a vector $\mathbf{y} = (y_1, y_2, \dots, y_N)$, $\mathbf{y} \in \mathbb{R}^N$.

The goal of the gradient-descent bit flipping decoders is to maximize the objective function $f(\mathbf{x})$ defined as follows [12]

$$f(\mathbf{x}) = \sum_{i=1}^{N} x_i y_i + \sum_{j=1}^{M} \prod_{i\in\mathcal{Q}(c_j)} x_i, \tag{1}$$

where the first term represents the correlation between the received vector and a potential solution, while the second term is *a penalty factor* that ensures that the global solution is a valid codeword. The non-linearity of the objective function makes searching for the codeword, that is the most correlated with the received vector, hard.

Iterative gradient-based algorithms, that are used to solve the optimization problem (1), associate *local energy* to each variable during a decoding iteration and flip the variables with energy below the threshold, for the predefined maximal number of iterations, denoted in this paper as $L_{\max}$. These methods have tendencies to oscillate between local optima and usually perform worse compared to the massage-passing decoders. In the following subsection, we present a learnable framework that matches the decoding operations to a certain code and overcomes the convergence problems of the state-of-the-art gradient-descent algorithms.

B. General decoder description

We first explain the operations conducted in each decoding iteration, then we highlight the introduced improvements and conditions under which our approach can be reduced to the state-of-the-art GDBF-w/M decoder [9] and our decoder previously described in [8].

We associate a decimal value, called *the potential* to each variable v_i during the ℓ-th decoding iteration, and denote it by $r_i^{(\ell)}$. It is initialized prior to the first iteration to the value received from the channel, i.e. $r_i^{(0)} = y_i$, $1 \le i \le N$. The potential of v_i in the ℓ-th iteration is calculated based on the local energy, denoted by $E_i^{(\ell)}$, with minimal value $E_{\min}^{(\ell)} = \min_{1\le i\le N} E_i^{(\ell)}$. The variable decisions $\hat{x}_i^{(\ell)}$ are obtained by $\hat{x}_i^{(\ell)} = \text{sign}(r_i^{(\ell)})$, where $\text{sign}(\cdot)$ denotes signum function.

Each decoding iteration consists of three parts: i) calculation of the local energy of variables, ii) updating potential of the variables, and iii) deciding to restart the decoder or change the momentum vector.

Energy calculation. Consider now the local energy $E_i^{(\ell)}$ calculated as follows

$$E_i^{(\ell)} = w_{1,i}^{(\ell)} y_i \hat{x}_i^{(\ell-1)} + w_{2,i}^{(\ell)} \sum_{j\in\mathcal{P}(v_i)} s_j^{(\ell)} + m_{l_i}, \tag{2}$$

where $w_{1,i}^{(\ell)}$ and $w_{2,i}^{(\ell)}$ are learnable weights associated to the i-th variable during the ℓ-th decoding iteration, $s_j^{(\ell)} = \prod_{n\in\mathcal{Q}(c_j)} \hat{x}_n^{(\ell-1)}$, while $m_{l_i} \in \{0, 1, \dots, I\}$, $I \in \mathbb{N}^+$, is the momentum correction term, chosen for each variable separately from a vector $\mathbf{m} = (m_1, m_2, \dots, m_{L'}, 0)$, based on the variable's past flipping activity. Namely, if the last flip of the i-th variable occurred during the t-th iteration we calculate the index term ℓ_i as

$$\ell_i = \min(\ell - t, L' + 1), \tag{3}$$

while ℓ_i prior to the decoding is initialized to $L'+1$. We restrict the influence of the momentum to L' consecutive iterations, while variable flips from other iterations are neglected.

Variable potential updates. In each decoding iteration ℓ, a set of *highlighted variables* is formed as follows

$$\mathcal{F}^{(\ell)} = \{v_i | E_i^{(\ell)} \le E_{\min}^{(\ell)} + \delta\}, \tag{4}$$

where $\delta \ge 0$ is a predefined margin, which enables flipping multiple bits, if needed. If $\delta = 0$ we flip a single bit during an iteration. We also define an auxiliary set $\mathcal{G}^{(\ell)}$ as follows

$$\mathcal{G}^{(\ell)} = \{v_i | v_i \notin \mathcal{F}^{(\ell)} \wedge \ell_i \le L''\}, \tag{5}$$

where $L'' \le L'$ is *an iteration window* in which we keep track of the flipping activity. Afterwards, variable potentials are updated according to the following

$$r_i^{(\ell)} = r_i^{(\ell-1)} - \mathbb{1}_{\mathcal{F}^{(\ell)}}(v_i)\hat{x}_i^{(\ell-1)}\delta_1^{(\ell)} + \mathbb{1}_{\mathcal{G}^{(\ell)}}(v_i)\hat{x}_i^{(\ell-1)}\delta_2^{(\ell)}, \tag{6}$$

where $\delta_1^{(\ell)} > 0$ and $\delta_2^{(\ell)} > 0$ are *potential increments* adapted across iterations, which steer variables potential towards positive or negative direction, and $\mathbb{1}_{\mathcal{A}}(x)$ is an indicator function of a set $\mathcal{A}$ defined as follows

$$\mathbb{1}_{\mathcal{A}}(x) = \begin{cases} 1 & \text{if } x \in \mathcal{A} \\ 0 & \text{if } x \notin \mathcal{A}. \end{cases} \tag{7}$$

Algorithm 1 gAD-GDBF decoding framework

```
Input: $\mathbf{y} = (y_i, y_2, \ldots, y_N) \in \mathbb{R}^N$
Output: $\mathbf{x} = (x_i, x_2, \ldots, x_N) \in \{\pm 1\}^N$
Initialization: $\ell = 1,\ \ r_i = y_i,\ \ \ell_i = L'+1, \forall i = 1, \ldots, N$
while $\ell \leq L_{\text{max}}$ do
    $x_i = \text{sign}(r_i),\ \forall i = 1, \ldots, N$
    $s_j = \prod_{i \in \mathcal{Q}(c_j)} x_i,\ \forall j = 1, \ldots, M$
    if $s_j = 1, \forall j = 1, \ldots, M$ then
        $\ell = L_{\text{max}}$
    else
        if $\ell \in \mathcal{M}$ then
            Update $\mathbf{m}$
        end if
        if $\ell \in \mathcal{S}$ then
            $r_i = y_i,\ \forall i = 1, \ldots, N$
        end if
        $E_{\min} = +\infty$
        for $i = 1, \ldots, N$ do
            $\ell_i = \min(\ell_i, L') + 1$
            $E_i = w_{1,i}^{(\ell)} y_i x_i + w_{2,i}^{(\ell)} \sum_{j \in \mathcal{P}(v_i)} s_j + m_{l_i}$
            $E_{\min} = \min\{E_{\min}, E_i\}$
        end for
        $\mathcal{F} = \{v_i | E_i \leq E_{\min} + \delta\}$
        $\mathcal{G} = \{v_i | v_i \notin \mathcal{F} \wedge \ell_i \leq L''\}$
        for $i = 1, \ldots, N$ do
            $r_i = r_i - \mathbb{1}_{\mathcal{F}}(v_i) x_i \delta_1^{(\ell)} + \mathbb{1}_{\mathcal{G}}(v_i) x_i \delta_2^{(\ell)}$
            if $x_i \neq \text{sign}(r_i)$ then
                $\ell_i = 0$
            end if
        end for
        $\ell = \ell + 1$
    end if
end while
```

Note that the sets $\mathcal{F}^{(\ell)}$ and $\mathcal{G}^{(\ell)}$ are disjoint and at most one increment ($\delta_1^{(\ell)}$ or $\delta_2^{(\ell)}$) is associated to a single variable.

Momentum updates. We allow momentum vector changes (updates) in predefined iterations. Prior to the decoding a set of indices $\mathcal{M} \subset \{1, 2, \ldots, L_{\text{max}}\}$ is defined, while the momentum vector is changed in all iterations $\ell \in \mathcal{M}$.

Decoding restarts. Prior to the decoding, a set of indices $\mathcal{S} \subset \{1, 2, \ldots, L_{\text{max}}\}$ is formed, and

$$\text{if } \ell \in \mathcal{S} \ \Rightarrow\ r_i^{(\ell)} = y_i,\ \forall i = 1, \ldots, N. \tag{8}$$

For convenience, we chose $\mathcal{S} \subseteq \mathcal{M}$, which means that every decoder restart is followed by the momentum update, while the momentum update may not lead to the decoder restart.

The proposed decoding framework we call **the generalized AD-GDBF (gAD-GDBF) decoder** and formally express it in Algorithm 1, where, for the reason of clarity, we abandon the dependency on the decoding iteration for all the parameters, which values will be re-written in the subsequent iteration.

The proposed scheme reduces to the GDBF-w/M decoder, described in [9], if the following condition are fulfilled

- The sets of restart indices and momentum updates are empty, i.e., $\mathcal{S} = \mathcal{M} = \emptyset$;
- The momentum values are organized in non-increasing order, i.e., $m_1 \geq m_2 \geq \ldots \geq m_{L'} \geq 0$;
- The variable potentials are initialized to $r_i^{(0)} = \text{sign}(y_i)$ and $\delta_1^{(\ell)} = 2$ and $\delta_1^{(\ell)} = 0$, for all $1 \leq i \leq N$, and $1 \leq \ell \leq L$;
- The weights $w_{1,i}^{(\ell)} = w$, $w > 0$ and $w_{2,i}^{(\ell)} = 1$, for all $1 \leq i \leq N$, and $1 \leq \ell \leq L$.

Our published AD-GDBF decoder, described in [8], is mostly used for regular LDPC codes and on binary symmetric channels, and does not allow different weights to be associated to different variable bits, nor does it use varying potential increments (it automatically flips bits from the set $\mathcal{F}$). Next we will explain the key features of the proposed algorithm.

We allow weights $w_{1,i}^{(\ell)}$ and $w_{2,i}^{(\ell)}$ to be chosen independently for each variable bit (or groups of bits) in order to adjust the decoder to a specific code structure, i.e., to compensate irregularity of a code. If all variables were scaled with the same values (for example w_1 and w_2), erroneous variables with the lower degrees will be harder to correct. For example, consider two isolated erroneous variables v_i and v_j, with degrees $|\mathcal{P}(v_i)| = 3$ and $|\mathcal{P}(v_j)| = 7$, respectively. If both variables have all unsatisfied checks, the variable v_i will have the energy $E_i = w_1 y_i - 3w_2$, if no previous flips occurred, while under the same condition, the energy of the variable v_j is $E_j = w_1 y_j - 7w_2$ and with high probability v_j will be flipped before the variable v_i. By adding learnable weights, we can change the behaviour of the decoder and force it to flip variables with lower degrees more frequently.

On the other hand, the momentum m_ℓ is an additive term with the purpose to reduce consecutive flipping of the same variable bits, as described in [9]. We also allow changes of momentum vector $\mathbf{m}$ after the predefined number of iterations, in order to prevent oscillatory behavior, commonly attributed to trapping sets. The frequent momentum updates can be seen as a pseudo-random perturbation of the energy function - behavior usually achieved only in probabilistic decoders.

Variable potential updates are another way to mimic behavior of probabilistic decoders (like [13]). Instead of flipping all variables from the set $\mathcal{F}^{(\ell)}$ automatically, we only push the variable potentials towards flipping, where only a portion of variables from $\mathcal{F}^{(\ell)}$ will change the sign. In a sense, the absolute potential of variables with the lowest energy is reduced, while the variables that were flipped in previous L'' iterations were "rewarded" and their absolute potential is increased. Although the momentum term in the energy function has the same responsibility, our numerical results have shown that the two approaches complement one another. In our decoding framework, the momentum term is restricted to L' iterations – two flips of the same variable separated for more than L' iterations are not prevented. On the other hand, a variable potential is accumulated from the start of the

TABLE I: Complexity of various decoders per decoding iteration on (63,45) / (127,106) / (255,215) BCH codes.

Decoder	# of learnable params.	# of multiplications	# of tanh of $\tanh^{-1}$	# of additions	# of comparisons	# of XORs
BP	- / - / -	846 / 1995 / 8920	432 / 1008 / 4480	864 / 2016 / 8960	- / - / -	- / - / -
NBP	495 / 1135 / 4735	1341 / 3130 / 13655	432 / 1008 / 4480	864 / 2016 / 8960	- / - / -	- / - / -
NOMS	495 / 1135 / 4735	495 / 1135 / 4735	- / - / -	1296 / 3024 / 13440	1242 / 2961 / 13320	846 / 1995 / 8920
gAD-GDBF	180 / 204 / 360	126 / 254 / 510	- / - / -	126 / 254 / 510	125 / 253 / 509	477 / 1114 / 4695

decoding (or from the previous decoder restart) and even after the momentum effect is lost, previously flipped variables will less likely to be flipped again.

Decoding restarts prevent the decoding process to converge to a wrong codeword and the detailed explanation of the technique can be found in [8], [11].

Due to space limitations, we do not show the gain achieved by each improvement we introduced separately; however, the numerical examples from Section III indicate that the exclusion of any modification will result in inferior performance.

C. Learning method

We can divide all the parameters of the algorithm, described in the previous subsection into two categories:

- *heuristics* - maximal number of iterations $L_{\max}$, momentum length and values, L' and I, respectively, increment window length L'', energy margin δ, the set $\mathcal{M}$;
- *learnable parameters* - energy wights $w_{1,i}^{(\ell)}$ and $w_{2,i}^{(\ell)}$, $(1 \le i \le N,\ 1 \le \ell \le L_{\max})$, variable potential increments $\delta_1^{(\ell)}$ and $\delta_2^{(\ell)}$, $(1 \le \ell \le L_{\max})$, $|\mathcal{M}|$ different momentum vectors $\mathbf{m}$, and the set of restarts $\mathcal{S}$, $(|\mathcal{S}| \le |\mathcal{M}|)$.

Values of the heuristics are chosen empirically and known in advance, i.e., prior to the optimization of the learnable parameters. Values of the learnable parameters are obtained through machine learning-type optimization. Note that in the general description of the decoder we allow different weights to be assigned to each variable during each decoding iteration, which can make the optimization complex. To reduce the number of learnable parameters we put the following restrictions:

- all the variables with the same degree have the same weight and
- the weights and the potential increments are constant between two momentum updates, i.e.,

$$w_{1,i}^{(\ell)} = w_{1,|\mathcal{P}(v_i)|}^{(k)}, \quad w_{2,i}^{(\ell)} = w_{2,|\mathcal{P}(v_i)|}^{(k)},$$
$$\delta_1^{(\ell)} = \delta_1^{(k)},\ \delta_2^{(\ell)} = \delta_2^{(k)}, \forall i = 1, \dots, N \wedge\ m < \ell \le k, \quad (9)$$

where $\{m, k\} \subset \mathcal{M} \cup \{0\}$, $m < k$, is such that there does not exist $n \in \mathcal{M}$, for which $m < n < k$. In other words, m and k are the two closest elements in $\mathcal{M} \cup \{0\}$. Recall that $\mathcal{S} \subseteq \mathcal{M}$, and $\mathcal{S}$ can be constructed by optimizing $|\mathcal{M}|$ binary indicators $\mathbb{1}_{\mathcal{S}}(x), x \in \mathcal{M}$. Thus, the total number of revised learnable parameters depends on the cardinality of $\mathcal{M}$ and the number of distinct variable degrees in a code (called D) i.e., it is equal to $|\mathcal{M}|(2D + L' + 3)$.

To further reduce the complexity of the optimization, we quantize the learnable parameters and allow them to take values only from finite discrete sets. Then we represent all the learnable parameters in binary form, append one after another, and use a genetic algorithm to optimize all the parameters jointly. The advantage of using genetic algorithm is that we can directly use error rate (bit or frame) as a criterion function and directly steer the learnable parameters toward minimizing error rate on predefined training sets. The detailed description of the optimization process can be found in our previous work [8], and we omit it here because of the space limitations.

D. Complexity analysis

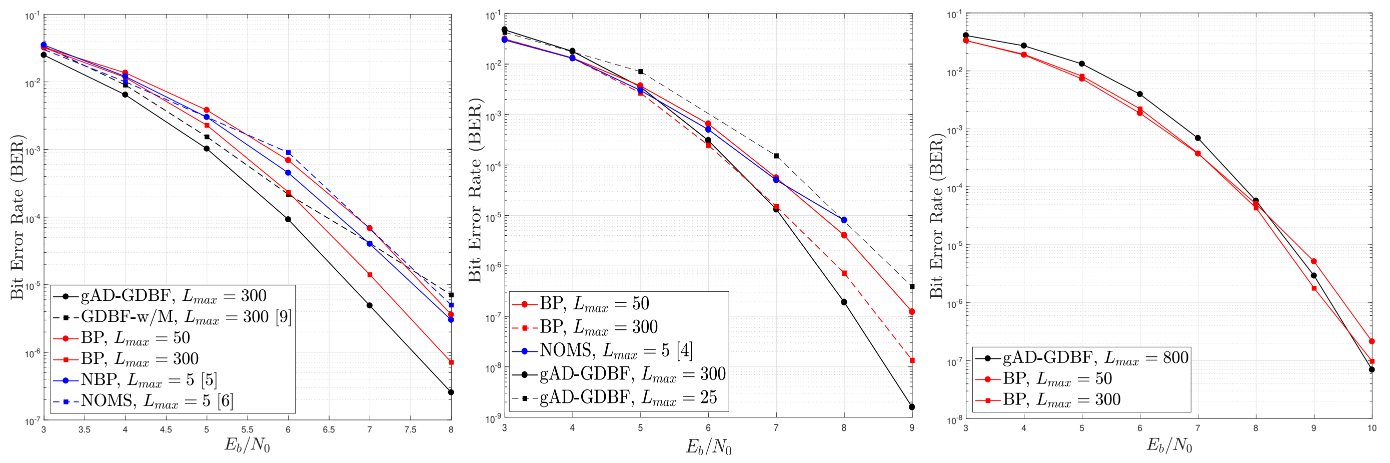
To compute the energy function of a variable, given in eq. (2), we need two floating point multiplications and additions, while the parity equation $s_j^{(\ell)}$, is performed with $|\mathcal{Q}(c_j)| - 1$ binary XOR operations. Thus, the total number of multiplications/additions per iteration is equal to $2N$, and we also need $E + N - M$ binary XOR operations. To calculate the minimum energy value we need $N - 1$ comparators, and additional N comparators to form the set $\mathcal{F}^{(\ell)}$. Note that we neglect the summation $S_i = \sum_j s_j^{(\ell)}$ as the majority of variables will have the maximal value of the summation $S_i = |P(v_i)|$ (all satisfied parity checks), and we can initialize S_i to $|P(v_i)|$ and subtract twice the number of unsatisfied checks for a small percentage of variables that have unsatisfied checks. Similar reasoning holds for the potential updates (eq. (6)), which are performed even less frequently. It follows that the density of the parity check matrix has a weaker influence on the complexity of the gAD-GDBF decoder, compared to the BP-based decoders.

In table I we present the number of operations required to perform a single iteration of the gAD-GDBF decoder for several BCH codes, assuming $|\mathcal{M}| = 6$. We also compare the complexity of the gAD-GDBF decoder with the BP and NBP decoders, based on the analysis from [14] and the NOMS algorithm, which complexity we estimate as in [8]. It can be observed that, compared to the BP decoder, the gAD-GDBF algorithm used on (63,45) BCH code requires more than 6 times less multipliers and adders, and for the BCH code (127,106) we observed more than 8 times complexity reduction.

III. Numerical results

The performance of the gAD-GDBF decoder is expressed in a form of the bit error rate (BER) dependency on the E_b/N_0 (energy per information bit to noise power spectral density ratio) are presented in Fig. 1 for several BCH codes. The optimized parameters of designed decoders and used parity check matrices are publicly available in [15].

We first verified the significance of the introduced improvements in the gAD-GDBF decoder, by showing its superiority compared to the GDBF-w/M decoder on (63,45) code. Additionally, we observed that the gAD-GDBF decoders, if run



(a) BCH (63,45) code (b) BCH (127,106) code (c) BCH (255,215) code

Fig. 1: Performance of the gAD-GDBF decoder on various BCH codes.

for $L_{\text{max}} = 300$ iterations on (63,45) and (127,106) codes provide gain close to 1.0 dB, compared to the BP decoder, run for $L_{\text{max}} = 50$ iterations. Interestingly, our decoders even outperform the BP decoder run for $L_{\text{max}} = 300$. It should be emphasized that increasing the number of iterations for the BP decoder beyond 300, does not improve the decoding, and the performance loss compared to the gAD-GDBF decoder cannot be reduced. Furthermore, we can see that the proposed decoder, run for 300 iterations, outperforms the NBP and NOMS decoders. These decoders are designed only for $L_{\text{max}} = 5$ iterations and it is unknown whether the loss, compared to the gAD-GDBF decoder, can be compensated by adding more iterations. Furthermore, we can see that the gAD-GDBF decoder, run for 25 iterations on (127,106) code, matches the performance of the NOMS decoder for $E_b/N_0 \approx 8$ dB, while the slopes of the BER curves indicate that the gAD-GDBF decoder will outperfom the NOMS decoder, for larger E_b/N_0 values. Finally, we can see that the gAD-GDBF decoder, run for $L_{\text{max}} = 800$ iterations on (255,215) code, for higher E_b/N_0, outperforms the BP decoder, run for $L_{\text{max}} = 50$ iterations and performs closely as the BP decoder with $L_{\text{max}} = 300$ iterations.

IV. Conclusion

Although deterministic, our decoder incorporates a key feature of probabilistic decoders – the ability to constantly improve the performance by using more decoding iterations. Furthermore, it represents a low-complexity solution and we can see that adding more iterations does not necessarily create insurmountable obstacles when it comes to the practical realizations. For example, on (255,215) BCH code, a single iteration of the gAD-GDBF decoder requires more than 17 times less multipliers, compared to the BP iteration, and even if we neglect other complex BP operations (like tanh), we still achieve higher coding gains than the BP decoder, with significantly lower complexity. For shorter codes, our design methods gave even better results and we were able to surpass the BP decoder with the same number of iterations.

References

[1] M. Shirvanimoghaddam, M. Mohammadi, R. Abbas, A. Minja, C. Yue, B. Matuz, G. Han, Z. Lin, W. Liu, Y. Li, S. Johnson, and B. Vucetic, "Short block-length codes for ultra-reliable low latency communications," *IEEE Comm. Mag.*, vol. 57, no. 2, pp. 130–137, Feb. 2019.

[2] V. Miloslavskaya and B. Vucetic, "Design of short polar codes for SCL decoding," *IEEE Trans. Commun.*, vol. 68, no. 11, pp. 6657–6668, Nov. 2020.

[3] E. Nachmani, Y. Be'ery, and D. Burshtein, "Learning to decode linear codes using deep learning," in *Proc. 54th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Sep. 2016.

[4] L. Lugosch and W. J. Gross, "Neural offset min-sum decoding," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, June 2017.

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

[6] E. Nachmani and Y. Be'ery, "Neural decoding with optimization of node activations," *IEEE Commun. Letters*, vol. 26, no. 11, pp. 2527–2531, Nov. 2022.

[7] K. Tian, C. Yue, C. She, Y. Li, and B. Vucetic, "A scalable graph neural network decoder for short block codes," 2022, [Online]. Available: https://arxiv.org/abs/2211.06962.

[8] S. Brkic, P. Ivanis, and B. Vasić, "Adaptive gradient descent bit-flipping diversity decoding," *IEEE Commun. Letters*, vol. 26, no. 10, pp. 2257–2261, Oct. 2022.

[9] V. Savin, "Gradient descent bit-flipping decoding with momentum," in *Proc. 2021 11th Inter. Symp. on Topics in Coding (ISTC)*, Aug. 2021.

[10] G. Sundararajan and E. Winstead, C.and Boutillon, "Noisy gradient descent bit-flip decoding for LDPC codes," *IEEE Trans. Commun.*, vol. 62, no. 10, pp. 3385–3400, Oct. 2014.

[11] T. Tithi, C. Winstead, and G. Sundararajan, "Decoding LDPC codes via noisy gradient descent bit-flipping with redecoding," 2015, [Online]. Available: http://arxiv.org/abs/1503.08913.

[12] T. Wadayama, K. Nakamura, M. Yagita, Y. Funahashi, S. Usami, and I. Takumi, "Gradient descent bit flipping algorithms for decoding LDPC codes," *IEEE Trans. Commun.*, vol. 58, no. 6, pp. 1610–1614, June 2010.

[13] O. A. Rasheed, P. Ivanis, and B. Vasić, "Fault-tolerant probabilistic gradient-descent bit flipping decoder," *IEEE Commun. Letters*, vol. 18, no. 9, pp. 1487–1490, Sep. 2014.

[14] G. Li, X. Yu, Y. Luo, and G. Wei, "A bottom-up design methodology of neural min-sum decoders for LDPC codes," *IET Communications*, vol. 17, no. 3, pp. 377–386, Mar. 2023.

[15] Generalized AD-GDBF decoder database. [Online]. Available: https://github.com/milojko94/gAD-GDBF