# Adaptive Gradient Descent Bit-Flipping Diversity Decoding

Srdan Brkic, *Member, IEEE*, Predrag Ivaniš *Senior Member, IEEE* and Bane Vasić *Fellow, IEEE*

***Abstract*—In this letter we propose a novel framework for designing decoders, for Low-Density Parity Check (LDPC) codes, that surpasses the frame error rate performance of Belief-Propagation (BP) decoding on binary symmetric channels. Its key component is the adaptation method, based on the genetic optimization algorithm, that is incorporated into the recently proposed Gradient Descent Bit-Flipping Decoding with Momentum (GDBF-w/M). We show that the resulting decoder outperforms all state-of-the-art probabilistic bit-flipping d ecoders a nd, additionally, it can be trained to perform beyond BP decoding, which is verified by numerical examples that include codes used in IEEE 802.3an and 5GNR standards. The proposed framework provides a systematic method for decoder optimization without requiring knowledge of trapping sets. Moreover, it is applicable to both regular and irregular LDPC codes.**

***Index Terms*—Belief-propagation, error-floors, g radient descent bit-flipping, g enetic a lgorithm, l ow-density parity-check codes.**

## I. Introduction

BELIEF propagation (BP) is the state-of-the-art algorithm used for decoding low-density parity-check (LDPC) codes, as it provides reasonably good decoding performance on a variety of communication channels. The BP decoder is also credited for LDPC codes approaching Shannon capacity on Gaussian noise channels and binary symmetric channels (BSCs). Its hardware-friendly versions are used in protocols that involve LDPC codes, like the fifth-generation standard for broadband cellular networks (5GNR), or IEEE 802.3an networks.

However, on short and medium length LDPC codes, the BP performance is far from the maximum-likelihood (ML) performance bounds. This behavior of the BP decoder is attributed to unavoidable dense structures in Tanner graphs of finite-length LDPC codes, referred to as trapping sets. Over the years, significant effort was made to identify such structures and to design more resilient decoders (e.g., see [1] and references therein). This led to the development of the Finite Alphabet Iterative Decoders (FAIDs), which surpass the BP on regular column weight-3 codes [2]. However, designing a FAID for higher column-weights or for irregular LDPC codes is challenging, as those codes contain a significantly larger variety and a larger number of trapping sets.

To circumvent this problem, recent attempts to build superior decoders are oriented towards incorporating artificial intelligence concepts into decoder design. Nachmani *et al.* showed in [3] that the BP decoder represents only a variant of sparse deep neural network (DNN) and that its performance can be improved via training and back-propagation weight updating. The idea was further elaborated by Lugosch and Gross [4] and especially by Xiao *et al.* [5], where recurrent quantified DNNs are used to design FAIDs that converge faster than the BP. In the most recent publication [6], the same authors showed that, for short regular LDPC codes, it is possible to build a diversity of complement DNN-based FAIDs. However, due to the complexity of the proposed DNN framework, it is uncertain if it can be extended to longer or higher column-weight codes. In another research direction, DNNs are not used to train the decoder, but rather as a part of the decoding process [7]–[9]. Although such decoding schemes can operate beyond BP, their complexity is high, and the majority of them are restricted to very short LDPC codes.

Until recently, bit-flipping decoders were not considered to be competitive with the BP decoding (and its simplifications). It has been reported that newly proposed solutions [10]–[14], that emerged from the gradient descent bit-flipping (GDBF) decoder, originally proposed by Wadayama *et al.* [15], in some cases, perform relatively close to the BP decoding on the BSC. The GDBF decoder tries to solve the ML decoding problem by using the gradient descent optimization. In each decoding iteration, the GDBF decoder flips bit values in order to minimize the ML objective function, i.e., bits with the lowest energy. However, being only an approximation of the ML decoder, the GDBF decoder frequently gets stuck at local optima (trapping sets). It is widely assumed that the optimal way to break a trapping set is to insert randomness into the decoding process, as the best bit-flipping decoders employ random generators [10]–[14]. It was shown in [16] that the probabilistic GDBF (PGDBF) decoder can approach even the ML decoding bound, if an unbounded number of iterations is allowed. For example, the Information Storage Bit Flipping (ISBF) decoder, proposed by Cui *et al.* [12], defines two flipping thresholds and flips only a portion of randomly chosen bits, with the energy beyond the thresholds. In the most recent publication [14], Savin added the momentum term to the PGDBF energy function and showed that (P)GDBF decoder with momentum ((P)GDBF-w/M) closely approaches the performance of the floating-point BP decoding.

In this letter we propose a novel framework for designing decoders on the BSC, that surpass the BP decoder in the error floor region. Our framework couples GDBF-w/M decoders with different parameters into a unified diversity decoder, while parameters of the component decoders are learned by

S. Brkic and P. Ivaniš are with the University of Belgrade, School of Electrical Engineering, 11000 Belgrade, Serbia (e-mails: srdjan.brkic@etf.rs, predrag.ivanis@etf.rs).

B. Vasić is with the University of Tucson, Department of ECE, Tucson, Arizona (e-mail: vasic@ece.arizona.edu).

employing genetic algorithm optimization. The constructed diversity decoder outperforms state-of-the-art PGDBF-w/M and ISBF decoders. More importantly, for the proposed framework, it is immaterial if an LDPC code is regular or irregular, low or high-variable degree or what the code rate is, and can be constructed without explicit knowledge of trapping sets. This - to the best of our knowledge - makes it unique. In addition, the diversity decoder is purely deterministic (it does not use random number generators), and for regular codes can be implemented by using only integer arithmetic.

## II. Adaptive GDBF Diversity Decoding

### A. State-of-the-art Decoder Description

Consider an arbitrary chosen LDPC code $(N, K)$, with a code rate $R = K/N$ and parity check matrix $\mathbf{H} = [h_{ji}]_{M\times N}$, used to increase reliability of transmission through the BSC with crossover probability $\alpha$. Each code bit (variable) $v_i$ ($N \geq i \geq 1$) is associated with a set of parity equations $\mathcal{P}(v_i) = \{j|h_{ji} = 1\}$, where the cardinality $\gamma_i = |\mathcal{P}(v_i)|$ represents the *weight* of the column, that corresponds to the variable $v_i$. Similarly, $\rho_j$ denotes weight (number of ones) of the $j$-th row of $\mathbf{H}$. If $\gamma_i = \gamma$, $\forall i$ and $\rho_j = \rho$ $\forall j$, the code is regular, otherwise we speak of an irregular code. In the rest of the manuscript, if not stated otherwise, we will assume regular codes. The output of the BSC (*channel vector*), is denoted by $\mathbf{y} = (y_1, y_2, \ldots, y_N)$, where $\mathbf{y} \in \{0, 1\}^N$.

The GDBF-w/M decoder, in each iteration, calculates the *inverse energy* of variables and flips the variables with the highest inverse energy. Let $\mathbf{x}^{(\ell)}$ be a vector of variable estimates from the $\ell$-th decoding iteration and corresponding syndrome is defined as $\mathbf{s}^{(\ell)} = \mathbf{x}^{(\ell)} \times \mathbf{H}^T$, where $T$ denotes matrix transposition. Initial variable estimates are received from the channel, i.e., $\mathbf{x}^{(0)} = \mathbf{y}$. Consider now the inverse energy associated with the $i$-th variable in the $\ell$-th iteration

$$E^{(\ell)}(v_i) = a(x_i^{(\ell)} \oplus y_i) + b \sum_{j\in\mathcal{P}(v_i)} s_j^{(\ell)} - m_{\ell_i}, \qquad (1)$$

where $a$ and $b$ are positive scaling constants[1], $\ell_i$ is the number of iterations, passed from the last flip of the variable $v_i$, while $\mathbf{m} = (m_1, \ldots, m_L)$ is a predefined *momentum vector* that contains non-negative integers sorted in non-increasing order [14], i.e., $m_1 \geq m_2 \ldots \geq m_{L'} \geq 0 = m_{L'+1} = \ldots = m_L$, where $L$ represents the maximal number of decoding iterations. Note that we limit the effect of momentum to at most $L'$ consecutive iterations, meaning $m_{\ell_i} = 0$ for all $\ell_i > L'$, while $\ell_i$ values are initialized to $L$ for all variables.

A flipping rule is the following

$$x_i^{(\ell+1)} = \begin{cases} \overline{x}_i^{(\ell)} & \text{if } E^{(\ell)}(v_i) = E_{\text{th}}^{(\ell)} \\ x_i^{(\ell)} & \text{otherwise,} \end{cases} \qquad (2)$$

where $\overline{x}$ represents complement of $x$ and

$$E_{\text{th}}^{(\ell)} = \max_{1\leq i\leq N} E^{(\ell)}(v_i). \qquad (3)$$

[1]In case of an irregular code it is possible to associate different scaling constants to variables with different weights.

Decoding halts when all parity checks are satisfied, i.e., $\mathbf{x}^{(\ell+1)} \times \mathbf{H}^T = \mathbf{0}$, or when the maximal number of iterations $L$ is reached – in which case the decoder *fails*.

In a probabilistic version of the algorithm, called the PGDBF-w/M decoder, variables with inverse energy $E_{\text{th}}^{(\ell)}$ are not flipped automatically, but with some predefined probability, that is channel and code specific.

To summarize, the GDBF-w/M decoder, $\mathbf{x}^{(\ell)} = \mathsf{Dec}(\mathbf{y})$, applied to its input $\mathbf{y}$ and producing the output $\mathbf{x}^{(\ell)}$ at the $\ell$-th iteration, is specified by the following parameters: the decoder input $\mathbf{y}$, the maximum number of iterations $L$, scaling values $a$ and $b$ and the momentum vector $\mathbf{m}$. The parameters are organized in a 5-tuple $(\mathbf{y}, L, a, b, \mathbf{m})$.

### B. Diversity Decoding and Parameters Adaptation

Consider a collection of $M_D$ decoders $\mathcal{D} = \{\mathsf{Dec}_1, \mathsf{Dec}_2, \ldots, \mathsf{Dec}_{M_D}\}$, wherein $\mathsf{Dec}_t$ is a component GDBF-w/M decoder with parameters $(\mathbf{z}_t, L_t, a_t, b_t, \mathbf{m}_t)$. The output of the $t$-th decoder is $\mathbf{x}_t = \mathsf{Dec}_t(\mathbf{z}_t)$ where $\mathbf{z}_t = (1 - r_t)\mathbf{x}_{t-1} + r_t\mathbf{y}$, wherein $r_t \in \{0, 1\}$ are binary "restart" switches (and $\mathbf{x}_0 = \mathbf{y}$). In other words, the component decoders are run sequentially, i.e., $\mathsf{Dec}_{t+1}$ is employed in the case of $\mathsf{Dec}_t$ failure, and the next component decoder has a choice of either: (i) using the codeword estimate/output of the previous decoder as an input ($r_t = 0$), or (ii) starting from the channel vector ($r_t = 1$). We refer to the above concatenation as an *adaptive diversity GDBF decoder (AD-GDBF)*, and denote it as $\mathcal{D}$. The values $r_t$, $a_t$, $b_t$, and $\mathbf{m}_t$ are subject to the numerical optimization, while in the rest of the manuscript we consider that $L_t$ is fixed and empirically chosen. Furthermore, we consider that the momentum length $L'$ is fixed for all the component decoders.

It is important to note that the restart switches play a critical role in the diversity decoder, and are motivated by findings reported in [11], [16], where it was shown that a probabilistic bit-flipping decoder, in some cases, provides a codeword estimate that contains significantly more errors than the channel output vector. The frequent momentum updates - while not random - may cause a similar behavior. We prevent this by allowing decoder restarts.

We next provide a back-of-the-envelope calculation of the number of possible decoder configurations. Let us bound the maximal momentum value by some positive integer $I$, i.e., $m_{\ell_i} \in \mathcal{S} = \{0, 1, \ldots, I\}$. Then, we form a set that collects all possible momentum vectors $\mathbf{m}$ as $\mathcal{R} = \{\mathbf{m}|m_i \in \mathcal{S} \wedge m_i \geq m_{i+1} \wedge m_i = 0, L \geq i > L'\}$, where according to the basic combinatorial analysis $|\mathcal{R}| = \binom{I+L'}{L'}$. Let us also, for the sake of simplicity of this calculation, assume that $a_t$ and $b_t$ are fixed for the entire diversity set, while we can optimize $M_D$ momentum vectors and $M_D - 1$ binary switches. It directly follows that the total number of realizations is $|\mathcal{R}|^{M_D} \times 2^{M_D-1}$. For example, if $M_D = 10$ and $L' = 5$, we can choose to design the decoder among $\approx 10^{20}$ diversity realizations. Note also that in [14] the momentum values were optimized through exhaustive search over a discretized set of values; however, in our decoder, due to increased number of diversity configurations, such search was not feasible.

## III. Decoder Selection Strategy

The goal is to optimize the diversity set $\mathcal{D}$, for a total number of iterations $L$ and the BSC crossover probability $\alpha$, in order to minimize frame error rate (FER), denoted by $\Phi(\mathcal{D}, \alpha)$,

$$\min_{\mathcal{D}} \Phi(\mathcal{D}, \alpha) \quad \text{s.t.} \sum_{t=1}^{M_D} L_t = L. \tag{4}$$

We propose that the optimization parameters are quantized to finite precision, in order to simplify the optimization process, as well as the actual realization of the decoder. For each optimization parameter, prior to optimization, we define a set of possible values. In the rest of the manuscript a set of possible values of a parameter $x$ is denoted as $\mathcal{L}(x)$; specially, $\mathcal{L}(\mathbf{m_t}) = \mathcal{R}$. Then, each optimization parameter can be encoded in a binary form, and the optimization problem (4) can be solved by the standard genetic algorithm (GA) [17].

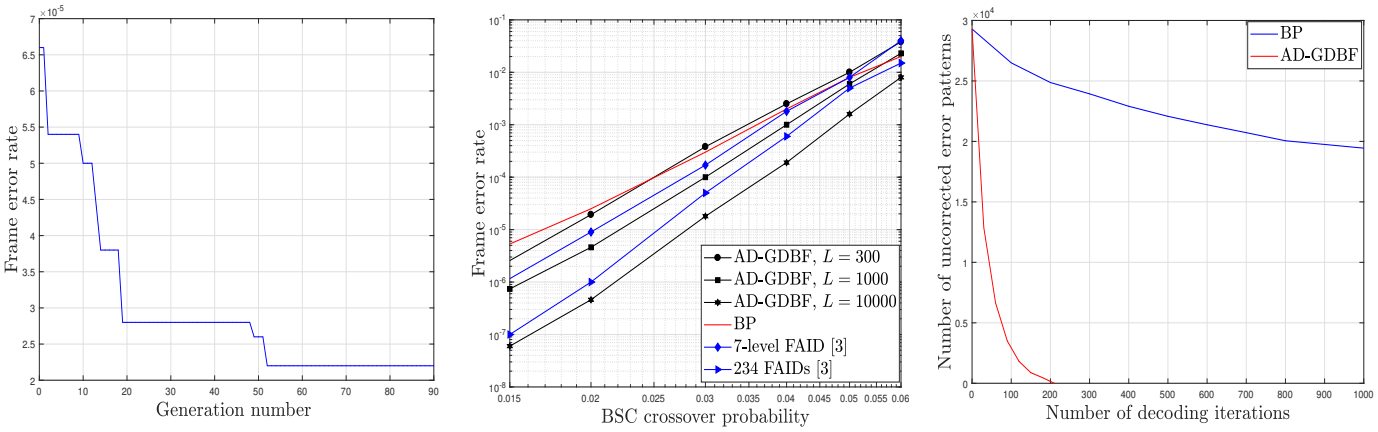
The first step of the optimization procedure is to collect error patterns, that will be used to train the decoder $\mathcal{D}$. We simply use the GDBF decoder with $a = 1$, $b = 1$ and $\mathbf{m} = \mathbf{0}$ (denoted by $\text{Dec}_0$) for $L_0$ iterations and perform Monte-Carlo simulation to obtain a set of uncorrectable error patterns $\mathcal{E}^{(0)}$. The cardinality of $\mathcal{E}^{(0)}$ is proportional to discrepancy between performance of $\text{Dec}_0$ and desired FER, which may be several orders of magnitude large and lead to training sets with sizes measured in tens of thousands patterns, in which case training all the component decoders jointly is computationally hungry. Instead, we propose to conduct training in several rounds. We denote the decoder, designed after the $k$-th training round, by $\mathcal{D}^{(k)} = \{\text{Dec}_1, \ldots, \text{Dec}_k\}$, and a pattern set correctable by this decoder $\mathcal{E}_{\mathcal{D}^{(k)}}$. Then, a set of uncorrectable patterns, used in the $(k+1)$-th training round, is $\mathcal{E}^{(k)} = \mathcal{E}^{(k-1)} \setminus \mathcal{E}_{\mathcal{D}^{(k)}}$. When, after a training round $k$, the number of uncorrectable patterns is below a predefined threshold $N_{\text{up}}$, i.e, $|\mathcal{E}^{(k)}| < N_{\text{up}}$, in the $(k+1)$-th training round, all the remaining component decoders $\{\text{D}_{k+1}, \ldots, \text{D}_{M_D}\}$ are jointly optimized. Given the fact that the initial set $\mathcal{E}^{(0)}$ contains limited portion of patterns uncorrectable with the GDBF decoder, there is a strong possibility that learning to decode all patterns from $\mathcal{E}^{(0)}$ leads to overfitting, and consequently the decoder designed in such way may not be optimal. This means that a relatively high positive number $N_{\text{low}}$ needs to be chosen and only if $|\mathcal{E}^{(k)}| \geq N_{\text{low}}$ the decoder $\mathcal{D}^{(k)}$ represents the solution. If $|\mathcal{E}^{(k)}| < N_{\text{low}}$, the set of uncorrectable patterns $\mathcal{E}^{(k-1)}$ needs to be expanded by performing Monte-Carlo simulation on the decoder $\mathcal{D}^{(k-1)}$. The optimization strategy is expressed in Algorithm 1.

We next design the AD-GDBF decoder for a famous quasi-cyclic code with length $N = 155$ and code rate $R = 0.4$, called the Tanner code. We collected 11250 error patterns, uncorrectable for the GDBF decoder, for $\alpha = 0.02$ and obtained $\Phi(\text{Dec}_0, 0.02) \approx 0.0045$. Then, by using the GA, we jointly trained $M_D = 10$ component decoders, with $L_t = 30$, $\forall t$. We randomly chose 20 binary chromosomes, with lengths calculated in Algorithm 1, to represent the initial population. In each iteration of the GA, called a generation, the number of corrected patterns from the training set is calculated and used as fitness associated with an individual chromosome. In Fig. 1 (a) we show how FER changes after each generation. Interestingly, even among randomly chosen decoders (that correspond to the initial population), good decoders can be found and FER before the optimization is approximately $6.5 \times 10^{-5}$. We observed that by increasing the diversity size $M_D$ with randomly chosen component decoders, FER is reduced. It follows that frequent parameter changes manifest as random perturbations of the inverse energy, and our decoder behaves similarly as probabilistic decoders, without employment of random generators. The GA reduces the number of iterations required to achieve the desired FER or, in the above example, reduces FER (approximately three times) for the same number of iterations. We achieved the same FER with random decoder selection only if 600 iterations were allowed. In Fig. 1 (b) we show the performance of the AD-GDBF decoders, designed for different numbers of iterations.

It is beyond the scope of this letter to provide proof that the performance of the AD-GDBF decoder approaches the ML decoding bound, but experimental evidence and our numerical results reveal that, after a sufficiently large number of iterations, the decoder always converges to a valid codeword - an observation that warrants further investigation. We strengthen our claim with an example, in which we show that the AD-GDBF decoder can be used to provide guaranteed error correction on the Tanner code. From [2] it is known that the best FAID can correct all weight-5 error patterns on the Tanner code, but fails on 29294 specific weight-6 patterns. We have trained our decoder to correct all weight-6 patterns, uncorrectable by the FAID, with only 210 iterations, as shown in Fig. 1 (c). The BP decoder, on the other hand, even with

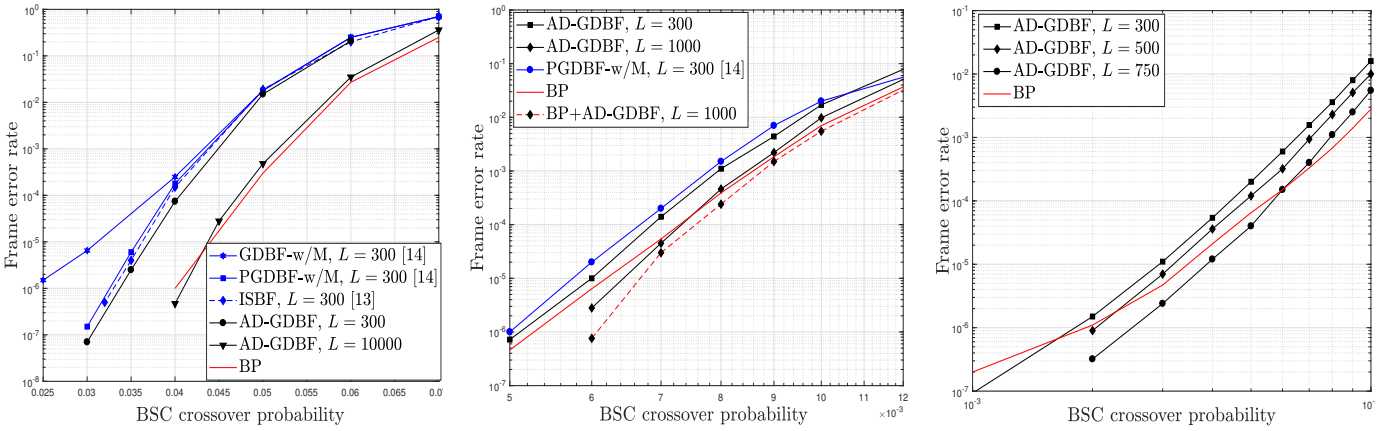
**Algorithm 1 Decoder selection strategy**

**Input:** $M_D$, $\mathcal{R}$, $\mathcal{L}(a_t)$ $\mathcal{L}(b_t)$, $L_t$, $N_{\text{up}}$, $N_{\text{low}}$, $\alpha$
$\text{Dec}_0 \leftarrow (\mathbf{y}, L_0, \mathbf{a}_0 = \mathbf{1}, \mathbf{b}_0 = \mathbf{1}, \mathbf{m}_0 = \mathbf{0})$
Monte-Carlo simulation on $\text{Dec}_0$ and $\alpha$ to obtain $\mathcal{E}^{(0)}$ $k \leftarrow 0$, $\delta \leftarrow 1$
**while** $k < M_D$ **do**
  $i \leftarrow k$
  **if** $|\mathcal{E}^{(i)}| > N_{\text{up}}$ **then**
    $k \leftarrow i + 1$
  **else**
    $k \leftarrow M_D$
  **end if**
  $\Delta = (k - i)(\lceil \log_2 |\mathcal{R}| \rceil + 1) - \delta$, $\delta \leftarrow 0$
  **Number of binary genes in a chromosome:**
  $N_b = \Delta + \sum_{j=i}^{k} \left[ \lceil \log_2 |\mathcal{L}(a_j)| \rceil + \lceil \log_2 |\mathcal{L}(b_j)| \rceil \right]$
  **GA fitness function:** maximize $|\mathcal{E}_{\mathcal{D}^{(k)}}|$
  **Perform GA to obtain** $\{\text{Dec}_{i+1}, \ldots, \text{Dec}_k\}$
  $\mathcal{E}^{(k)} = \mathcal{E}^{(t)} \setminus \mathcal{E}_{\mathcal{D}^{(k)}}$
  **if** $|\mathcal{E}^{(k)}| < N_{\text{low}}$ **then**
    Monte-Carlo simulation on $\mathcal{D}^{(i)}$ and $\alpha$ to obtain $\mathcal{E}_{\text{add}}$
    $\mathcal{E}^{(i)} \leftarrow \mathcal{E}^{(i)} \cup \mathcal{E}_{\text{add}}$, $k \leftarrow i$
  **end if**
**end while**
**Output:** $\mathcal{D}^{(M_D)} = \{\text{Dec}_1, \ldots, \text{Dec}_{M_D}\}$




(a) The GA optimization (b) Frame error rate (c) Uncorrectable weight-6 error patterns

Fig. 1: Performance of the AD-GDBF on the Tanner code.




(a) iRISC code ($R = 0.5, \gamma = 4$) (b) IEEE 802.3an ($R = 0.84, \gamma = 6$) (c) 5GNR irregular code ($R = 0.75$)

Fig. 2: Performance of the AD-GDBF decoder on various LDPC codes.

1000 iterations, cannot correct the majority of patterns.

## IV. Numerical results

In this section, we present numerical results obtained by Monte-Carlo simulation. For each considered LDPC code and the maximum number of iterations $L$, we choose relatively low BSC crossover probability $\alpha$ and design the AD-GDBF decoder, assuming $N_{\text{low}} = 50$, $N_{\text{up}} = 1000$ and $L_0 = 30$. For regular codes, we have $\mathcal{L}(a_t) = \mathcal{L}(b_t) = \{1, 2, 3, 4\}$, $L' = 4$ and $I = 5$, while the number of iterations $L_t$ is code dependent, and chosen empirically, with the constraint $L_1 \geq L_0$, to ensure that with high probability $\text{Dec}_1$ corrects all the error patterns correctable with $\text{Dec}_0$. Typically, we adapt momentum and scaling values after every 30 to 50 iterations. For the irregular code, the scaling values are rational numbers, in a range between 1 and 7, chosen from a set with 64 different values. Due to space limitations, we here do not give exact learned momentum and scaling parameter values. Instead, they are publicly available together with parity check matrices in [18]. We run flooding scheduled BP decoders for 100 iterations, given the fact that further increasing the number of iterations will not significantly affect FER.

In Fig. 2 (a) we examine performance of the AD-GDBF decoder on a quasi-cyclic (QC) code, called the iRISC code, with $\gamma = 4$, code rate $R = 0.5$ and length $N = 1296$. We show superiority of the AD-GDBF decoder compared to (P)GDBF-w/M and ISBF decoders, for the same number of decoding iterations considered in [13] and [14] ($L = 300$). For the case of $L = 300$ iterations, the presented AD-GDBF performance curve is obtained by combining two decoders, one designed for the crossover probability $\alpha = 0.03$ and another for other operational points, marked as black circles in Fig. 2 (a).

In Fig. 2 (b) we show the AD-GDBF decoder performance on the code proposed in IEEE 802.3an standard ($R = 0.84$, $N = 2048$) and again illustrate its superiority compared to the PGDBF-w/M decoders and its possibility to surpass the BP decoder. To show adaptability of our framework to certain error-patterns, we have also designed a different AD-GDBF decoder, which is employed only when the BP decoder fails to correct an error pattern. We witness that applying the AD-GDBF decoder as the post-processor for $L = 1000$ iterations can lower the error-floor of the BP decoder by an order of magnitude. Finally, in Fig. 2 (c), we show the performance of the 5GNR code with $N = 1276$ and $R = 0.75$ and validate the proposed framework on an irregular code.

TABLE I: Complexities of different decoders.

| | AD-GDBF | OMS | ISBF |
|---|---|---|---|
| Additions | $(2+\beta)NL$ | $(2\gamma N+\rho M)L$ | $>NL$ |
| MC | $(N-1)L$ | $M(2\rho-3)L$ | $>NL$ |
| EC | $NL$ | $M\rho L$ | $>NL$ |
| XOR | $(N+M(\rho-1))L$ | $M(2\rho-1)L$ | $(N+M(\rho-1))L$ |

## V. Conclusion

In this letter we proposed a decoding framework, capable of surpassing the BP decoder in the error-floor r egion. We verified the performance of our decoder on numerous examples and showed its effectiveness, especially on codes with higher code rates. The main feature of the proposed solution is the possibility to adapt to error patterns specific to a certain LDPC code. Thus, quality of the decoder is mostly influenced by error pattern sets, used to train the decoder.

## Appendix (A note on Complexity)

Here we give an estimate of the complexity of the AD-GDBF decoder, when applied to regular codes, and compare it with the fixed p oint O ffset M in-Sum ( OMS) a nd t he ISBF decoders (Table I). The AD-GDBF decoder employs integer adders, integer comparators and 2-input XORs. Note that calculating the inverse energy requires two integer additions per code bit, while the energy threshold can be obtained by using $N-1$ magnitude comparators (MC). Finally, selecting bits for flipping c an b e d one w ith $N$ e quality comparators (EC). It is worth noting that multiplications are unnecessary, because scaling with 2 and 4 can be accomplished through decimal point shifting, whereas multiplication of an integer $x$ by 3 can be accomplished through the use of an adder ($x+2x$). We denote by $\beta$, $0 \leq \beta \leq 2$, the number of such adders per decoding iteration. We neglect i) operations required to calculate the values $\ell_i$, $1 \leq i \leq N$, defined in Section II-A, and ii) counters of unsatisfied checks, as the majority of variables will have all satisfied checks. We estimate the complexity of the OMS decoder by counting operations listed in [19].

A single AD-GDBF decoder iteration requires approximately $3(\rho-1)M/N$ times fewer comparators and $(2\gamma+\rho M/N)/(2+\beta)$ times fewer adders, than an OMS decoding iteration. For the Tanner code, we designed the AD-GDBF decoder that, if performs 300 iterations (with $\beta=0.9$), requires the same number of adders, as the OMS decoder that performs 97 iterations, and the same number of comparators, as the OMS decoder that performs only 83 iterations. We also see that the AD-GDBF decoder, designed for the IEEE 802.3an code, that performs 1000 iterations (with $\beta=0.35$), is superior in the error-floor region compared to the BP (performs 100 iterations), while at the same time has the complexity of the OMS decoder with approximately 140 iterations.

The complexity of the AD-GDBF decoder designed for the code given in Fig. 2 (a) is the same as the complexity of the GDBF-w/M decoder, as scaling is performed without using additional adders [18]. Furthermore, the PGDBF-w/M decoder requires additional random number generators (RNGs), which means that it is more complex than the AD-GDBF decoder. The ISBF decoder requires fewer additions, compared to the AD-GDBF decoder, but more comparators and additional RNGs.


## Acknowledgement

Srdan Brkic and Predrag Ivaniš acknowledge the support of the Science Fund of the Republic of Serbia, grant No 7750284, Hybrid Integrated Satellite and Terrestrial Access Network - hi-STAR. Bane Vasić acknowledges the support of NSF under grants CCF-1855879, CCF-2106189, CCSS-2027844, CCSS-2052751 and CCF-2100013 and NASA through the Strategic University Research Partnerships (SURP) program. Bane Vasić has disclosed an outside interest in Codelucida to the University of Arizona.


## References


[1] N. Raveendran, D. Declercq, and B. Vasić, "A sub-graph expansion-contraction method for error floor computation," *IEEE Trans. Commun.*, vol. 68, no. 7, pp. 3984–3995, July 2020.

[2] D. Declercq, B. Vasić, S. Planjery, and E. Li, "Finite alphabet iterative decoders–part II: Towards guaranteed error correction of LDPC codes via iterative decoder diversity," *IEEE Trans. Commun.*, vol. 61, no. 10, pp. 4046–4057, Oct. 2013.

[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] X. Xiao, B. Vasić, R. Tandon, and S. Lin, "Designing finite alphabet iterative decoders of LDPC codes via recurrent quantized neural networks," *IEEE Trans. Commun.*, vol. 68, no. 7, pp. 3963–3974, July 2020.

[6] X. Xiao, N. Raveendran, B. Vasić, S. Lin, and R. Tandon, "FAID diversity via neural networks," in *Proc. 2021 11th Inter. Symp. on Topics in Coding (ISTC)*, Aug. 2021.

[7] A. Buchberger, C. Hager, H. Pfistery, L. Schmalenz, and A. G. Amat, "Learned decimation for neural belief propagation decoders," in *Proc. ICASSP 2021 - 2021 IEEE Inter. Conf. on Acoustics, Speech and Sig. Process. (ICASSP)*, June 2021.

[8] F. Liang, C. Shen, and F. Wu, "An iterative BP-CNN architecture for channel decoding," *IEEE Journal of Select. Topics in Sig. Process.*, vol. 12, no. 1, pp. 144–159, Feb. 2018.

[9] J. He, "A deep learning-aided post-processing scheme to lower the error floor of LDPC codes," in *Proc. 2020 IEEE 20th Inter. Conf. on Commun. Technology (ICCT)*, Oct. 2020.

[10] 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.

[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] H. Cui, J. Lin, and Z. Wang, "An improved gradient descent bit-flipping decoder for LDPC codes," *IEEE Trans. Circuits and Systems I: Regular Papers*, vol. 66, no. 8, pp. 3188–3200, Aug. 2019.

[13] ——, "Information storage bit-flipping decoder for LDPC codes," *IEEE Trans. Very Large Scale Integ. Syst.*, vol. 28, no. 11, pp. 2464–2468, Nov. 2020.

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

[15] 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.

[16] B. Vasić, P. Ivanis, D. Declercq, and K. LeTrung, "Approaching maximum likelihood performance of LDPC codes by stochastic resonance in noisy iterative decoders," in *Proc. 2016 Inform. Theory and Applications Workshop (ITA 2016)*, Feb. 2016.

[17] M. Melanie, *An Introduction to Genetic Algorithms*. Cambridge, Massachusetts: The MIT Pres, 1996.

[18] "Database of codes and decoders for analysis of AD-GDBF," *https://github.com/brka05/AD-GDBF-decoders*.

[19] V. Petrovic, M. Markovic, D. E. Mezeni, L. Saranovac, and A. Radosevic, "Flexible high throughput QC-LDPC decoder with perfect pipeline conflicts resolution and efficient hardware utilization," *IEEE Trans. Circuits and Systems–I: Regular papers*, vol. 67, no. 12, pp. 5454–5467, Dec. 2020.