Gradient Descent Bit-Flipping Decoding with momentum

Valentin Savin

Gradient Descent Bit-Flipping Decoding with Momentum

Valentin Savin, CEA-LETI, Université Grenoble Alpes, France (valentin.savin@cea.fr)

***Abstract*—In this paper, we propose a Gradient Descent Bit-Flipping (GDBF) decoding with momentum, which considers past updates to provide inertia to the decoding process. We show that GDBF or randomized GDBF decoders with momentum may closely approach the floating-point Belief-Propagation decoding performance, and even outperform it in the error-floor region, especially for graphs with high connectivity degree.**

I. Introduction

Bit-Flipping (BF) is the simplest form of iterative decoding for codes defined by bipartite graphs, requiring only one 1-bit message per graph node, and per iteration. Compared to message-passing (MP) decoders – exchanging multi-bit *extrinsic messages* along the graph edges – BF yields a significant reduction of the computational time and space complexity. As one may expect, this complexity reduction comes at the price of a significant degradation of the error correction performance. BF decoding was already introduced by Gallager in his seminal paper [1], as a very first example of simple, but poor performance approach to iterative decoding. Probabilistic decoding came then into play, latter reformulated in terms of Belief-Propagation (BP), paving the way for many MP decoders subsequently proposed in the literature. In a sense, the original BF decoding has been relegated to a secondary (and for many years almost inexistent) role.

However, the emergence of massive data rate communications over the last few years, has motivated a renew of interest for BF or BF-based decoding, seen as a possible approach towards meeting the very stringent requirements in terms of throughput/latency and energy efficiency. It is worth mentioning here the reformulation of BF decoding as a gradient descent solution to an optimization problem, proposed in [2]. The corresponding decoding algorithm, referred to as Gradient Descent Bit-Flipping (GDBF), is appealing in practice due to its simplicity, and since it answers an optimization problem that may, in principle, yield a solution to the ML decoding problem (see Section II). However, its main drawback is that it suffers from many local optimum traps, due to the high non-linearity of the objective function. Therefore, even if GDBF performs significantly better than the original BF decoding proposed by Gallager, or other variants of the BF decoding proposed in the literature, its performance is still behind that of more powerful MP decoders, such as BP or Min-Sum (MS).

The current solution to further improving the GDBF decoding performance relies on randomization. To some extent, the approach is reminiscent of noisy decoders, intensively investigated in the literature over the last years [3]–[6]. The difference between noisy and randomized decoders is that in the former case, noise is an external aggression that perturbs the decoding operations, while in the latter case, noise (or strictly speaking, randomness) has desired statistical properties and is an integral part of decoding operations. Moreover, it has been recently shown in the literature that randomness allows GDBF decoders to escape from local optima, thus improving their error correction performance. Randomized versions of the GDBF decoder – *e.g.*, Probabilistic GDBF (PGDBF) [7] and Noisy GDBF (NGDBF) [8] – have been shown to considerably narrow the gap to more powerful MP decoders. It is worth noting that both PGDBF and NGDBF introduce randomness in the bit flipping rule, and they carry strong similarities to each other. The PGDBF is however a hard-decision decoding algorithm, thus it mainly applies to the binary symmetric channel (BSC) model, while NGDBF applies to more general soft-output channels.

In this paper, we investigate alternative approaches to further improve the GDBF decoding performance, inspired by techniques used in gradient descent optimization. We propose a GDBF decoding with momentum, which considers past updates to provide inertia to the decoding process. We show that GDBF or randomized GDBF decoders with momentum may closely approach the floating-point BP decoding performance, and even outperform it in the error-floor region, especially for graphs with high degree of connectivity.

II. Randomized GDBF Decoders

We consider an LDPC code defined by a bipartite (Tanner) graph H, with N variable-nodes (corresponding to coded bits) and M check-nodes (corresponding to parity-check equations). We denote by $H(n)$ the set of check-nodes connected to a variable-node $n = 1, \ldots, N$, and by $H(m)$ the set of variable-nodes connected to a check-node $m = 1, \ldots, M$. We assume either a binary symmetric channel (BSC), or a binary-input additive white Gaussian noise (AWGN) channel. For simplicity with shall assume that in both cases the input alphabet of the channel is $\{-1, +1\}$, thus transmitted codewords take values in $\{-1, +1\}^N$ rather than $\{0, 1\}^N$. The maximum likelihood (ML) decoding is equivalent to finding a codeword $\underline{\boldsymbol{x}} = (x_1, \ldots, x_N)$, having the maximum correlation with the received word $\underline{\boldsymbol{y}} = (y_1, \ldots, y_N)$. Let $E(\underline{\boldsymbol{x}})$ be the objective function, also referred to as *energy function*, defined by:

$$E(\underline{\boldsymbol{x}}) \triangleq \alpha \sum_{n=1}^{N} x_n y_n + \sum_{m=1}^{M} \prod_{n \in H(m)} x_n, \;\; \forall \underline{\boldsymbol{x}} \in \{-1, +1\}^N \quad (1)$$

This work was partially supported by the French Agence Nationale de la Recherche (ANR), under grant number ANR-15-CE25-0006 (NAND project), and by the IPCEI programme of the European Commission, and the French Minister of Finance through DGE organization.

The first term of the objective function is the correlation between $\underline{\boldsymbol{x}}$ and $\underline{\boldsymbol{y}}$ (omitting the multiplicative coefficient α), while the second term is the sum of the (bipolar) syndromes of $\underline{\boldsymbol{x}}$. The multiplicative coefficient $\alpha > 0$ controls the contribution of the correlation term to the objective function. Clearly, the second term is maximized (equal to M) for any codeword $\underline{\boldsymbol{x}}$. Hence, if the vector $\underline{\boldsymbol{x}}$ maximizing $E(\underline{\boldsymbol{x}})$ is a codeword, it is necessarily the ML decoding solution.

The approach in [2] is to maximize $E(\underline{\boldsymbol{x}})$ (or minimize $-E(\underline{\boldsymbol{x}})$) by using a variant of the gradient descent method, known as coordinate descent: an iterative algorithm that successively minimizes along coordinate directions (x_n). It turns out that the corresponding decoding algorithm is a variant of the BF decoding, referred to as Gradient Descent BF (GDBF). It consists of flipping the bits with lowest *local energy* values, where the local energy of a bit x_n, denoted by $E_n(\underline{\boldsymbol{x}})$, is defined as:

$$E_n(\underline{\boldsymbol{x}}) \triangleq \alpha x_n y_n + \sum_{m\in H(n)} \prod_{n'\in H(m)} x'_n \tag{2}$$

We shall simply denote $E_n \triangleq E_n(\underline{\boldsymbol{x}})$, when no confusion is possible. Thus, at each decoding iteration, the set of bit-flips is given by

$$\mathcal{F} \triangleq \{x_\nu \mid E_\nu \leq E_{th}\}, \tag{3}$$

where E_{th} is a threshold value, referred to as *inversion threshold*. For the BSC model, it has been proposed in [7] to use $E_{\text{th}} = E_{\text{min}} \triangleq \min_{n=1,\ldots,N} E_n$, meaning that the set $\mathcal{F}$ contains the bits minimizing the local energy value (at each iteration). In this work, we use $E_{\text{th}} = E_{\text{min}} + \delta$, where δ is a predetermined value. We shall use $\delta = 0$ for the BSC (or discrete-output channels), but $\delta > 0$ for the AWGN channel (or continuous-output channels). According to the above description, GDBF is a *multi-bit flip* decoder, as multiple bits can be flipped at each iteration.

To escape from local minima, the PGDBF decoder [7] integrates a random perturbation of the bit-flip rule, consisting of flipping each bit $x_\nu \in \mathcal{F}$ with some probability $p < 1$ (the value of p is determined empirically). The PGDBF decoder has been initially proposed and investigated for the BSC, but in this work we extend its use to both BSC and AWGN channels.

We note that for the AWGN channel, an alternative approach is based on the NGDBF decoder [8], which uses additive white Gaussian noise to perturb the local energy values (prior to determining the bit-flip set $\mathcal{F}$).

III. Randomized GDBF Decoders with Momentum

Gradient descent with momentum has first been introduced in [9], as a method to accelerate the convergence rate of the gradient descent algorithm, and several extensions have been subsequently proposed in the literature, with different choices for the momentum parameter (see [10] and references therein). Momentum is also used for stochastic gradient descent, where the objective function is learned through samples in a training data set. It is one of the simplest extensions to gradient descent that has been successfully used for decades in the training of artificial neural networks [11].

Gradient descent with momentum remembers the update $\Delta x_n^{(\ell)} \triangleq x_n^{(\ell)} - x_n^{(\ell-1)}$ at each iteration ℓ, and determines the next update as a linear combination of the gradient and the previous update. This pushes the next update into the same direction as the previous update. Due to the discrete nature of our optimization problem (since $x_n^{(\ell)} \in \{\pm 1\}$), pushing in the same direction amounts to reducing the chances of a bit x_n being flipped back again, after a bit-flip occurred. To do so, we add a *momentum term* to the local energy value of a bit x_n, as follows.

$$E_n \triangleq \alpha x_n y_n + \sum_{m\in H(n)} \prod_{n'\in H(m)} x'_n + \rho(l_n), \tag{4}$$

where $l_n \geq 1$ is equal to the number of iterations since the last flip of x_n ($l_n = 1$ if x_n has been flipped at the previous iteration, $l_n = 2$ if x_n has been flipped two iterations ago, and so on). We shall assume that the momentum *lasts for L iterations*, meaning that $\rho(l_n) = 0$ if $l_n > L$. Hence, we define ρ as a vector of L non-increasing positive values:

$$\rho \triangleq [\,\rho(1) \geq \rho(2) \geq \cdots \geq \rho(L) > 0\,] \tag{5}$$

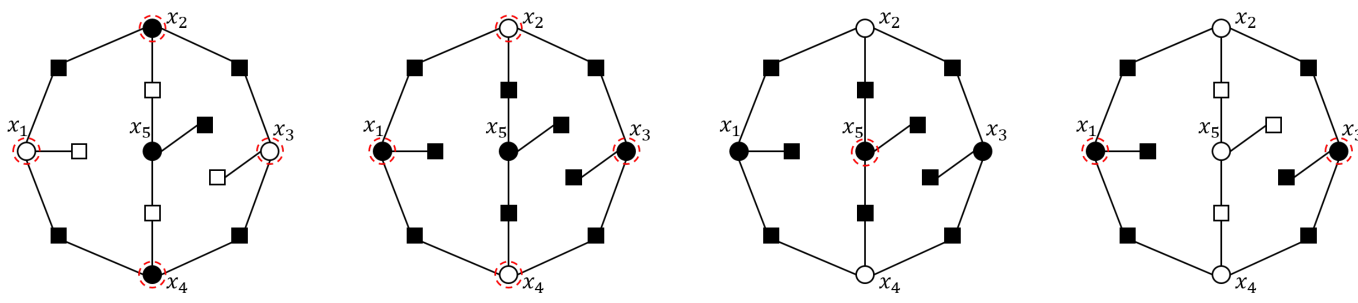
and shall further set $\rho(L+1) \triangleq 0$. Since there is no momentum when the decoding starts, l_n values are initialized to $L + 1$. The l_n value is incremented by 1 at each iteration (without exceeding the maximum $L + 1$ value), and set to 0 each time the bit x_n is flipped (such that it is incremented to 1 the next iteration).

The PGDBF decoding with momentum is described in Algorithm 1 (according to our assumption, the BSC has in-

Algorithm 1 PGDBF decoding with momentum (w/M)

```
Input:  $\underline{\boldsymbol{y}} = (y_1, \ldots, y_N) \in \mathbb{R}^N$        ▷ received word
Output: $\underline{\boldsymbol{x}} = (x_1, \ldots, x_N) \in \{-1, +1\}^N$     ▷ estimated codeword
Parameters: $\alpha > 0$ (correlation coef.), $p \in ]0, 1]$ (bit-flip probability)
            $\delta \geq 0$ (defines inversion threshold), $\rho$ (momentum)

 1: for all $n = 1, \ldots, N$ do                          ▷ initialization
 2:     $x_n = \text{sign}(y_n)$;
 3:     $l_n = L + 1$;
 4: end for
 5: for all Iter $= 1, \ldots,$ Iter$_{\text{max}}$ do                 ▷ iteration loop
 6:     for all $m = 1, \ldots, M$ do $c_m = \prod_{n\in H(m)} x_n$;   ▷ syndrome
 7:     if $c_m = 1$, $\forall m = 1, \ldots, M$ then exit the iteration loop;
 8:     for all $n = 1, \ldots, N$ do            ▷ local energy computation
 9:         $l_n = \min(l_n, L) + 1$;
10:         $E_n = \alpha x_n y_n + \sum_{m\in H(n)} c_m + \rho(l_n)$;
11:     end for
12:     $E_{\text{th}} = \min_{n=1,\ldots,N} E_n + \delta$;               ▷ inversion threshold
13:     for all $n = 1, \ldots, N$ do                           ▷ bit-flipping
14:         if $E_n \leq E_{\text{th}}$ and rand() $< p$ then
15:             $x_n = -x_n$;
16:             $l_n = 0$;
17:         end if
18:     end for
19: end for
```



(a) $\ell = 1$, GDBF and GDBF-w/M (b) $\ell = 2$, GDBF (c) $\ell = 2$, GDBF-w/M (d) $\ell = 3$, GDBF-w/M

Figure 1. Trajectories of GDBF and GDBF-w/M decoders, for an error pattern corresponding to a (5,5) trapping set

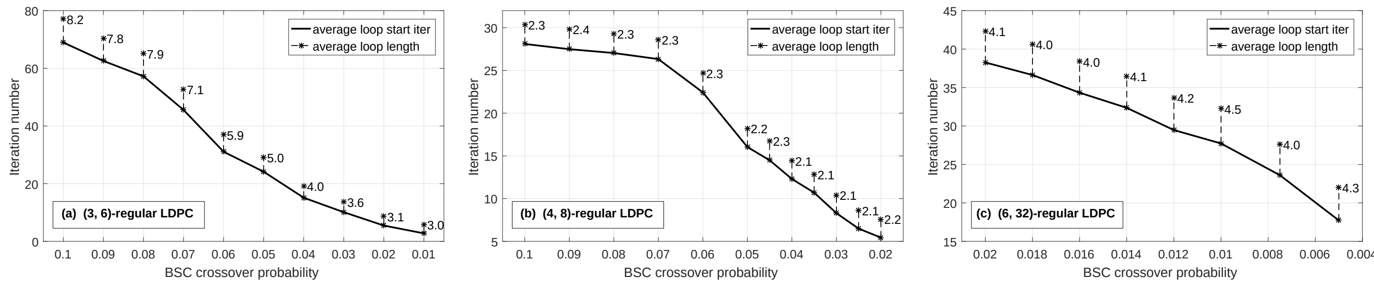


Figure 2. Average loop starting iteration and average loop length (regular LDPC codes, BSC)

put/output alphabet $\{-1, +1\}$). The parameters of the decoder are $\alpha > 0$ (the correlation coefficient), $p \in]0, 1]$ (the bit-flip probability), $\delta \geq 0$ (used to define the inversion threshold, which is set to 0 for the BSC), and ρ (the momentum). GDBF with momentum can be seen as a particular case, by taking the bit-flip probability parameter $p = 1$.

Escaping attractors: To illustrate the inertial effect of the GDBF-w/M decoder, we consider in Fig. 1 the decoding of an error pattern corresponding to a $(5, 5)$ trapping set of an LDPC code with variable-nodes of degree 3 (for a comparison with the PGDBF decoder, see the similar analysis in [7, Fig. 4]). Fig. 1(a) illustrates the error pattern, where variable-nodes are depicted as circles and check-nodes as squares, with full markers corresponding to a binary one (or bipolar -1, using our previous convention), and empty markers to binary zero (or bipolar $+1$). The iteration number is denoted as ℓ. In the first iteration, variable-nodes $x_1, \cdots, x_4$ have minimum local energy $E_n = 0$, $\forall n = 1, \ldots, 4$. Therefore, these nodes are flipped in both GDBF and GDBF-w/M decoders, which is indicated in the figure by the surrounding dashed red circles. In the second iteration, shown in Fig 1(b) for the GDBF decoder, the minimum local energy is obtained again for the same variable-nodes, since $E_n = -4$, $\forall n = 1, \ldots, 4$. Hence, these nodes are flipped back, and the GDBF decoder cannot converge, as it will repeat again and again the same operations. Fig 1(c) shows the second iteration for the GDBF-w/M decoder. In this case a momentum term is added to the local energy value of the bits that have been flipped in the first iteration. We will assume here that the momentum lasts only one iteration, and $\rho(1) = 3$. We get $E_n = -1$, $\forall n = 1, \ldots, 4$, and the minimum local energy is obtained for $E_5 = -2$. Thus, the momentum term prevented the variable-nodes $x_1, \ldots, x_4$ being flipped back again, and a new variable-node, namely x_5 is flipped in the second iteration. In the third iteration, shown in Fig 1(d), we get $E_1 = E_3 = -4$, $E_2 = E_4 = -2$, and $E_5 = 3$ (where we took into account the momentum term added to variable-node x_5). Variable-nodes x_1 and x_3 are then flipped, leading to successful decoding.

Momentum optimization: To determine the *length* (L) of the momentum, we start by investigating the behavior of the conventional GDBF decoder. Let us assume for the moment that the decoder is run for a possibly infinite number of iterations, and it only stops if the syndrome check condition is satisfied, meaning that the estimated $\underline{\boldsymbol{x}}$ is a codeword. We shall refer to the $\underline{\boldsymbol{x}}$ vector as the *state* of the decoder. Clearly, the GDBF decoder is completely deterministic, and its state at some iteration only depends on its state at the previous iteration. Hence, assuming an infinite number of decoding iterations, the decoder will eventually reach either a codeword state, or a state that has been visited before, in which case it gets caught in a loop, due to its deterministic behavior. If one keeps a history of the visited states, it is possible to detect when the decoder gets caught in a loop, *i.e.*, when $\underline{\boldsymbol{x}}^{(\ell_1)} = \underline{\boldsymbol{x}}^{(\ell_2)}$ for some iterations $\ell_1 < \ell_2$. We shall refer to ℓ_1 as the loop starting iteration and to $\ell_2 - \ell_1$ as the loop length. Fig. 2 shows the average loop starting iteration and the average loop length, for three regular LDPC codes, over the BSC:

(a) $(3, 6)$-regular LDPC code (rate = 0.5), length 1296 bits,
(b) $(4, 8)$-regular LDPC code (rate = 0.5), length 1296 bits,

(c) $(6,32)$-regular LDPC code (rate = 0.84), of length 2048 bits, from the IEEE 802.3an standard.

In all cases, the average loop starting iteration decreases with decreasing crossover probability of the BSC, meaning that the decoder gets caught in a loop earlier, as the channel gets better. The average loop length (indicated by the height of vertical bars) varies much less with the channel, especially for the higher degree codes: from 2.3 to 2.1 for the $(4,8)$-regular code, and from 4.08 to 4.01 for the $(6,32)$-regular code.

Since the momentum is aimed at preventing the decoder getting caught in a loop, we use momentum of length L equal (or close) to the average loop length of the GDBF decoder, as determined by simulation. Then, for a given moment length L, we search for momentum values $\rho_{max} \geq \rho(1) \geq \rho(2) \geq \cdots \geq \rho(L) > 0$, that yield the best decoding performance. To do this, we consider $\rho(\ell)$ in a discrete set of values, using some quantization step (to reduce the search space, we have used a quantization step to 0.5). Although the above optimization process may be long and laborious, it is an offline optimization process, that needs to be performed only once, then the optimized momentum is integrated to the decoding algorithm. In order to avoid exhaustive search of all possible solutions, it might be advantageously combined with search heuristics based for instance on genetic algorithms.

IV. Numerical Results

In this section, the decoding performance of BF-based decoders (BF, GDBF/PGDBF, and GDBF/PGDBF with momentum (w/M)) is assessed against that of more powerful MP decoders (BP and MS). Table I summarizes the various parameters used by the GDBF/PGDBF (w/M) decoders. The list of parameters used by each decoder is also indicated in the table. In our simulations, MP decoders perform 50 decoding iterations, with flooded scheduling, while BF-based decoders perform 300 decoding iterations.

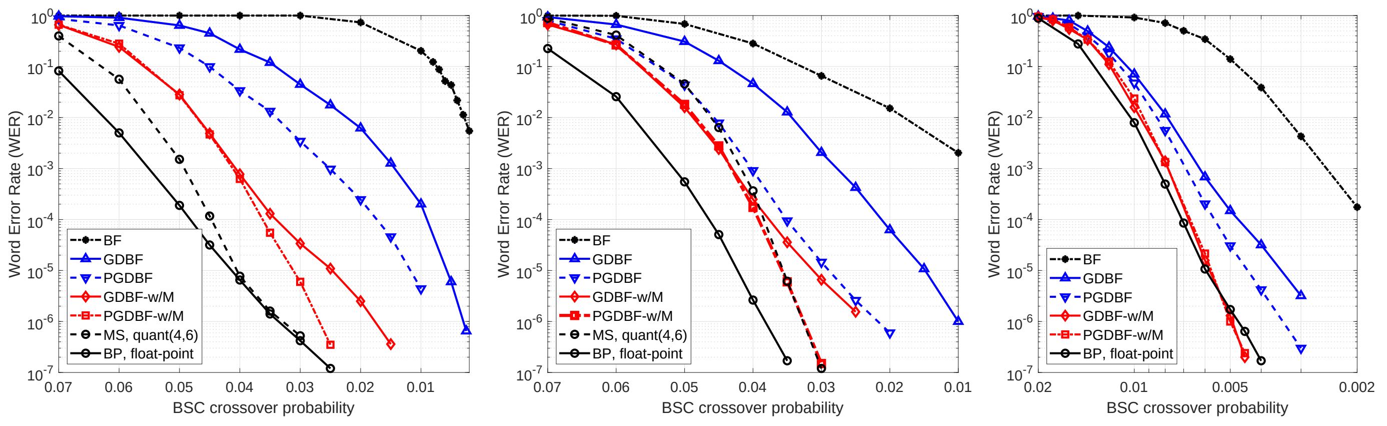
BSC channel: Simulation results for the BSC are shown in Fig. 3. Regarding MP decoders, we consider the floating point BP decoder, and the finite-precision MS decoder, with 4-bit messages. For the *(3,6)-regular code*, momentum significantly improves the performance of both GDBF and PGDF decoders. It is worth noticing that GDBF-w/M decoder significantly outperforms PGDBF, although the former does not make use of randomness. The PGDBF-w/M decoder makes use of both randomness and momentum, closely approaching the performance of the floating-point BP and finite-precision MS decoders, especially in the error floor region. For the *(4,8)-regular code*, the GDBF-w/M decoder performs slightly better than PGDF, but they both exhibit an error floor at word error rate WER $\approx 10^{-4}$. The PGDBF-w/M decoder does not show any error floor down to WER $= 10^{-7}$, and exhibits virtually the same decoding performance as the finite-precision MS decoder. Finally, for the *(6,32)-regular code*, it can be seen that both GDBF-w/M and PGDBF-w/M decoders exhibit virtually the same decoding performance, closely approaching and even outperforming the floating-point BP decoding in the error floor region.

AWGN channel: Simulation results for the AWGN channel (with bipolar ± 1 inputs) are shown in Fig. 4. Both BP and MS

Table I
Parameters used by GDBF/PGDBF (w/M) decoders

Channel	LDPC Code	correlation coef. α	inversion thresh. δ	bit-flip proba. p	momentum ρ
BSC	$(3,6)$-regular	0.5	0	0.9	$[2,2,2,1]$
	$(4,8)$-regular	1.0	0	0.9	$[4,2,1]$
	$(6,32)$-regular	2.0	0	0.8	$[4,3,2,1]$
AWGN	$(4,8)$-regular	1.8	1.1	0.9	$[2,2,2,2,2,1,1]$
	$(6,32)$-regular	4.5	1.2	0.8	$[3,3,2,1]$

Parameters used by each decoder: GDBF(α,δ), PGDBF(α,δ,p), GDBF-w/M(α,δ,ρ), PGDBF-w/M(α,δ,p,ρ)



(a) (3,6)-regular LDPC (b) (4,8)-regular LDPC (c) (6,32)-regular LDPC

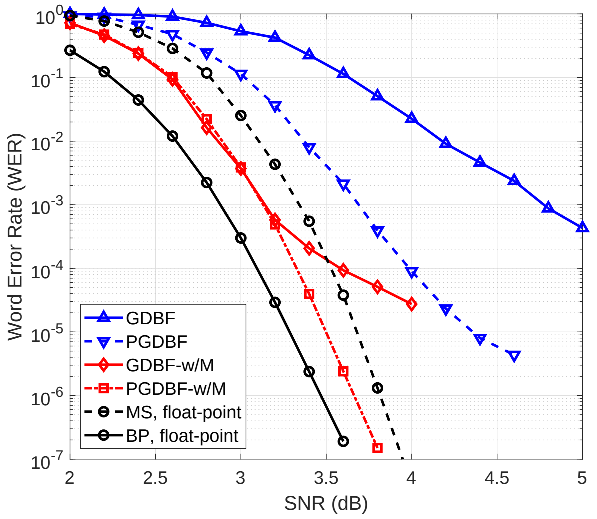
Figure 3. Decoding performance for the BSC

decoders are implemented in floating-point precision. GDBF-w/M and PGDBF-w/M decoders show virtually the same decoding performance in the waterfall region, outperforming the floating-point MS decoder, but the GDBF-w/M decoder exhibits a higher error floor. The PGDBF-w/M decoder closely approaches (within less than $0.2\,\mathrm{dB}$ for the (4,8)-regular code) or achieves virtually the same performance (for the (6,32)-regular code) as the floating-point BP decoder. This is all the more remarkable for soft-output channels, and demonstrates the effectiveness of the momentum technique for avoiding local minima in gradient descent based decoding.

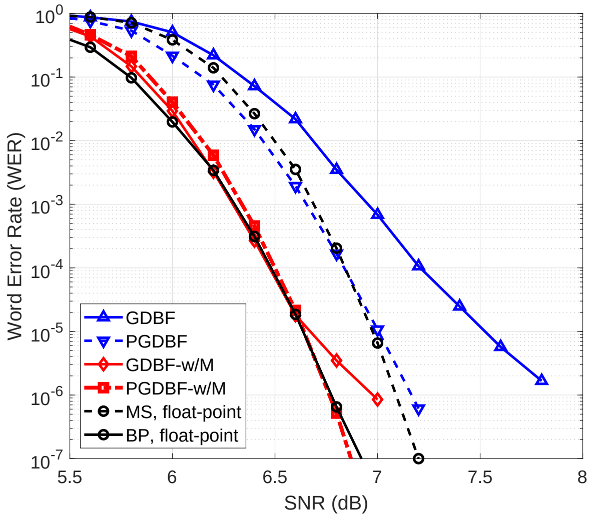
V. Conclusion and Perspectives

This paper reported on a new approach aimed at improving the decoding performance of randomized GDBF decoders, inspired by the momentum technique used in gradient descent optimization. We showed that GDBF or randomized GDBF decoders with momentum may closely approach the floating-point BP decoding performance, and may even outperform it in the error-floor region, especially for graphs with high connectivity degree. This makes the proposed technique particularly relevant to low error-floor applications, since LDPC codes designed for such applications are usually defined by bipartite graphs with higher degree of connectivity.

The results presented in this paper open a number of perspectives, regarding the optimization of the momentum parameters, for a given bipartite graph (or the optimization of both randomization and momentum parameters, in case of randomized GDBF decoders). To optimize the decoding performance in the error floor region, an analytical model based on absorbing Markov chains could be used to quantify the contribution of dominant trapping/absorbing sets to the word error rate [12]. A different approach is to learn optimal decoding parameters, by using deep reinforcement learning techniques. BF decoding *de facto* behaves as a deep neural network (DNN), with input/output layers corresponding to the input/output of the decoder, and hidden layers corresponding to the decoding iterations. We believe that DNN models for GDBF-based decoders may be used to learn decoding parameters (*e.g.*, randomization and momentum parameters), but also to learn weights and bias to improve decoding performance, while taking into account the specific code structure (short cycles, trapping/absorbing sets, etc.).



(a) (4,8)-regular LDPC



(b) (6,32)-regular LDPC

Figure 4. Decoding performance for the AWGN channel

References

[1] R. G. Gallager, "Low density parity check codes," MIT Press, Cambridge, 1963, research Monograph series.

[2] T. Wadayama, K. Nakamura, M. Yagita, Y. Funahashi, S. Usami, and I. Takumi, "Gradient descent bit flipping algorithms for decoding LDPC codes," in *IEEE International Symposium on Information Theory and Its Applications (ISITA)*, 2008, pp. 1–6.

[3] L. R. Varshney, "Performance of LDPC codes under faulty iterative decoding," *IEEE Trans. Inf. Theory*, vol. 57, no. 7, pp. 4427–4444, 2011.

[4] S. Yazdi, H. Cho, and L. Dolecek, "Gallager-B decoder on noisy hardware," *IEEE Transactions on Commmunications*, vol. 66, no. 5, pp. 1660–1673, 2013.

[5] C. Kameni Ngassa, V. Savin, E. Dupraz, and D. Declercq, "Density evolution and functional threshold for the noisy min-sum decoder," *IEEE Transactions on Communications*, vol. 63, no. 5, pp. 1497–1509, 2015.

[6] E. Dupraz, D. Declercq, B. Vasic, and V. Savin, "Analysis and design of finite alphabet iterative decoders robust to faulty hardware," *IEEE Transactions on Communications*, vol. 63, no. 8, pp. 2797–2809, 2015.

[7] O. Al Rasheed, P. Ivaniš, and B. Vasić, "Fault-tolerant probabilistic gradient-descent bit flipping decoder," *IEEE Communications Letters*, vol. 18, no. 9, pp. 1487–1490, 2014.

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

[9] Y. Nesterov, "A method of solving a convex programming problem with convergence rate $o(1/k^2)$," *Soviet Mathematics Doklady*, vol. 27, no. 2, pp. 372–376, 1983.

[10] V. Apidopoulos, J.-F. Aujol, C. Dossal, and A. Rondepierre, "Convergence rates of an inertial gradient descent algorithm under growth and flatness conditions," 2018, hAL Preprint, https://hal.archives-ouvertes.fr/hal-01965095.

[11] M. D. Zeiler, "ADADELTA: an adaptive learning rate method," 2012, arXiv preprint arXiv:1212.5701.

[12] P. Ivaniš and B. Vasić, "Error errore eicitur: A stochastic resonance paradigm for reliable storage of information on unreliable media," *IEEE Transactions on Communications*, vol. 64, no. 9, pp. 3596–3608, 2016.

Appendix A
Additional Data

The $(3,6)$-regular and $(4,8)$-regular LDPC codes used for the simulations in this paper are quasi-cyclic LDPC codes, with base matrix of size 12×24 and expansion factor $z = 54$. Hence, both codes have length $24 \times 54 = 1296$ bits. The base matrices of the two codes are given below. The parity check matrix of a code is obtained by replacing each non-zero entry $b \geq 0$ in the base matrix by a square matrix of size $z \times z$, defined as the circular *right-shift* of the identity matrix by b positions. Entries $b = -1$ in the base matrix are replaced by all-zero square matrices of size $z \times z$.

(a) $(3,6)$-regular LDPC code (rate $= 0.5$), length 1296 bits

$$\begin{bmatrix}
49 & -1 & -1 & -1 & -1 & 43 & -1 & -1 & -1 & -1 & 50 & -1 & -1 & -1 & -1 & 2 & -1 & 27 & -1 & -1 & -1 & -1 & -1 & 49 \\
-1 & -1 & -1 & 10 & 41 & -1 & -1 & -1 & -1 & 52 & -1 & -1 & 32 & -1 & -1 & -1 & -1 & -1 & 50 & -1 & 50 & -1 & -1 & -1 \\
-1 & -1 & 20 & -1 & -1 & -1 & -1 & 20 & -1 & -1 & -1 & 51 & -1 & 10 & -1 & -1 & 47 & -1 & -1 & -1 & -1 & -1 & 33 & -1 \\
-1 & 24 & -1 & -1 & -1 & -1 & 22 & -1 & 53 & -1 & -1 & -1 & -1 & -1 & 31 & -1 & -1 & -1 & -1 & 18 & -1 & 47 & -1 & -1 \\
10 & -1 & -1 & -1 & 15 & -1 & -1 & -1 & -1 & -1 & 2 & -1 & -1 & -1 & -1 & 50 & -1 & 13 & -1 & -1 & -1 & -1 & -1 & 53 \\
-1 & -1 & 44 & -1 & -1 & 6 & -1 & -1 & -1 & -1 & -1 & 29 & -1 & 40 & -1 & -1 & 16 & -1 & -1 & -1 & 13 & -1 & -1 & -1 \\
-1 & 2 & -1 & -1 & -1 & -1 & -1 & 13 & 41 & -1 & -1 & -1 & -1 & -1 & 42 & -1 & -1 & -1 & -1 & 48 & -1 & 49 & -1 & -1 \\
-1 & -1 & -1 & 36 & -1 & -1 & 24 & -1 & -1 & 50 & -1 & -1 & 12 & -1 & -1 & -1 & -1 & -1 & 10 & -1 & -1 & -1 & 48 & -1 \\
-1 & -1 & 47 & -1 & 50 & -1 & -1 & -1 & -1 & -1 & 0 & -1 & -1 & -1 & -1 & 9 & -1 & 7 & -1 & -1 & -1 & -1 & -1 & 28 \\
-1 & 24 & -1 & -1 & -1 & -1 & -1 & 51 & -1 & 38 & -1 & -1 & -1 & -1 & 6 & -1 & -1 & -1 & -1 & 23 & -1 & 16 & -1 & -1 \\
6 & -1 & -1 & -1 & -1 & -1 & 5 & -1 & -1 & -1 & -1 & 13 & -1 & 3 & -1 & -1 & 29 & -1 & -1 & -1 & 16 & -1 & -1 & -1 \\
-1 & -1 & -1 & 35 & -1 & 16 & -1 & -1 & 37 & -1 & -1 & -1 & 4 & -1 & -1 & -1 & -1 & -1 & 24 & -1 & -1 & -1 & 29 & -1
\end{bmatrix}$$

(b) $(4,8)$-regular LDPC code (rate $= 0.5$), length 1296 bits

$$\begin{bmatrix}
11 & -1 & -1 & -1 & 27 & -1 & -1 & -1 & 33 & 16 & -1 & -1 & -1 & 44 & -1 & -1 & 44 & -1 & 8 & -1 & -1 & -1 & -1 & 0 \\
-1 & 25 & -1 & -1 & -1 & 31 & 29 & -1 & -1 & -1 & 29 & -1 & -1 & -1 & 36 & -1 & -1 & 34 & -1 & 15 & -1 & -1 & 17 & -1 \\
-1 & -1 & 44 & 4 & -1 & -1 & -1 & 11 & -1 & -1 & -1 & 2 & 50 & -1 & -1 & 52 & -1 & -1 & -1 & -1 & 30 & 33 & -1 & -1 \\
27 & -1 & -1 & -1 & 34 & -1 & 20 & -1 & -1 & 20 & -1 & -1 & -1 & 13 & -1 & -1 & 27 & -1 & 4 & -1 & -1 & -1 & -1 & 27 \\
-1 & 42 & -1 & 22 & -1 & -1 & -1 & 11 & -1 & -1 & -1 & 44 & -1 & -1 & 4 & 14 & -1 & -1 & -1 & -1 & 45 & 17 & -1 & -1 \\
-1 & -1 & 24 & -1 & -1 & 10 & -1 & -1 & 10 & -1 & 18 & -1 & 2 & -1 & -1 & -1 & -1 & 19 & -1 & 38 & -1 & -1 & 31 & -1 \\
-1 & -1 & 40 & -1 & -1 & 35 & -1 & -1 & 31 & 19 & -1 & -1 & 3 & -1 & -1 & 42 & -1 & -1 & -1 & 42 & -1 & -1 & 39 & -1 \\
-1 & 29 & -1 & 0 & -1 & -1 & -1 & 29 & -1 & -1 & 5 & -1 & -1 & -1 & 47 & -1 & -1 & 28 & -1 & -1 & 28 & 41 & -1 & -1 \\
9 & -1 & -1 & -1 & 7 & -1 & 20 & -1 & -1 & -1 & -1 & 1 & -1 & 19 & -1 & -1 & 5 & -1 & 25 & -1 & -1 & -1 & -1 & 41 \\
-1 & -1 & 53 & -1 & -1 & 3 & -1 & -1 & 26 & -1 & 3 & -1 & -1 & -1 & 30 & -1 & -1 & 5 & -1 & 35 & -1 & -1 & 44 & -1 \\
-1 & 4 & -1 & -1 & 4 & -1 & -1 & 5 & -1 & -1 & -1 & 13 & 42 & -1 & -1 & 50 & -1 & -1 & -1 & -1 & 36 & 38 & -1 & -1 \\
39 & -1 & -1 & 17 & -1 & -1 & 36 & -1 & -1 & 34 & -1 & -1 & -1 & 46 & -1 & -1 & 12 & -1 & 8 & -1 & -1 & -1 & -1 & 15
\end{bmatrix}$$