Dynamic Weighted Bit-Flipping Decoding Algorithms for LDPC Codes

Tofar C.-Y. Chang, Student Member, IEEE and Yu T. Su, Senior Member, IEEE
emails: {tofar.cm96g, ytsu}@nctu.edu.tw.

Abstract—Bit-flipping decoding of LDPC codes is of low complexity but gives inferior performance in general. We propose new checksum weight generation and flipped-bit selection (FBS) rules to enhance their performance. From belief propagation’s viewpoint, the checksum and its weights determine the beliefs a check node (CN) passes to its connected variable nodes (VNs) which then update their beliefs about the associated bit decisions by computing the corresponding flipping functions (FFs). Our FF includes a weighted sum of checksums but unlike existing FFs, we adjust the weights associated with each checksum in every decoding iteration with some beliefs inhibited if necessary. Our new FBS rule takes more information into account in determining the bits to be flipped. These two modifications represent our efforts to track more closely the evolutions of both CNs and VNs’ reliabilities. To reduce the decoder complexity, we further suggest two selective weight-updating schedules. Different combinations of the new FBS rule and known or new FFs offer various degrees of performance improvements. Numerical results indicate that the decoders using the new FF and FBS rule yield performance close to that achieved by the sum-product algorithm and the reduced-complexity selective weight-updating schedules incur only minor performance loss.

Index Terms—LDPC codes, belief propagation, bit-flipping decoding, flipped bit selection.

I. INTRODUCTION

LDPC codes have been shown to asymptotically give near-capacity performance when the sum-product algorithm (SPA) is used for decoding [1]. Gallager proposed two alternatives that use only hard-decision bits [2]. These so-called bit-flipping (BF) algorithms flip one or a group of bits based on the values of the flipping functions (FFs) computed in each iteration. The FF associated with a variable node (VN) is a reliability metric of the corresponding bit decision and depends on the binary-valued checksums of the VN’s connected check nodes (CNs). Although BF algorithms are much simpler than the SPA, their performance is far from optimal. To reduce the performance gap, many variants of Gallager’s BF algorithms have been proposed. Most of them tried to improve the VN’s reliability metric (the FF) and/or the method of selecting the flipped bits, achieving different degrees of bit error rate (BER) and convergence rate performance enhancements at the cost of higher complexity.

The class of weighted bit-flipping algorithms [3]–[6] assign proper weights to the binary checksums. Each weight can be regarded as a reliability metric on the corresponding checksum and is a function of the associated soft received channel values. Another approach called gradient descent bit-flipping (GDBF) algorithm was proposed recently [8]. Instead of using a weighted checksum based FF, the GDBF algorithm derives its FF by computing the gradient of a nonlinear objective function which is equivalent to the log-likelihood function of the bit decision with checksum constraints. It was shown that the GDBF algorithm outperforms most known WBF algorithms when the VN degree is small.

For the WBF algorithms, the weights are decided by the soft received channel values and remain unchanged throughout the decoding process. Since the weights reflect the decoder’s belief on the checksums which, in turn, depend on those of the associated VNs’ FF and bit decisions, the associated checksum weights should be updated accordingly. In this paper, we present dynamic weighted BF (DWBF) algorithms that assign dynamic checksum weights which are updated according to a nonlinear function of the associated VNs’ FF values. As we shall show, the nonlinear function has the effects of quarantining unreliable checksums while dynamically adjusts the more reliable checksums’ beliefs. We also suggest two selective weight-updating schedules to provide different performance-complexity trade-offs.

The single-bit BF algorithms flip only the least reliable bit thus result in slow convergence rates. For this reason, many a multiple-flipped-bit selection rule was suggested [8], [9]–[14]. By simultaneously flipping the selected bits, a BF decoder can offer rapid convergence but, sometimes, at the expense of performance loss. A bit selection rule may consist of simple threshold comparisons or include a number of steps involving different metrics. It is usually designed assuming a specific FF is used and may not be suitable when a different FF or metric is involved. As the FF value may not provide sufficient statistic for making a tentative bit decision, we propose a new flipped-bit selection (FBS) rule that takes into account both the FF value and other information from related CNs.

The efficiencies of using the proposed checksum weight, FF, weight-updating schedules, and FBS rule jointly or separately with existing designs are evaluated by examining the corresponding numerical error rate and convergence behaviors. We show that our single-bit DWBF algorithm provides significant performance improvement over the existing single-bit GDBF and WBF algorithms. Our FBS rule works very well with different FFs and outperforms known FBS rules. Moreover, the selective weight-updating schedules suffer little performance degradation while offer significant complexity reduction when the CN and VN degrees are small.

The rest of this paper is organized as follows. In Section II we define the basic system parameters and give a brief overview of various BF decoders and the associated FBS rules. In Section III we introduce our checksum weight updating...
rule and the associated FF. Two weight-updating schedules, the DWBF algorithm and its performance along with some other single-bit algorithms’ are presented in Section [V]. We develop a new FBS rule and analyze different decoders’ complexities in Section [V]. The error rate and convergence behaviors of various multi-bit BF decoders using the new FBS rule is also given in the same section. Finally, conclusion remarks are drawn in Section [VI].

II. BACKGROUNDS AND RELATED WORKS

A. Notations and the Basic Algorithm

We denote by \((N, K) (d_c, d_e)\) a regular binary LDPC code \(C\) with VN degree \(d_c\) and check node (CN) degree \(d_e\), i.e., \(C\) is the null space of an \(M \times N\) parity check matrix \(H = [H_{mn}]\) which has \(d_c\) 1’s in each column and \(d_e\) 1’s in each row. Let \(\mathbf{u}\) be a codeword of \(C\) and assume that the BPSK modulation is used so that a codeword \(\mathbf{u} = (u_0, u_1, \cdots, u_{N-1})\), \(u_i \in \{0, 1\}\), is mapped into a bipolar sequence \(\mathbf{x} = (x_0, x_1, \cdots, x_{N-1})\) = \((1 - 2u_0, 1 - 2u_1, \cdots, 1 - 2u_{N-1})\) for transmission.

The equivalent baseband transmission channel is a binary-input Gaussian-output channel characterized by additive zero-mean white Gaussian noise with two-sided power spectral density of \(N_0/2\) W/Hz. Let \(\mathbf{y} = (y_0, y_1, \cdots, y_{N-1})\) be the sequence of soft channel values obtained at the receiver’s coherent matched filter output. The sequence \(\mathbf{z} = (z_0, z_1, \cdots, z_{N-1})\), where \(z_i \in \{0, 1\}\), is obtained by taking hard-decision on each component of \(\mathbf{y}\). Let \(\hat{\mathbf{u}} = (\hat{u}_0, \hat{u}_1, \cdots, \hat{u}_{N-1})\) be the tentative decoded binary sequence and each components of \(\mathbf{y}\).

B. Flipping Functions of BF Decoding Algorithms

Gallager proposed that a simple sum of binary checksums be used as the FF [2]

\[
E_n = - \sum_{m \in M(n)} (1 - 2s_m). \tag{2}
\]

implies that the FF value is inversely proportional to the bit decision reliability as it is an increasing function of the number of nonzero checksums (i.e., unsatisfied check nodes, UCNs).

By taking into account soft-valued channel information and assigning checksum weights, later modifications of Gallager’s FF can be described by the following general formula

\[
E_n = - \alpha_1 \cdot \phi(\hat{u}_n, y_n) - \sum_{m \in M(n)} w_{mn} (1 - 2s_m), \tag{3}
\]

where \(\alpha_1 > 0\), \(\phi(\hat{u}_n, y_n)\) is a reliability metric involving channel value and/or bit decision, and, to be consistent with [2], \(w_{mn} \geq 0\).

For the weighted BF (WBF) algorithm [3], \(\phi(\hat{u}_n, y_n) = 0\) and \(w_{mn}\) is

\[
w_{mn} = \min_{n' \in N(m)} |y_{n'}|, \tag{4}
\]

The modified WBF (MWBF) algorithm [4] has \(\phi(\hat{u}_n, y_n) = |y_{n'}|\) while the improved MWBF (IMWBF) algorithm [5] uses the same \(\phi(\hat{u}_n, y_n)\) but replaces the checksum weight by

\[
w_{mn} = \min_{n' \in N(m) \setminus n} |y_{n'}| \tag{5}
\]

for the belief passed from \(c_m\) to \(v_n\) should exclude that originated from \(v_n\). For the reliability ratio based WBF (RRWBF) algorithm [6], \(\phi(\hat{u}_n, y_n) = 0\) and

\[
w_{mn} = 1/w_{mn}' = \left(\beta \max_{n' \in N(m)} |y_{n'}| \right)^{-1}, \tag{6}
\]

where \(\beta\) is the normalizing factor to ensure that \(\sum_{n \in N(m)} w_{mn}' = 1\).

The GDBF algorithm of Wadayama et al. [8] applies the gradient descent method to minimize

\[
f(\hat{u}) = - \sum_{n=0}^{N-1} y_n (1 - 2\hat{u}_n) - \sum_{m=0}^{M-1} (1 - 2s_m) \tag{7}
\]
with respect to \((1 - 2\hat{u}_n)\) and obtains the FF
\[
E_n = -y_n(1 - 2\hat{u}_n) - \sum_{m \in \mathcal{M}(n)}(1 - 2s_m),
\]
(8)
which is equivalent to assigning \(\alpha_1 = 1, \phi(\hat{u}_n, y_n) = y_n(1 - 2\hat{u}_n),\) and \(w_{mn} = 1\) in [3].

C. Flipped Bit Selection Rules

For the algorithms mentioned in the Section II-B, only the bit(s) related to the VN having the largest FF value \(E_n\) is (are) flipped in each iteration, i.e., the FB set is
\[
\mathcal{B} = \{n | n = \arg \max_i E_i\}. \quad (9)
\]
As mentioned before, \(|\mathcal{B}| = 1\) and the corresponding convergence is often very slow if \(E_n\) has a soft-valued information term.

Flipping several bits in each iteration simultaneously can improve the convergence speed. The simplest FBS rule for multi-bit BF decoding uses the FB set
\[
\mathcal{B} = \{n | E_n \geq \Delta\}, \quad (10)
\]
where the threshold \(\Delta\) can be a constant or be adaptive. The optimal adaptive threshold was derived by Gallager [2], assuming that no cycle appears in the code graph. Since practical finite-length LDPC codes usually have cycles and the optimal thresholds can only be found through time-consuming simulations, two ad-hoc methods which automatically adjust \(\Delta\) were suggested in [11] and [14]. In the adaptive threshold BF (ATBF) algorithm [11], the initial \(\Delta\) is found by simulation and subsequent thresholds are a monotonically non-increasing function of the decoding iterations. The adaptive MWBF (AMWBF) algorithm [14] adjusts the threshold by
\[
\Delta = E^* - |E^*| \left[1 - \frac{w_H(s)}{M}\right], \quad (11)
\]
where \(E^* = \max_n E_n\) and \(w_H(s)\) is the Hamming weight of the syndrome vector \(s\).

Sometimes, a tentative decoded vector \(\hat{u}\) may reappear several times during the decoding process and form a decoding loop. This may be caused, for example, by the event that an even number of bits associated with a CN are flipped, leading to an unchanged checksum and then oscillating bit decisions. To eliminate the occurrence of loops, the AMWBF algorithm includes the loop detection scheme of [7] in its FBS rule so that if a loop is detected, the most reliable bit in \(\mathcal{B}\) is removed. The parallel weighted BF (PWBF) algorithm [9] tries to reduce the loop occurrence probability by having every UCN \((s_m = 1)\) send a constant flip signal (FS) to its least reliable linked VN (based on the FF of the IMWBF algorithm) and flips the bits in
\[
\mathcal{B} = \{n | F_n \geq \Delta_{FS}\}, \quad (12)
\]
where \(\Delta_{FS}\) is a constant optimized by simulations,
\[
F_n = \sum_{m \in \mathcal{M}(n)} q_{mn}s_m, \quad (13)
\]
and \(q_{mn}\) is given by
\[
q_{mn} = \begin{cases} 1, & n = \arg \max_{n' \in \mathcal{N}(m)} E_{n'} \vspace{1em} \cr 0, & \text{otherwise} \end{cases} \quad (14)
\]
Since the above remedy can only eliminate loops with a certain probability, the improved PWBF (IPWBF) algorithm employs the loop detection scheme of [7] and when a loop is detected, it removes the bit(s) receiving the smallest \(F_n\) from \(\mathcal{B}\). This algorithm also adds a bootstrapping step and a delay-handling procedure to further improve the bit selection accuracy but achieves limited improvement for the codes with high column degrees such as Euclidean geometry (EG) LDPC codes.

A hybrid GDBF (HGDBF) algorithm was proposed in [8]. In this algorithm, single- and multi-bit BF decoding is performed alternatively and an escape process is used for preventing the decoding process from being trapped in local minima/loops. Two extensions of the HGDBF algorithm were considered in [12] and [13] which require less complexity at the expense of inferior performance.

III. CHECKSUM BELIEF AND DYNAMIC WEIGHTS

A. BF Decoding and Checksum Weights

In line with the belief propagation (BP) based SPA, \(E_n\) is similar to the total log-likelihood ratio (LLR) of \(v_n\) and \(-w_{mn}(1 - 2s_m)\) in [3] is analogous to the belief sent to \(v_n\) by \(c_m\). Unlike SPA, however, for (4)-(6) and (8), the latter remains unchanged unless an \(\hat{u}_n\), \(n \in \mathcal{N}(m)\) has been flipped, which leads to a sign change of the belief. The general FF format, (3), includes two major terms that represent the decoder’s belief of a bit’s tentative decision based respectively on its channel decision) and the beliefs on the related checksums. Since the channel values remain fixed, the checksum term should be given adjusted weights at least in the later iterations when the beliefs on checksums change.

Although the flipping operation changes the reliability metric of \(\hat{u}_n\) and the related checksums, all the FFs used in known BF decoders use static \(w_{mn}\) thereby can neither reflect the dynamic of VNs’ belief propagations nor offer self-adjustment capability in accurately updating bit reliability information. We present dynamical weight generation method in this section.

B. Dynamic Weights and New Flipping Function

The review on BF algorithms in Section II indicates clearly that the FF value is a proper explicit or implicit reliability metric of a VN’s decision. As a checksum in turn is a function of the associated VNs’ decisions, the corresponding checksum weights should be updated according to the current FF values. A reasonable candidate checksum weight is therefore given by
\[
E_{n'}^{(l)} = \min_{n' \in \mathcal{N}(m) \setminus n} -E_n^{(l)}, \quad (15)
\]
where \(E_n^{(l)}\) is the FF value of \(v_n\) in the \(l\)th iteration. However, [15] may result in negative weights which is inconsistent with [2] and [3]; both are increasing functions of the nonzero
checksum number. To have proper positive checksum weights based on \( E_n \), we consider the likelihood ratio

\[
\Lambda(E_n) = \frac{\Pr(E_n|H_0)}{\Pr(E_n|H_1)},
\]

where \( H_0 \) and \( H_1 \) denote the hypotheses that \( \hat{u}_n = u_n \) and \( \hat{u}_n \neq u_n \), respectively. The conditional probability density function (pdf), \( \Pr(E_n|H_i) \), is the pdf of those \( E_n \)'s associated with a correct or incorrect tentative bit decision at a given iteration. It is to be interpreted as a conditional pdf averaged over all VNs. The basic decision theory tells us that the optimal decision rule is given by

\[
\Lambda(E_n) \geq \frac{\pi_i}{p_i} (C_{01} - C_{10}) \quad \text{and} \quad \frac{\pi_i}{p_i} (C_{10} - C_{00})
\]

where \( \pi_i = \Pr(H_i \text{ is true}) \) and \( C_{ij} \) is the “cost” for accepting \( H_j \) while \( H_i \) is true. Unlike the conventional Bayesian minimizing error probability setting, both the costs and the a priori probabilities are difficult to assess. For a BF decoder, a tentative decision, except for the initial iteration, is determined 

\[
\text{priori probabilities are difficult to assess but the optimal }
\]

tentative decision, \( FF \), and checksums, as elaborated in more details below, suggest that we modify (15) as

\[
\text{below, suggest that we modify (15) as}
\]

\[
\Omega(x) = \begin{cases} x - \eta, & x \geq \eta \\ 0, & x < \eta \end{cases}
\]

The clipping operator, \( \Omega(x) \), besides ensuring only positive weights are used, can be interpreted as the decision for a CN to send no message to other linked VNs when the associated FF values fail to exceed the threshold, which bears the flavor of “stop-and-go” algorithms that pass a CN-to-VN belief only if it is deemed reliable. Note that a checksum \( s_m \) is determined by \( d_m \) bit decisions, and if \( s_m = 0 \) and there is only one unreliable decision \( \hat{u}_n \) (\( E_n < \eta \)) among VNs in \( \mathcal{N}(m) \), the checksum is likely to be valid and the decision is in fact correct. Hence \( c_m \) should modify \( E_n \) to increase \( \Lambda(E_n) \) but not pass its beliefs \( -r_m^{(l)}(1 - 2s_m) \) to other connected (reliable) VNs \( \{n' \in \mathcal{N}(m) \setminus n\} \). In doing so, \( E_n \) has a local (among \( \mathcal{N}(m) \)) maximum FF decrease and the probability of reversing the bit decision is reduced. On the other hand, if \( s_m = 1 \), \( \hat{u}_n \) is likely to be only local incorrect decision, the above rule will result in a local maximum FF increase and thus a higher probability of being flipped. When more than one \( E_n > \eta \), \( n \in \mathcal{N}(m) \) are clipped, no belief is sent from \( c_m \) as the checksum itself is unreliable. The temporary suspension of some message propagation induced by \( \Omega(x) \) also has the desired effect of containing the damage an incorrect belief may have done and preventing the decoder from being trapped in a local minimum. The above discourse confirms that (19) does fulfill the goal that weight updating should have FFs, checksums, and bit decisions join a cohesive effort in improving the performance of a BF decoder.

The FF defined by (3) using recursive weight update rule (18) tends to make the check belief part of the FF, \( -\sum_{m \in \mathcal{M}(n)} w_{mn}(1 - 2s_m) \), starts to grow exponentially after most of the correctable bits were flipped and the number of UCNs decreases to just a few. It is conceivable that these estimates should be given different weight with more remote estimates having less weights. This can be done by having the check belief multiplied by a damping factor, \( 0 < \alpha_2 < 1 \), as can be found in many recursive adaptive filters. As for the optimal clipping threshold \( \eta \), we are unable to determine since the closed-form expressions for \( \Pr(E_n|H_i) \) are practically unobtainable for reasons mentioned before. Some simulation
Furthermore, for those VNs whose FF values change from
linked checksums should have the highest updating priority.

With the above ideas in mind, we consider a new FF based on
(18) and (19) using \( \eta = 0 \):
\[
E_n^{(l)} = -y_n(1-2\hat{u}_n) - \alpha_2 \sum_{m \in M(n)} r_{mn}^{(l-1)}(1-2s_m), \tag{20}
\]
where \( 0 < \alpha_2 < 1 \) is a positive damping constant to be
optimized by numerical experiments.

IV. NEW SINGLE-BIT BF DECODING ALGORITHMS

In this section, we introduce a class of single-bit dy-
namic weighted BF (S-DWBF) decoding methods based on
the checksum weight \( \text{(18)} \), the FF \( \text{(20)} \), and three different
weight-updating schedules. These schedules provide trade-offs
between computational complexity and error-rate performance.

A. Single-DWBF Decoding and Weight-Updating Schedules

Combining (18) and the new FF (20), we obtain the DWBF
algorithm or, for simplicity, Algorithm 2.

Algorithm 2: Decoding Algorithm

Initialization Set \( l = 0 \), \( \hat{u} = z \). Initialize \( r_{mn}^{(l)} \) by (5)
for each \( n \in \mathcal{N}(m), m = 0, 1, \ldots, M-1 \) and let \( E_n^{(l)} = -y_n(1-2\hat{u}_n) \)
for \( n = 0, 1, \ldots, N-1 \).

Step 1 Compute \( s_m \) for all \( m = 0, 1, \ldots, M-1 \). If \( s = 0 \) or
\( l = l_{\text{max}} \), stop decoding and output \( \hat{u} \); otherwise, let
\( l = l + 1 \).

Step 2 For \( n = 0, 1, \ldots, N-1 \), compute \( E_n^{(l)} \) by (20).

Step 3 Update \( B \) and \( \forall n \in B \), flip \( \hat{u}_n \) and let \( E_n^{(l)} = -E_n^{(l)} \).

Step 4 For all \( n \in \mathcal{N}(m), m = 0, 1, \ldots, M-1 \), update \( r_{mn}^{(l)} \)
by (18) and go to Step 1.

For hard-decision decoding, \( -y_n(1-2\hat{u}_n) \) in (20) is
replaced by \(-1-2z_n(1-2\hat{u}_n)\) and \( r_{mn}^{(l)} \) is initialized as 1.

In Step 3, we invert the sign of the flipped VNs' FF values
before computing new \( r_{mn}^{(l)} \) 's. When we consider the FB set
\( \mathcal{A} \), Algorithm 2 flips only one bit (i.e., \( |B| = 1 \)) at each
iteration. Nevertheless, most FF values will change because of
the recursive nature of (18) and (20). Therefore, in Step 4, we
need to update the checksum beliefs for almost all CNs. This
is one of the prices we have to pay when the dynamic weights
instead of the conventional constant weights are assigned
to the checksums. To distinguish from other updating rules
to be discussed in the following paragraphs, we call this
updating rule as the full weight-updating (belief-propagation)
schedule (FWUS) and the resulting decoding algorithm as
the S-DWBF-F algorithm for simplicity. To lessen the computing
load of FWUS, we reduce the number of updated beliefs by
prioritizing the CNs' beliefs and update only those with higher
priority.

We first notice that, to ensure that the newest updated
information be broadcasted, the weights of the flipped bit's
linked checksums should have the highest updating priority.
Furthermore, for those VNs whose FF values change from
one side of the clipping threshold \( \eta \) of (19) to the other side
and undergo a reliability inversion, their related checksum
weights should be renewed as well. With these considerations,
the selective weight-updating schedule A (SWUS-A) updates
only those checksums (CNs) whose indices lie in
\[
G_A^{(l)} \triangleq \{ m | m \in M(n), n \in B \}.
\]

The time-expanded factor graph (TEFG) shown in Fig.
2 is a simple example illustrating how SWUS-A behaves,
assumed that the only VN which generates \( G_A^{(l)} = \{1,2\} \) is \( v_4 \). We denote this VN by \( \bullet \), the CNs visited (selected) by
the schedule by \( \blacksquare \), and the VNs which receive new CN beliefs
by \( \circ \).

When the DWBF algorithm is used along with (2) and
SWUS-A, we substitute Step 4 of Algorithm 2 by

Step 4 Update \( G_A^{(l)} \), and then update \( r_{mn}^{(l)} \) by (18) \( \forall n \in \mathcal{N}(m), \)
m \( \in G_A^{(l)} \), and let \( r_{mn}^{(l)} = r_{mn}^{(l-1)} \) \( \forall n \in \mathcal{N}(m), m \notin G_A^{(l)} \).

Then, go to Step 1.

The resulting decoder is called the S-DWBF-A algorithm.

Since only a few VNs received updated beliefs from the
selected CNs, some \( E_n \) 's are likely to remain constant for
many iterations (e.g., \( v_8 \)) or even during the whole decoding
process. To spread the updated beliefs to more VNs, we expand
the updated CN set to include both \( G_A^{(l)} \) and \( G_B^{(l)} \).
The performance of the known single-bit BF algorithms and S-DWBF-A/B/F algorithms for MacKay (816, 272)(4, 6) rate-0.333 LDPC code (816.44.878 16 and (1023, 781)(32, 32) rate-0.763 EG-LDPC code are plotted in Figs. 3 and 4 where we refer to these two codes as Code 1 and 2, respectively. The performance of SPA or the normalized BP (NBP) [15] is also given there for reference purpose.

Fig. 3 shows the BER performance of Code 1 with $l_{\text{max}} = 150$. For the S-DWBF-A, S-DWBF-B, and S-DWBF-F decoders, the numerically-optimized $\alpha_2$ values are 0.66, 0.44, and 0.35 whereas for the IMWBF decoder, we found $\alpha_1 = 0.2$. Our extensive simulation concluded that the optimal reliability threshold $\eta$ in the IMWBF algorithm uses $\eta = 0$ is used for all DWBF algorithms. At BER=$10^{-5}$, we observe that the S-DWBF-B and S-DWBF-F algorithms have 2.5 dB and 2.6 dB gains against the RRWBF algorithm; the simple S-DWBF-A algorithm achieves a much smaller 0.7 dB gain as it limits its weight update to a very limited range. The performance of Code 2 with $l_{\text{max}} = 50$ is shown in Fig. 4. Unlike Code 1, Code 2 has a much higher $d_v$, and the check belief part of the GDBF algorithm’s FF, $-\sum_{m \in M(n)} (1 - 2s_m)$, thus dominates the FF value after a few iterations and its performance is similar to that of Gallager’s BF algorithm, especially when SNR is high. To improve its performance we insert a damping factor $\alpha_3$ so that (21) becomes

$$E_n = -y_n (1 - 2\hat{u}_n) - \alpha_3 \sum_{m \in M(n)} (1 - 2s_m).$$

This modification is equivalent to multiplying the second summation of (12) by $\alpha_3$ and the damping factor is analogously to the Lagrange multiplier in (the checksum) constrained optimization. The optimal $\alpha_3$ for (21) is close to 1/17 for Code 2. Referring to (3), the IMWBF algorithm uses $\alpha_1 = 1.8$ and the S-DWBF-A (B) decoder uses $\alpha_2 = 0.33$ (0.3) in (20). Due to the high VN/CN degrees of Code 2, almost all CNs are updated by S-DWBF-B algorithm after two or three iterations, yielding performance similar to that of the S-DWBF-F decoder. The same figure shows that the S-DWBF-A decoder provides about 0.25 dB performance gain with respect to the IMWBF decoder at BER = $10^{-5}$ and the S-DWBF-B algorithm offers additional 0.1 dB gain.

Table II presents the average number of CNs visited (in the associated ETFGs) by different schedules, normalized by the total number of frames simulated ($10^6$) for Code 1.
TABLE II: Average Number of Visited CNs in S-DWBF algorithms

| Code | Iteration | S-DWBF-A | S-DWBF-B |
|------|-----------|----------|----------|
| 1    | 10        | 15.7     | 137.2    |
|      | 30        | 10.1     | 88.7     |
|      | 50        | 8.3      | 76.4     |
| 2    | 5         | 68.7     | 1023.0   |
|      | 10        | 38.7     | 1023.0   |
|      | 20        | 32.5     | 1002.5   |

Fig. 5: Conditional FF distributions for the S-DWBF-F algorithm in decoding Code 1, SNR = 4 dB.

and Code 2 at selected iterations. Although the S-DWBF-F algorithm has the best BER performance among the single-bit algorithms, it requires higher computational complexity. By contrast, the S-DWBF-A/B algorithms provide trade-offs between complexity and performance. Furthermore, for both selective weight-updating schedules, the number of visited CNs decreases when one proceeds with more iterations as the numbers of flipped bits and reliability-inverted VNs decreases.

We plot the FF value distributions for the S-DWBF-F algorithm in Fig. 5. In contrast to Fig. 1, where the separation between the two conditional pdfs exhibits little variation, our DWBF algorithm is able to pull $\Pr(E_n|H_0)$ away from $\Pr(E_n|H_1)$ as the decoding process evolves. Since the reliability of a decoder decisions based on $E_n$ depends on the separation (distance) between the two pdfs, the improved separation is certainly welcome. As mentioned before, we use $\eta = 0$ in (14) for all S-DWBF algorithms. Although the optimal clipping threshold is unknown, Fig. 5 does convince us that 0 is a valid and convenient choice and the FF with the proposed dynamic checksum weighting does give a much better VN reliability reference.

### V. New Flipped-Bit Selection Method

In Section II, we mentioned that (10) is a simple set for selecting multiple flipped bits. In practice, it is necessary to add the option $B = \{n|n = \arg \max_i E_i\}$ in case $B = \emptyset$. Such an FBS rule is labeled as the M1 FBS rule in the subsequent discussion. Recall that the PWBF algorithm uses the FF of the IMWBF algorithm as the VN reliability metric with the FB set (12) determined by the FS count $\hat{F}_n$ of (13). That is, a CN sends a flip signal to its most unreliable linked VN only and the VNs which receive sufficient number of reliability warnings (flip signals) shall be flipped. It turns out that, with this extra filtering of VN-to-CN messages ($E_n$), and selective CN-to-VN flip signal passing, the PWBF algorithm is able to outperform the IMWBF decoder in both convergence rate and error rate [9]. This performance gain motivates us to ponder if a more elaborated flipping decision strategy that uses more information can bring about further performance improvement.

#### A. Flipping Intensity

Let $U_n \triangleq \sum_{m \in \mathcal{M}(n)} s_m, \mu_m \triangleq \max_{n \in \mathcal{N}(m)} U_n, \lambda_m \triangleq \arg \max_{n \in \mathcal{N}(m)} E_n$, and $\mathcal{M}'(n) = \{m|m \in \mathcal{M}(n), \lambda_m = n\}$. With these notations, we define the flipping intensity (FI) of (received by) $v_n$ as

$$
\hat{F}_n = \sum_{m \in \mathcal{M}'(n)} \theta_0 s_m \delta(U_{\lambda_m} - \mu_m) + \theta_1 s_m [1 - \delta(U_{\lambda_m} - \mu_m)], \quad (22)
$$

where $\theta_0 > \theta_1 \geq 0$ and $\delta(x)$ is the Kronecker delta function. The above definition implicitly implies that $\hat{F}_n = 0$ if $\mathcal{M}'(n) = \emptyset$ and only UCNs have a say in deciding FI. It also implies that a VN has a nonzero FI only if it has the largest FF value among $\mathcal{N}(m)$ and if it is connected to the largest number of UCNs among its peers in $\mathcal{N}(m)$, the associated FI should be even higher ($\theta_0 > \theta_1$). In both cases, a UCN $c_m$ will send a non-negative message to the VN with the highest FF value in the set $\mathcal{N}(m)$. However, if $c_m$ is a passed CN (PCN) ($s_m = 0$) and $d_c$ is small, it often implies that the tentative decisions of its linked VNs are all correct. Hence if the flipped bits are to be selected by checking whether the associated FI is greater than a threshold, $v_{\lambda_m}$ should have a smaller probability of being chosen. This can be done by having the PCN send a drag message $\theta_2(s_m - 1)$. But if there is doubt that $c_m$ is connected to even incorrect bit decisions, the PCN has better not sending such a message. We decide that this is likely to be the case if $U_{\lambda_m} \neq \mu_m$ for this inequality means that at least one VN in $\mathcal{N}(m)$ has more connected UCNs than $v_{\lambda_m}$. With UCNs and PCNs contributing opposite signals, we modify (22) for all $n$, $0 \leq n < N$ as

$$
\hat{F}_n = \sum_{m \in \mathcal{M}'(n)} [\theta_2(s_m - 1) + \theta_0 s_m] \delta(U_{\lambda_m} - \mu_m) + \theta_1 s_m [1 - \delta(U_{\lambda_m} - \mu_m)] \quad (23)
$$
where \( \theta_2 \geq 0 \). On the other hand, when \( d_c \) is large, it is less likely that \( s_m = 0 \) automatically implies correct decisions on all its linked bits and we thus stick to \( 22 \), having no PCN to contribute to \( \Delta FI \). Although the thresholds \( \theta_i \)'s can be any nonnegative real numbers, to simplify implementation, we let \( \theta_i \)'s be nonnegative integers such that the \( \Delta FI \) is integer-valued.

### B. Flipped-Bit Selection Rule

A simple \( \Delta FI \)-based FBS rule is to flip the bits in the FB set \( B = \{ n | \tilde{F}_n \geq \Delta FI \} \). But the optimal threshold \( \Delta FI \) is not easy to determine especially for a code with low VN degree. A smaller threshold may cause incorrect flipping decisions while a large threshold tends to slow down the convergence or even cause decoding failure as no VN meets the flipping requirement. To overcome this dilemma, we select a relative high \( \Delta FI \) threshold and use the FB set \( B = \{ n | \tilde{F}_n \geq \Delta FI \} \) if it is nonempty. Otherwise, \( B = \{ n | U_n = \max_{i \in T} U_i \} \) where \( T \triangleq \{ n | \tilde{F}_n = \max_j \tilde{F}_j \} \). We summarize below the new FBS rule as **Algorithm 3** which, for convenience of reference, is called M2 FBS rule.

**Algorithm 3 Flipped Bit Selection Rule M2**

1. For \( n = 0, 1, \ldots, N - 1 \), compute \( \tilde{F}_n \) by \( 22 \) or \( 23 \).
2. Find \( B = \{ n | \tilde{F}_n \geq \Delta FI \} \). If \( B \neq \emptyset \), stop; otherwise, go to Step 3.
3. Update \( T \) and find \( B = \{ n | U_n = \max_{i \in T} U_i \} \).

Loop-detection/breaking procedures can be included in our FBS algorithm if necessary. The loop detection scheme used \( 22 \) is an appropriate choice. When a loop is detected, we generate a disturbance on the tentative decoded sequence by switching to the FB set

\[
B = \{ n | U_n = \max_{i \in T} U_i \}. \tag{24}
\]

### C. Numerical Results

Note the proposed M2 FBS rule is independent of the FF and can be used in conjunction with different FFs no matter whether the checksum weights are constant or not. Different combinations of the FF, the weight-updating schedule, and the FBS rule used lead to different decoding algorithms. The error-rate performance and decoding speed of various combinations are presented in this subsection.

1) **Abbreviations and parameter values:** For convenience of reference, we adopt a systematic labeling scheme similar to that used in Section [VI-B]. We describe a decoding method by groups of capital letters separated by hyphens. The first group is used to indicate if single (S) or multiple (M) bits are to be flipped in an iteration and, for the latter case, if the simple FF based (M1) or the more complicated FI-based (M2) FBS rule is adopted. The second group contains the abbreviation of the known or proposed algorithm such as IMWBF, GDBF or DWBF, denoting the FF used. The third group tells whether a selective (A or B) or the full (F) weight-updating schedule is used. For examples, M1-DWBF-A refers to the decoder that uses the M1 FBS rule, the DWBF (**Algorithm 2**) decoding procedure, and SWUS-A for checksum weight update, and M2-GDBF is the decoder that uses the M2 FBS rule and the GDBF algorithm. For known constant weight algorithms without FBS modification and SWUS, we keep conventional abbreviations like AMWBF, IPWBF, and HGDBF only.

For the decoders based on the new FBS rule M2, we use the FI thresholds \( \theta_0 = 3 \), \( \theta_1 = 2 \), and \( \theta_2 = 1 \), respectively. For the M1/M2-DWBF-A and M1/M2-DWBF-B algorithms using Code 1, \( \alpha_2 \)'s are 0.58 and 0.35, respectively while \( \alpha_1 \), \( \Delta FS \) are \( (0.2, 1) \) and \( \alpha_1 = 0.2 \) for the IPWBF and M2-IMWBF algorithms and \( \alpha_1 = 0.2 \) for the AMWBF algorithm. These parameter values are obtained through numerical optimization. We also find \( \Delta FI = 1 \) for both the M2-GDBF and M1-DWBF-A/B algorithms. For simplicity, the FF clipping threshold \( \eta \) is set to zero. The remaining parameters needed for the HGDBF and IPWBF algorithms are also optimized.

For decoding Code 2 with the M1/M2-DWBF-A and M1/M2-DWBF-B algorithms, the damping factor \( \alpha_2 \)'s are 0.33 and 0.4. Furthermore, \( \Delta FI = 4 \) and 1 for the M1-DWBF-A and M1-DWBF-B decoders, respectively. When using the IPWBF algorithm, we set \( \alpha_1 = 0.5 \) and \( \alpha_1 = 0.2 \) for the AMWBF algorithm. These parameter values are obtained through numerical optimization. The M2-IMWBF algorithm uses \( \alpha_1 = 0.2 \), \( \Delta FS = 1 \); while the M2-GDBF algorithm uses the modified FF \( 21 \) and \( \alpha_3 = 0.5 \).

2) **BER and FER performance:** Fig. 6 shows the BER performance of different multi-bit BF algorithms for Code 1 when \( f_{\max} = 50 \). The effectiveness of the M2 FBS rule can also be verified by comparing the required \( E_b/N_0 \) for BER = \( 10^{-5} \): the M2-IMWBF decoder outperforms the IPWBF decoder by approximately 1.7 dB and the M2-GDBF algorithm has a 0.4 dB gain over the HGDBF algorithm. The DWBF algorithms yields significant BER performance improvement even with the simple M1 rule, and when the M2 rule is used its performance becomes very close to that provided by the SPA algorithm when BER < \( 10^{-6} \).

The convergence behaviors of these algorithms are shown in
Figs. 7 and 8. The results show that the M2 rule gives better BER performance and, for both the DWBF and M2-GDBF algorithms, the convergence rate is improved as well.

Note that in Figs. 6-8, loop-detecting/breaking schemes are activated for all but the M2-DWBF-B algorithm. In general, loops are much less likely to occur in a DWBF decoder than in a static CN weight decoder. When the FWUS or SWUS-B is used, we are unable to detect any loop for both codes in all our simulation efforts whence a loop breaker is not needed. This is because the time-varying checksum weights of the DWBF algorithm and wider message propagation ranges of FWUS or SWUS-B schedules have made the BF decision related variables, $E_n, U_n, \mu_m$, and $\tilde{F}_n$, to have much larger dynamic ranges.

The BER (with $l_{\max} = 20$) and frame error rate (FER) convergence performance of various multi-bit BF decoders for Code 2 are respectively presented in Fig. 9 and 10. By comparing the two sets of BER curves, M2-GDBF versus HGDBF and M2-IMWBF versus AMWBF, we verify the effectiveness of the new FBS (M2) rule. Although the M2-IMWBF algorithm yields the same converged BER as the IP-WBF decoder for this code, it gives better FER performance in the first few iterations. We also find that the M2-DWBF-A (B) decoder is superior to the M1-DWBF-A (B) decoder in both BER performance and decoding speed. Only the AMWBF and HGDBF decoders need a loop-breaker in decoding Code 2 since for the other decoders, loops are rarely detected.

We want to remark that although the M1 FBS rule is simpler, our simulations indicate that the M2 FBS rule can significantly reduce the probability of decoding loops no matter it is used in conjunction with the DWBF, GDBF, or IMWBF decoders. This is particular useful when using conventional FFs to decode low-degree codes.

D. Complexity Analysis

The computing complexity of a BF decoding algorithm consists of three parts: a) weight/belief update, b) FF update, and c) flipped bits selection. Once new beliefs, $-r_m(1 - 2s_m)$, are
available, the FF update is just adding all returned CN beliefs and \(-\alpha_1|y_n|\) or \(-y_n(1-2\hat{u}_n)\). There is little difference among the BF decoders in FF computing. Therefore, to compare the complexity of different BF decoders, we only have to consider a) and c). Updating the weights associated to CN \(c_m\), i.e., the beliefs to be sent to its connected VNs, requires 2\(d_e - 3\) real comparisons for finding the smallest and second smallest \(-E_{\alpha}, n \in N(m)\) and two real comparisons with the threshold \(\eta\). As for c), Step 1 of Algorithm 4 requires \(d_e - 1\) real and \(d_c\) integer comparisons per CN for finding the corresponding \(\lambda_m\) and \(\mu_m\) and checking if \(U_{\lambda_m} = \mu_m\) in computing \(\tilde{F}_n\) via (23) or (24). If \(B = \emptyset\), Step 3, which calls for additional integer comparisons, is needed. The FBS rule of the IPWBF algorithm involves the computation of FSs (12)–(13) which, on the average, needs \(M_U/(d_e - 1)\) real comparisons per CN where \(M_U\) is the average UCN number per iteration. Since the HGDBF and AMWBF algorithms use (10) in their FBS rule, they do not have to perform these operations.

Table III shows the average FBS operations per codeword (a total of \(10^6\) codewords were simulated) at selected iterations for various decoding algorithms. A single FBS operation for the IPWBF decoder and those using the M2 BFS rule refer to the respective computing efforts analyzed in the preceding paragraph. Since Code 1 uses (23) as FI, all CNs have to select flipped bits, the average number of CN operations is simply the product of \(M\) (the number of CNs) and the corresponding FER at a given iteration. Code 2 uses (22) instead; the average number is the product of the average UCN number and the corresponding FER. The definition of the average visited CN number is the same as that defined in Table II. This table and Fig. 6–10 provide information for considering tradeoffs between performance and complexity when combining different FBS rules, weight-updating schedules, and FFs. All FF updates discussed have almost the same computation complexity; the only exception is that used by GDBF algorithms, both (5) and (21) are simpler than other FFs.

Among the the decoding algorithms compared in Table III, the IPWBF algorithm uses a simpler FBS operation (14) but it has to perform a delay-handling process in every iteration plus an initial bootstrapping step. These two extra operations need off-line computing effort in searching for the corresponding optimal parameter values. They also require additional storage and computing complexity. Although the HGDBF algorithm needs not make the FF value and UCN number comparisons, three real thresholds, one for the multi-bit flipping mode and two for the escape (loop-breaking) process are required in its FBS rule, resulting extra off-line search, real-time comparisons, and random variable generation. Other offline efforts include the searches for \(\alpha_1\) (M2-IMWBF), \(\alpha_2\) (M1- and M2-DWBF-A/B), and \(\Delta_{FI}\), which is located within \(-d_e\theta_2 \leq \Delta_{FI} \leq d_e\theta_1\) and should be jointly optimized with the \(\alpha_i\)’s. For the M2-GDBF algorithm, only \(\Delta_{FI}\) is needed to be optimized.

The loop-breaking scheme (24) is simpler than those used by other decoding algorithm and more effective than the methods used by the IPWBF and AMWBF algorithms which remove the bit(s) having maximum \(F_n\) or \(E_n\) from \(B\); when \(B = \emptyset\), the decoding process will be forced to terminate after the removal. Instead of reducing \(|B|\), the escape process (24) and that used by the HGDBF algorithm perturb the tentative decoded sequence to break a loop. The latter, however, has to generate a Gaussian random variable and perform real comparison for every VN.

VI. Conclusion

A typical BF decoding algorithm for LDPC codes consists of three major components, namely VN belief (FF) computing, CN (checksum) belief update, and FBS rule. These three components determine the performance of a BF decoder. As the FF format for all BF decoding algorithms is very similar, the parallel BP operation is a major complexity concern for LDPC decoders, we thus prioritize the beliefs and propose selective BP (weight-updating) schedules to reduce the implementation complexity and provide trade-offs between complexity and performance.

Different combinations of checksum belief update method, FBS rule and selective BP result in different decoder structures. We simulate the error rate and convergence performance of various combinations and the resulting numerical behaviors confirm the effectiveness of our new designs.

REFERENCES

[1] D. J. C. MacKay, “Good error-correcting codes based on very sparse matrices,” IEEE Trans. Inf. Theory, vol. 45, pp. 399-431, Mar. 1999.
[2] R. G. Gallager, *Low-density parity-check codes*, Cambridge, MA: MIT Press, 1963.

[3] Y. Kou, S. Lin and M. Fossorier, “Low-density parity-check codes based on very sparse matrices,” *IEEE Trans. Inf. Theory*, vol. 47, pp.2711-236, Nov. 2001.

[4] J. Zhang and M. Fossorier, “A modified weighted bit-flipping decoding of low-density parity-check codes,” *IEEE Commun. Lett.*, vol. 8, no. 3, pp. 165-167, Mar. 2004.

[5] M. Jiang, C. Zhao, Z. Shi, and Y. Chen, “An improvement on the modified weighted bit flipping decoding algorithm for LDPC codes,” *IEEE Commun. Lett.*, vol. 9, no. 9, pp. 814-816, Sep. 2005.

[6] F. Guo and L. Hanzo, “Reliability ratio based weighted bit-flipping decoding for low-density parity-check codes,” *IEEE Electron. Lett.*, vol. 40, no. 21, pp. 1356-1358, Oct. 2004.

[7] T. Ngatched, F. Takawira, and M. Bossert, “A Modified bit-flipping decoding algorithm for low-density parity-check codes”, in *Proc. 2007 IEEE Int. Conf. Commun.*, Glasgow, 2007, pp. 653-658.

[8] T. Wadayama et al., “Gradient descent bit flipping algorithms for decoding LDPC codes,” *IEEE Trans. Commun.*, vol. 58, no. 6, pp. 1610-1614, Jun. 2010.

[9] X. Wu, C. Zhao, and X. You, “Parallel weighted bit-flipping decoding,” *IEEE Commun. Lett.*, vol. 11, no. 8, pp. 671-673, Aug. 2007.

[10] G. Li, G. Feng, “Improved parallel weighted bit-flipping decoding algorithm for LDPC codes,” *IET Commun.*, vol. 3, Iss. 1, pp. 91-99, Jun. 2009.

[11] J. Cho and W. Sung, “Adaptive threshold technique for bit-flipping decoding of low-density parity-check codes,” *IEEE Commun. Lett.*, vol. 14, no. 9, pp. 857-859, Sep. 2010.

[12] R. Haga and S. Usami, “Multi-bit flip type gradient descent bit flipping decoding using no thresholds,” in *Proc. 2012 Int. Sym. Inform. Theory and its Appl.*, Honolulu, 2012, pp. 6-10.

[13] M. Ismail et al., “Low latency low power bit flipping algorithms for LDPC decoding,” in *Proc. IEEE 21st Int. Sym. Personal Indoor and Mobile Radio Commun.*, Instanbul, 2010, pp. 278-282.

[14] T. Chen, “Adaptive-weighted multibit-flipping decoding of low-density parity-check codes based on ordered statistics,” *IET Commun.*, vol. 7, Iss. 14, pp. 1517-1521, Jul. 2013.

[15] M. Yazdani, S. Hemati, and A. Banihashemi, “Improving belief propagation on graphs with cycles,” *IEEE Commun. Lett.*, vol. 8, no. 1, pp. 57-59, Jan. 2004.

[16] D. J. C. MacKay, *Encyclopedia of sparse graph codes* [Online]. Available: http://www.inference.phy.cam.ac.uk/mackay/ codes/data.html