Noisy Gradient Descent Bit-Flip Decoding for LDPC Codes

Gopalakrishnan Sundararajan, *Student Member, IEEE*, Chris Winstead, *Senior Member, IEEE*, and Emmanuel Boutillon, *Senior Member, IEEE*

***Abstract*—A modified Gradient Descent Bit Flipping (GDBF) algorithm is proposed for decoding Low Density Parity Check (LDPC) codes on the binary-input additive white Gaussian noise channel. The new algorithm, called Noisy GDBF (NGDBF), introduces a random perturbation into each symbol metric at each iteration. The noise perturbation allows the algorithm to escape from undesirable local maxima, resulting in improved performance. A combination of heuristic improvements to the algorithm are proposed and evaluated. When the proposed heuristics are applied, NGDBF performs better than any previously reported GDBF variant, and comes within 0.5 dB of the belief propagation algorithm for several tested codes. Unlike other previous GDBF algorithms that provide an escape from local maxima, the proposed algorithm uses only local, fully parallelizable operations and does not require computing a global objective function or a sort over symbol metrics, making it highly efficient in comparison. The proposed NGDBF algorithm requires channel state information which must be obtained from a signal to noise ratio (SNR) estimator. Architectural details are presented for implementing the NGDBF algorithm. Complexity analysis and optimizations are also discussed.**

***Index Terms*—GDBF, bit flipping, LDPC, weighted bit flipping, noisy GDBF.**

I. Introduction

LOW DENSITY PARITY CHECK (LDPC) codes gained considerable research attention in recent years. Due to their powerful decoding performance, LDPC codes are increasingly deployed in communication standards. The performance and cost of using LDPC codes are partly determined by the choice of decoding algorithm. LDPC decoding algorithms are usually iterative in nature. They operate by exchanging messages between basic processing nodes. Among the various decoding algorithms, the soft decision Belief Propagation (BP) algorithm and the approximate Min-Sum (MS) algorithm offer the best performance on the binary-input additive white Gaussian noise (AWGN) channel [1], [2], but these algorithms require a large number of arithmetic operations repeated over many iterations. These operations must be implemented with some degree of parallelism to support the throughput requirements of modern communication systems [3], [4]. As a result, LDPC decoders can be highly complex devices.

Significant effort has been invested to develop reduced-complexity decoding algorithms known "bit-flipping" decoders. These algorithms are similar in complexity to hard-decision decoding algorithms, but obtain improved performance by accounting for soft channel information. In most bit-flipping algorithms, the symbol node updates are governed by an *inversion function* that estimates the reliability of received channel samples. The inversion function includes the received channel information in addition to the hard-decision syndrome components obtained from the code's parity-check equations. In the so-called *single* bit-flipping algorithms, the least reliable bit is flipped during each iteration. In *multiple* bit-flipping algorithms, any bit is flipped if its reliability falls below a designated threshold, hence multiple bits may be flipped in parallel, allowing for faster operation.

A recently emerged branch of the bit-flipping family is Gradient Descent Bit Flipping (GDBF), which formulates the inversion function as a gradient descent problem. GDBF algorithms demonstrate a favorable tradeoff between performance and complexity relative to other bit-flipping algorithms. One difficulty for GDBF algorithms is that they are affected by undesirable local maxima which cause the decoder to converge on an erroneous message. Various schemes have been proposed to avoid or escape local maxima, but require additional complexity due to multiple thresholds or computing a global function over the code's entire block length. In this work, we propose an improved version of the GDBF algorithm, called Noisy GDBF (NGDBF) that offers a low-complexity solution to escape spurious local maxima. The proposed method works by introducing a random perturbation to the inversion function. The resulting algorithm provides improved performance and requires only local operations that can be executed fully in parallel (except for a global binary stopping condition). The proposed NGDBF algorithm comprises a set of heuristic methods that are empirically found to provide good performance for typical codes. Simulation results indicate that the NGDBF's optimal noise variance is proportional to the channel noise variance. This introduces a possible drawback compared to previous GDBF algorithms: NGDBF requires knowledge of the channel noise variance, which must be obtained from an estimator external to

Manuscript received February 7, 2014; revised June 24, 2014 and August 20, 2014; accepted August 29, 2014. Date of publication September 9, 2014; date of current version October 17, 2014. This work was supported by the US National Science Foundation under award ECCS-0954747, and by the Franco-American Fulbright Commission for the Exchange of Scholars. The work also used resources of the CPER PALMYRE II, with funding from FEDER and the region of Brittany, France. The work of G. Sundararajan is supported by a Sant Graduate Innovation Fellowship at Utah State University. The associate editor coordinating the review of this paper and approving it for publication was M. Lentmaier.

G. Sundararajan and C. Winstead are with the Department of Electrical and Computer Engineering, Utah State University, Logan, UT 84322-4120 USA (e-mail: gopal.sundar@aggiemail.usu.edu; chris.winstead@usu.edu).

E. Boutillon is with the lab-STICC, UMR 6285, Université de Bretagne Sud, 56321 Lorient, France (e-mail: emmanuel.boutillon@univ-ubs.fr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCOMM.2014.2356458

TABLE I
SUMMARY AND COMPARISON OF BIT-FLIPPING ALGORITHMS

Algorithm	Performance	Architecture	Arithmetic Complexity
BFA [5]	Poor	Parallel	Minimum
PBFA [6]	Fair	Parallel	Low
WBF [2]	Fair	Serial	Moderate
MWBF [24]	Good	Serial	Moderate
IMWBF [25]	Excellent	Serial	High
PWBF [7]	Excellent	Parallel	High
S-GDBF [9]	Fair	Serial	Low
M-GDBF [9]	Good	Mixed	Low
H-GDBF [9]	Excellent	Mixed	High
AT-GDBF [10]	Good	Parallel	Low
IGDBF [12]	Excellent	Mixed	Moderate
RRWGDBF [11]	Excellent	Parallel	High
Stoch. MTFM [17]	Excellent	Parallel	Low
*S-NGDBF	Fair	Serial	Low
*M-NGDBF	Good	Parallel	Low
*SM-NGDBF	Excellent	Parallel	Low-Moderate

* New algorithms described in this paper.

the decoder. Because of the heuristic nature of these results, the paper is organized to present the algorithm's technical details and empirical results first, followed by theoretical analyses that provide explanations for some of the observed results.

The remainder of this paper is organized as follows: Section II discusses the related work on bit-flipping algorithms, as well as some recently reported decoding algorithms that benefit from noise perturbations. Section III describes notation and summarizes the proposed NGDBF algorithm and its heuristic modifications. Section IV presents simulation results, and offers a comparative analysis of the various heuristics. Section V presents architectural simplifications and complexity analysis. Section VI presents an evaluation and comparison of the algorithms' convergence to the global maximum-likelihood (ML) solution, and Section VII presents an analysis of some of NGDBF's heuristics—namely threshold adaptation and syndrome weighting—which are interpreted in terms of evolving ML decisions on the local neighborhoods of symbol nodes. Conclusions are presented in Section VIII.

II. RELATED WORK

This section presents a review of bit-flipping algorithms and other methods related to the new NGDBF algorithms described in this article. As an aid to the reader, a qualitative summary of the considered bit-flipping algorithms and their comparative characteristics are provided in Table I. The performance and complexity comparisons in Table I are qualitative estimates made solely within the family of bit-flipping algorithms.

The original bit-flipping algorithm (BFA) was introduced by Gallager in his seminal work on LDPC codes [5]. Gallager's BFA is a hard-decision algorithm for decoding on the binary symmetric channel (BSC), in which only hard channel bits are available to the decoder. To correct errors, the BFA computes a sum over the adjacent parity-check equations for each bit in the code. If, for any bit, the number of adjacent parity violations exceeds a specified threshold, then the bit is flipped. This process is repeated until all parity checks are satisfied, or until a maximum iteration limit is reached. The BFA has very low complexity since it only requires, in each iteration, a summation over binary parity-check values for each symbol; however the BFA provides weak decoding performance. Miladinovic *et al.* considered a probabilistic BFA (PBFA) which adds randomness to the bit-flip decision, resulting in improved performance [6]. In PBFA, when a bit's parity-check sum crosses the flip threshold, it is flipped with probability p. The parameter p is optimized empirically and is adapted towards one during successive iterations.

Kou *et al.* introduced the Weighted Bit-Flipping (WBF) algorithm which improves performance over the BFA by incorporating soft channel information, making it better suited for use on the Additive White Gaussian Noise (AWGN) channel and other soft-information channels [2]. In the WBF algorithm, all parity-check results are weighted by a magnitude that indicates reliability. For each parity-check, the weight value is obtained by finding the lowest magnitude in the set of adjacent channel samples. During each iteration, a summation E_k is computed over the adjacent weighted parity-check results for each symbol position k. The symbol with the maximum E_k (or minimum, depending on convention) is flipped. The weights are only calculated once, at the start of decoding, however the WBF algorithm requires at every iteration a summation over several weights for each symbol—a substantial increase in complexity compared to the original BFA. In addition to the increased arithmetic complexity, WBF has two major drawbacks: first, a potentially large number of iterations are required because only one bit may be flipped in each iteration. Second, the algorithm must perform a global search to find the maximum E_k out of all symbols, resulting in a large latency per iteration that increases with codeword length, thereby hindering a high-throughput implementation.

Researchers introduced several improvements to the WBF. Zhang *et al.* introduced the Modified WBF (MWBF) algorithm, which obtained improved performance with a slight increase in complexity. Jiang *et al.* described another Improved MWBF (IMWBF) algorithm which offered further improvement by using the parity-check procedure from the MS algorithm to determine the parity-check weights—another substantial increase in complexity. Both of these methods inherit the two key drawbacks associated with single-bit flipping in the WBF algorithm.

Recently, Wu *et al.* introduced a Parallel WBF (PWBF) algorithm, which reduces the drawbacks associated with single-bit flipping in the other WBF varieties [7]. In the PWBF algorithm, the maximum (or minimum) E_i metric is found within the subset of symbols associated with each parity-check. The authors of [7] also developed a theory relating PWBF to the BP and MS algorithms, and showed that PWBF has performance comparable to IMWBF [8]. In the PWBF algorithm, it is still necessary to find the maximum E_i from a set of values, which costs delay, but the set size is significantly reduced compared to the other WBF methods, and it is independent of codeword length. In spite of these improvements, PWBF retains the complex arithmetic associated with IMWBF.

To reduce the arithmetic complexity of bit-flipping algorithms, Wadayama *et al.* devised the GDBF algorithm as

a gradient-descent optimization model for the ML decoding problem [9]. Based on this model, the authors of [9] obtained single-bit and multi-bit flipping algorithms that require mainly binary operations, similar to the original BFA. The GDBF methods require summation of binary parity-check values, which is less complex than the WBF algorithms that require summation over independently weighted syndrome values. The single-bit version of the GDBF algorithm (S-GDBF) requires a global search to discover the least reliable bit at each iteration. The multi-bit GDBF algorithm (M-GDBF) uses local threshold operations instead of a global search, hence achieving a faster initial convergence. In practice, the M-GDBF algorithm did not always provide stable convergence to the final solution. To improve convergence, the authors of [9] adopted a mode-switching strategy in which M-GDBF decoding is always followed by a phase of S-GDBF decoding, leveraging high-speed in the first phase and accurate convergence in the second.

Although the mode-switching strategy provided a significant benefit, the algorithm was still subject to spurious local maxima. Wadayama *et al.* obtained further improvements by introducing a "hybrid" GDBF algorithm (H-GDBF) with an escape process to evade local maxima. The H-GDBF algorithm obtains performance comparable to MS, but the escape process requires evaluating a global objective function across all symbols. When the objective function crosses a specified threshold during the S-GDBF phase, the decoder switches back to M-GDBF mode, then back to S-GDBF mode, and so on until a valid result is reached. To date, H-GDBF is the best performing GDBF variant, but requires a maximum of 300 iterations to obtain its best performance, compared to 100 for M-GDBF and S-GDBF. The major disadvantages of this algorithm are its use of multiple decoding modes, the need to optimize dual thresholds for mode switching and bit flipping, the global search operation and the global objective function used for mode switching. These global operations require an arithmetic operation to be computed over the entire code length, and would be expensive to implement for practical LDPC codes with large codeword length.

Several researchers proposed alternative GDBF algorithms to obtain fully parallel bit-flipping and improved performance. Ismail *et al.* proposed an Adaptive Threshold GDBF (AT-GDBF) algorithm that achieves good performance without the use of mode-switching, allowing for fully-parallel operation [10]. The same authors also introduced an early-stopping condition (ES-AT-GDBF) that significantly reduces the average decoding iterations at lower Signal to Noise Ratio (SNR). Phromsa-ard *et al.* proposed a more complex Reliability-Ratio Weighted GDBF algorithm (RRWGDBF) that uses a weighted summation over syndrome components with an adaptive threshold to obtain reduced latency [11]. The RRWGDBF method has the drawback of increased arithmetic complexity because it performs a summation of weighted syndrome components, similar to previous WBF algorithms. Haga *et al.* proposed an improved multi-bit GDBF algorithm (IGDBF) that performs very close to the H-GDBF algorithm, but requires a global sort operation to determine which bits to flip [12].

These GDBF algorithms can be divided into two classes: First, the low-complexity class, which includes S-GDBF and M-GDBF, AT-GDBF and RRWGDBF; in this class, mode-switching M-GDBF is the best performer. For low-complexity algorithms, the typical maximum number of iterations is reported as $T = 100$. Second is the high-performance class, which includes H-GDBF and IGDBF. In the high-performance class, significant arithmetic complexity is introduced and a larger number of iterations is reported, $T = 300$. H-GDBF is the best performer in this class, and in this paper we consider H-GDBF as representative of the high-performance GDBF algorithms.

In this work, we propose a new Noisy GDBF algorithm with single-bit and multi-bit versions (S-NGDBF and M-NGDBF, respectively). The M-NGDBF algorithm proposed in this work employs a single threshold and also provides an escape from the neighborhood of spurious local maxima, but does not require the mode-switching behavior used in the original M-GDBF. The proposed algorithm also avoids using any sort or maximum-value operations. When using the threshold adaptation procedure borrowed from AT-GDBF, as described in Section III-D, the proposed M-NGDBF achieves performance close to the H-GDBF and IGDBF methods at high SNR, with a similar number of iterations. We also introduce a new method called *Smoothed* M-NGDBF (SM-NGDBF) that contributes an additional 0.3 dB gain at the cost of additional iterations. It should be noted that Wadayama *et al.* proposed using a small random perturbation in the H-GDBF thresholds [9]; the NGDBF methods use a larger perturbation in combination with other heuristics to obtain good performance with very low complexity.

Because of its reliance on pseudo-random noise and single-bit messages, the proposed NGDBF algorithms bear some resemblance to the family of stochastic iterative decoders that were first introduced by Gaudet and Rapley [13]. One of the authors (Winstead) introduced stochastic decoding for codes with loopy factor graphs [14], and Sharifi-Tehrani *et al.* later demonstrated stochastic decoding for LDPC codes [15], [16]. High throughput stochastic decoders have been more recently demonstrated by Sharifi Tehrani *et al.* [17]–[19] and by Onizawa *et al.* [20]. Stochastic decoders are known to have performance very close to BP, allow for fully-parallel implementations, and use very simple arithmetic while exchanging single-bit messages. They may therefore serve as an appropriate benchmark for comparing complexity against the proposed SM-NGDBF algorithm (an analysis of comparative complexity is presented in Section V-C).

In addition to recent work on low-complexity decoding, there has also been some exploration of noise-perturbed decoding using traditional MS and BP algorithms. Leduc *et al.* demonstrated a beneficial effect of noise perturbations for the BP algorithm, using a method called dithered belief propagation [21]. Kameni Ngassa *et al.* examined the effect of noise perturbations on MS decoders and found beneficial effects under certain conditions [22]. The authors of [23] offered the conjecture that noise perturbations assist the MS algorithm in escaping from spurious fixed-point attractors, similar to the hypothesis offered in this paper to motivate the NGDBF algorithm.

Up to now, there is not yet a developed body of theory for analyzing noise-perturbed decoding algorithms, and the recent

research on this topic tends to adopt a heuristic approach. In this paper we also adopt the heuristic approach, and demonstrate through empirical analysis that noise perturbations improve the performance of GDBF decoders.

III. Proposed Noisy GDBF Algorithm

A. Notation

Let $\mathbf{H}$ be a binary $m \times n$ parity check matrix, where $n > m \geq 1$. To $\mathbf{H}$ is associated a binary linear code defined by $\mathcal{C} \triangleq \{c \in F_2^n : \mathbf{Hc} = 0\}$, where F_2 denotes the binary Galois field. The set of bipolar codewords, $\hat{\mathcal{C}} \subseteq \{-1, +1\}^n$, corresponding to $\mathcal{C}$ is defined by $\hat{\mathcal{C}} \triangleq \{(1-2c_1), (1-2c_2), \ldots, (1-2c_n) : c \in \mathcal{C}\}$. Symbols are transmitted over a binary input AWGN channel defined by the operation $\mathbf{y} = \hat{\mathbf{c}} + \mathbf{z}$, where $\hat{\mathbf{c}} \in \hat{\mathcal{C}}$, $\mathbf{z}$ is a vector of independent and identically distributed Gaussian random variables with zero mean and variance $N_0/2$, N_0 is the noise spectral density, and $\mathbf{y}$ is the vector of samples obtained at the receiver.

We define a decision vector $\mathbf{x} \in \{-1, +1\}^n$. We say that $\mathbf{x}(t)$ is the decision vector at a specific iteration t, where t is an integer in the range $[0, T]$, and T is the maximum number of iterations permitted by the algorithm. In iterative bit-flipping algorithms, the decision values may be flipped one or more times during decoding. We will often omit the dependence on t when there is no ambiguity. The decision vector is initialized as the sign of received samples, i.e., $x_k(t=0) = \text{sign}(y_k)$ for $k = 1, 2, \ldots, n$.

The parity-check neighborhoods are defined as $\mathcal{N}(i) \triangleq \{j : h_{ij} = 1\}$ for $i = 1, 2, \ldots, m$, where h_{ij} is the (i,j)th element of the parity check matrix $\mathbf{H}$. The symbol neighborhoods are defined similarly as $\mathcal{M}(j) \triangleq \{i : h_{ij} = 1\}$ for $j = 1, 2, \ldots, n$. The code's parity check conditions can be expressed as bipolar syndrome components $s_i(t) \triangleq \prod_{j \in \mathcal{N}(i)} x_j(t)$ for $i = 1, 2, \ldots, m$. A parity check node is said to be *satisfied* when its corresponding syndrome component is $s_i = +1$.

B. GDBF Algorithm

The GDBF algorithm proposed in [9] was derived by considering the maximum likelihood problem as an objective function for gradient descent optimization. The standard ML decoding problem is to find the decision vector $\mathbf{x}_{\text{ML}} \in \hat{\mathcal{C}}$ that has maximum correlation with the received samples $\mathbf{y}$:

$$\mathbf{x}_{\text{ML}} = \arg\max_{\mathbf{x} \in \hat{\mathcal{C}}} \sum_{k=1}^{n} x_k y_k. \tag{1}$$

To include information from the code's parity check equations, the syndrome components are introduced as a penalty term, resulting in the objective function proposed by Wadayama *et al.*:

$$f(\mathbf{x}) = \sum_{k=1}^{n} x_k y_k + \sum_{i=1}^{m} s_i. \tag{2}$$

In the GDBF algorithm, a stopping criterion is used to enforce the condition $\mathbf{x} \in \hat{\mathcal{C}}$, i.e., any allowable solution $\mathbf{x}$ must be a valid codeword. Under this constraint, a solution that maximizes the objective function (2) is also a solution to the ML problem defined by (1). This is because for any valid codeword $\mathbf{x}$, the summation $\sum_{i=1}^{m} s_i$ is constant and equal to m. Since the objective functions in (1) and (2) differ only by a constant term, they must have the same maxima and minima.

By taking the partial derivative with respect to a particular symbol x_k, the local inversion function is obtained as

$$E_k = x_k \frac{\partial f(\mathbf{x})}{\partial x_k} = x_k y_k + \sum_{i \in \mathcal{M}(k)} s_i. \tag{3}$$

Wadayama *et al.* showed that the objective function can be increased by flipping one or more x_k with the most negative E_k values. The resulting iterative maximization algorithm is described as follows:

Step 1: Compute syndrome components $s_i = \prod_{j \in \mathcal{N}(i)} x_j$, for all $i \in \{1, 2, \ldots, m\}$. If $s_i = +1$ for all i, output x and stop.

Step 2: Compute inversion functions. For $k \in \{1, 2, \ldots, n\}$ compute

$$E_k = x_k y_k + \sum_{i \in \mathcal{M}(k)} s_i.$$

Step 3: Bit-flip operations. Perform one of the following:
- a) Single-bit version: Flip the bit x_k for $k = \arg\min_{k \in \{1,2,\ldots,n\}} E_k$.
- b) Multi-bit version: Flip any bits for which $E_k < \theta$, where $\theta \in \mathbb{R}^-$ is the *inversion threshold*.

Step 4: Repeat steps 1 to 3 till a valid codeword is detected or maximum number of iterations is reached.

The inversion threshold is a negative real number, i.e., $\theta < 0$, to ensure that only bits with negative-valued E_k are flipped. The optimal value of θ is found empirically, as discussed in Section IV-C. The single-bit GDBF algorithm (S-GDBF) incurs a penalty in parallel implementations due to the requirement of finding the minimum from among n values. The multi-bit version (M-GDBF) is trivially parallelized, but does not converge well because there tend to be large changes in the objective function after each iteration. The objective function increases rapidly during initial iterations, but is not able to obtain stable convergence unless a mechanism is introduced to reduce the flipping activity during later iterations. In this paper, we consider two such mechanisms: *mode-switching* and *adaptive thresholds*.

To improve performance, the authors of [9] proposed a mode-switching modification for M-GDBF, controlled by a parameter $\mu \in \{0, 1\}$: During a decoding iteration, if $\mu = 1$ then step 3b is executed; otherwise step 3a is executed. At the start of decoding, μ is initialized to 1. After each iteration, the global objective function (2) is evaluated. If, during any iteration t, $f(\mathbf{x}(t)) < f(\mathbf{x}(t-1))$, then μ is changed to 0. This modification adds complexity to the algorithm, but also significantly improves performance. In the sequel (Section III-D), it is explained that AT-GDBF eliminates the need for mode-switching by using the strictly parallel mechanism of adaptive thresholds [10].

C. Noisy GDBF

To provide a low-complexity mechanism to escape from local maxima in the GDBF algorithm, we propose to introduce a random perturbation in the inversion function. Based on this approach, we modify the step 2 of the GDBF algorithm as follows:

Step 2: Symbol node update. For $k = 1, 2, \ldots, n$ compute

$$E_k = x_k y_k + w \sum_{i \in \mathcal{M}(k)} s_i + q_k,$$

where $w \in \mathbb{R}^+$ is a syndrome weight parameter and q_k is a Gaussian distributed random variable with zero mean and variance $\sigma^2 = \eta^2 N_0/2$, where $0 < \eta \leq 1$. All q_k are independent and identically distributed.

In this step, a syndrome weighting parameter w is introduced. Syndrome weighting is motivated by the local maximum likelihood analysis presented in Section VII. Typically w and η are close to one, and are found through numerical optimization. The optimal values for w and η are code dependent, and are found to be weakly SNR dependent in some cases.

Throughout this paper, we refer to this algorithm and its variants as *Noisy* GDBF (NGDBF). Both single-bit and multi-bit versions are possible, and are indicated as S-NGDBF and M-NGDBF, respectively. In this paper, mode-switching is never used in association with NGDBF; instead, threshold adaptation is employed as explained in the next subsection.

The perturbation variance proportional to $N_0/2$ was chosen based on an intuition that the algorithm's random search region should cover the same distance as the original perturbation introduced by the channel noise. The noise-scale parameter η is introduced to fine-tune the optimal perturbation variance for each code. The effect of η on performance is studied empirically in Section IV-D. For some codes, good performance is obtained when using a single SNR-independent value for η. In other cases, η must be varied to get the best performance at different SNR values.

D. Threshold Adaptation

Methods of threshold adaptation were previously investigated to improve the convergence of multi-bit flipping algorithms. In this paper we consider a local Adaptive Threshold GDBF (AT-GDBF) algorithm described by Ismail *et al.* [10] in which a separate threshold θ_k is associated with each symbol node. For $k = 1, 2, \ldots, n$, the threshold θ_k is adjusted in each iteration by adding these steps to the M-GDBF algorithm:

Step 0: Initialize $\theta_k(t = 0) = \theta$ for all k, where $\theta \in \mathbb{R}^-$ is the global initial threshold parameter.

Step 3b: For all k, compute the inversion function E_k. If $E_k(t) \geq \theta_k(t)$, make the adjustment $\theta_k(t+1) = \theta_k(t)\lambda$, where λ is a global adaptation parameter for which $0 < \lambda \leq 1$. If $E_k(t) < \theta_k(t)$, flip the sign of the corresponding decision x_k.

In practice, the adaptation parameter λ must be very close to one. The case $\lambda = 1.0$ is equivalent to non-adaptive M-GDBF. According to the authors of [10], AT-GDBF obtains the same performance as M-GDBF with mode-switching, hence it enables fully parallel implementation with only local arithmetic operations. In the sequel we will show that threshold adaptation significantly improves performance in the M-NGDBF algorithm, at the cost of some additional complexity in the bit-flip operations.

E. Output Decision Smoothing

Convergence failures in the M-NGDBF algorithm may arise from excessive flipping among low-confidence symbols. This may occur as a consequence of the stochastic perturbation term. In this situation, the decoder may converge *in mean* to the correct codeword, but that does not guarantee that it will satisfy all parity checks at any specific time prior to the iteration limit T. More precisely, suppose the decoder is in an initially correct state, i.e., initially all $x_k = \hat{c}_k$. When the inversion function is computed for some x_k, there is a non-zero probability of erroneous flipping due to the noise contribution:

$$p_{f,k} = \Pr\left(x_k y_k + w \sum_{i \in \mathcal{M}(k)} s_i + q_k < \theta \right). \tag{4}$$

Now suppose that p_f is the least among the $p_{f,k}$ values among all symbols. Then the probability P_F that at least one erroneous flip occurs in an iteration is bounded by

$$P_F \geq 1 - (1 - p_f)^n. \tag{5}$$

This probability approaches one as $n \to \infty$ for any $p_f > 0$. For a sufficiently large code, it would be unlikely to satisfy all checks in a small number of iterations, even if all decisions are initially correct.

This problem may be compensated by introducing an up/down counter at the output of every x_k. The counter consists of a running sum X_k for each of the N output decisions. At the start of decoding, the counters are initialized at zero. During each decoding iteration, the counter is updated according to the rule

$$X_k(t+1) = X_k(t) + x_k(t) \tag{6}$$

If the stopping criterion is met (i.e., all parity checks are satisfied) then x_k is output directly. If the iteration limit T is reached without satisfying the stopping condition, then the smoothed decision is $\overline{x}_k = \mathrm{sign}(X_k)$. In practice, the summation in (6) can be delayed until the very end of decoding. This saves activity in the up/down counter and hence reduces power consumption. Results in the sequel are obtained with summation only over the interval from $t = T - 64$ up to T. When using this procedure, we refer to the algorithm as the "smoothed" M-GDBF method, or in shortened form as SM-NGDBF.

IV. Simulation Results

A. BER Performance

The proposed NGDBF algorithms were simulated on an AWGN channel with binary antipodal modulation using various

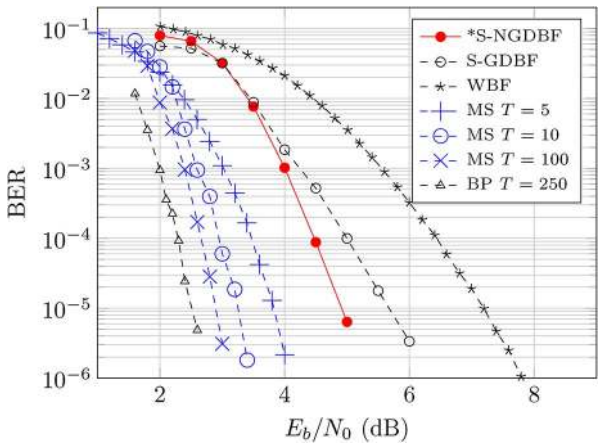

Fig. 1. BER versus E_b/N_0 curves for S-NGDBF with $T = 100$ and $\eta = 1.0$ using the rate 1/2 PEGReg504x1008 code simulated over an AWGN channel with binary antipodal modulation. The newly proposed S-NGDBF algorithm is indicated by an asterisk $(*)$.

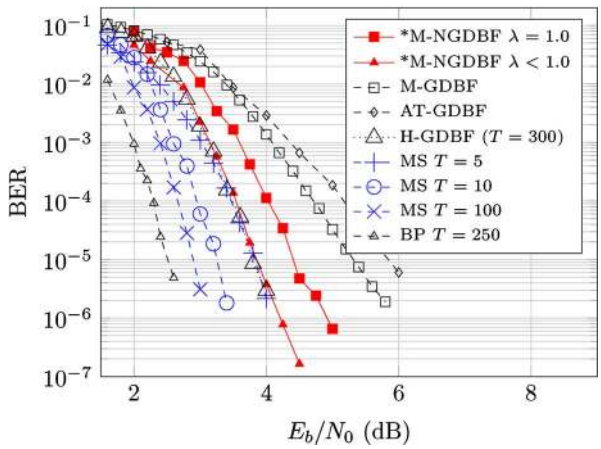

Fig. 2. BER versus E_b/N_0 curves for M-NGDBF with $T = 100$ using the rate 1/2 PEGReg504x1008 code simulated over an AWGN channel with binary antipodal modulation. The newly proposed M-NGDBF algorithms (adaptive and non-adaptive) are indicated by asterisks $(*)$. Several other known algorithms are shown for comparison, including M-GDBF ($T = 100$), AT-GDBF ($T = 100$) and H-GDBF with escape process ($T = 300$).

regular LDPC codes selected from MacKay's online encyclopedia [26] (all selected codes are partially irregular in parity-check degree, but are still considered regular codes). For each code, the NGDBF decoding parameters, including θ, λ, η, and w, were optimized one at a time, holding fixed values for all but one parameter. The free parameter was adjusted using a successive approximation procedure, repeating the BER simulation in each trial, and iteratively shrinking the search domain until the best value was found. This procedure was repeated for each parameter to obtain good-performing parameters.

All NGDBF algorithms were evaluated for the rate 1/2 (3, 6) regular LDPC code identified as PEGReg504x1008 in MacKay's encyclopedia, which is commonly used as a benchmark in previous papers on WBF and GDBF algorithms. Because our primary attention is directed at SM-NGDBF, additional simulations were performed to verify this algorithm on the rate 1/2 regular (4, 8) code identified as 4000.2000.4.244, and on the rate 0.9356 (4, 62) code identified as 4376.282.4.9598. Unless stated otherwise, all simulations use double precision floating-point arithmetic, channel samples are saturated at $Y_{\max} = 2.5$, and the syndrome weighting is $w = 0.75$.

In each simulation, comparison results are provided for the BP algorithm with 250 iterations, for the MS algorithm with 5, 10, and 100 iterations. Additional appropriate comparisons are described for each result presented in this section. The MS results presented here represent the strict MS algorithm, i.e., they do not reflect performance for offset-MS or normalized-MS. For the M-GDBF results, the mode-switching procedure was used to obtain the best performance in all cases.

To verify the beneficial effect of added noise, we first verified the S-NGDBF algorithm for the PEGReg504x1008 code with $T = 100$. The results are shown in Fig. 1, with comparative results for S-GDBF ($T = 100$), WBF ($T = 100$), MS and BP. The results show a gain approaching 1 dB for S-NGDBF compared to S-GDBF. This provides a basic empirical validation for the NGDBF concept.

Simulation results for the M-NGDBF algorithm are shown in Fig. 2. The results in this figure were obtained for the PEGReg504x1008 code with $T = 100$. The M-NGDBF results are shown for the non-adaptive case $(\lambda = 1.0)$ and for the adaptive-threshold case with initial threshold $\theta = -0.9$ and $\eta = 0.95$, where η is the noise-scale parameter described in Section III-C. For SNR < 3.5 dB, the best performance was obtained with an adaptation parameter of $\lambda = 0.99$. At higher SNR values, λ was decreased to 0.97 at 3.5 dB, 0.94 at 4.0 dB, and 0.9 at 4.25 dB and 4.5 dB. The performance of adaptive M-NGDBF is nearly identical to that of H-GDBF with escape process, which requires $T = 300$, and is also very close to MS with $T = 5$.

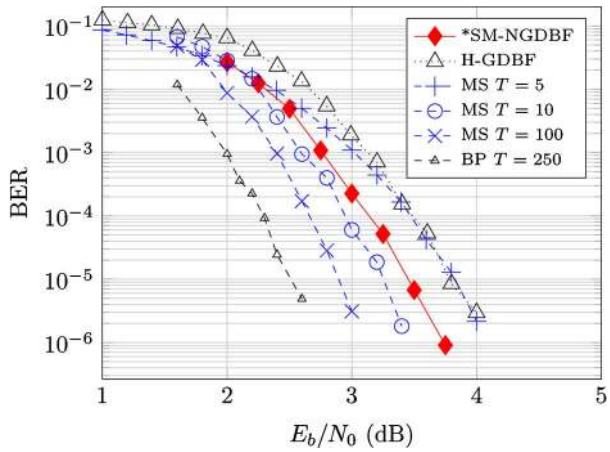

Fig. 3. BER versus E_b/N_0 curves for SM-NGDBF and H-GDBF with $T = 300$ using the rate 1/2 PEGReg504x1008 code over an AWGN channel with binary anitpodal modulation. The proposed algorithms are indicated by an asterisk $(*)$.

Results for the SM-NGDBF algorithm are shown in Fig. 3. For SM-NGDBF with $T = 100$, performance was equal to M-NGDBF (i.e., there was no gain from smoothing when $T = 100$), so these results are not shown. The most improved results were obtained with $T = 300$, the same number of iterations used for H-GDBF with escape process. SM-NGDBF is found to achieve about 0.3 dB of coding gain compared to H-GDBF, for the same value of T. Compared to M-NGDBF, SM-NGDBF is found to achieve about 0.3 dB of coding gain, at the cost of

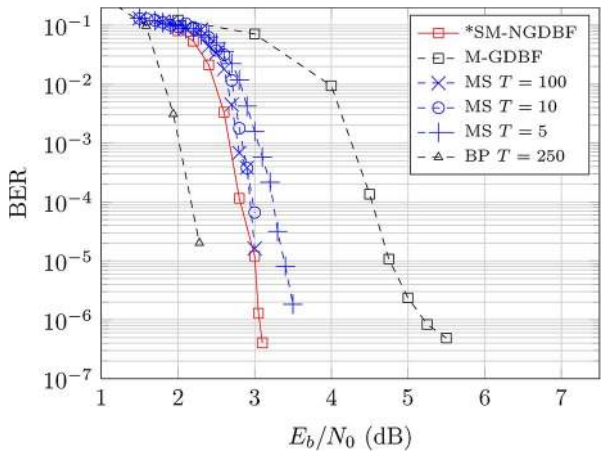

Fig. 4. BER versus E_b/N_0 curves for SM-NGDBF with $T = 300$ using the rate 1/2 4000.2000.4.244 code over an AWGN channel with binary antipodal modulation. These results were obtained using $\theta = -0.9$, $\lambda = 0.99$, and η varied between 0.625 at low SNR (below 2.8) and 0.7 at higher SNR (above 2.8).

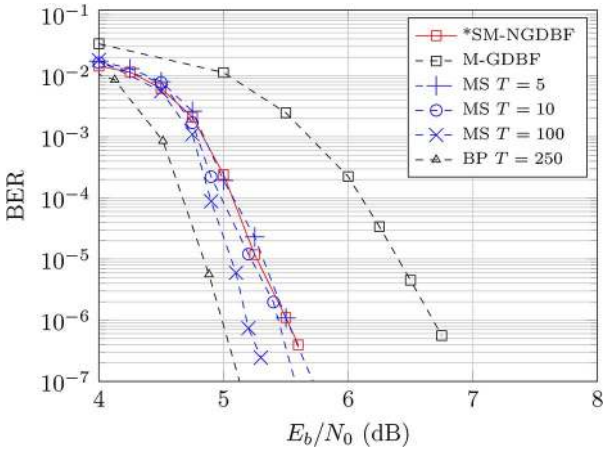

Fig. 5. BER versus E_b/N_0 curves for SM-NGDBF with $T = 300$ using the rate 0.9356 4376.282.4.9598 code over an AWGN channel with binary antipodal modulation. Several other algorithms are also shown for comparison. These results were obtained using $\theta = -0.7$, $\eta = 0.65$ and $\lambda = 0.993$. The dynamic range and syndrome weighting were also modified for this simulation, using $Y_{\max} = 2.0$ and $w = 0.1875$.

additional iterations ($T = 300$ for SM-NGDBF vs $T = 100$ for M-NGDBF).

To confirm robust performance of the SM-NGDBF algorithm, it was simulated for two other LDPC codes, yielding the results shown in Figs. 4 and 5. These results confirm that SM-NGDBF achieves good performance on codes with higher variable-node degree (in the case of Fig. 4) and for codes with rates above 0.9 (in the case of Fig. 5). In both cases, SM-NGDBF remains competitive with MS decoding.

Among the previously reported bit-flip algorithms, the best performance was achieved by Wadayama *et al.*'s H-GDBF algortihm with escape process. Haga and Usami's IGDBF achieved nearly identical performance to H-GDBF. Both H-GDBF and IGDBF allow a maximum of 300 iterations to achieve the best performance. Our results indicate that the adaptive M-NGDBF algorithm equals the H-GDBF performance with a maximum of only 100 iterations. Furthermore SM-NGDBF exceeds the H-GDBF performance when

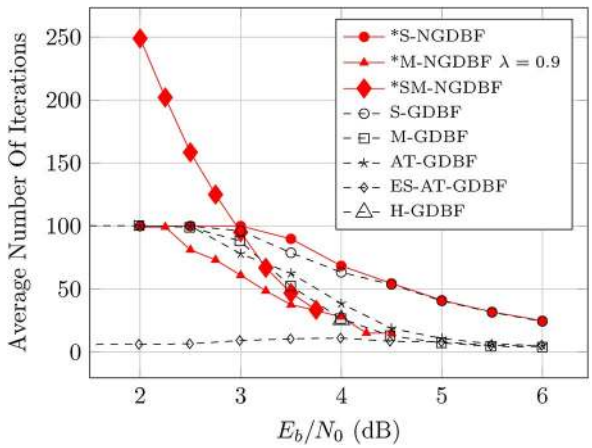

Fig. 6. Average number of iterations versus E_b/N_0 curves for existing GDBF and proposed NGDBF algorithms using PEGReg504x1008 code in an AWGN channel, maximum iterations limited to 100 except for SM-NGDBF and H-GDBF, which have $T = 300$. The newly proposed algorithms are indicated by an asterisk $(*)$.

300 iterations are allowed. These results may be interpreted in two ways. First, the adaptive M-NGDBF algorithm requires fewer iterations and is less complex than H-GDBF, but achieves the same performance (Fig. 2)—hence it can be interpreted as a gain in speed and complexity over H-GDBF. Second, by using additional iterations with output smoothing, the speed improvement can be traded for additional coding gain (Fig. 3).

B. Average Iterations per Frame

Fig. 6 shows the average number of iterations per frame as a function of E_b/N_0, using the PEGReg504x1008 code. This plot considers results for M-NGDBF and S-GDBF with $T = 100$, and for the SM-NGDBF algorithm which has $T = 300$. The comparison curves show previously known GDBF algorithms with $T = 100$, and also H-GDBF with $T = 300$. For the H-GDBF algorithm, the full iteration profile was not disclosed, but it was stated to be 25.6 iterations at an SNR of 4 dB [9] (shown as a single point in Fig. 6). From the plot, we see that the S-NGDBF provides no benefit in iteration count compared to previous algorithms. The M-NGDBF algorithms are comparable to previous alternatives; only the Early Stopping (ES) AT-GDBF algorithm converges faster than M-NGDBF, and this advantage disappears at higher SNR. At high SNR, i.e., $E_b/N_0 \geq 5$ dB, the average iteration count is nearly the same for the M-NGDBF, SM-NGDBF, and ES-AT-GDBF methods.

At high SNR, the SM-NGDBF algorithm has the same average iterations as M-NGDBF. Although SM-NGDBF requires $T = 300$—three times higher than M-NGDBF—on average these algorithms require the same number of iterations when operating at the same SNR. As an alternative comparison, we compare the average number of iterations needed to achieve a given BER performance. Fig. 7 shows the average iterations per frame plotted against the measured BER for M-NGDBF and SM-NGDBF. When compared for the same BER, the number of iterations needed for SM-NGDBF is on average double the number of iterations required for M-NGDBF. This result shows that the performance gain of SM-NGDBF comes at the cost of increased average iterations.

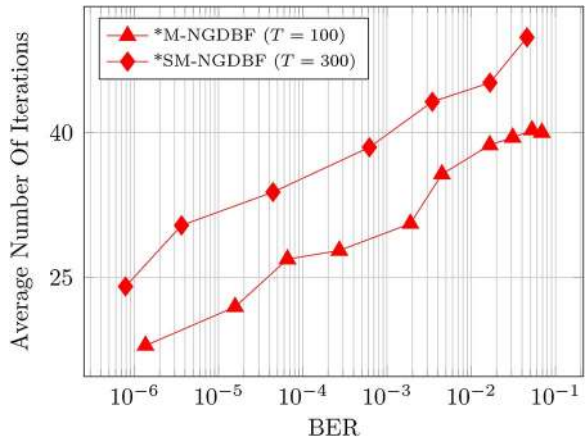

Fig. 7. Average number of iterations versus BER for the proposed M-NGDBF and SM-NGDBF algorithms.

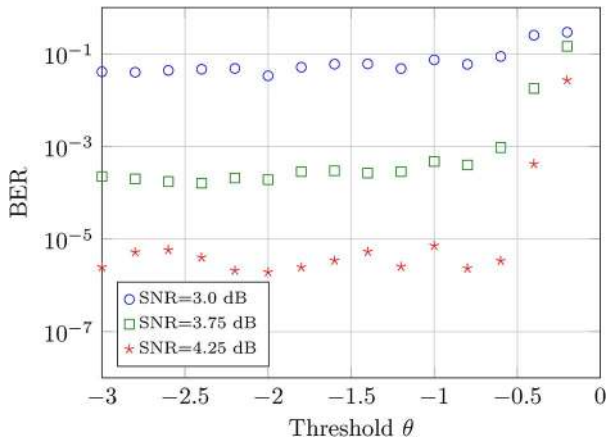

Fig. 9. Threshold sensitivity of adaptive M-NGDBF algorithm with parameters $\lambda = 0.9$, $T = 100$, $\eta = 1.0$ for the PEGReg504x1008 code.

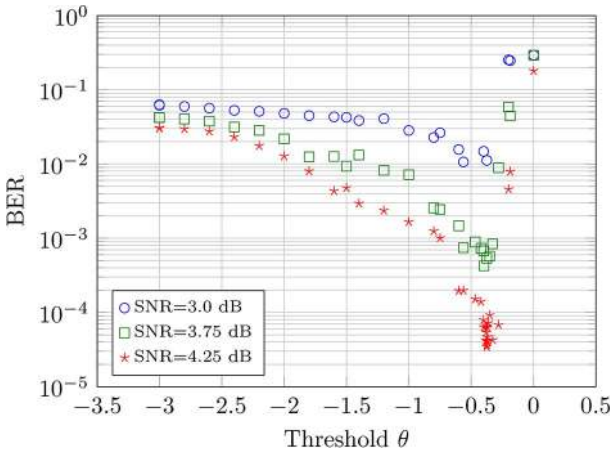

Fig. 8. Threshold sensitivity of the non-adaptive M-NGDBF algorithm with parameters $\lambda = 1.0$, $T = 100$, $\eta = 1.0$ for the PEGReg504x1008 code.

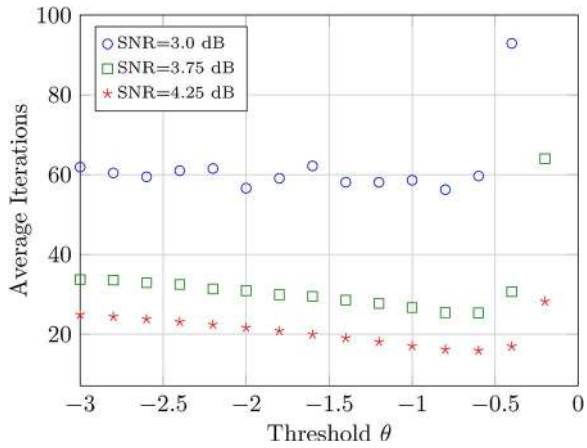

Fig. 10. Average iterations for adaptive M-NGDBF as a function of the initial threshold parameter θ for the PEGReg504x1008 code. The remaining parameters are $\lambda = 0.9$ and $T = 100$.

Since the average number of iterations for SM-NGDBF tends to be small, the smoothing operation is only used in a fraction of received frames. This is because the smoothing operation is only applied when the number of iterations exceeds $T - 64$. For the PEGReg504x1008 code, the smoothing operation was found to be required for only 6.1% of decoded frames when simulated at an SNR of 2.75 dB, 1.45% of frames at 3.0 dB, 0.51% of frames at 3.25 dB, and 0.16% of frames at 3.5 dB.

C. Sensitivity to Threshold Parameter θ

The optimal threshold values for the non-adaptive M-NGDBF algorithm (i.e., with $\lambda = 1.0$) were found empirically through a numerical search. Results from that search are shown in Fig. 8 for the PEGReg504x1008 code, in which the algorithm's BER is shown as a function of the threshold parameter. From this figure, it can be seen that the M-NGDBF algorithm is highly sensitive to the value of θ, which may prove problematic if the algorithm is implemented with fixed-point arithmetic at lower precision.

The adaptive M-NGDBF algorithm was simulated in a similar way, and the results shown in Fig. 9 reveal much less sensitivity to θ. The reduced threshold sensitivity is expected because the local thresholds θ_k are iteratively adjusted during decoding. Since it will take some number of iterations for the θ_k to settle, the optimal initial threshold should be chosen as the value that minimizes the average iterations per frame. Fig. 10 shows the average number of iterations per frame as a function of θ. The iteration count is seen to be only weakly a function of θ, with the minimum appearing at $\theta = -0.6$.

For adaptive M-NGDBF, the optimal value of λ is found through a similar empirical search. Results from that search are shown in Fig. 11 for the PEGReg504x1008 code. These results reveal a smooth relationship between BER and λ, allowing the optimal value of λ to be found reliably. The sensitivity revealed in Fig. 11 may prove to be difficult for implementations with quantized arithmetic; this problem is analyzed and resolved in Section V-A.

D. Sensitivity to Perturbation Variance

The NGDBF algorithms' performance is sensitive to the precise variance of the noise perturbation terms. As with the θ and λ parameters, the optimal value of the noise-scale

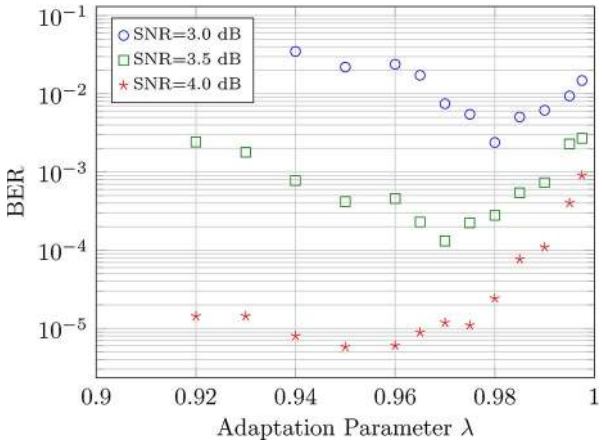

Fig. 11. Sensitivity of performance for M-NGDBF relative to the global adaptation parameter λ, with parameters $\theta = -0.9$, $T = 100$, $\eta = 0.96$, for the PEGReg504x1008 code.

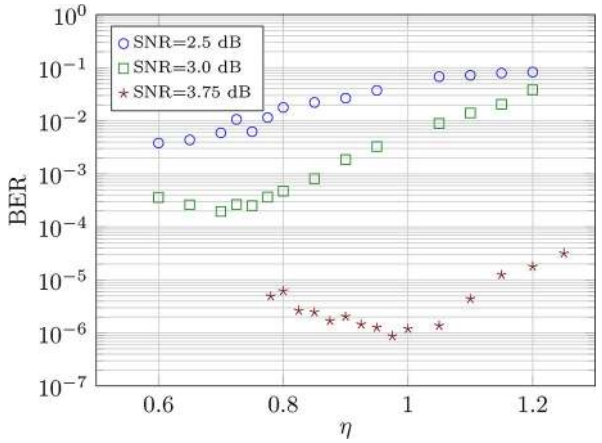

Fig. 12. Sensitivity of performance for SM-NGDBF relative to the noise scale parameter η.

parameter η is found through an empirical search. This search may produce different values for different codes and at different SNR values. Example results are shown in Fig. 12 for the SM-NGDBF algorithm simulated on the PEGReg504x1008 code. These results show that the optimal η is typically somewhat less than one, and tends to increase toward one at higher SNR.

E. Effects of Quantization on NGDBF

In this section we consider the performance of the M-NGDBF algorithm when implemented with limited precision. The algorithm was simulated with quantized arithmetic using Q bits by applying a uniform quantization with $N_Q = 2^Q$ levels in the range $[-Y_{\max}, Y_{\max}]$ (zero is excluded). The quantized channel sample $\tilde{y}_k$ is given by the quantization function $g(y)$:

$$g(y) = \text{sign}(y) \left(\left\lfloor \frac{|y| N_Q}{2Y_{\max}} \right\rfloor + \frac{1}{2} \right) \left(\frac{2Y_{\max}}{N_Q} \right). \qquad (7)$$

The quantization function is used to obtain quantized values. The vector of quantized channel samples is denoted by $\tilde{\mathbf{y}}$, and each quantized channel sample is $\tilde{y_k} = g(y_k)$. The same function is used to obtain the quantized inversion threshold, $\tilde{\theta} = g(\theta)$, the noise perturbation, $\tilde{q_k} = g(q_k)$, and the syndrome

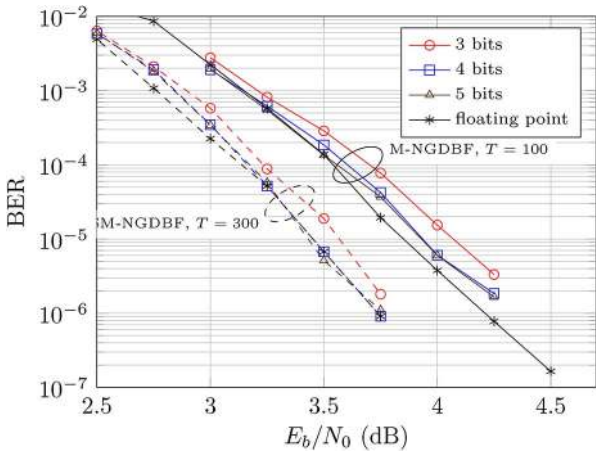

Fig. 13. BER versus E_b/N_0 curves for quantized implementations of the proposed M-NGDBF and SM-NGDBF algorithms, using the PEGReg504x1008 code on an AWGN channel with binary antipodal modulation. Solid curves indicate results for M-GDBF with $T = 100$, and dashed curves indicate results for SM-NGDBF with $T = 300$.

weight parameter, $\tilde{w} = g(w)$. After quantization, the inversion function is

$$\tilde{E}_k(t) = x_k(t)\tilde{y}_k + \tilde{w} \sum_{i \in \mathcal{M}(k)} s_i + \tilde{q}_k(t). \qquad (8)$$

The adaptive M-NGDBF and SM-NGDBF algorithms were simulated using the quantized inversion function with the PEGReg504x1008 code. The BER results are shown in Fig. 13. The quantized simulations reported in this section also use quantized threshold adaptation and the noise sample reuse method described in Section V. The results show that the algorithm is very close to unquantized performance when $Q = 3$, and the best BER performance is reached when $Q = 4$. There is a diminishing benefit to BER when $Q > 4$, however an additional effect is observed in the Frame Error Rate (FER) results shown in Fig. 14. Here we see an "error flare" effect for all cases for M-NGDBF, i.e., when output smoothing is not used. The flare improves when Q is increased. For SM-NGDBF, i.e., when output smoothing is used, the flare evidently does not occur at all, or occurs at a very low FER. These simulations were performed with parameter values $Y_{\max} = 1.7 - 1.75$, $\lambda = 0.98 - 0.99$, $\theta = -0.7$, and $w = 0.67 - 0.75$. Parameter values were adjusted within these ranges to optimize for BER performance.

V. Architecture Considerations

This section considers practical concerns for implementing the adaptive SM-NGDBF algorithm. These concerns include limited-precision arithmetic and architectural simplifications.

A. Implementing Threshold Adaptation

In Section IV, threshold adaptation was shown to provide a significant performance improvement to the M-NGDBF and SM-NGDBF algorithms. When threshold adaptation is applied, as described in Section III-D, each symbol node must independently implement threshold scaling by parameter λ

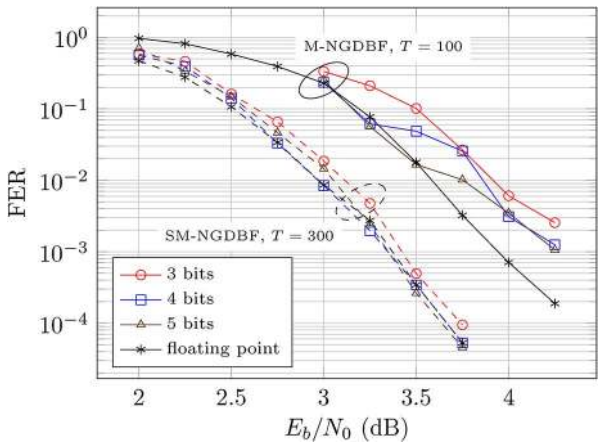

Fig. 14. FER versus E_b/N_0 curves for quantized implementations of the proposed M-NGDBF and SM-NGDBF algorithms, using the PEGReg504x1008 code on an AWGN channel with binary antipodal modulation. Solid curves indicate results for M-GDBF with $T = 100$, and dashed curves indicate results for SM-NGDBF with $T = 300$.

during every iteration. If implemented with arbitrary precision, this would require implementing multiplication and division operations. When the algorithm is implemented with limited precision, however, only a small number of quantized threshold values are required. The threshold adaptation procedure can therefore be expressed as

$$\theta_k(t+1) = \begin{cases} \theta_k(t)\lambda & x_k \text{ not flipped} \\ \theta_k(t) & x_k \text{ flipped} \end{cases} \tag{9}$$

If quantized arithmetic is used with low precision, and if λ is close to one (as is commonly the case), then it is possible that $g(\theta_k\lambda) = \theta_k$, where $g(\cdot)$ is the quantization function defined by (7). This case represents a failure of threshold adaptation because the local threshold is never able to change.

To avoid adaptation failures in quantized arithmetic, we introduce the symbol u_k as a counter for non-flip events. The counter is initialized at the start of decoding as $u_k(t=0) = 0$, and the counter is incremented according to the rule

$$u_k(t+1) = u_k(t) + \frac{1+\delta_k(t)}{2} \tag{10}$$

where $\delta_k(t) = \text{sign}(\tilde{E}_k(t) - \tilde{\theta}_k(t))$. Then the threshold at iteration t can be expressed as

$$\theta_k\left(u_k(t)\right) = \theta\lambda^{u_k(t)}, \tag{11}$$

so the quantized threshold value is then given by

$$\tilde{\theta}_k\left(u_k(t)\right) = g\left(\theta\lambda^{u_k(t)}\right). \tag{12}$$

In a finite-precision implementation, the quantized threshold $\tilde{\theta}_k(t)$ only changes for certain values of u_k. We say that an *adaptation event* occurs for some $u_k(t) = \tilde{\tau}$ if $\tilde{\theta}(\tilde{\tau}) \neq \tilde{\theta}(\tilde{\tau} - 1)$. Since u_k can only be changed by zero or one during any iteration, the threshold adaptation can be implemented by storing a pre-computed list of adaptation events $(\tilde{\theta}, \tilde{\tau})$. Threshold adaptation can thus be implemented using a simple combinational logic circuit that detects when $u_k = \tau^{(i)}$ and outputs the corresponding $\tilde{\theta} = \tilde{\theta}^{(i)}$.

TABLE II
THRESHOLD ADAPTATION EVENTS FOR $\theta = -0.9$, $\lambda = 0.99$, $Y_{\max} = 2.5$

	$Q=3$		$Q=4$		$Q=5$	
i	$\tilde{\theta}^{(i)}$	$\tilde{\tau}^{(i)}$	$\tilde{\theta}^{(i)}$	$\tilde{\tau}^{(i)}$	$\tilde{\theta}^{(i)}$	$\tilde{\tau}^{(i)}$
0	-0.9375	0	-0.7812	0	-0.8594	0
1	-0.3125	37	-0.4688	37	-0.7031	15
2			-0.1562	106	-0.5469	37
3					-0.3906	65
4					-0.2344	106
5					-0.0781	175

Table II shows threshold adaptation events for an example design with $\theta = -0.9$, $\lambda = 0.99$, $T = 300$, and $Y_{\max} = 2.5$. The table shows the threshold values $\tilde{\theta}^{(i)}$ and the corresponding adaptation level $\tilde{\tau}^{(i)}$ at which the threshold value becomes active. Only two unique threshold values occur when $Q = 3$; three values occur when $Q = 4$; and six values occur when $Q = 5$. There is typically a small number of distinct threshold values, because the values only span a small portion of the quantization range.

B. Simplification of Noise Sample Generation

The NGDBF algorithms require generating a Gaussian distributed random number at each symbol node during each iteration. Gaussian random number generators add significant hardware complexity. To simplify the implementation, we considered using only a single Gaussian Random Number Generator (RNG). The random samples are shifted from one symbol node to the next using a shift-register chain, as shown in Fig. 15. This method requires a powerup initialization so that all shift registers are pre-loaded with random samples. Simulations were performed using this method to obtain the results shown in Figs. 13 and 14, which come very close to the floating-point performance.

As a further simplification, uniform noise samples may be used in place of Gaussian samples, but with an associated performance loss. When repeating the cases from Figs. 13 and 14 using uniformly distributed noise samples, a performance loss of 0.1–0.2 dB was observed. Because this performance loss is undesirable, in the remainder of this paper we will only consider Gaussian distributed noise samples.

C. Complexity Analysis

The foregoing considerations are combined to arrive at the top-level architecture shown in Fig. 15. In addition to the shown architecture, a channel SNR estimator is required to obtain σ. The symbol node implementation is shown in Fig. 16 and the check node implementation is shown in Fig. 17. The check node implementation is uncomplicated and standard, requiring only $d_c - 1$ binary XNOR operations per parity-check node. The symbol node requires one ordinary counter and one up/down counter, a $(\tilde{\theta}, \tilde{\tau})$ memory and a signed adder with three Q-bit inputs and d_v single-bit inputs. Four single-bit operations are also required, including a toggle flip-flop, a sign multiplier (equivalent to an XNOR operation) and two inverters.

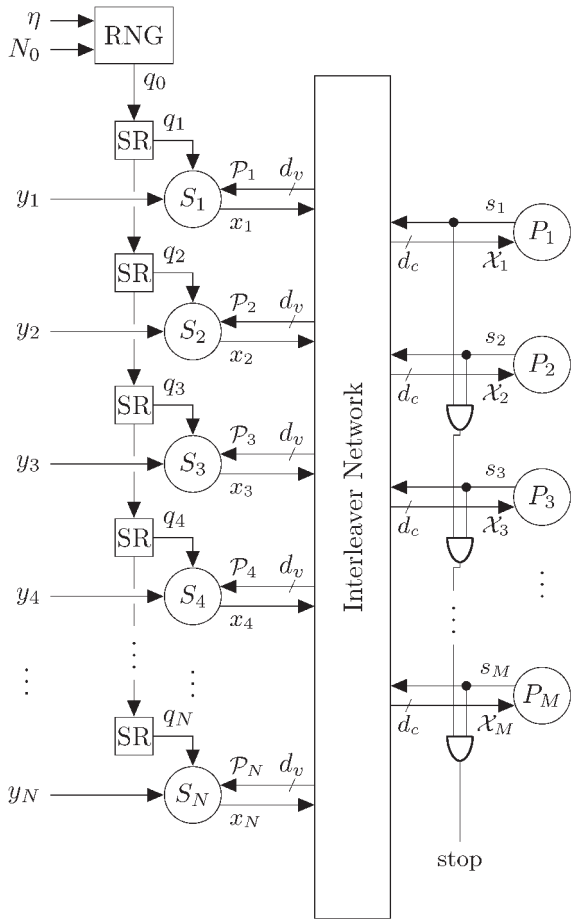

Fig. 15. Architecture of the NGDBF decoder. Gaussian-distributed noise samples are produced serially at the output of a Random Number Generator (RNG). The RNG requires inputs η and N_0, and the latter must be generated by a channel parameter estimator (not shown). A shift-register (SR) chain is used to distribute the random Gaussian samples that serve as the q_k perturbations. The symbol $\mathcal{P}_i$ indicates the set of syndrome messages that arrive at symbol node S_i, corresponding to the index set $\mathcal{N}(i)$. The symbol $\mathcal{X}_j$ is the set of messages that arrive at parity-check node P_j, corresponding to the index set $\mathcal{M}(j)$.

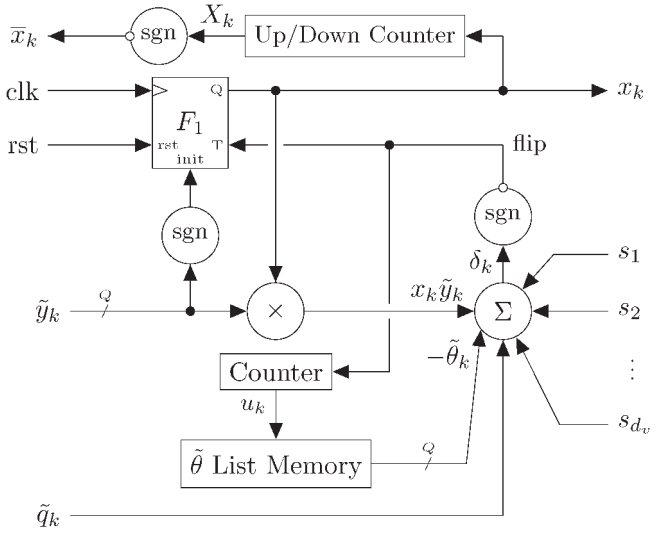

Fig. 16. Symbol node schematic. F_1 is a toggle flip-flop. The s_i messages are locally indexed. The sgn operator refers to sign-bit extraction. The multiplication $\otimes$ is binary, as it applies only to the sign bit of $\tilde{y}_k$.

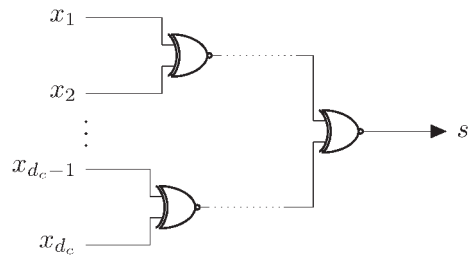

Fig. 17. Check node schematic showing a tree of XNOR operations over $d_c - 1$ input messages. The x_i messages are locally indexed.

The most complex operation is the multi-input adder. To remove the weight parameter $\tilde{w}$ from the syndrome inputs, we require that all $\tilde{y}_k$, $\tilde{q}_k$, and $\tilde{\theta}$ values are pre-scaled by the factor $\tilde{w}^{-1}$ (this pre-scaling is not expressly indicated in Fig. 16). Then the scaled inversion function is

$$\tilde{E}_k \tilde{w}^{-1} = x_k \tilde{y}_k \tilde{w}^{-1} + \tilde{q}_k \tilde{w}^{-1} + \sum_{i \in M(k)} s_i \qquad (13)$$

and the flip decision can be expressed as the sign of the difference $\delta_k = \tilde{E}_k \tilde{w}^{-1} - \tilde{\theta}_k \tilde{w}^{-1}$. This detail allows for the simplified adder implementation shown in Fig. 16. The effective complexity of this operation is that of two Q-bit binary adders and a d_v-bit adder.

Based on this proposed architecture, it is possible to make some high-level complexity comparisons against other related decoding methods. When making comparisons at this level, it is not possible to make strong predictions about power consumption, throughput, gate count or energy efficiency, but it is possible to make some interesting observations about the algorithms' comparative features.

1) Comparison With Previous GDBF Algorithms: Previously reported GDBF algorithms do not depend on the channel SNR, so NGDBF introduces a fixed complexity cost (i.e., the cost is independent of the code's length) because it requires a channel parameter estimator. At minimum, all GDBF algorithms require $d_c - 1$ XNOR operations in each parity-check node. At the symbol nodes, they require addition over the d_v single-bit syndromes in each symbol node, which has gate complexity equivalent to a d_v-bit adder. A second Q-bit addition is needed to incorporate the channel information. In the S-GDBF algorithm, the minimum metric must be found, requiring $n - 1$ comparisons. Since a comparison can be implemented using a signed adder, we say that the S-GDBF algorithm requires a total of $3n - 1$ additions and $m(d_c - 1)$ XNOR operations. In the M-GDBF algorithm, the global comparison is not required, but a comparison must still be made in each symbol node to implement the threshold operation, hence M-GDBF requires $3n$ additions. The SM-NGDBF algorithm requires $3n$ adders, and also requires an additional $2n$ counters (a counter requires fewer gates than an adder). A single RNG module and a channel parameter estimator are also required, but these are fixed overhead that does not scale with n.

2) Comparison With the MS Algorithm: The MS algorithm does not require channel SNR information. NGDBF again incurs a fixed complexity penalty due to channel parameter estimation. In a single iteration, the MS algorithm requires at

least $2d_v$ additions for every symbol node. For every parity-check node, $2d_c$ comparisons and $2d_c - 1$ XNOR operations are needed. MS decoders typically allow an internal dynamic range that exceeds the channel quantization of Q bits, so the arithmetic is assumed to be quantized on $Q + D$ bits, where $D \geq 0$ is the number of extra bits to accommodate the larger dynamic range. The messages exchanged between symbol and parity-check nodes are also comprised of $Q + D$ bits.

The gate requirements are clearly less for SM-NGDBF compared to MS. Based on the foregoing analysis, and using the (3, 6) PEGReg504x1008 code as an example, SM-NGDBF requires 75% fewer additions and comparisons. In terms of message routing, all GDBF algorithms exchange a total of $n + m$ single-bit signals in each iteration, compared to $2nd_v(Q + D)$ for MS. In the example code, assuming channel quantization with $Q = 5$ and $D = 3$, this means the required signal routing is reduced by 78.27%. Based on these comparisons, we may conclude that the GDBF algorithms (including SM-NGDBF) require substantially less circuit area than MS. This analysis is not sufficient to evaluate throughput or power efficiency, since those figures depend on a variety of circuit-level considerations such as critical path delay and average switching activity in combination with the average number of iterations per frame.

Another aspect of complexity is the algorithms' decoding latency. On first inspection, we observe that the MS algorithm requires much fewer iterations than the SM-NGDBF algorithm. For example, only 4.1 iterations are needed on average for MS decoding (assuming $T = 10$ with stoppping condition) on the PEGReg504x1008 code, operating at 3.5 dB. The SM-NGDBF algorithm, at the same SNR, requires an average of 47 iterations. While it appears that latency is much greater for SM-NGDBF, we must also account for the latency per iteration in the two algorithms. In typical implementations, MS decoders utilize multiple clock cycles per iteration; for example, Zhang *et al.* used 12 clock cycles per iteration [27], which we use here as a representative value. Due to SM-NGDBF's comparatively low gate and routing complexity, we expect an SM-NGDBF decoder to require only one clock cycle per iteration. We may therefore estimate the average latency of MS decoding at 49 clock cycles, compared to 47 clock cycles for SM-NGDBF. We therefore anticipate that an eventual implementation of SM-NGDBF could be comparable to previous MS implementations in terms of total latency.

3) Comparison With Stochastic Decoders: The M-NGDBF algorithm bears some similarity to stochastic LDPC decoders, as was mentioned in Section II. Stochastic LDPC decoders are known to provide performance within 0.5 dB of BP while exchanging single-bit messages with low-complexity logic processing. Stochastic decoders also require channel SNR estimation, so they share this fixed complexity cost with NGDBF. The most efficient stochastic decoding strategy is the Tracking Forecast Memory (TFM) described by Tehrani *et al.* [18]. The TFM-based decoder requires $2d_c - 1$ XOR operations at each parity-check node, nearly twice the number of XNOR operations needed by GDBF algorithms. At each symbol node, $2d_v$ Q-bit adders and d_v comparisons are used, for a total of $3nd_v$ equivalent additions. Some additional supporting logic is also required, including a Linear Feedback Shift Register (LFSR) to generate random bits, and a control circuit to regulate the inputs to the TFM adder.

Tehrani *et al.* also described a reduced-complexity Majority TFM (MTFM) design which reduces the required additions to approximately $3n$ [17], making it very close to SM-NGDBF. The MTFM decoder exchanges a total of $2nd_v$ single-bit messages per iteration, compared to $n + m$ for GDBF algorithms. For the PEGReg504x1008 code, SM-NGDBF exchanges about 50% fewer single-bit messages than an MTFM stochastic decoder for the same code. In terms of total iterations, MTFM-based stochastic decoders require about 20–40 iterations at higher SNR [17], whereas SM-NGDBF requires a comparable number at 30–50 iterations for similar SNR values. We may conclude that these algorithms have very similar complexity, but SM-NGDBF should require less circuit area due to the reduced message signal routing, and because fewer XNOR operations are required.

VI. Convergence Analysis

The NGDBF algorithms are built on the more general concept of noise-perturbed gradient descent optimization. The optimization task is the maximum likelihood (ML) decoding problem specified by (1), with the corresponding objective function $f(\mathbf{x})$ defined by (2), as described in Section III-B. The objective function is a non-linear function and has many local maxima. For gradient-descent optimization methods, local maxima are the major source of sub-optimality. NGDBF rests on the hypothesis that the noisy perturbation is beneficial for escaping from local maxima, thereby improving the likelihood of obtaining the correct global maximum. This section examines that hypothesis by analyzing a detailed case example of convergence dynamics, in which NGDBF is compared to other GDBF algorithms. We expect that the algorithms' comparative convergence errors should follow the same order as their comparative BER performance.

All GDBF and NGDBF algorithms attempt to maximize $f(\mathbf{x})$ by iteratively adjusting $\mathbf{x}$. By using the stopping condition requiring that all parity-checks are satisfied—i.e., that $\prod_{i=1}^{m}(1 + s_i)/2 = 1$—the GDBF algorithms enforce the constraint that candidate solutions are codewords in $\hat{\mathcal{C}}$, so long as decoding completes before reaching the maximum iteration count. For any ML-decodable case, the original transmitted codeword $\hat{\mathbf{c}} \in \hat{\mathcal{C}}$ should also be the ML solution. Then the global maximum for $f(\mathbf{x})$ is given by

$$f_{\max} = \sum_{k=1}^{n} \hat{c}_k y_k + \sum_{i=1}^{m} \prod_{j \in \mathcal{N}(i)} \hat{c}_j$$

$$= \sum_{k=1}^{n} \hat{c}_k y_k + m. \tag{14}$$

Fig. 18 shows the behavior of the objective functions evaluated for several algorithms as a function of iterations. Results are shown for the original GDBF and the proposed S-NGDBF algorithms for a simulated ML-decodable case with E_b/N_0 value of 4 dB. In the case of the S-GDBF algorithm, the objective function value gradually increases with the number of

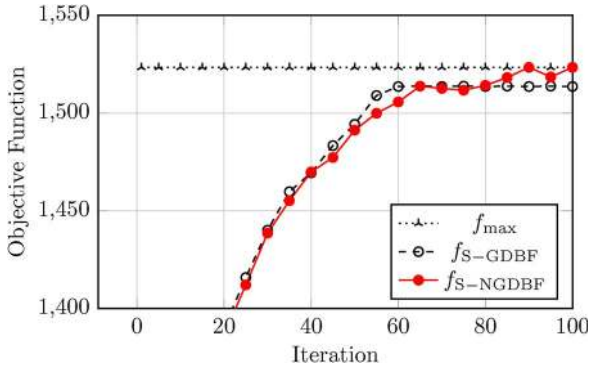

Fig. 18. Convergence behavior of the S-GDBF and S-NGDBF algorithms for a single frame sampled at $E_b/N_0 = 4$ dB. The true maximum is $f_{\max} = 1523$. The S-GDBF algorithm is able to obtain a maximum value of 1514. S-NGDBF obtains the global maximum in this case.

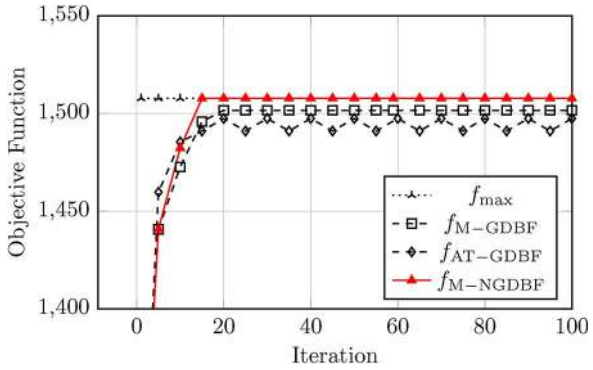

Fig. 19. Convergence behavior of the M-GDBF and M-NGDBF algorithms for a single frame sampled at $E_b/N_0 = 4$ dB. Threshold adaptation is used in the case of M-NGDBF algorithm, with $\lambda = 0.99$. The true maximum is $f_{\max} = 1508$. The M-GDBF algorithm is able to obtain a maximum value of 1502; the AT-GDBF algorithm is able to obtain a value of 1497; M-NGDBF obtains the global maximum in this case.

iterations. However, after 60 iterations the rise eventually stops and the objective function flattens out. This flat part corresponds to a local maximum. S-NGDBF reaches the global maximum value after 90 iterations, indicating that the S-NGDBF algorithm is able to escape from the spurious local maximum. A similar comparison is shown in Fig. 19 for the M-GDBF and M-NGDBF algorithms. The figure demonstrates that, for this example, M-GDBF is stuck in a local maximum, but M-NGDBF is able to escape from the local maximum and obtain the global solution.

The results shown in Figs. 18 and 19 represent single cases, and only partially demonstrate the superior convergence of NGDBF algorithms. To gain more insight into the convergence properties, we performed statistical analysis on the objective functions of several GDBF and NGDBF algorithms over many frames. For each algorithm, the convergence error was measured at the final iteration T and averaged over F transmitted frames. The convergence error is defined by

$$\epsilon_{\text{alg}} = \frac{1}{F}\sum_{i=1}^{F}\left(f_i\left(\mathbf{x}^{(i)}(T)\right) - f_{\max,i}\right). \tag{15}$$

where the subscript "alg" is replaced with the appropriate algorithm name, i is the index of a unique sample frame,

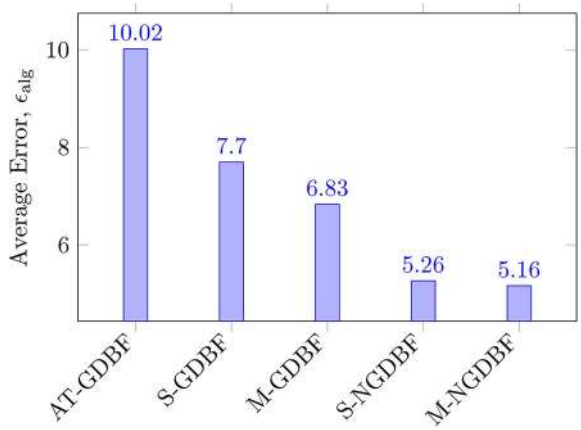

Fig. 20. Average convergence error for several GDBF and NGDBF algorithms at $E_b/N_0 = 5$ dB. Threshold adaptation is used for M-NGDBF, with $\lambda = 0.99$. For all algorithms, the maximum number of iterations is $T = 100$.

and the superscripted $\mathbf{x}^{(i)}(T)$ indicates the solution obtained for the ith frame. The subscript i is added to f and $f_{\max}$ to emphasize their dependence on the received channel samples. For the ith sampled frame, the transmitted message is $\hat{\mathbf{c}}^{(i)}$, the received channel samples are $\mathbf{y}^{(i)}$, and the objective function is maximized by

$$f_{\max,i} = \sum_{k=1}^{n} \hat{c}_k^{(i)} y_k^{(i)} + m. \tag{16}$$

Fig. 20 shows the average convergence error values for the tested algorithms at E_b/N_0 of 5 dB. The total number of frames F is 100. From Fig. 20, we see that the average convergence error for the proposed NGDBF algorithms is lower compared to the average error of the previously known GDBF algorithms. This shows that the NGDBF algorithms are more likely, on average, to arrive in the neighborhood of the correct solution, and are therefore more likely to have a better error correcting performance than the other GDBF algorithms.

VII. Local Maximum Likelihood Interpretation

The GDBF and NGDBF algorithms are developed based on heuristic approaches for combinatorial optimization of the global ML objective function. In this section, we provide a theoretical analysis to motivate the use of threshold adaptation and syndrome weighting. To explain the beneficial effects of these heuristics, we consider the Local Maximum Likelihood (LML) bit-flip decision at the symbol node level given the local information from the channel and adjacent partial syndrome values. The LML analysis predicts a pattern by which the flip decisions should evolve as the decoder converges toward an error-free codeword. When using threshold adaptation and syndrome weighting heuristics with a GDBF algorithm, the evolution of flip decisions is brought into closer correspondence with the LML decisions. During the initial iterations, LML decisions are found to be mainly determined by the channel information. In later iterations, the LML decisions are more heavily influenced by the partial syndrome values. We show that this behavior is very close to that of GDBF under threshold

adaptation. We further propose that GDBF can be improved by introducing a weight factor to the syndrome components, so that the local flip decisions evolve similarly to the LML decisions.

In this section, we introduce one minor change in notation. Since only scalar values are considered in this section, boldfaced letters are used to indicate random variables instead of vector quantities. We consider the problem of gradient descent decoding on a local channel sample $\tilde{y}_k$ and a set of d_v adjacent syndromes s_i, $i \in \mathcal{M}(k)$. The channel sample is assumed to be quantized using the quantization procedure described in Section IV-E. For a binary-input AWGN channel, we may obtain the probability masses for $\tilde{y}_k$ conditioned on the transmitted symbol $\hat{\mathbf{c}}_k$.

$$\Pr\left(\tilde{y}_k|\hat{\mathbf{c}}_\mathbf{k}=-1\right)=F_{-1}\left(\tilde{y}_k^+\right)-F_{-1}\left(\tilde{y}_k^-\right), \tag{17}$$

$$\Pr\left(\tilde{y}_k|\hat{\mathbf{c}}_\mathbf{k}=+1\right)=F_{1}\left(\tilde{y}_k^+\right)-F_{1}\left(\tilde{y}_k^-\right), \tag{18}$$

where $\tilde{y}_k^+$ and $\tilde{y}_k^-$ are the upper and lower boundary points of the quantization range that contains $\tilde{y}_k$, and F_{-1} and F_1 are cumulative Gaussian distribution functions with variance $\sigma^2 = N_0/2$ and means -1 and $+1$, respectively.

Initially, the decision $x_k(t=0)$ has error probability $p_e^{(0)} = P(x_k \neq \hat{\mathbf{c}}_k)$ given by $p_e = F_1(0)$. Recalling that the syndrome values are given by $s_i = \prod_{j\in\mathcal{N}(i)} x_j$, we have $s_i = x_k\nu_{ik}$ where ν_{ik} is the partial syndrome at parity-check i, excluding the influence of symbol node k. Finally, we define S_k as the penalty term $S_k = \sum_{i\in\mathcal{M}(k)} s_i$. The penalty term can also be expressed as $S_k = (\sum_{i\in\mathcal{M}(k)} \nu_{ik})x_k$.

From this we directly obtain the partial syndrome error probabilities $p_c = \Pr(\nu_{ik} \neq \hat{\mathbf{c}}_k)$ by enumerating over combinations in which an odd number of symbol errors has occurred out of $d_c - 1$ independent, identically distributed neighbors:

$$p_c = \sum_{j=1}^{\lceil \frac{d_c-1}{2} \rceil} \binom{d_c-1}{2j-1}(1-p_e)^{d_c-2j}p_e^{2j-1}. \tag{19}$$

The probability $P(n_e)$ of having n_e errors among the partial syndrome values ν_{ik} is thus:

$$P(n_e) = \binom{d_v}{n_e} p_c^{n_e}(1-p_c)^{d_v-n_e}. \tag{20}$$

If $x_k = \hat{\mathbf{c}}_k$, then n_e errors give a penalty $S_k = d_v - 2n_e$ (summation of $d_v - n_e$ syndrome $+1$ and n_e syndromes -1). Symmetrically, if $x_k = -\hat{\mathbf{c}}_k$, then $S_k = 2n_e - d_v$. In other words, knowing the observation S_k, we can deduce:

$$P(S_k|x_k = \hat{\mathbf{c}}_\mathbf{k}) = P\left(n_e = (d_v - S_k)/2\right) \tag{21}$$

and

$$P(S_k|x_k = -\hat{\mathbf{c}}_\mathbf{k}) = P\left(n_e = (d_v + S_k)/2\right) \tag{22}$$

Then the LML decision is

$$\hat{x}_{k,\mathrm{LML}} = \arg\max_{x_k} \Pr(\tilde{y}_k|x_k)\Pr(S_k|x_k). \tag{23}$$

$$\underset{p_e\,(0)\,=\,0.0672}{\begin{bmatrix}1&1&1&1\\1&1&1&1\\1&1&1&1\\1&1&1&1\\1&1&1&1\\1&1&1&-1\\1&1&1&-1\\1&1&-1&-1\end{bmatrix}} \rightarrow \underset{p_e\,=\,0.5p_e\,(0)}{\begin{bmatrix}1&1&1&1\\1&1&1&1\\1&1&1&-1\\1&1&1&-1\\1&1&1&-1\\1&1&1&-1\\1&1&-1&-1\\1&1&-1&-1\end{bmatrix}} \rightarrow \underset{p_e\,=\,0.01p_e\,(0)}{\begin{bmatrix}1&1&1&-1\\1&1&1&-1\\1&1&1&-1\\1&1&1&-1\\1&1&-1&-1\\1&1&-1&-1\\1&1&-1&-1\\1&1&-1&-1\end{bmatrix}}$$

Fig. 21. Example of evolution of the LML flip matrix Φ for a (3, 6) LDPC code, with $Q = 4$ and an $E_b/N_0 = 3.50$ dB. The initial value of P_e is 0.0672. Since Φ is symmetric with $\Phi(i,j) = \Phi(N_Q + 1 - i, d_v + 2 - j)$, only the top $N_Q/2$ rows are shown.

To relate the LML result to bit flipping algorithms, it can be expressed as an LML flip decision ϕ, defined by

$$\phi(x_k, \tilde{y_k}, S_k) = \operatorname{sign}\log\left(\frac{\Pr(\tilde{y}_k|x_k)\Pr(S_k|x_k)}{\Pr(\tilde{y}_k|-x_k)\Pr(S_k|-x_k)}\right). \tag{24}$$

If $\phi = -1$, then the optimal decision is to flip x_k.

To visualize the LML behavior on a quantized channel, we arrange the decisions in a *flip matrix* Φ that expresses all possible states; the rows of Φ correspond to the possible channel sample values, and the columns correspond to the possible values of S_k. There are $d_v + 1$ possible values of $S_k : -d_v, -d_v + 2, \ldots, d_v - 2, d_v$. We index these values in ascending order as $S_k^{(j)}$, $j = 1, 2, \ldots, d_v + 1$. The possible $\tilde{y}_k$ values are similarly indexed in ascending order as $\tilde{y}_k^{(i)}$, $i = 1, 2, \ldots, N_Q$. Then Φ is an $N_Q \times (d_v + 1)$ matrix with entries $\phi_{i,j} = \phi(x_k, \tilde{y}_k^{(i)}, S_k^{(j)})$. For a given locally received $\tilde{y}_k$ and S_k, if the corresponding $\phi_{ij} = -1$, then the corresponding decision x_k should be flipped.

The LML flip decision depends on the partial syndrome error probabilities, which change in successive iterations. To understand how the LML decision evolves across iterations, we suppose that the bit error probability is a function of the iteration number, t, and that $p_e(t)$ is decreasing with successive iterations. As the error probability decreases, the flip matrix is found to evolve from an initial pattern in which decisions are heavily dependent on $\tilde{y}_k$, with increasing dependence on S_k in later iterations as $p_e(t)$ decreases toward zero. An example of this evolution is shown in Fig. 21, for the case $x_k = 1$ with parameters $Y_{\max} = 1.5$, $\sigma = 0.668$, $Q = 4$, and $d_v = 3$. For a (3, 6) LDPC code, this corresponds to $E_b/N_0 = 3.50$ dB.

With threshold adaptation, the GDBF algorithm's behavior is similar to the LML flip matrix. Initially, the threshold θ is set to a significantly negative value, say $\theta = -1.0$. For a given set of parameters, we may obtain a matrix E of values for the inversion function, with members $E_{i,j} = \tilde{y}_k^{(i)} + S_k^{(j)}$. By applying the threshold θ to all the elements of E, we obtain the flip matrix for the GDBF algorithm. As the threshold is adapted toward zero, the flip matrix evolves to place increased weight on the syndrome information, similar to the LML flip matrix evolution. The GDBF flip matrices do not correspond perfectly to the LML predictions. To bring closer agreement, a weight factor is introduced, giving a modified weighted inversion function

$$\tilde{E}_k = x_k\tilde{y}_k + w\sum_{i\in M(k)} s_i + \tilde{q}_k, \tag{25}$$

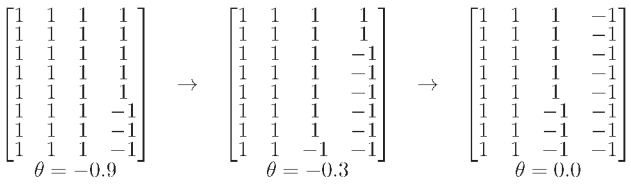

Fig. 22. Evolution of the flip matrix for the weighted GDBF algorithm with threshold adaptation. Use of threshold adaptation and syndrome weighting achieves qualitative agreement with the LML flip decisions in Fig. 21.

The best value for w is found empirically and may be code dependent. Based on the parameters $\sigma = 0.668$, $Y_{\max} = 1.5$, $w = 0.75$, and $d_v = 3$, the flip matrix evolution corresponding to $\tilde{E}_k$ is shown in Fig. 22.

The relationship between LML and the GDBF heuristics is not an exact correspondence. The LML analysis predicts the desirable behavior of the flip matrix over time during a successful decoding event. When GDBF is augmented by introducing the threshold adapation and syndrome weighting heuristics, its behavior is brought into approximate correspondence with the LML prediction. This provides a new theoretical motivation for using these heuristic methods, which has not been addressed in the previous literature.

VIII. Conclusion

This paper introduced a collection of novel Noisy GDBF algorithms, based on a noisy gradient descent heuristic, that outperforms existing GBDF algorithms. We found that previous GDBF algorithms, including the S-GDBF, M-GDBF, and AT-GDBF algorithms, are significantly improved when combined with the noise perturbation. Additional heuristic improvements were introduced that achieved a significant performance benefit in comparison to the best known versions of GDBF, achieving performance comparable to the standard min-sum algorithm for several LDPC codes. We also provided an architecture for implementing the new algorithm with quantized channel information, and showed that its implementation complexity is quite low compared to min-sum or stochastic decoding. The NGDBF algorithms do require estimation of the channel SNR, which introduces a fixed complexity cost that does not affect the previously known GDBF of MS algorithms.

The NGDBF decoding algorithms are based on a heuristic approach. To gain additional validation for those heuristics, we examined the convergence characteristics and found that, on average, the NGDBF algorithm converges closer to the global maximum whereas other algorithms are more frequently trapped in suboptimal local maxima. We also examined the approximate local ML solution for bit-flip behavior. From this analysis, we proposed using a weight factor to bring GDBF closer to the LML behavior. As a result of these analyses, we obtained a new bit-flipping decoding algorithm that avoids using any global search or sort operations. The resulting algorithm is feasibly a competitor to the popular min-sum algorithm, since it requires less computational effort while maintaining good BER and FER performance.

Acknowledgment

The authors would like to express his thanks to the anonymous reviewers of IEEE Transactions on Communications for their comments, which helped to improve the quality of the manuscript.

References

[1] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *IEEE Electron. Lett.*, vol. 33, no. 6, pp. 457–458, Mar. 1997.

[2] Y. Kou, S. Lin, and M. Fossorier, "Low-density parity-check codes based on finite geometries: A rediscovery and new results," *IEEE Trans. Inf. Theory*, vol. 47, no. 7, pp. 2711–2736, Nov. 2001.

[3] M. Mansour and N. Shanbhag, "High-throughput LDPC decoders," *IEEE Trans. VLSI Syst.*, vol. 11, no. 6, pp. 976–996, Dec. 2003.

[4] F. Guilloud, E. Boutillon, J. Tousch, and J.-L. Danger, "Generic description and synthesis of LDPC decoders," *IEEE Trans. Commun.*, vol. 55, no. 11, pp. 2084–2091, Nov. 2007.

[5] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inf. Theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962.

[6] N. Miladinovic and M. P. C. Fossorier, "Improved bit-flipping decoding of low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 51, no. 4, pp. 1594–1606, Apr. 2005.

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

[8] X. Wu *et al.*, "New insights into weighted bit-flipping decoding," *IEEE Trans. Commun.*, vol. 57, no. 8, pp. 2177–2180, Aug. 2009.

[9] 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.

[10] M. Ismail, I. Ahmed, and J. Coon, "Low power decoding of LDPC codes," *ISRN Sensor Netw.*, vol. 2013, pp. 650740-1–650740-12, 2013.

[11] T. Phromsa-ard *et al.*, "Improved gradient descent bit flipping algorithms for LDPC decoding," in *Proc. 2nd Int. Conf. DICTAP*, 2012, pp. 324–328.

[12] R. Haga and S. Usami, "Multi-bit flip type gradient descent bit flipping decoding using no thresholds," in *Proc. ISITA*, 2012, pp. 6–10.

[13] V. C. Gaudet and A. C. Rapley, "Iterative decoding using stochastic computation," *Electron. Lett.*, vol. 39, no. 3, pp. 299–301, Feb. 2003.

[14] C. Winstead, V. C. Gaudet, A. C. Rapley, and C. Schlegel, "Stochastic iterative decoders," in *Proc. ISIT*, 2005, pp. 1116–1120.

[15] S. Sharifi Tehrani, W. J. Gross, and S. Mannor, "Stochastic decoding of LDPC codes," *IEEE Commun. Lett.*, vol. 10, no. 10, pp. 716–718, Oct. 2006.

[16] S. Sharifi Tehrani, S. Mannor, and W. J. Gross, "Fully parallel stochastic LDPC decoders," *IEEE Trans. Signal Process.*, vol. 56, no. 11, pp. 5692–5703, Nov. 2008.

[17] S. Sharifi Tehrani *et al.*, "Majority-based tracking forecast memories for stochastic LDPC decoding," *IEEE Trans. Signal Process.*, vol. 58, no. 9, pp. 4883–4896, Sep. 2010.

[18] S. Sharifi Tehrani, A. Naderi, G.-A. Kamendje, S. Mannor, and W. J. Gross, "Tracking forecast memories for stochastic decoding," *J. Signal Process. Syst.*, vol. 63, no. 1, pp. 117–127, Apr. 2011.

[19] S. Sharifi Tehrani *et al.*, "Relaxation dynamics in stochastic iterative decoders," *IEEE Trans. Signal Process.*, vol. 58, no. 11, pp. 5955–5961, Nov. 2010.

[20] N. Onizawa, W. J. Gross, T. Hanyu, and V. C. Gaudet, "Clockless stochastic decoding of low-density parity-check codes: Architecture and simulation model," *J. Signal Process. Syst.*, vol. 76, no. 2, pp. 185–194, Aug. 2014.

[21] F. Leduc-Primeau, S. Hemati, S. Mannor, and W. Gross, "Dithered belief propagation decoding," *IEEE Trans. Commun.*, vol. 60, no. 8, pp. 2042–2047, Aug. 2012.

[22] C. K. Ngassa, V. Savin, and D. Declercq, "Unconventional behavior of the noisy min-sum decoder over the binary symmetric channel," in *Proc. Int. Workshop ITA*, San Diego, CA, USA, Feb. 2014, pp. 1–10.

[23] C. K. Ngassa, V. Savin, and D. Declercq, "Min-sum-based decoders running on noisy hardware," in *Proc. IEEE GLOBECOM*, Dec. 2013, pp. 1879–1884.

[24] J. Zhang and M. P. C. 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.

[25] 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.
[26] D. J. C. MacKay, Encyclopedia of Sparse Graph Codes. [Online]. Available: http://www.inference.phy.cam.ac.uk/mackay/codes/data.html, accessed: 2014-06-20
[27] Z. Zhang, V. Anantharam, M. J. Wainwright, and B. Nikolic, "An efficient 10GBASE-T ethernet LDPC decoder design with low error floors," *IEEE J. Solid-State Circuit*, vol. 45, no. 4, pp. 843–855, Apr. 2010.

Gopalakrishnan Sundararajan was born in Madras, Chennai, Tamil Nadu, India. He received the B.Eng. degree from Anna University, India, and the M.S. degree in electrical and computer engineering from Oklahoma State University, Stillwater, OK, USA, in 2010. He is currenly working towards the Ph.D. degree in electrical and computer engineering at Utah State University, Logan, UT, USA. His research intersts include novel LDPC decoding algorithms and their VLSI implementation, fault tolerant circuit techniques, low power and process variation tolerant circuit design. Gopalakrishnan received a Sant Graduate Innovation Fellowship from Utah State University's College of Engineering in 2014.

Chris Winstead (SM'11) received the B.S. degree in electrical and computer engineering from the University of Utah, Salt Lake City, UT, USA, in 2000, and the Ph.D. degree from the University of Alberta, Edmonton, AB, Canada, in 2005. He is currently with the ECE Department at Utah State University, Logan, UT, USA, where he holds the rank of Associate Professor. His research interests include reliable wireless communication systems, implementation of error-correction algorithms, low-power electronics and fault-tolerant VLSI circuits. Dr. Winstead received the NSF Career award for research in low-energy wireless communication circuits, and is a Fulbright scholar. He is a member of the Tau Beta Pi engineering honor society.

Emmanuel Boutillon received the Diploma in engineering in 1990 and the Ph.D. degree in 1995, both from the Telecom Paris Tech, Paris, France. From 1995 to 2000, he was an Assistant Professor in Telecom Paris Tech. In 1998, he spent a sabbatical year at the University of Toronto, Toronto, ON, Canada. In 2000, he moved to the University of Bretagne Sud as a Professor. He headed the LESTER lab from 2005 up to end of 2007 and, since 2008, he is the head of CACS department (lab-STICC). In 2011, he had a sabbatical year at INICTEL-UNI, Lima (Peru). His research interests are on the interactions between algorithm and architecture in the field of wireless communications and high speed signal processing. In particular, he works on Turbo Codes and LDPC decoders.