# Numerical Issues Affecting LDPC Error Floors

Brian K. Butler and Paul H. Siegel
Department of Electrical and Computer Engineering, University of California San Diego, La Jolla, CA 92093
butler@ieee.org, psiegel@ucsd.edu

***Abstract*—Numerical issues related to the occurrence of error floors in floating-point simulations of belief propagation (BP) decoders are examined. Careful processing of messages corresponding to highly-certain bit values can sometimes reduce error floors by several orders of magnitude. Computational solutions for properly handling such messages are provided for the sum-product algorithm (SPA) and several variants.**

## I. Introduction

Belief propagation (BP) decoders based upon the sum-product algorithm (SPA) are widely used to decode error-correcting codes that have a sparse graphical representation, including, most notably, low-density parity-check (LDPC) codes. This paper addresses numerical issues arising in the implementation of SPA decoding and several of its variants that have an impact on the occurrence and severity of frequently observed error floors in the decoder performance curves. (For background on BP and LDPC coding, the reader is referred to [1]–[3].)

In [4], MacKay and Postol examined *near codewords* associated with the error floors that they observed in BP decoding of Margulis-type codes over the additive white Gaussian noise (AWGN) channel. Shortly thereafter, Richardson [5] wrote a seminal paper on the error floors of LDPC codes in the setting of the AWGN channel and the binary symmetric channel, identifying decoder-dependent, error-prone substructures in the Tanner graph, dubbed *trapping sets*, that were responsible for the observed floors. Later, Han and Ryan also studied error floors and their properties for LDPC codes used on the AWGN channel [6]. With the exception of [5], which used a 5-bit hardware simulation to generate error-rate curves, it appears that the error floors reported in these prior studies were found with floating-point (FP) simulations of the BP decoder.

A quantization scheme, such as FP, imposes a limited range and finite resolution on the values to be represented. Additionally, the use of non-linear functions in a finite-precision environment may dramatically further limit the domains and/or images of said functions. This paper addresses these issues of range and resolution of the messages in several SPA variants implemented in FP. Recent work has shown that the error floors of variable-regular LDPC codes, such as the Margulis code considered by the authors of [3]–[6], are largely the result of numerical problems associated with the processing of highly certain messages in the BP decoder implementation [7]–[9].

This research was supported in part by NSF Grants CCF-0829865 and CCF-1116739 and by the Center for Magnetic Recording Research at UCSD.

This work improves the range of several known SPA variants, introduces a new SPA variant, and presents simulation results for two variable-regular LDPC codes whose error floors are reduced by orders of magnitude by addressing these numerical range issues. Motivated by these observations, Zhang and Siegel [10] have devised several small-bit-width decoders that substantially lower the observed error floors of several LDPC codes. These decoders use non-uniform quantization techniques that accept reduced message resolution for highly certain messages for the sake of increased message range.

## II. Background

### A. Code and Channel

LDPC codes are defined by the null space of a parity check matrix $\mathbf{H}$. The codewords are the set of column vectors $\mathcal{C}$, such that $\mathbf{c} \in \mathcal{C}$ satisfies $\mathbf{Hc} = \mathbf{0}$ over a particular field. A given code can be described by many different $\mathbf{H}$ matrices.

The $\mathbf{H}$ matrix over GF(2) may be associated with a bipartite graph $B = (V, C, E)$, called a *Tanner graph*, in which the vertex set may be partitioned into two disjoint sets $V$ and $C$. The set of variable nodes $V$ represent the symbols of the codeword that are sent over the channel and correspond to the columns of the parity-check matrix. The set of check nodes $C$ enforce the parity-check equations represented by the rows of $\mathbf{H}$. Each edge $e \in E$ of a Tanner graph joins a variable node $v_i \in V$ to a check node $c_j \in C$. The $(j, i)$th entry of $\mathbf{H}$ is 1 if $(v_i, c_j) \in E$ and is 0 otherwise.

**Assumption.** We are only concerned with codes over the binary field GF(2). Also, we assume binary antipodal signaling over the AWGN channel for the purposes of illustration, but the fundamental numerical issues are independent of these assumptions.

After encoding, each binary element of codeword $\mathbf{c}$ is transmitted over the AWGN channel as a binary antipodal symbol $t_i \in \{+1, -1\}$. Every received symbol $r_i$ is simply the sum $t_i + n_i$ of the transmitted symbol plus independent and identically-distributed (i.i.d.) Gaussian noise of zero-mean and variance $\sigma^2$. This yields a channel SNR of $1/\sigma^2$ or $2RE_b/N_0$, where $R$ is the rate of the code, $E_b$ is the received energy per information bit for coherent bandpass detection, and $N_0$ is the one-sided power spectral density of the noise.

### B. Sum-Product Algorithm (SPA)

We next describe the sum-product algorithm (SPA) that is one of the most widely used forms of BP decoding for LDPC

codes. For codes with a cycle-free Tanner graph representation, SPA decoding on such a graph is optimal, in the sense that it is equivalent to symbol-wise, maximum a-posteriori (MAP) decoding. Such codes are generally not attractive [11]. Although no longer optimal when used to decode LDPC codes whose Tanner graph representations have cycles, SPA decoding has nevertheless been found to provide near-optimal performance, at least in certain ranges of signal-to-noise ratio.

As is often done, we will first implement our decoder simulation in the log-domain; this removes the need for normalization and is also closer to the approximations used in hardware implementations. In describing the algorithm, we use the notation $\mathcal{N}(i)$ to indicate the set of check nodes adjacent to variable node $v_i$, and we use $\mathcal{N}(j) \setminus i$ to denote all variable nodes adjacent to check node $c_j$, excluding variable node $v_i$.

First, the received symbols $r_i$ are converted to log-likelihood ratios (LLRs) defined by

$$\lambda^{[i]} = \ln \frac{P(r_i|t_i = +1)}{P(r_i|t_i = -1)}.$$

For the AWGN channel, this becomes $\lambda^{[i]} = 2r_i/\sigma^2$.

The LLR $\lambda^{[i]}$ is the initial message passed from variable node $v_i$ to each of its adjacent check nodes in the Tanner graph. That is, $\lambda_0^{[i\to j]} = \lambda^{[i]}$ for all $i, j$ such that $(v_i, c_j) \in E$. The iteration counter $l$ is then initialized to 1.

On the first half-iteration of iteration $l$, the message sent from check node $c_j$ to the adjacent variable node $v_i$ is given by

$$\lambda_l^{[i\leftarrow j]} = 2 \tanh^{-1}\left[\prod_{k\in\mathcal{N}(j)\setminus i} \tanh \frac{\lambda_{l-1}^{[k\to j]}}{2}\right], \qquad (1)$$

for each edge in the graph. During the second half-iteration, the return message sent from variable node $v_i$ to adjacent check node $c_j$ is given by

$$\lambda_l^{[i\to j]} = \lambda^{[i]} + \sum_{k\in\mathcal{N}(i)\setminus j} \lambda_l^{[i\leftarrow k]}. \qquad (2)$$

If the early termination logic detects a codeword or if the iteration counter $l$ exceeds the maximum allowed count, the iterations are halted. Otherwise, the iteration counter is incremented by 1, and the new check-to-variable-node messages are computed using (1). Upon the completion of the iterations, the decision for symbol $i$, denoted $\hat{t}_l^{[i]}$, is set equal to the sign of the incoming message sum at variable node $v_i$, that is,

$$\hat{t}_l^{[i]} = \operatorname{sign}\Big(\lambda^{[i]} + \sum_{k\in\mathcal{N}(i)} \lambda_l^{[i\leftarrow k]}\Big).$$

If $B$ has no cycles, the message sum is equivalent to $\ln \frac{P(\mathbf{r}|t_i=+1)}{P(\mathbf{r}|t_i=-1)}$, the MAP decision statistic.

### C. Floating-Point Notation

Floating-point (FP) formatting offers an economical computer representation (*i.e.*, quantization) of real numbers covering a wide dynamic range with a significant level of precision.

The elements of an FP number are stored separately: the sign bit, the exponent, and the significand. Table I shows the basic binary formats included in IEEE Standard 754-2008, which uses base-2 representation [12]. The parameters $p$ and $emax$ denote the number of binary digits in the significand (i.e., *precision*) and the maximum exponent, respectively. The standard requires the minimum exponent be $emin = 1 - emax$. Since normalized FP numbers are expressed with the radix point after the first binary digit of the significand, the maximum supported FP value may be computed to be

$$\overbrace{(1.111\ldots111)_2}^{p \text{ ones}} \cdot 2^{emax} = (2 - 2^{1-p})\ 2^{emax}.$$

TABLE I
FLOATING-POINT BASIC BINARY FORMATS OF IEEE STANDARD 754 [12].

| Name | Bits stored | $p$ bits | $emax$ |
|---|---|---|---|
| Single-precision (SP) | 32 | 24 | +127 |
| Double-precision (DP) | 64 | 53 | +1023 |
| Quadruple-precision (QP) | 128 | 113 | +16,383 |

Normalization of binary FP numbers is required by the standard when possible. These *normalized* numbers must have 1 as the most significant bit of the significand, which need not be stored. Very small FP numbers may become subnormal (or "denormalized") when they are too small to be normalized. Subnormal numbers are supported by the standard and do not use the full precision available. The smallest normalized positive value is $2^{emin}$ and the smallest subnormal positive value is

$$\overbrace{(.000\ldots001)_2}^{p-1 \text{ bits}} \cdot 2^{emin} = 2^{emin+1-p}$$

per [12, §3.3]. The available binary FP values in the immediate vicinity of 1 are the following:

$$1 - 3 \cdot 2^{-p}, 1 - 2 \cdot 2^{-p}, 1 - 2^{-p}, 1, \text{ and } 1 + 2^{1-p}. \qquad (3)$$

### D. Numerical Problem in FP

Direct implementation of (1) yields numerical problems at high LLRs. Letting $y = \tanh(\lambda/2) < 1$, we may express $y$ as

$$y = \frac{1 - e^{-\lambda}}{1 + e^{-\lambda}} = 1 - \frac{2e^{-\lambda}}{1 + e^{-\lambda}}$$

in order to find when FP quantizes $y$ to 1. Ideally, the values of $y$ near 1 are rounded to the closest quantized level listed in (3). So to be rounded-up to 1, $y \geq 1 - 2^{-p}/2$ must be satisfied, which is equivalent to

$$\frac{2e^{-\lambda}}{1 + e^{-\lambda}} \leq 2^{-(p+1)} \text{ and} \qquad (4)$$

$$\lambda + \ln(1 + e^{-\lambda}) \geq (p + 2) \ln 2. \qquad (5)$$

We may very accurately approximate (5) by $\lambda \geq (p + 2) \ln 2$ or 38.1230949 in DP-FP (64-bit IEEE 754) with its $p = 53$ bits of precision.

As an argument of $\pm 1$ will cause the $\tanh^{-1}$ function to overflow, protection from high magnitude LLRs must be added to (1) or (2) to ensure numerical integrity or an alternative solution not using $\tanh^{-1}$ must be found. Thus,

preventing $\tanh^{-1}$ overflow by limiting LLRs will result in a maximum producible LLR magnitude. Our examination of published error-floor results suggests that such LLR limiting (or "saturating" or "clipping") is commonly employed.

## III. Preferred SPA Solution

In this section we describe our preferred SPA solution, its numerical limits, and speed issues. The relationship known as the Jacobian logarithm is

$$\begin{aligned}\ln(e^x + e^y) &= \ln(e^x) + \ln\left(1 + e^{y-x}\right) \\ &= \ln(e^y) + \ln\left(1 + e^{x-y}\right) \\ &= \max(x, y) + \ln\left(1 + \exp(-|x - y|)\right).\end{aligned} \tag{6}$$

Using (6) and other identities, an alternative exact pairwise check node reduction may be derived [13], [14]. For example, if we set $\mathcal{N}(j) \setminus i = \{k_1, k_2\}$, the check-node message in (1) can be computed as follows:

$$\begin{aligned}\lambda_l^{[i\leftarrow j]} = &\operatorname{sign}\lambda_{l-1}^{[k_1\to j]} \cdot \operatorname{sign}\lambda_{l-1}^{[k_2\to j]} \cdot \\ &\min\left(\left|\lambda_{l-1}^{[k_1\to j]}\right|, \left|\lambda_{l-1}^{[k_2\to j]}\right|\right) \\ &+ \ln\left(1 + \exp\left[-\left|\lambda_{l-1}^{[k_1\to j]} + \lambda_{l-1}^{[k_2\to j]}\right|\right]\right) \\ &- \ln\left(1 + \exp\left[-\left|\lambda_{l-1}^{[k_1\to j]} - \lambda_{l-1}^{[k_2\to j]}\right|\right]\right)\end{aligned} \tag{7}$$

The check-node-message re-formulation in (7) contains no possibility of overflow, regardless of the LLR magnitude, which for DP-FP extends to approximately $1.798 \times 10^{308}$. The only potential for overflow in the SPA is now just the addition operation within (2) at extremely high LLRs.

The computer implementation of (7) does not necessarily have a large impact on simulation speed. A single hyperbolic tangent evaluation consumes nearly the same number of CPU cycles as four exponential or logarithmic evaluations on a modern processor in our experiments. The most significant impact is that (7) forces us to organize computations pairwise. Hu et al. present a substantial speed improvement by organizing computation pairs onto a trellis by the forward-backward algorithm, which computes *all* the output messages of the check node [13].

The next level of potential speed optimization is to approximate the $\ln(1 + \exp(-|x|))$ operations, for which results lie in the interval $(0, \ln 2]$. There has been significant work in this area, much of it focused towards hardware implementation [13], [15]. For simulation, we have often adopted the following two-piece linear approximation of Richter et al. [16].

$$\ln\left(1 + e^{-|x|}\right) \approx \begin{cases} 0.6 - 0.24\,|x|, & \text{if } |x| < 2.5 \\ 0, & \text{otherwise.} \end{cases}$$

We have noted losses of about $0.02$ dB in the AWGN waterfall region of the FER curve for the $(2640, 1320)$ Margulis code with this approximation, while the simulation runs nearly $4$ times faster. This is an acceptable trade-off for a decoder simulation tool used to study the error floor.

## IV. Additional SPA Formulations

In this section we address the numerical issues of other versions of the SPA implemented using FP computer processing. As explained in Section II-D, the $\tanh(\lambda/2)$ function loses accuracy and rounds to 1 for $\lambda \geq (p+2)\ln 2$. Thus, changing to larger FP formats for added precision increases the LLR limits only linearly. This section explores alternative formulations of SPA, some of which increase LLR dynamic range.

### A. Min-Sum Algorithm (MSA)

The min-sum algorithm (MSA) uses the following approximation to the check-node-update expression (1):

$$\lambda_l^{[i\leftarrow j]} = \min_{k\in\mathcal{N}(j)\setminus i}\left|\lambda_{l-1}^{[k\to j]}\right| \cdot \prod_{k\in\mathcal{N}(j)\setminus i} \operatorname{sign}\lambda_{l-1}^{[k\to j]}. \tag{8}$$

Since this expression is equivalent to (7) with the two logarithmic terms assumed to be zero, (8) also won't overflow. However, the decoder performance losses that have been observed by using MSA have ranged from 0.60 to 1.22 dB [15] in AWGN, depending on the code. To reduce these losses, variations on (8) known as normalized-BP and offset-BP have been used successfully [3], [15]. Note that as LLRs get very large, the MSA closely approximates the SPA as the two logarithmic terms of (7) become relatively small.

### B. Gallager's Involution Transform (GIT)

In [1], Gallager proposed another version of the check-node-update expression (1), using logarithms to replace the multiplications with additions. The resulting check-node update becomes

$$\begin{aligned}\lambda_l^{[i\leftarrow j]} = &\prod_{k\in\mathcal{N}(j)\setminus i} \operatorname{sign}\left(\lambda_{l-1}^{[k\to j]}\right) \cdot \\ &\Phi\Bigg(\sum_{k\in\mathcal{N}(j)\setminus i} \Phi\left(\left|\lambda_{l-1}^{[k\to j]}\right|\right)\Bigg),\end{aligned}$$

where we define

$$\Phi(x) \triangleq -\ln\tanh\left(\frac{x}{2}\right) = \ln\left(\frac{1+e^{-x}}{1-e^{-x}}\right), \tag{9}$$

for $x > 0$. Since the function $\Phi(x)$ is its own inverse, *i.e.*, $\Phi(\Phi(x)) = x$, for all $x > 0$, this technique is sometimes called Gallager's involution transform (GIT). Because the function $\Phi(x)$ transforms values between domains in which addition is the primary means of computing, this technique was originally proposed for low-complexity hardware implementation of the SPA. We are also aware of its appearance in SPA simulation code, in spite of the fact that addition is no faster than multiplication on a modern FP processor.

Both expressions in (9) suffer finite-precision problems prior to taking the logarithm. As explained in Section II-D, $\tanh(\lambda/2)$ is rounded to 1 for $\lambda \geq (p+2)\ln 2$, which would yield $\Phi(\lambda) = -\ln 1 = 0$. Similarly, $1 - e^{-x}$ and $1 + e^{-x}$ are rounded to 1 for $x \geq (p+1)\ln 2$ and $x > p\ln 2$, respectively. Thus, the computational limit of LLR magnitude using (9) is at best $\lambda \geq (p+2)\ln 2$.

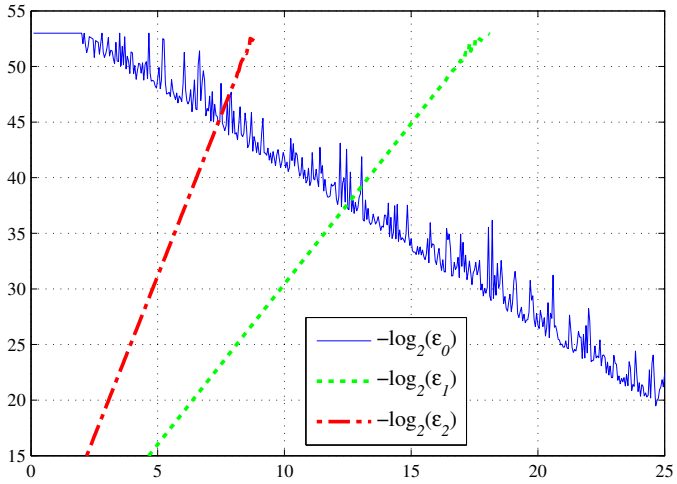


Fig. 1. Numerical accuracy of several versions of $-\ln\tanh(x/2)$ plotted as significant bits vs. $x$ (LLR value) using DP-FP computations

### C. Gallager's Involution Transform, Amended (GIT2)

Given the following series expansion in the range $x > 0$:

$$\ln\left(1+e^{-x}\right) = e^{-x} - \frac{e^{-2x}}{2} + \frac{e^{-3x}}{3} + \ldots,$$

the series expansion of (9) is readily found to be

$$\ln\left(\frac{1+e^{-x}}{1-e^{-x}}\right) = 2\left[e^{-x} + \frac{e^{-3x}}{3} + \frac{e^{-5x}}{5} + \ldots\right], \qquad (10)$$

for $x > 0$. This can clearly be approximated by a small number of terms as $x$ grows large. What has been gained is that (10) will not round to zero until $x > (emax + p)\ln 2$, thus providing a substantial increase in LLR dynamic range. Also, the value $e^{-x+\ln 2}$ will not begin to lose accuracy as it "denormalizes" until $x > emax \ln 2$.

Our next step is to find the cross-over in accuracy between (9) and (10) with a limited number of terms. Make the following computations:

$$\begin{aligned}
\Phi_0(x) &\triangleq -\ln\tanh\left(\frac{x}{2}\right),\\
\Phi_1(x) &\triangleq 2e^{-x},\\
\Phi_2(x) &\triangleq 2e^{-x} + \frac{2e^{-3x}}{3}, \quad \text{and}\\
\epsilon_i(x) &\triangleq \left|\Phi_i(x) - \Phi(x)\right| / \Phi(x),
\end{aligned}$$

where we compute $\Phi(x)$ using our best approximation at each $x$ and not (9) explicitly. The expression for $\epsilon_i(x)$ computes the relative error of the corresponding computation $\Phi_i(x)$.

In Fig. 1 we plot the relative error of each computation as $-\log_2(\epsilon_i)$ versus $x$ to show the accuracy in bits of the several computational versions of $\Phi(x)$ in DP-FP. We observe a cross-over of the lower bound of the accuracy of $\Phi_0(x)$ and the accuracy of the single-term power series $\Phi_1(x)$ at $x \approx 12.4$ and 37.2 bits of accuracy, which is still quite good. Thus, computationally, the following approximation to (9) has much greater dynamic range than directly implementing (9) on a computer, at acceptable accuracy for many applications:

$$\Phi(x) \approx \begin{cases} -\ln\tanh\left(\frac{x}{2}\right), & 0 < x < 12.4\\ e^{-x+\ln 2}, & x \geq 12.4. \end{cases} \qquad (11)$$

The value returned by (11) will not round to zero until $x > (emax + p)\ln 2$ or 745.8 in DP-FP. If even more accuracy is desired, additional terms may be employed at an earlier cross-over. For instance, Fig. 1 shows that the cross-over in the accuracy of $\Phi_0(x)$ and the two-term power series $\Phi_2(x)$ occurs at $x \approx 7.35$ with over 44 bits of accuracy. However, no larger dynamic range is achievable while employing Gallager's involution transform.

Our proposed computation (11) achieves a factor of $(emax + p)/(p + 2)$ increase in the LLR dynamic range of (9), which is approximately a 20-fold improvement for DP-FP. Also, at LLR magnitudes larger than 12.4, this simple modification achieves higher accuracy with reduced computational complexity than (9).

### D. Likelihood Ratio (LR)

An alternative to using $\tanh$ in the SPA is to perform the required computations in the likelihood ratio (LR) domain [17], [18]. Interestingly, the non-uniform FP quantization of LRs maps to nearly uniform quantization in the LLR domain. If we let $L^{[i]} \triangleq \exp\lambda^{[i]}$, $L_l^{[i\to j]} \triangleq \exp\lambda_l^{[i\to j]}$, and so on, the variable-node update becomes

$$L_l^{[i\to j]} = L^{[i]} \prod_{k\in\mathcal{N}(i)\backslash j} L_l^{[i\leftarrow k]} \quad \text{or} \qquad (12)$$

$$L_l^{[i\to j]} = \exp\Big(\ln L^{[i]} + \sum_{k\in\mathcal{N}(i)\backslash j} \ln L_l^{[i\leftarrow k]}\Big). \qquad (13)$$

Letting $\mathcal{N}(j)\backslash i = \{k_1, k_2\}$, the pairwise check-node update may be computed as

$$L_l^{[i\leftarrow j]} = \frac{1 + L_{l-1}^{[k_1\to j]} L_{l-1}^{[k_2\to j]}}{L_{l-1}^{[k_1\to j]} + L_{l-1}^{[k_2\to j]}}. \qquad (14)$$

Andrews [18] notes that multiplicative overflow within (14) is avoided if the input LRs are limited to $L_{l-1}^{[k\to j]} < \sqrt{2^{emax}(2 - 2^{1-p})}$, which corresponds to an LLR limit of about 354.89 in DP-FP. We have found a numerical improvement to this computation which doubles its LLR range. That improvement is to appear in the full version of this paper.

Note that the variable-node update (12) is more sensitive to multiplication overflow than (14) for $d_v \geq 3$. However, (12) may be re-stated as (13) which has no intermediate overflows. We may gracefully limit the argument of the exponential to restrict the output LR range. Of course, (13) is more complex to evaluate.

### E. Likelihood Difference (LD) or Tanh Domain

Another alternative is to perform the computations in the likelihood difference (LD) domain, on the interval $(-1, +1)$ [17]. This is variously known as the $\tanh$ or soft-bit domain [19]. If we let $\delta^{[i]} \triangleq P(r_i|t_i = -1) - P(r_i|t_i = +1)$ and so on, the pairwise variable-node update becomes

$$\delta_l^{[i\to j]} = \frac{\delta_l^{[i\leftarrow k_1]} + \delta_l^{[i\leftarrow k_2]}}{1 + \delta_l^{[i\leftarrow k_1]}\delta_l^{[i\leftarrow k_2]}}, \tag{15}$$

while the check-node update is simply

$$\delta_l^{[i\leftarrow j]} = \prod_{k\in\mathcal{N}(j)\setminus i} \delta_{l-1}^{[k\to j]}. \tag{16}$$

The dominant numerical issue for the check-node update is the resolution of $\delta$ in the neighborhood of $\delta = \pm 1$. The LD messages closest to absolute certainty in FP are $\pm(1 - 2^{-p})$. Since LLRs are related to LDs by $\lambda = \ln(1+\delta) - \ln(1-\delta)$, the greatest nearly certain message available in LD is equivalent to an LLR magnitude of

$$\begin{aligned} \lambda &= \ln(2 - 2^{-p}) - \ln(2^{-p}) \\ &= (p+1)\ln 2 + \ln(1 - 2^{-p-1}) \approx (p+1)\ln 2, \end{aligned} \tag{17}$$

where our approximation error is less than the available resolution of FP. Note that the variable-node update (15) may suffer from rounding to $\pm 1$ issues and divide by 0 errors for highly certain messages, but our focus in this section has been on the limitations of the check-node updates. Thus, for comparative purposes, we use (17) to represent the equivalent LLR limits of this formulation.

### F. Offset-Likelihood Difference (OLD)

The magnitude of a quantization error in (normalized) FP is roughly proportional to the amplitude of the represented value. Thus, FP quantizes with greater absolute accuracy close to 0 than close to 1. This leads us to propose that LDs be offset such that check-node computations are in the form $f = 1 - |\delta| = 1 - \tanh|\lambda/2|$, so that highly certain messages are near $f = 0$. Since the check-node update for LD (16) is odd in every input and $\operatorname{sign}(\delta) = \operatorname{sign}(\lambda)$, we may simply handle the sign at the conclusion of the check-node-message calculation.

To begin the derivation of the check-node computation for the offset-likelihood difference (OLD) algorithm we note that

$$f_l^{[i\leftarrow j]} = 1 - \prod_{k\in\mathcal{N}(j)\setminus i} \left|\delta_{l-1}^{[k\to j]}\right|$$

simplifies significantly when performed pairwise. Letting $\mathcal{N}(j) \setminus i = \{k_1, k_2\}$, the check-node computation becomes

$$f_l^{[i\leftarrow j]} = f_{l-1}^{[k_1\to j]} + f_{l-1}^{[k_2\to j]} - f_{l-1}^{[k_1\to j]} \cdot f_{l-1}^{[k_2\to j]}. \tag{18}$$

If we wish to perform the variable-node update in the LLR domain, then we note that the transformations between domains are relatively simple, since

$$f_i = \frac{2e^{-|\lambda_i|}}{1 + e^{-|\lambda_i|}} = \frac{2}{1 + e^{|\lambda_i|}} \tag{19}$$

and

$$|\lambda_i| = \ln\left(\frac{2 - f_i}{f_i}\right). \tag{20}$$

**Algorithm 1** Offset-LD Check-Node Update for One Edge

```
1: procedure OFFSET_LD_CN(n, λ)
2:     f ← 0                          ▷ f holds the magnitude, 0 < f ≤ 1
3:     s ← +1                         ▷ s holds the sign, s ∈ {+1, −1}
4:     for i ← 1, n do
5:         g ← 2 * exp(−|λ[i]|)/ [1 + exp(−|λ[i]|)]
6:         f ← f + g − f * g
7:         s ← s * sign(λ[i])
8:     end for
9:     λ_out ← s * ln((2−f)/f)
10:    return λ_out
11: end procedure
```

The pairwise calculation of (18) generalizes easily to the recursion of Algorithm 1. By definition, the range of $f$ is the interval $(0, 1]$. For simplicity Algorithm 1 is written to compute a single output message; however, a check node must compute an output for each edge. Thus, the transformed LLR values from (19) on line 5 of Algorithm 1 may be precomputed (just once) for all output messages. Furthermore, we may organize the pairwise computations on a trellis [13].

The computational complexity of Algorithm 1 is less than that of (7); however the dynamic range is also less. The numerical dynamic range is limited due to *underflowing* (*i.e.*, rounding to zero) $2e^{-|\lambda|}$ in (19). This was shown in Section IV-C to occur as LLRs exceed $(emax + p)\ln 2$.

The other numerical concern is overflowing $2/f$ in (20) when $f$ is so small that it is significantly denormalized. However, for small $f$ (*e.g.*, $f_i < 2^{-p}$), we may accurately restate (20) as $|\lambda_i| = \ln 2 - \ln f_i$, which has no major numerical concerns and fewer operations.

### G. Summary of SPA Formulations

Table II summarizes the findings on LLR limits with respect to check-node inputs. The traditional formulations based on $\tanh$ operations in (1), Gallager's involution transform, and LD are all very limited in their usable LLR range. Our preferred approach (7) has a range 306 orders of magnitude greater.

Fig. 2 shows the output accuracy of a CN using 64-bit FP calculations for several SPA formulations. The MSA approximation peaks at low LLRs and then improves substantially, while offset-BP is more accurate than MSA at low LLR and less accurate at high LLR. The approximation by Richter et al. [16] has substantially less error than the other approximations throughout the LLR range. Finally, the vertical lines indicate two exact formulations reaching their upper LLR limit and their error beginning a linear growth with respect to LLR.

## V. Hybrid SPAs and Rescaling

Fig. 2 also suggests that hybrid SPA solutions – that is, switching from a precise formulation to an approximation as

TABLE II
UPPER LLR LIMITS OF SPA FORMULATIONS WITH RESPECT TO CHECK NODE INPUTS

| Technique | LLR-equivalent limit (approx.) | LLR limit for DP-FP | LLR limit for QP-FP |
|---|---|---|---|
| (1) | $(p+2)\ln 2$ | 38.12 | 79.72 |
| (7) | $2^{emax+1}$ | $1.798 \times 10^{308}$ | $1.190 \times 10^{4932}$ |
| MSA | $2^{emax+1}$ | $1.798 \times 10^{308}$[a] | $1.190 \times 10^{4932}$[a] |
| GIT | $(p+2)\ln 2$ | 38.12 | 79.72 |
| GIT2 | $(emax+p)\ln 2$ | 745.8 | 11434 |
| LR | $(emax+1)\ln 2/2$ | 354.9 | 5678 |
| LD | $(p+1)\ln 2$ | 37.43 | 79.02 |
| OLD | $(emax+p)\ln 2$ | 745.8 | 11434 |

[a]MSA decoder approximates SPA with a performance loss of 0.6 to 1.22 dB.

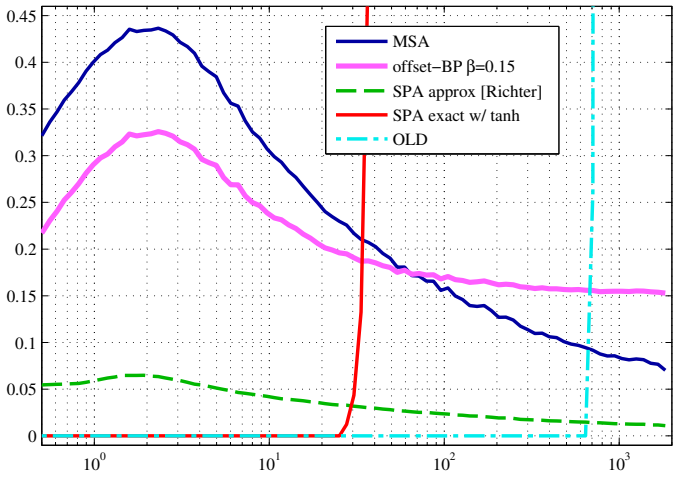


Fig. 2. RMS LLR error versus mean input LLR value, $m_\lambda$, for check node calculation with 4 input LLRs having i.i.d. Gaussian distribution with variance $\sigma_\lambda^2 = 2m_\lambda$.

LLRs grow – may be acceptable. In fact, one of our SPA decoder implementations in DP-FP switches from the Richter et al. approximation to MSA once a very large LLR, say $2^{p+3}$, is first seen. At such a point in decoding, the approximation error of the less-complex MSA is less than the resolution of the values for most of the messages in the graph.

Also, it is simple to push the LLR limits beyond that shown in Table II for formulations that already support large LLRs. At very large LLRs the update rules are largely insensitive to scaling, and MSA is always insensitive to scaling. So, once we detect an extremely large LLR in decoding, say $10^{305}$ in DP-FP, we may rescale all LLRs (including the LLRs from the channel), substantially to provide additional headroom. This alleviates the need to consider the QP-FP format for more LLR range. We recognize that, when quantizing FP in the LLR domain (as we prefer) and rescaling, the quantization steps get larger as LLR magnitudes increase. We believe this effect to be tolerable when operating in the error floor region as LLRs grow exponentially in trapping set conditions [7]–[9].

## VI. FLOATING-POINT RESOLUTION

Since range is not the only issue in quantization, we briefly examine floating-point (FP) resolution for the several domains

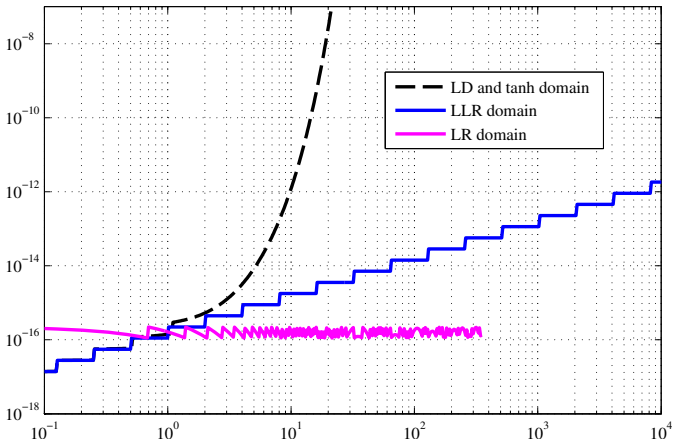


Fig. 3. 64-bit FP resolution in LLR units versus equivalent LLR $\lambda$.

covered. Fig. 3 shows the DP-FP resolution (or quantization step-size) of several domains plotted in terms of LLR resolution versus equivalent LLR. Since the curve for "LLR domain" is in its native domain, the resolution takes on discrete values. It is approximately proportional to the value represented, with a proportionality constant of $\alpha = 2^{-52.5} = 1.6 \times 10^{-16}$.

We can plot the other domains using a simple transformation. For instance, in the LR domain ($L = e^\lambda$) of Section IV-D, we find the FP-quantization steps of $L$ as a function of $\lambda$ are approximately

$$\Delta L \approx \alpha L = \alpha e^\lambda \text{ in LR units.}$$

To transform small changes in $L$ to LLR units we need only multiply by the magnitude of the derivative $|d\lambda/dL| = 1/L$ to produce

$$\left|\frac{d\lambda}{dL}\right| \Delta L \approx \alpha L/L = \alpha \text{ in LLR units,}$$

approximately a constant value. As shown in Fig. 3 the true LR resolution in LLR units dithers about our approximate result until it runs out of range. The GIT2 and OLD domains yield a resolution overlapping the LR result, but with smaller step-size for GIT2 at LLRs less than 0.2. For the LD or $\tanh$ domain of we perform the same analysis and produce the top curve in Fig. 3, which shows the step-size growing rapidly.

## VII. SIMULATION

We present performance simulation results for two codes to demonstrate the techniques of this paper. Fig. 4 shows the frame error rate (FER) of the $(2640, 1320)$ Margulis code, which is a $(3, 6)$-regular LDPC code. We show MacKay and Postol's [4] results in addition to our own. While they show the start of an error floor at $10^{-6}$ at $E_b/N_0 = 2.4$ dB, other studies have found the floor to be substantially higher [3], [5], [6]. MacKay and Postol note that this error floor is caused by certain near codewords [4]. Our own simulation, using a non-saturating SPA decoder running for a maximum of 200 iterations, showed no sign of a floor down to $10^{-8}$ and significantly lower. In fact, resorting to techniques similar to

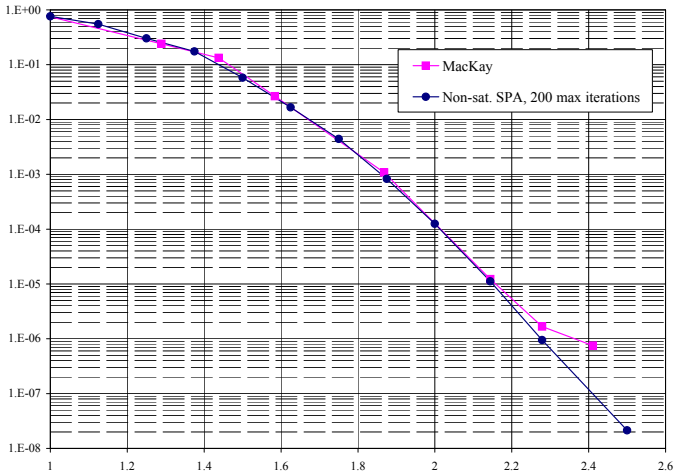


Fig. 4. FER vs. $E_b/N_0$ in dB for the Margulis LDPC code in AWGN.

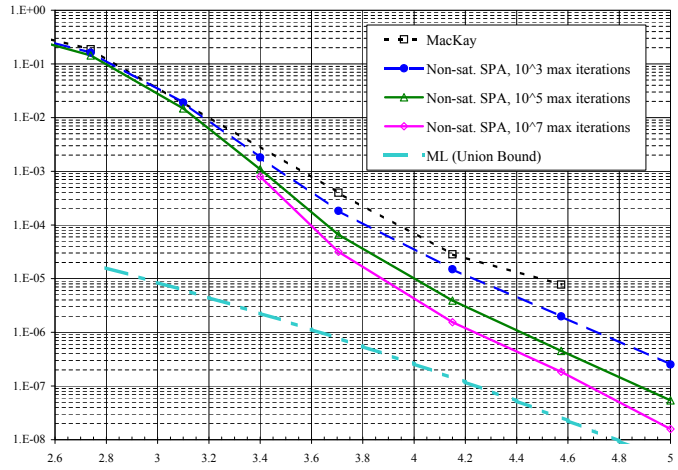


Fig. 5. FER vs. $E_b/N_0$ in dB for $n = 1057$ LDPC code in AWGN.

importance sampling we found an error-rate contribution due to the supposedly dominant near codewords of just $2 \times 10^{-11}$ at $E_b/N_0 = 2.8$ dB for this decoder [9].

Fig. 5 shows the FER results for an LDPC code listed as 1057.244.3.457 in [20]. It is a $(3, 13)$-regular code, with block length $n = 1057$, rate $R \approx 0.77$, and $d_{min} = 8$. Again, we compare our results to those of MacKay. We used our non-saturating SPA decoder with three settings for the maximum number of iterations: $10^3$, $10^5$, and $10^7$. The average number of decoder iterations performed at $E_b/N_0 = 4.574$ dB was $2.290$, $2.344$, and $4.08$, respectively. Despite the small increase in average iterations required, significant performance improvements were obtained by using the larger maximum number of iterations.

## VIII. Conclusion

We have addressed numerical limitations on the allowable size of log-likelihood ratios (LLRs) in floating-point (FP) simulations of several formulations of the sum-product algorithm (SPA). We have described preferred techniques to accommodate very large LLR values and even unbounded range through rescaling. Additionally, we have proposed simple numerical improvements to Gallager's involution transform that extends its dynamic range by a factor of about 20 (for double-precision FP computations). Finally, we introduced a new exact SPA formulation, "offset-likelihood difference," which supports a moderate LLR range with low computational complexity.

## Acknowledgment

The authors would like to thank Kenneth Andrews and Ido Tal for the helpful discussions.

## References

[1] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA: M.I.T. Press, 1963.

[2] T. J. Richardson and R. L. Urbanke, *Modern Coding Theory*. Cambridge, UK: Cambridge Univ. Press, 2008.

[3] W. E. Ryan and S. Lin, *Channel Codes: Classical and Modern*. Cambridge, UK: Cambridge Univ. Press, 2009.

[4] D. J. C. MacKay and M. J. Postol, "Weaknesses of Margulis and Ramanujan–Margulis low-density parity-check codes," in *Proc. of the 2nd Irish Conf. on the Math. Foundations of Comput. Sci. and Inf. Tech. (MFCSIT)*, ser. Electronic Notes in Theoretical Comput. Sci., vol. 74, Galway, Ireland, 2003.

[5] T. J. Richardson, "Error-floors of LDPC codes," in *Proc. of the 41st Annu. Allerton Conf.*, Monticello, IL, Oct. 2003, pp. 1426–1435.

[6] Y. Han and W. E. Ryan, "Low-floor decoders for LDPC codes," *IEEE Trans. Commun.*, vol. 57, no. 6, pp. 1663–1673, Jun. 2009.

[7] J. Sun, "Studies on graph-based coding systems," Ph.D. dissertation, Dept. Elect. and Comput. Eng., Ohio State Univ., Columbus, 2004.

[8] B. K. Butler and P. H. Siegel, "Error floor approximation for LDPC codes in the AWGN channel," in *Proc. of the 49st Annu. Allerton Conf.*, Monticello, IL, Sep. 2011, pp. 204–211.

[9] ——, "Error floor approximation for LDPC codes in the AWGN channel," Feb. 2012, submitted to *IEEE Trans. Inf. Theory*. [Online]. Available: http://arxiv.org/abs/1202.2826

[10] X. Zhang and P. H. Siegel, "Quantized min-sum decoders with low error floor for LDPC codes," in *Proc. IEEE Int. Symp. on Inform. Theory*, Cambridge, MA, Jul. 2012, pp. 2881–2885.

[11] T. Etzion, A. Trachtenberg, and A. Vardy, "Which codes have cycle-free Tanner graphs?" *IEEE Trans. Inf. Theory*, vol. 45, no. 6, pp. 2173–2181, Sep. 1999.

[12] *IEEE Standard for Floating-Point Arithmetic*, IEEE Std. 754-2008, Aug 29, 2008.

[13] X.-Y. Hu, E. Eleftheriou, D.-M. Arnold, and A. Dholakia, "Efficient implementations of the sum-product algorithm for decoding LDPC codes," in *Proc. IEEE Global Telecommun. Conf.*, vol. 2, San Antonio, TX, Nov. 2001, pp. 1036–1036E.

[14] A. Anastasopoulos, "A comparison between the sum-product and the min-sum iterative detection algorithms based on density evolution," in *Proc. IEEE Global Telecommun. Conf.*, vol. 2, San Antonio, TX, Nov. 2001, pp. 1021–1025.

[15] J. Chen and M. P. C. Fossorier, "Density evolution for BP-based decoding algorithms of LDPC codes and their quantized versions," in *Proc. IEEE Global Telecommun. Conf.*, vol. 2, Taipei, Taiwan, Nov. 2002, pp. 1378–1382.

[16] G. Richter, G. Schmidt, M. Bossert, and E. Costa, "Optimization of a reduced-complexity decoding algorithm for LDPC codes by density evolution," in *Proc. IEEE Int. Conf. on Commun.*, vol. 1, Seoul, South Korea, May 2005, pp. 642–646.

[17] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 498–519, Feb. 2001.

[18] K. Andrews, "LDPC codes: Design and shortcomings," Mar. 2001, unpublished.

[19] S. L. Howard, V. C. Gaudet, and C. Schlegel, "Soft-bit decoding of regular low-density parity-check codes," *IEEE Trans. Circuits Syst. II*, vol. 52, no. 10, pp. 646–650, Oct. 2005.

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