Optimization of LDPC Finite Precision Belief Propagation Decoding with Discrete Density Evolution

D. Declercq and F. Verdier

ETIS ENSEA/UCP/CNRS UMR-8051
6, avenue du Ponceau 95014 Cergy-Pontoise (France)
e-mail: declercq@ensea.fr, verdier@ensea.fr

Abstract: In this paper, we study the impact of the finite precision coding of the messages used for LDPC Belief Propagation decoding. The finite precision deteriorates the code performance, and we explain theoretically this performance loss with a quantized version of Density Evolution. Then, we point out the weaknesses of the quantized Belief Propagation decoder and propose a modification of the algorithm that reduces the gap between finite precision decoding and infinite precision decoding.

1. Introduction

In this paper, we will investigate theoretically the effect of finite precision implementation on LDPC decoders. A noticeable breakthrough among works on LDPC codes has been the work of Richardson and Urbanke [1] who showed that irregular LDPC codes can be parametrized and that the asymptotical behaviour of LDPC codes could be related to these parameters. Using this approach, called Density Evolution, it is possible to predict the gap between the decoding threshold of a particular LDPC code and the Shannon limit, and to optimize the code parameters in order to minimize this gap.

Meanwhile, LDPC codes are being considered for various communication applications such as multiuser transmission, DAB/DVB, wireline ADSL transmission as proposed to the ITU SG15/Q4 committee, and data storage. This rises the interest in designing efficient hardware implementation of LDPC codes.

In a hardware implementation of LDPC codes, the Belief Propagation decoder has to tackle quantized values of the messages, which deteriorates the code performance. In this paper, we study theoretically with a quantized version of density evolution the impact of the finite precision coding of the data values used in the LDPC decoder. This study helped us to characterize with accuracy the finite precision LDPC decoder behavior and led us to a modification of the decoder that improves the error correcting performance in the case of finite precision decoding.

The paper will be organized as follows: in section 2, we recall the notations regarding LDPC codes and the Belief Propagation decoder helpful for the rest of the paper. Section 3 starts by the study of the finite precision coding impact on the code performance with a quantized density evolution. Then, the modification of the LDPC decoder is presented and we show both theoretically and with simulations the performance gain that we obtain. The advantage of our approach is illustrated through simulations on the regular (3, 6) LDPC code.

2. Basics on LDPC decoding

H is the LDPC \((M \times N)\) parity check matrix, where \(N\) denotes the codeword length and \(M\) the number of parity checks (if \(R\) is the code rate, \(M = (1 - R)N\)). A codeword \(c\) of length \(N\) is such that \(Hc = 0\).

A LDPC code may also be represented by its factor graph which is a bipartite graph with two kinds of nodes: data nodes representing the codeword bits and function nodes representing the parity checks [2]. An irregular LDPC code is specified by two polynomials: \(\lambda(x) = \sum_{i=2}^{\text{max}} \lambda_i x^{i-1}\) and \(\rho(x) = \sum_{j=2}^{\text{max}} \rho_j x^{j-1}\) where \(\lambda_i\) is the fraction of edges which are connected to a degree \(i\) data node and \(\rho_j\) is the fraction of edges which are connected to a degree \(j\) check node.

The decoder input consists in the likelihood ratios computed from the AWGN channel observations \(y_n\) and is given by \(u_0(n) = 2y_n/\sigma^2\), where \(\sigma^2\) is the noise variance. We will denote \(u^{(l)}\) the messages entering the variable nodes at the decoding iteration \(l\), and \(v^{(l)}\) the message entering a check node at iteration \(l\). Both \(u^{(l)}\) and \(v^{(l)}\) are given in log density ratio form: \(\log \left( \frac{p(c=0|l)}{p(c=1|l)} \right)\) (cf. figure 1). Using these notations, the output message of a degree \(i\) data node is:

\[
v_m^{(l)} = u_0 + \sum_{k=1,k \neq m}^i u_k^{(l-1)}
\]

The message update through a check node is a convolutional product and it is more efficient to perform it in the Fourier domain. The function \(f(x) = \log \tanh(|x|/2)\) transforms a log-density message in the log-Fourier domain, and using \(g(x) = 2 \tanh^{-1} \exp(x)\) as the inverse function of \(f(x)\), we get at the output of a degree \(j\) check node:

\[
u_m^{(l)} = g \left( \sum_{k=1,k \neq m}^j f(v_k^{(l)}) \right) \times \prod_{k=1,k \neq m}^j \text{sign}(v_k^{(l)})
\]

After \(N_{it}\) iterations of Belief Propagation decoding, we take a decision on the bit BPSK-value, given by the sign of the \textit{a posteriori} probability.

3. Finite Precision Density Evolution

Now we will study theoretically the effect on finite precision data coding on the performance of the LDPC
codes. We will make this study through density evolution tools applied to discrete random variables. This idea is not new since it has been presented for example in [3], but the discrete density evolution depends on the assumptions that are made on the finite precision coding and the version of the Belief Propagation algorithm that is used. We present in this Section discrete density evolution applied to equations (1) and (2), assuming that the functions \( f(x) \) and \( g(x) \) are stored in Look Up Tables (LUT). We do not claim that the results we obtain could be applied to another implementation of Belief Propagation. We will see however that our approach has some advantages compared to existing work.

The following assumptions that we made on the hardwire implementation will lead to modifications in the density evolution process: (i) the messages which represent log-density ratios are stored on \( n \) bits. We will differentiate the \( n_i \) bits used for the integer part of the messages and the \( n_f \) bits used for the fractional part of the messages. Therefore, a \( n \) bits message \( u \) takes its values in \([-2^{n_i}+2^{-n_f}:2^{-n_f}:2^{n_i}-2^{-n_f}]\) (in Matlab notations). We will denote this finite precision \( Q_n, n_f \). (ii) the functions \( f(x) \) and \( g(x) \) used in equation (2) are stored in Look Up Tables (LUT), and we assume that the precision of the tables input/output are the same as for the messages, that is the output of the LUT is a \( Q_n, n_f \)-value.

Density Evolution of the discrete messages is obviously reduced to the evolution of histograms, representing the discrete densities. The histograms containers are centered on the values \([-2^{n_i}+2^{-n_f}:2^{-n_f}:2^{n_i}-2^{-n_f}]\). Discrete density evolution can be summarized in 4 steps (see figure 1 for the notations).

**step 1:** For a degree \( i \) data node, we assume that there are \((i-1)\) independent \( Q_n, n_f \)-messages entering a data node plus the \( Q_n, n_f \)-message coming from the channel likelihood ratio \( u_0 \). Taking into account the different data node degrees, the density of \( u \) is a mixture of densities, with the coefficients of \( \lambda(x) \) being the mixing parameters. The convolution does not change the quantum of the message values, so we just need to saturate the resulting mixture of convolutions. We denote the saturation of the density the following way (the same notation is used for 2-side or 1-side densities):

\[
p\left(v^{(i)}\right) = \text{Sat} \left\{ \sum_{i=2}^{t_{\text{max}}} \lambda_i p\left(u_0\right) \otimes p\left(u^{(i)}\right)^{(i-1)} \right\}
\]

**step 2:** Then, we must make a change of variables in order to express the density of \( f(u^{(i)}) \). Because we need to take into account separately the sign of the messages, we will consider the decomposition of the density:

\[
p\left(v^{(i)}\right) = p^+\left(v^{(i)}\right) + p^-\left(v^{(i)}\right)
\]

\[
\left\{
\begin{array}{l}
p^+(x) = p(x) \mathbb{I}_{[0,2^{-n_f}:2^{n_i},-2^{-n_f}]}
p^-(x) = p(x) \mathbb{I}_{[-2^{n_i}+2^{-n_f}:2^{-n_f}:0]}
\end{array}
\right.
\]

The change of variables \( \Gamma(\cdot) \) corresponding to \( f(x) = \log \tanh(|x|/2) \) does not preserve the quantum of messages so we have to perform an histogram equalization - both for \( \Gamma \{p^+(u^{(i)})\} \) and \( \Gamma \{p^-(u^{(i)})\} \). This histogram equalization is just a re-ordering of the probability weights that are centered on \([f(0): f(2^{-n_f}): f(2^{n_i}-2^{-n_f})]\) around the values \([-2^{n_i}+2^{-n_f}: 2^{-n_f}: 0]\).

\[
p^{\pm} \left(f\left(u^{(i)}\right)\right) = \text{Sat} \left\{ \text{Equal} \left( \Gamma \{p^{\pm}\left(u^{(i)}\right)\} \right) \right\}
\]

**step 3:** Now we have \((j-1)\) \( Q_n, n_f \)-messages \( f(u^{(i)}) \) entering the check node. The output of this check node is still a \( Q_n, n_f \)-value, and the sign of this \( Q_n, n_f \)-value is given by the product of the signs of the input messages. Therefore, the sign of the output message is negative if and only if an odd number of input messages are negative. This leads to the following density evolution through a degree \( j \) check node:

\[
p^+_j \left(f\left(u^{(i)}\right)\right) = \sum_{k=0}^{j-1} \binom{j-1}{k} p^+ \left(f\left(u^{(i)}\right)\right)^{\otimes(j-1-k)} \otimes p^- \left(f\left(u^{(i)}\right)\right)^{\otimes k}
\]

\[
p^-_j \left(f\left(u^{(i)}\right)\right) = \sum_{k=0}^{j-1} \binom{j-1}{k} p^+ \left(f\left(u^{(i)}\right)\right)^{\otimes(j-1-k)} \otimes p^- \left(f\left(u^{(i)}\right)\right)^{\otimes k}
\]

Using these expressions, the densities corresponding to the different signs of the output message come from a mixture of \( t_{\text{max}} \) densities and need also to be saturated:

\[
p^{\pm} \left(f\left(u^{(i)}\right)\right) = \text{Sat} \left\{ \sum_{j=2}^{t_{\text{max}}} \rho_j p^{\pm}_j \left(f\left(u^{(i)}\right)\right) \right\}
\]

**step 4:** Now we have to make the change of variable \( \Gamma^{-1}(\cdot) \) corresponding to \( g(x) \) in order to go back to the \( u^{(i)} \) domain. Again, this change of variable is done using an histogram equalization and we get finally the density \( p\left(u^{(i)}\right) = p^+\left(u^{(i)}\right) + p^-\left(u^{(i)}\right) \), with:

\[
p^{\pm} \left(u^{(i)}\right) = \text{Sat} \left\{ \text{Equal} \left( \Gamma^{-1} \{p^{\pm}\left(f\left(u^{(i)}\right)\right)\} \right) \right\}
\]

Let us recall that the discrete density evolution that we just described is dependant on the chosen algorithm implementation. In particular, because we suppose that the functions \( f(x) \) and \( g(x) \) are tabulated
We can notice that the number of bits for the integer part has a direct consequence on the error floor level. For coding precisions with the same integer precision, the more bits are used in \( n_i \), the lower is the error floor. Note that this error floor comes from the finite precision coding of the messages and not from a finite codeword length. As for the number of bits for the fractional part, it determines the threshold value. Above this threshold, the finite precision Belief Propagation has a bit error probability corresponding to the error floor level. This is seen in figure 2 since the error floor appears approximately at the same SNR for coding precisions with the same \( n_f \). The more bits are used in \( n_f \), the closer is the threshold to the infinite precision threshold (as therefore to the Shannon Capacity).

In order to verify experimentally the theoretical behavior of the Belief Propagation algorithm, we have simulated some finite precision cases for the regular \((t_c = 3, t_r = 6)\) LDPC code.

We can see on figure 3 that the comments we have made for the Density Evolution curves are verified by simulation. Essentially, when \( n_i \) increases, the error floor gets lower and when \( n_f \) increases, the convergence appears at a lower SNR. Of course, the behavior is verified, but the predicted values of the thresholds and the error floors are different due to the finite length of the codewords.

### 4. Finite Precision Constrained Modification of Belief Propagation

The study presented in the last Section led us to modify the decoding algorithm in order to take more precisely into account the effect of the finite precision on the decoding steps. In particular, the fact that the functions \( f(x) \) and \( g(x) \) are stored in finite lookup tables reduces greatly the performance of LDPC codes. One solution is to increase the size of those LUT, but this will also increase the size of the required memory on the architecture. The problem in the quantization of the function \( f(x) \) is that it is very sensitive to quantization errors introduced for values of \( x \) close to 0. Moreover, we can remark that equation (2) is still valid for any base of logarithm. We propose then to adapt the base of the logarithms of messages such that the LUT of the function \( f(x) \) are efficiently stored - that is we get the minimum quantization errors for small values of \( x \).

This modification of Belief Propagation is then simply to multiply (or divide) the messages by a constant, and is achieved only by tabulating a different function in the LUT. This modification of the Belief Propagation algorithm is totally adapted to the hardware choices that we have made and works only because we use a sub-optimal (quantized) version of Belief Propagation. It would not increase the performance for an infinite precision decoder. In order to choose the optimal logarithm base, we have adapted the discrete density evolution presented in the last Section. Note that it only involves modifications during the changes of variables \( \Gamma(\cdot) \) and \( \Gamma^{-1}(\cdot) \). The op-
As seen on figure 4 there exists an optimal value of the logarithm base that depends on the coding precision and the LDPC code. In terms of hardware storage, we could comment the figure 4 by noticing that the threshold is approximately the same for the Q3.4 precision and the optimal base as for the Q3.6 precision and the natural base. This means that theoretically, we could store the messages in finite precision with 2 bits less using the optimal logarithm base, without sacrificing performance.

We have confirmed these remarks by simulations on figure 5. The plotted curves are obtained with $N = 40000$, and $N_{it} = 100$ decoding iterations. The theoretical behaviour of the modified-base algorithm is verified. The performance improvement in the case of the Q3.4 precision is approximately $0.15dB$ while it is only $0.05dB$ for the Q3.6 precision. Note that with the modified logarithm base, the performance of the Q3.4 precision is better than for the Q3.6 precision with natural seed, allowing and effective 2-bits coding gain on the messages, with no loss of performance. In [4], we have tabulated the optimal bases for various irregular LDPC codes and various coding precisions.

5. Conclusion

In this paper, we have studied through a quantized version of Density Evolution the effect of finite precision Belief Propagation. Although it is specific to a given hardware implementation of LDPC decoders, our study helped us to explain theoretically some comments that have been made in the finite precision literature [5], regarding the separate effect of integer and fractionnal precisions. Then, we have proposed a slight modification of the finite precision Belief Propagation in order to reduce the performance loss with regards to its infinite precision version. By changing the tabulated functions used in the decoding steps, we have gained $0.05dB - 0.2dB$ depending on the precision used. The approach we have introduced in this paper in order to improve the performance of a sub-optimal version of Belief Propagation has some connections with the work of Chen and Fossorier in [6]. It would be interesting to explore more deeply the connections between these works.

REFERENCES