1 Low-Density Parity Check codes

1.1 Basic introduction in LDPC codes

Before we begin a discussion of low density parity check codes (LDPC) we review some basics of linear block codes. We shall see that the other codes considered so far in this book (parallel concatenation, serial concatenation, product turbo codes) all can be framed in a common language: that of low density parity check codes. For simplicity we will assume that all codes are binary. We shall also see that LDPCs are conceptually very simple, they are defined in terms of a sparse parity check matrix. The decoder uses the *message passing algorithm* which we discuss at length shortly. We provide a complete description and proof of the decoding algorithm. For those not so interested in proofs, we give pseudo-code implementations of the decoders that requires very little knowledge of how and why the message passing algorithm works. The message passing algorithm is highly parallel and is ideally suited for high data rate applications. Careful programming of the decoder results a surprisingly simple decoder.

Before proceeding we review some basics of linear block codes. An (n, k) block code $\mathcal{C}$ is a mapping between a k-bit message (row) vector m, and an n length codeword vector c. The code $\mathcal{C}$ is linear if is a $k-$ dimensional subspace of an $n-$dimensional binary vector space V_n. The code can also be viewed as a mapping of k-space to n-space by a $k \times n$ generator matrix G,where $c = mG$. The rows of G constitute a *basis* of the code subspace. The *dual space*, $\mathcal{C}^{\perp}$ consists of all those vectors in V_n orthogonal to $\mathcal{C}$, namely for all $c \in \mathcal{C}$ and all $d \in \mathcal{C}^{\perp}$, $< c, d >= 0$. The rows of an $n-k \times n$ parity check matrix H constitute a basis for $\mathcal{C}^{\perp}$. It follows that for all $c \in \mathcal{C}$, $cH^T = 0$. A code is completely specified by either G or H, but neither are unique.

Low density parity check codes were invented by Gallager [**?**, **?**]. Their description is among the simplest of any code. A low density parity check code is one where the parity check matrix is binary and sparse, where most of the entries are zero and only a small fraction are 1's. In its simplest form the parity check matrix is constructed at random subject to some rather weak constraints on H. A $t - regular$ LDPC is one where the column weight (number of ones) for each column is exactly t resulting in an average row weight of $nt/(n-k)$. One might fix the row weight to be exactly $s = nt/(n-k)$. An (s, t)-regular LDPC is one where both row and column weights are fixed.

The following parity check matrix H is an LDPC matrix with $t = 2$

$$H = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{1}$$

and for any valid codeword c

$$H = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_5 \\ c_6 \end{bmatrix} = 0 \tag{2}$$

This expression (2) serves as the starting point for constructing the decoder. The matrix/vector multiplication in (2) defines a set of *parity checks*, which for the specific example are

$$\begin{aligned} p_o &= c_0 \oplus c_3 \oplus c_4 \oplus c_5 \oplus c_6 \\ p_1 &= c_1 \oplus c_3 \oplus c_5 \\ p_2 &= c_1 \oplus c_2 \oplus \\ p_3 &= c_0 \oplus c_2 \oplus c_4 \oplus c_6 \end{aligned} \tag{3}$$

A complete discussion of the message passing decoder is given in the next section. Next, we address encoding. By defining an LDPC code in terms of H alone it is not obvious what consitutes the set C of valid codewords. Furthermore we need to specify the generator matrix G for the encoder. A straightforward way of doing this is to first reduce H to systematic form $H_{sys} = [I_{n-k}|P]$. In principle this is simple using Gaussian elimination and some column reordering. As long as H is full (row) rank, H_{sys} will have $n-k$ rows. There is some probability, however, that some of the rows of H are linearly dependent. In that case H is not full rank and the resulting H will be in systematic form, albeit with fewer rows. Once H is in systematic form, it is easy to confirm that a valid (systematic) generator matrix is $G_{sys} = \left[P^T|I_k\right]$ since $GH^T = 0$. It is interesting to note that the H_{sys} no longer has fixed column or row weight, and with high probability P is dense. The denseness of P can make the encoder quite complex.

As an example the parity check matrix in (2) can be reduced to systematic form as

$$H_{systematic} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \tag{4}$$

which can be expressed as $H_{systematic} = [I_{n-k}|P]$. This immediately determines the systematic generator matrix $G = \left[-P^T|I\right]$.

1.2 Relation of other codes to LDPCs

. In this section we show that conventional parallel and serial concatenated turbo codes are low density parity check codes, making the class of codes it

presents to be quite powerful.

Consider first a convolutional code. a parallel concatenated (Berrou) turbo code shown in Figure YY (ADD TEXT).

A binary low-density parity check (LDPC) code is a binary linear code defined by a sparse parity check matrix H. Suppose that the parity check matrix A has N columns and M rows. Then the codeword consists of N bits which satisfy M checks, where the location of a 1 in the parity check matrix indicates which that a bit is involved in a parity check. The total length of the codeword is N bits, the number of message bits is $K = N - M$, and the rate of the code is $\frac{K}{N}$, assuming that the matrix is full rank. As shown in Figure **??**, it is possible to associate a bipartite graph to this parity check matrix, consisting of N bit nodes (indicated by the white circles) and M check nodes (indicated by the black squares), with an edge between the bit and check nodes if there is a 1 in the parity check matrix. It is known how to design irregular parity check matrices (ref. [**?**]) to achieve performance extremely close to the Shannon limit of the channel. In this paper, we consider regular parity check matrices, where each column has a fixed number of 1's and each row has a fixed number of 1's, since this serves as a common benchmark of performance.

For now we shall consider a simple AWGN channel model wihout ISI. Later we shall show how the ISI channel of Figure 5 connects to the LDPC. Let $r = x + n$ where $x = (x_0, \ldots, x_{N-1})$ is the codeword out of the LDPC encoder and r is the received version of x and where n is an IID vector of Gaussian random variables where each component has variance σ^2. We assume that $x_j = 0$(resp. $x_j = 1$) are transmitted as a -1 (resp. $+1$). The decoding problem is to estimate the c's from the r's, by computing $LLR^{\text{posterior}}(x_j) = log\left(\frac{p(x_j=0|r)}{p(x_j=1|r)}\right)$ for all j. If $LLR^{\text{posterior}} > 0, \hat{x}_j = 0$ otherwise $\hat{x}_j = 1$ The N code bits must satisfy all parity checks, and we will use this fact to compute the posterior probability (APP) $p(x_j = b|S_j, r)$, where S_j is the event that all parity checks associated with x_j have been satisfied.

1.3 Algorithms for LDPC decoder

The message passing algorithm is an APP algorithm only if the code graph has no cycles. LDPC codes have cycles, so strictly speaking the message passing algorithm does not compute the APPs. However the algorithm performs remarkably well. The cycle-free requirement implies that all code bits $x_0, \ldots, x_{N-1}$ are independent, which clearly they are not. At the end of this section algorithms are given for implementing the message passing algorithm in both probability and log domains. We now derive the message passing algorithm for LDPCs from first principles. Prior to decoding the decoder has the following: a parity check matrix H, its bipartite graph, and N channel outputs r. Let $M(j)$ be the set of parity nodes connected to the code bit x_j, and let $N(m)$ be the set of bit nodes connected to the m'th parity check.

1.4 Message passing algorithm

The derivation given here most closely follows the description given by Gallager. Using the assumption of code bit independence and Baye's rule the APP $p(x_j = b|S_j, r)$ can be rewritten as

$$p(x_j = b|S_j, r) = Kp(r_j|x_j = b)p(S_j|x_j = b, r) \tag{5}$$

where K is a constant for both $b = 0, 1$ and can be ignored. The first term is easy to compute, which for Gaussian noise

$$p(r_j|x_j = b) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp \frac{(r_j + (-1)^b)^2}{2\sigma^2} \tag{6}$$

and for a BSC (with crossover probability p)

$$p(r_j|x_j = 0) = p^{r_j}(1-p)^{1-r_j} \tag{7}$$
$$p(r_j|x_j = 1) = p^{1-r_j}(1-p)^{r_j} \tag{8}$$

The second term in (6) is the probability that all parity checks connected to x_j are satisfied given r and $x_j = b$. Note that $S_j = \{S_{0j}, \ldots, S_{kj}\}$ is a collection of events, where S_{mj} is the event that the m'th parity node connected to x_j is satisfied. Again by independence of the code bits $(c_0, \ldots, c_{N-1})$ this can be written as

$$p(S_j|x_j = b, r) = p(S_{0j}, S_{1j}, \ldots, S_{kj}|x_j = b, r) = \prod_{m \in M(j)} p(S_{mj}|x_j = b, r) \tag{9}$$

where $p(S_{mj}|x_j = b, r)$ is the probability that the m'th parity check connected to the bit x_j is satisfied given $x_j = b$ and r. If $b = 0$ this is the probability that the code bits other than x_j connected to the m'th parity check have an even number of 1's. If $b = 1$, the other code bits must have odd parity. Using this fact we next show that $p(S_{mj}|x_j = b, r)$ has a relatively simple form.

As a preliminary calculation, suppose two bits satisfy a parity check constraint $x_1 \oplus x_2 = 0$, and it is known that $p_1 = P(x_1 = 1)$ and $p_2 = P(x_2 = 1)$. Let $q_1 = 1 - p_1$ and $q_2 = 1 - p_2$. Then the probability that the check is satisfied is

$$\begin{aligned} P(x_1 \oplus x_2 = 0) &= (1-p_1)(1-p_2) + p_1 p_2 \\ &= 2p_1 p_2 - p_1 - p_2 + 1 \end{aligned}$$

which can be rewritten as

$$2P(x_1 \oplus x_2 = 0) - 1 = (1-2p_1)(1-2p_2) = (q_0 - p_0)(q_1 - p_1). \tag{10}$$

Now suppose that $L + 1$ bits satisfy an even parity-check constraint $x_0 \oplus x_1 \oplus x_2 \oplus \cdots \oplus x_L = 0$, as pictured in the factor graph in Figure **??**.

Then for known probabilities $\{p_1, p_2, ..., p_L\}$ corresponding to the bits $\{x_1, x_2, ..., x_L\}$, it is possible to generalize (11) to find the probability distribution on the binary sum $z_L = x_1 \oplus x_2 \oplus \cdots \oplus x_L$, where $z_L = z_{L-1} \oplus x_L$

$$\begin{aligned} 2P(z_L = 0) - 1 &= (1 - 2P(z_{L-1} = 1))(1 - 2p_L) \\ &= (2P(z_{L-1} = 0) - 1)(1 - 2p_L) \end{aligned}$$

where $p_L = P(x_L = 1)$. Applying this recursively yields

$$2P(z_L = 0) - 1 = \prod_{i=1}^{L} (1 - 2p_i). \tag{11}$$

or

$$P(z_L = 0) = \frac{1}{2}\left(1 + \prod_{i=1}^{L}(1 - 2p_i)\right) = \frac{1}{2}\left(1 + \prod_{i=1}^{L}(q_i - p_i)\right). \tag{12}$$

Similarly it is possible to show

$$P(z_L = 1) = \frac{1}{2}\left(1 - \prod_{i=1}^{L}(q_i - p_i)\right). \tag{13}$$

Returning to our calaculation of $p(S_{mj}|x_j = b, r)$, if $x_j = 0$ then we use $P(z_L = 0)$ and if $x_j = 1$ we use $P(z_L = 1)$.

$$p(S_{mj}|x_j = 0, r) = \frac{1}{2}\left(1 + \prod_{n'=N(m)\backslash j}\left(q^0_{mn'} - q^1_{mn'}\right)\right) \tag{14}$$

$$p(S_{mj}|x_j = 1, r) = \frac{1}{2}\left(1 - \prod_{n'=N(m)\backslash j}\left(q^0_{mn'} - q^1_{mn'}\right)\right) \tag{15}$$

where $q^0_{mn'}$ is the probability that code bit $x_{n'}$ is zero, given r and excluding any information about $x_{n'}$ from parity check m. This exclusion is needed because we desire extrinsic knowledge about $x_{n'}$ from its parity checks to get extrinsic knowledge about x_j. Note that the rightmost product in (16) is over all code bits connected to the m'th parity node except for x_j, since we are interested in the even or odd parity of the bits other than x_j.

Combining these results with (6) and (10) we get the final expressions for the APPs.

$$\begin{aligned} p(x_j = 0|S_j, r) &= Kp(r_j|x_j = 0)p(S_j|x_j = 0, r) \\ &= p(r_j|x_j = 0)\prod_{m\in M(j)}\frac{1}{2}\left(1 + \prod_{n'=N(m)\backslash j}\left(q^0_{mn'} - q^1_{mn'}\right)\right) \end{aligned} \tag{16}$$

$$
\begin{aligned}
p(x_j = 1|S_j, r) &= Kp(r_j|x_j = 1)p(S_j|x_j = 1, r) \\
&= p(r_j|x_j = 1) \prod_{m \in M(j)} \frac{1}{2} \left(1 - \prod_{n' = N(m) \backslash j} \left(q^0_{mn'} - q^1_{mn'} \right) \right) \qquad (17)
\end{aligned}
$$

Careful inspection of the APPs in (20) one sees that many of the calculations can be done in parallel. Furthermore, some can be viewed as "parity node" computations and others as "bit node" computations, for example, for $b = 1$, $p(x_j = 1|S_j, r)$ is

$$
\underbrace{\overbrace{p\left(y \mid x_j = 1\right)}^{\text{prior}} \prod_{m \in M(j)} \overbrace{\frac{1}{2} \left(1 - \prod_{n' = N(m) \backslash j} \left(q^0_{mn'} - q^1_{mn'} \right) \right)}^{\text{parity node}}}_{\text{bit node}} . \qquad (18)
$$

The notation can be simplified by letting $\delta q_{mj} = q^0_{mj} - q^1_{mj}$ and defining parity check equations as

$$
\begin{aligned}
r^0_{mj} &= \frac{1}{2} \left(1 + \prod_{n' = N(m) \backslash j} \delta q_{mn'} \right) \\
r^1_{mj} &= \frac{1}{2} \left(1 - \prod_{n' = N(m) \backslash j} \delta q_{mn'} \right) . \qquad (19)
\end{aligned}
$$

With this new notation, we obtain the message passing algorithm given in Appendix A.1.

For the BSC, the expressions in (20) can be simplified a great deal. We begin by fleshing out what happens in the first iteration of the decoder. The probabilities q^0_{mj} and q^1_{mj} get initialized as

$$
q^0_{mj} = p(r_j|x_j = 0) = p^{r_j}(1 - p)^{1 - r_j} \qquad (20)
$$

$$
q^1_{mj} = p(r_j|x_j = 1) = p^{1 - r_j}(1 - p)^{r_j} \qquad (21)
$$

Depending on the value of the received bit r_j the initialized q^i_{mj}'s take on only one of two values. The expressions in (19) can be simplified as follows. First recognize that if $r_j = 0$

$$
\begin{aligned}
q^0_{mj} - q^1_{mj} &= p^{r_j}(1 - p)^{1 - r_j} - p^{1 - r_j}(1 - p)^{r_j} \qquad (22) \\
&= 1 - 2p \qquad (23)
\end{aligned}
$$

and if $r_j = 1$, $q^0_{mj} - q^1_{mj} = -(1 - 2p)$. This implies that (19) can be rewritten

as

$$\begin{aligned} r^0_{mj} &= \frac{1}{2}\left(1 + \prod_{n'=N(m)\backslash j} (1-2p)\,(-1)^{r_{n'}}\right) \\ &= \frac{1}{2}\left(1 + (1-2p)^{|N(m)|-1} \prod_{n'=N(m)\backslash j} (-1)^{r_{n'}}\right) \\ r^1_{mj} &= \frac{1}{2}\left(1 - (1-2p)^{|N(m)|-1} \prod_{n'=N(m)\backslash j} (-1)^{r_{n'}}\right) \end{aligned} \tag{24}$$

Note that the product on the RHS of both of these terms evaluates to either $+1$ or -1 depending on the number of 1's in the set $N(m)\backslash j$ and is very easy to compute given the value of the parity check computed for node m. Finally, r^0_{mj} and r^1_{mj} take only one of two values, making this easy to implement in hardware, with no real calculations required. Namely

$$\begin{aligned} r^0_{mj} &= \frac{1}{2}\left(1 - (1-2p)^{|N(m)|-1}\right) \text{ if the parity of bits with indices from } N(m)\backslash j \text{is odd} \\ &= \frac{1}{2}\left(1 + (1-2p)^{|N(m)|-1}\right) \text{ if the parity of bits with indices from } N(m)\backslash j \text{is even} \\ r^1_{mj} &= \frac{1}{2}\left(1 - (1-2p)^{|N(m)|-1}\right) \text{ if the parity of bits with indices from } N(m)\backslash j \text{is even} \\ &= \frac{1}{2}\left(1 + (1-2p)^{|N(m)|-1}\right) \text{ if the parity of bits with indices from } N(m)\backslash j \text{is odd} \end{aligned} \tag{25}$$

Proceeding further the APPs can be simplified as

$$\begin{aligned} p(x_j = 0 \mid S_j, r) &= p(r_j \mid x_j = 0) \prod_{m \in M(j)} \frac{1}{2}\left(1 \pm (1-2p)^{|N(m)|-1}\right) \\ p(x_j = 1 \mid S_j, r) &= p(r_j \mid x_j = 1) \prod_{m \in M(j)} \frac{1}{2}\left(1 \pm (1-2p)^{|N(m)|-1}\right) \end{aligned} \tag{26}$$

which reduces to

$$\begin{aligned} p(x_j = 0 \mid S_j, r) &= f^0_j \prod_{m \in M^{odd}(j)} \frac{1}{2}\left(1 - (1-2p)^{|N(m)|-1}\right) \prod_{m \in M^{even}(j)} \frac{1}{2}\left(1 + (1-2p)^{|N(m)|-1}\right) \\ p(x_j = 1 \mid S_j, r) &= f^1_j \prod_{m \in M^{odd}(j)} \frac{1}{2}\left(1 + (1-2p)^{|N(m)|-1}\right) \prod_{m \in M^{even}(j)} \frac{1}{2}\left(1 - (1-2p)^{|N(m)|-1}\right) \end{aligned} \tag{27}$$

Where $M^{odd}(j)$ are the set of nodes connected to x_j with odd parity, and $M^{even}(j)$ are those with even parity. Finally, the first iteration calculations,

which are extremely easy to calculate given the hard decision results of the parity checks, are:

$$\begin{aligned} p(x_j = 0 \mid S_j, r) &= f_j^0 \frac{1}{2}\left(1-(1-2p)^{|N(m)|-1}\right)^{|M^{odd}(j)|} \frac{1}{2}\left(1+(1-2p)^{|N(m)|-1}\right)^{|M^{even}(j)|} \\ p(x_j = 1 \mid S_j, r) &= f_j^1 \frac{1}{2}\left(1+(1-2p)^{|N(m)|-1}\right)^{|M^{odd}(j)|} \frac{1}{2}\left(1-(1-2p)^{|N(m)|-1}\right)^{|M^{even}(j)|} \end{aligned} \quad (28)$$

1.5 Message passing algorithm in log domain

Next we compute $LLR^{\text{posterior}}(x_j) = \log \frac{p(x_j=0|r,S_j)}{p(x_j=1|r,S_j)}$. Before we take the log of the expressions in (20) we use the following fact:

$$\prod_i a_i = \left(\prod_i sgn(a_i)\right) * \exp\left(\sum_i \log(|a_i|)\right) \quad (29)$$

and the following definition:

$$\tanh(\frac{x}{2}) = \frac{e^x - 1}{e^x + 1} \quad (30)$$

First

$$\begin{aligned} LLR^{\text{posterior}}(x_j) &= \log \frac{p(x_j = 0|r, S_j)}{p(x_j = 1|r, S_j)} \\ = \log \frac{p(r_j|x_j = 0)}{p(r_j|x_j = 1)} &+ \log \prod_{m \in M(j)} \frac{1 + \prod_{n'=N(m)\setminus j} (q^0_{mn'} - q^1_{mn'})}{1 - \prod_{n'=N(m)\setminus j} (q^0_{mn'} - q^1_{mn'})} \end{aligned} \quad (31)$$

Letting $\delta q_{mn'} = q^0_{mn'} - q^1_{mn'}$ and $LLR(q^0_{mn'}) = \log \frac{q^0_{mn'}}{q^1_{mn'}}$, simple substitution gives $\delta q_{mn'} = \tanh(LLR(\frac{q^0_{mn'}}{2}))$. With this one can write (34) as

$$\begin{aligned} \frac{2r_j}{\sigma^2} &+ \sum_{m \in M(j)} \log \frac{1 + s_{mj} e^{A_{mj}}}{1 - s_{mj} e^{A_{mj}}} \\ = \frac{2r_j}{\sigma^2} &- \sum_{m \in M(j)} \log \frac{1 - s_{mj} e^{A_{mj}}}{1 + s_{mj} e^{A_{mj}}} \\ = \frac{2r_j}{\sigma^2} &- \sum_{m \in M(j)} \log \left(-\frac{s_{mj} e^{A_{mj}} - 1}{s_{mj} e^{A_{mj}} + 1}\right) \end{aligned} \quad (32)$$

where

$$\begin{aligned} s_{mj} &= \prod_{n'=N(m)\backslash j} sgn(\delta q_{mn'}) = \prod_{n'=N(m)\backslash j} sgn\left(LLR(q^0_{mn'})\right) \\ A_{mj} &= \sum_{n'=N(m)\backslash j} \log(|\tanh(\frac{LLR(q^0_{mn'})}{2})|) \end{aligned} \tag{33}$$

Referring to (34)The argument of the log() in (35) is always positive. The equation (35) can be simplified further to

$$\begin{aligned} \frac{2r_j}{\sigma^2} &- \sum_{m\in M(j)} s_{mj} \log\left(-\tanh(\frac{A_{mj}}{2})\right) \\ \frac{2r_j}{\sigma^2} &- \sum_{m\in M(j)} s_{mj} \log\left(|\tanh(\frac{A_{mj}}{2})|\right) \end{aligned} \tag{34}$$

Letting $\Psi(x) = \log\left(|\tanh(\frac{x}{2})|\right)$ (37) becomes

$$LLR^{\text{posterior}}(x_j) = \frac{2r_j}{\sigma^2} - \sum_{m\in M(j)} s_{mj}\Psi(A_{mj}) \tag{35}$$

where

$$A_{mj} = \sum_{n'=N(m)\backslash j} \log(|\tanh(\frac{LLR(q^0_{mn'})}{2})|) = \sum_{n'=N(m)\backslash j} \Psi(LLR(q^0_{mn'})) \tag{36}$$

This leads to the log-domain decoding algorithm given in Section ZZZ. For the binary symmetric channel the first iteration calculation (41) in log domain, $LLR^{\text{posterior}}(x_j) = \log\frac{p(x_j=0|r,S_j)}{p(x_j=1|r,S_j)}$. The first iteration calculations are easy to calculate given the hard decision results of the parity checks:

$$LLR^{\text{posterior}}(x_j) = \log\left(\frac{f^0_j\left(1-(1-2p)^{|N(m)|-1}\right)^{|M^{odd}(j)|}\left(1+(1-2p)^{|N(m)|-1}\right)^{|M^{even}(j)|}}{f^1_j\left(1+(1-2p)^{|N(m)|-1}\right)^{|M^{odd}(j)|}\left(1-(1-2p)^{|N(m)|-1}\right)^{|M^{even}(j)|}}\right) \tag{37}$$

$$\begin{aligned} &= \log\left(\frac{f^0_j}{f^1_j}\right) \\ &+ |M^{odd}(j)|\log\left(1-(1-2p)^{|N(m)|-1}\right) + |M^{even}(j)|\log\left(1+(1-2p)^{|N(m)|-1}\right) \\ &- |M^{odd}(j)|\log\left(1+(1-2p)^{|N(m)|-1}\right) - |M^{even}(j)|\log\left(1-(1-2p)^{|N(m)|-1}\right) \end{aligned} \tag{38}$$

and

$$
\begin{aligned}
LLR^{\text{posterior}}(x_j) &= \log\left(\frac{f_j^0}{f_j^1}\right) \\
&+ |M^{odd}(j)|\left[\log\left(1-(1-2p)^{|N(m)|-1}\right) - \log\left(1+(1-2p)^{|N(m)|-1}\right)\right] \\
&+ |M^{even}(j)|\left[\log\left(1+(1-2p)^{|N(m)|-1}\right) - \log\left(1-(1-2p)^{|N(m)|-1}\right)\right] \qquad (39)
\end{aligned}
$$

Let

$$
C_j^1 = \log\left(1-(1-2p)^{|N(m)|-1}\right) - \log\left(1+(1-2p)^{|N(m)|-1}\right) \qquad (40)
$$

which can precomputed based on the value of p and is independent of j. Recall (9), so,

$$
LLR^{\text{posterior}}(x_j) = (1-2r_j)\log\left(\frac{1-p}{p}\right) + C_j^1\left(|M^{odd}(j)| - |M^{even}(j)|\right) \qquad (41)
$$

so the first iteration requires only a parity check calculation and evaluation of two constants that depend on p.

1.5.1 Probability domain decoding algorithm

0) **Initialization**. The variables $q_{mn}^0$are initialized to

for $j = 0, \ldots, N-1$

$q_{mj}^0 = f_i^0 = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp \frac{(r_j+1)^2}{2\sigma^2}$

$q_{mi}^1 = f_i^1 = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp \frac{(r_j-1)^2}{2\sigma^2}$

end

1) **Parity node updates**. Let $\delta q_{mj} = \delta q_{mj}^0 - q_{mj}^1$.

for $j = 0, \ldots, N-1$

for $m \in M(j)$

$$\delta r_{mj} = \prod_{i' \in N(m) \backslash j} \delta q_{mi'}$$

$r_{mj}^0 = \frac{1}{2}(1 + \delta r_{mj})$

$r_{mj}^1 = \frac{1}{2}(1 - \delta r_{mj})$

end

end

2) **Bit node updates**. The constants α and β are chosen to make $q^0 + q^1 = 1$

for $j = 0, \ldots, N-1$

$$q_{mj}^0 = \alpha_{mj} f_j^0 \prod_{m' \in M(j) \backslash m} r_{m'j}^0$$

$$q_{mj}^1 = \beta_{mj} f_j^1 \prod_{m' \in M(j) \backslash m} r_{m'j}^1$$

end

The q_j^b's are updated as:

for $j = 0, \ldots, N-1$

$$q_j^0 = \alpha_j f_j^0 \prod_{m' \in M(j)} r_{m'j}^0$$

$$q_j^1 = \beta_j f_j^1 \prod_{m' \in M(j)} r_{m'j}^1$$

end

3) **Verify parity checks**.

for $j = 0, \ldots, N-1$

If $(q_j^0 > 0.5)$ then $\hat{x}_j = 0$

else $\hat{x}_j = 1$

end

Does $H^T \hat{x} = 0$? If yes, done.

4) **If done, quit. If not, got to 1)** .

1.5.2 Log domain decoding algorithm

Let $\Psi(x) = \log(\tanh(\frac{x}{2}))$ and $A_{mj} = \sum_{n' \in N(m) \backslash j} \Psi(LLR(q^0_{mn'}))$.

0) **Initialization**. The variables q^0_{mj} are initialized to

for $j = 0, \ldots, N-1$
$LLR(q^0_{mj}) = \log(\frac{q^0_{mj}}{q^1_{mj}}) = \frac{2r_j}{\sigma^2}$
end

1) **Parity node updates**.

for $j = 0, \ldots, N-1$
$R_{mj} = s_j \Psi(-A_{mj})$
end

2) **Bit node updates**.

for $j = 0, \ldots, N-1$
$LLR(q^0_j) = \sum_{m \in M(j)} \Psi(R_{mj})$
end

The $LLR(q^0_{mj})$'s is updated as:

for $j = 0, \ldots, N-1$
$LLR(q^0_{mj}) = LLR(q^0_j) - R_{mj}$
end

3) **Verify parity checks**.

for $j = 0, \ldots, N-1$
If $LLR(q^0_j) > 0$ then $\hat{x}_j = 0$
else $\hat{x}_j = 1$
end

Does $H^T \hat{x} = 0$? If yes, done.

4) **If done, quit. If not, got to 1)** .