Iterative Plurality-Logic and Generalized Algorithm B Decoding of q-ary LDPC Codes

Kristin Jagiello and William E. Ryan
kristin@angeldojo.com and ryan@ece.arizona.edu
ECE Dept., University of Arizona, Tucson, AZ 85721

Abstract - We examine hard-decision decoding algorithms for q-ary LDPC codes. We first examine the performance of majority-logic decoding and iterative majority-logic decoding, including an improvement we call iterative plurality-logic decoding (iPLgD). We then show how iPLgD can be improved by tweaking it into what we call the generalized (Gallager) Algorithm B. We show how optimum generalized Algorithm B decision thresholds may be obtained semi-analytically for the q-ary symmetric channel. We additionally introduce a weighted Algorithm B decoder for the q-ary codes over the binary symmetric channel. Finally, we compare the performance of the various algorithms for decoding q-ary LDPC codes with that of Reed-Solomon codes.

I. INTRODUCTION

In this paper we re-introduce the iterative majority logic decoding (MLgD) algorithm of Massey [1] and show how it should be modified when applied to q-ary LDPC codes. The modification leads to what we call the iterative plurality decoding algorithm. Moreover, we point out that these algorithms, when cast as message-passing algorithms, do not pass extrinsic information. When the iterative plurality decoding algorithm is modified to incorporate extrinsic information, the result is a generalization to q-ary LDPC codes of Gallager’s so-called Algorithm B [2]. We derive equations which allow the determination of the optimum generalized Algorithm B threshold for the q-ary symmetric channel. Also, because q-ary LDPC codes are frequently considered for the binary symmetric channel, we introduce a modification of the generalized Algorithm B for this channel called the weighted Algorithm B. Finally, we present some simulation results which demonstrate the aforementioned algorithm improvements.

II. MLgD AND ITERATIVE MLgD

In this section, we first review majority-logic decoding for non-binary LDPC codes over GF(q), following [3]. (See also the pioneering work on this topic [1].) We will then present an iterative majority-logic decoding (iMLgD) algorithm for nonbinary LDPC codes, followed by an improved iMLgD algorithm. Our discussion will focus on (γ, ρ)-regular nonbinary LDPC codes, where γ is the column weight and ρ is the row weight of a code’s m × n parity-check matrix H. Extension of our discussion to irregular LDPC codes is trivial.

We will denote the rows of H by h₀, h₁, ..., hₘ₋₁ and the m components of the syndrome s = zHᵀ by sᵢ = z · hᵢ, for i = 0, 1, ..., m − 1. In this expression, z = v + e is the received word, where v is the length-n transmitted codeword and e is the q-ary error word added by the channel.

We assume that no two columns and no two rows in H have more than one location in common containing nonzero elements. That is, we assume no 4-cycles exist in the code’s Tanner graph. As a result of this row-column constraint, the γ check equations that involve code symbol v_j will involve any other code symbol at most once. We say that these check equations are orthogonal on v_j. The γ check equations orthogonal on v_j have the form sᵢ = hᵢ,jzⱼ + ∑ᵢ̸=j hᵢ,lz_l, where i takes the γ values in the set {0, 1, ..., m − 1} for which hᵢ,j ≠ 0. It is convenient for the decoding of v_j to introduce normalized syndrome components (or normalized syndromes, for brevity), given [3] by

\[ \hat{s}_i = h^{-1}_{i,j}s_i = z_j + h^{-1}_{i,j}\sum_{l\neq j}h_{i,l}z_l \] (1)

\[ e_j + h^{-1}_{i,j}\sum_{l\neq j}h_{i,l}e_l. \] (2)

The set of γ such normalized syndromes that check on v_j will be denoted by \( \hat{S}_j \).

Now assume there are \( |\gamma/2| \) or fewer errors in the received word z = v + e and consider the decoding of symbol v_j. If one of the errors is in position j, then \( \gamma - |\gamma/2| + 1 \) or more (i.e., a majority) of the elements of \( \hat{S}_j \) will equal \( e_j \). If none of the errors are in position j, then at most \( |\gamma/2| \) of the elements of \( \hat{S}_j \) will be nonzero, that is \( \gamma - |\gamma/2| \) (at least half) of the elements of \( \hat{S}_j \) will equal zero.

A \( |\gamma/2| \)-error-correcting decoding algorithm follows from these conclusions: For \( j = 0, 1, ..., n - 1 \),

1) if a strict majority of the syndrome values \( \hat{S}_j = \{\hat{s}_i\} \) that check on v_j are equal to a common nonzero value, \( \alpha \), then \( \hat{e}_j = \alpha \) and the decoded symbol is \( \hat{v}_j = z_j - \alpha \); else

2) if at least half of the syndrome values that check on v_j are equal to zero, then \( \hat{e}_j = 0 \) and the decoded symbol is \( \hat{v}_j = z_j \); else

3) declare a decoding failure.

This algorithm is called the one-step majority-logic decoding algorithm or, simply, the majority-logic decoding (MLgD) algorithm. As mentioned, this algorithm is capable only of correcting error patterns with at most \( |\gamma/2| \) errors. Over
fifty years ago, when this algorithm was invented and when codes in use were quite short, this capability was reasonable, particularly in view of the low complexity of the algorithm. However, by today’s standards, this error-correction capability is unacceptable. Consider, for example, that a length-10,000, rate-1/2, (4, 8)-regular LDPC code would only be capable of correcting double-error patterns with the MLgD algorithm.

Motivated by this point and the algorithm’s low complexity, we consider the iterative MLgD (iMLgD) algorithm [1], [4]. The iMLgD algorithm is quite effective when applied to both binary and non-binary LDPC codes. To our knowledge, this paper contains the first application of the iMLgD algorithm to LDPC codes. Moreover, we later show how the iMLgD algorithm can be improved in two ways with no essential change in decoding complexity.

The iterative MLgD algorithm iteratively employs a modified version of the MLgD algorithm. In the modified MLgD algorithm as in the original MLgD algorithm, the first conditional statement sets \( \hat{e}_j = \alpha \) and the second statement sets \( \hat{e}_j = 0 \). As for the third statement in the algorithm, since the estimate \( \hat{e}_j \) of \( e_j \) would be undecided at this point, but hopefully resolved in a future iteration, it would be set as \( \hat{e}_j = 0 \) in the modified algorithm; a decoding failure is not declared. Moreover, this modified third statement can be combined with the second statement and made unconditional. The resulting modified MLgD algorithm to be used in the iMLgD decoder is: For \( j = 0, 1, \ldots, n - 1 \),

1) if a strict majority of the syndrome values \( \{\hat{s}_i\} \) that check \( v_j \) are equal to a common nonzero value \( \alpha \), then set \( \hat{e}_j = \alpha \); else
2) set \( \hat{e}_j = 0 \).

If the number of errors does not exceed \( \left\lfloor \gamma/2 \right\rfloor \), the MLgD is guaranteed to correct all of the errors in one step. If the number of errors exceeds \( \left\lfloor \gamma/2 \right\rfloor \), a single execution of the MLgD may correct all of the errors, although it often results in uncorrected error values. When the number of errors exceeds \( \left\lfloor \gamma/2 \right\rfloor \), the error values that are estimated are usually correct. Of course, as the Hamming weight of \( e \) increases, the probability of the estimated error values being correct decreases.

Because the code is linear and the estimated error values are usually correct, the logical step is to subtract the impact of the estimated error values on the syndrome \( s \), and then, in a second iteration, try to correctly estimate the residual errors using the updated syndrome. After the second iteration, the syndrome is again modified using the most recently found errors, and further error estimates are made. Iterations continue until the syndrome vector becomes the zero vector or a maximum number of iterations has occurred. Figure 1 is a block diagram of the iterative MLgD decoder. It is formally described in Algorithm 1.

**Example 1:** Using a “random” construction algorithm, we designed a (1000, 500) 256-ary LDPC code satisfying the row-column constraint with column weight \( \gamma = 8 \) (and average row weight 16). The MLgD is guaranteed to correct only \( \gamma/2 = 4 \) 256-ary symbol errors. In this example, we examine how many more the MLgD can correct and, more importantly, how many more the iMLgD can correct. To do this, we studied the decoding behavior of these algorithms by simulating 10,000 randomly generated error patterns of selected weights \( w \), where \( w \in \{1, 2, \ldots, 5\} \) for the MLgD and \( w \in \{5, 10, 15, 20, 30, 35, 40\} \) for the iMLgD. For the iMLgD, the maximum number of iterations was set to \( L = 20 \). We use as our performance metric, \( P_{uc}(w) \), the measured probability of uncorrectable error, that is, the fraction of the weight-\( w \) error patterns tested that were not fully corrected. For the MLgD decoder, we found that \( P_{uc}(w) = 0 \) for \( w < 5 \) and \( P_{uc}(5) \cong 7 \times 10^{-2} \). Beyond \( w = 5 \), \( P_{uc}(w) \) quickly converged to unity. For the iMLgD decoder we found that \( P_{uc}(w) = 0 \) for \( w < 40 \) and \( P_{uc}(40) \cong 5 \times 10^{-4} \). Beyond \( w = 40 \), \( P_{uc}(w) \) gradually converged to unity. □

### III. Iterative Plurality-Logic Decoding

The MLgD is also known as a threshold decoder [1] because an error estimate \( \hat{e}_j \) is only set to a nonzero value if the number of equal normalized syndromes exceeds the threshold \( \gamma/2 \). However, the MLgD algorithm was originally designed for binary codes and with the intent that it would only be executed once. That is, the goal for the MLgD algorithm was to attain the guaranteed error-correction capability of \( \left\lfloor \gamma/2 \right\rfloor \) error in one decoding step. However, for the iterative MLgD algorithm, which has the MLgD processor at its core, a threshold of \( \gamma/2 \) is too large for nonbinary codes (although we were able to demonstrate a vast improvement over the non-iterative MLgD decoder in the above example).

For example, consider again the 256-ary (1000, 500) LDPC code discussed in the example above. With \( \gamma = 8 \), the
threshold that must be exceeded before an error estimate \( \hat{e}_j \) is set to a nonzero value is five. However, consider the situation in which there are six errors affecting the syndromes orthogonal on position \( j \). Assume that one of the six errors is in position \( j \). Then, the five other errors can be dispersed among at most five of the eight elements in \( \tilde{S}_j \). Thus, at least three elements of \( \tilde{S}_j \) will be affected only by the error in position \( j \), and these normalized syndromes will all equal \( e_j \) (see (2)). Moreover, since we are discussing a 256-ary code, it is unlikely that more than three of the five elements not affected by \( e_j \) will be identical (and not equal to \( e_j \)). Thus, a reasonable reduced-threshold MLgD processor assigns \( \hat{e}_j = \alpha \) if a (strict, no ties) plurality of the syndrome elements in \( \tilde{S}_j \) are equal to \( \alpha \); otherwise, the processor assigns \( \hat{e}_j = 0 \). If half of the normalized syndromes are equal to one element and the other half are equal to some other element, \( e_j \) is arbitrarily set to one of the two elements. In view of the plurality condition, the reduced-threshold decoder can be called an iterative plurality-logic decoder, iPLgD.

By reducing the threshold from a majority to a plurality in the MLgD core decoding step, the iterative MLgD algorithm is able to proceed past error patterns that would have otherwise stumped it. A variation of the iPLgD is an iMLgD with an adaptive threshold. That is, rather than reduce the threshold at the outset, the threshold is decremented after each iteration in which the decoder fails to make progress on the correction of errors. Each of these algorithms fit within the framework of Figure 1 and Algorithm 1. In the iPLgD case, the MLgD module at the core of the iterative decoder simply decides in favor of the normalized syndromes holding a plurality. In the iMLgD case with a decrementing threshold, the MLgD processor is initialized with some threshold \( t = t_0 \), where \( t_0 \leq \lfloor \gamma/2 \rfloor + 1 \), and \( t \) is decremented when no apparent progress is made in the correction of errors, that is, when the number of nonzero syndrome values does not decrease.

Example 2: We repeated the experiments of Example 1 using the iPLgD decoder with \( t = 3 \). That is, the decoding decision in favor of a plurality, provided the number of common symbols was at least \( t = 3 \). Randomly generated error patterns of weights \( w \in \{5, 10, 15, 20, 30, 35, 40, 50, 60, 70, 80, 90, 95\} \) were examined. We found that \( P_{\text{uc}}(w) = 0 \) for \( w < 95 \) and \( P_{\text{uc}}(95) \approx 2 \times 10^{-4} \). Beyond \( w = 95 \), \( P_{\text{uc}}(w) \) slowly converged to unity. 

IV. CONNECTIONS TO GALLAGER’S ALGORITHM B

The presentations of the iMLgD and iPLgD algorithms in the previous sections were presented in a classical fashion, along the lines of [1], [4]. They can, however, be presented in a more modern fashion in the language of message-passing decoding. For example, for the iMLgD, each updated syndrome computation in Fig. 1 can be represented as a check node (CN) computation. Moreover, each MLgD computation in Fig. 1 can be represented as a variable node (VN) computation.

Specifically, during iteration \( \ell \), VN \( v_j \) takes as inputs the syndrome components \( \tilde{S}_j(\ell) = \{\tilde{s}_i(\ell)\} \) computed from its neighboring CNs. A single output message \( \hat{v}_j(\ell) = \text{MLgD}(z_j, \tilde{S}_j(\ell)) \) is computed and sent to all of its neighbors, where

\[
\text{MLgD} \left( z_j, \tilde{S}_j(\ell) \right) = \begin{cases} 
  z_j - \alpha, & \text{majority of } \tilde{S}_j(\ell) \text{ equal } \alpha \\
  z_j, & \text{else}
\end{cases}
\]

(3)

The values \( \hat{v}_j(\ell) \) are initialized to the received values, \( \hat{v}_j(0) = z_j \), and the syndrome values \( \tilde{s}_i(\ell) \) are initialized to zero, \( \tilde{s}_i(0) = 0 \). Turning now to the CN updates, CN \( c_i \) takes as inputs the values \( V_i(\ell) = \{\hat{v}_j(\ell)\} \) and computes its corresponding syndrome values in a manner similar to (1): \( \tilde{s}_i(\ell) = \hat{v}_j(\ell) + h_{ij}^{-1} \sum_{l \neq j} h_{ij} \hat{v}_l(\ell) \). All outputs from CN \( c_i \) convey this same message, \( \tilde{s}_i(\ell) \).

The iPLgD decoder can likewise be described as a message-passing algorithm. The only changes would be that in equation (3) “MLgD” would be changed to “PLgD” and “majority” would be changed to “plurality”. The iPLgD decoder may be improved somewhat if rather than a simple plurality rule, a threshold is utilized, as described in the previous section.

However, there is one crucial component of modern message-passing decoding that is missing in these iterative algorithms, namely, extrinsic information. That is, in both the VN and CN update equations described above, all node inputs are used to compute the message, and there is one message for all node outputs. By contrast, in modern algorithms the outgoing message on an edge omits the incoming message on that edge in its computation, thereby passing along only extrinsic information.

If we modify the iPLgD decoder with a threshold \( t \) so that it computes extrinsic information, the result is essentially a generalization of Gallager’s so-called Algorithm B. This algorithm, which we denote by gen-Alg-B, is described below. Note that it assumes a Tanner graph in which the edges are “weighted” by the nonzero \( q \)-ary elements \( h_{ij} \) of the code’s parity-check matrix. Its performance advantages will be detailed in Section VII. A technique for computing the optimum threshold \( t^* \) for gen-Alg-B is described in the next section.

Algorithm 2 Generalized Algorithm B

0. Initialize: VN-to-CN message \( M_{ji} \) from VN \( v_j \) to CN \( c_i \) is set as \( M_{ji} = h_{ij} z_j \).

1. CN-to-VN update: CN-to-VN message \( N_{ij} \) from CN \( c_i \) to VN \( v_j \) is \( N_{ij} = h_{ij}^{-1} \sum_{j \neq j} M_{ji} \).

2. VN-to-CN update: VN-to-CN message \( M_{ji} \) from VN \( v_j \) to CN \( c_i \) is set as follows: If \( t \) or more incoming messages \( \{N_{ij}, j^{'i} \neq i\} \) equal some common symbol \( \alpha \), \( M_{ji} = h_{ij} \alpha \); else, \( M_{ji} = h_{ij} z_j \). (If the counts of more than one finite field symbol exceed \( t \), choose in favor of the received symbol \( z_j \) for ties, or of the maximum count otherwise.)

3. Decisions: Compute the code symbol decision \( \hat{v}_j(\ell) \) by considering all incoming messages, \( \{z_j, N_{ij}, \text{ all } i^{'j}\} \), to VN \( v_j \). If \( t \) or more values are equal some common value \( \alpha \), \( \hat{v}_j(\ell) = \alpha \); else, \( \hat{v}_j(\ell) = z_j \). (If the counts of more than one finite field symbol exceed \( t \), choose in favor of the received symbol \( z_j \) for ties, or of the maximum count otherwise.)

4. Stopping criterion: If a codeword is found or if reached the maximum number of iterations, stop; else, go to Step 1.
the error rate of the messages from each variable node at the zero codeword was transmitted. Thus, the channel model of the decoding algorithm [5], we may (and will) assume that regularity of the code’s decoding graph, and the “symmetry” binary symmetric channel used log 

Example 3: We repeated the experiments of Examples 1 and 2 using the gen-Alg-B decoder with \( t = 3 \). Randomly generated error patterns of weights \( w \in \{5, 10, 15, 20, 30, 35, 40, 50, 60, 70, 80, 90, 95\} \) were examined. We found that \( P_{\text{dc}}(w) = 0 \) for \( w < 95 \) and \( P_{\text{dc}}(95) \approx 1 \times 10^{-4} \). Beyond \( w = 95 \), \( P_{\text{dc}}(w) \) slowly converged to unity. So its error correction performance is very similar to that of the iPLgD decoder when 20 iterations are used. In Section VII we will see that gen-Alg-B converges to the correct codeword more quickly.

V. ON THE OPTIMUM ALGORITHM B THRESHOLD

Ardakani and Kschischang [5] showed that the Alg-B decoder was the optimum message-passing decoder with binary messages on the binary symmetric channel. It is possible that the gen-Alg-B decoder is optimum as well on the \( q \)-ary symmetric channel, but we leave that problem as a topic for future research. In this section, we present a semi-analytic technique for determining the optimum gen-Alg-B threshold on the \( q \)-ary symmetric channel as a function of \( q \), the channel error rate, and the variable and check node degrees \( (d_v, d_c) \) of a regular \( q \)-ary LDPC code. Note that the binary symmetric channel used \( \log_2(q) \) times for each \( q \)-ary symbol is not a \( q \)-ary symmetric channel.

We follow Gallager [2] who found the optimum threshold for the binary case (binary code, binary channel). We assume a very long \((d_v, d_c)\)-regular \( q \)-ary LDPC code on the \( q \)-ary symmetric channel. Thus, the error rate is assumed to be constant across the messages passed from the variable nodes. We denote this common message error rate at the \( i \)th iteration by \( p_i \). Because of the channel symmetry, the regularity of the code’s decoding graph, and the “symmetry” of the decoding algorithm [5], we may (and will) assume that the zero codeword was transmitted. Thus, the channel model depicted in Fig. 2 suffices for our discussion. Observe that the error rate of the messages from each variable node at the first decoding iteration is precisely the channel error rate \( p_0 \) presented in the figure.

Consider now the variable node message error rate \( p_{i+1} \) after iteration \( i + 1 \). This probability may be written as

\[
p_{i+1} = p_0 \Pr (E_1 \mid E_0) + (1-p_0) \Pr (E_2 \mid \overline{E}_0) \tag{4}
\]

where we define the events above as

- \( E_1 \) = the VN processor decides the channel symbol is correct and uses it as the message to the neighboring CN
- \( E_0 \) = channel symbol is erroneous
- \( E_2 \) = the VN processor decides the channel symbol is wrong and passes a different value to the neighboring CN
- \( \overline{E}_0 \) = channel symbol is correct

As we will see, the probabilities on the right-hand side of (4) are functions of \( p_i \). An outgoing message from a VN with an incorrect (i.e., nonzero) channel symbol will also be nonzero (hence, incorrect) if the decoding algorithm does not decide against the channel symbol. In accordance with the gen-Alg-B, this occurs with probability

\[
\Pr (E_1 \mid E_0) = \Pr \left( \sum_{j=0}^{t-1} \binom{d_v-1}{j} (1-P)^{d_c-1-j} \right) \tag{5}
\]

where

\[
P = \Pr (\text{CN-to-VN message equals zero}) = \sum_{\ell=0}^{d_c-1} \Pr (F(\ell, d_c-1)) = \left(1-p_i\right)^{d_c-1} + \frac{1}{q-1} \sum_{\ell=2}^{d_c-1} \binom{d_c-1}{\ell} p_i^\ell \left(1-p_i\right)^{d_c-1-\ell} \tag{6}
\]

where \( F(\ell, d_c-1) \) is the event that there are \( \ell \) errors (nonzero values) among the \( d_c-1 \) messages into the CN and they sum to zero. Note that \( \Pr (F(0, d_c-1)) \) is the first term in (6), which is the probability that all \( d_c-1 \) messages equal zero (and, hence, sum to zero). Clearly, \( \Pr (F(1, d_c-1)) = 0 \) since the single nonzero value cannot by itself sum to zero; hence, the absent \( s = 1 \) term in (6). The \( \ell \)th term in the sum is the probability that there are \( s \) nonzero values among the \( d_c-1 \) incoming messages and the factor \( \frac{1}{q-1} \) is the probability that the \( s \) nonzero \( q \)-ary values will sum to zero. (With probability \( \frac{1}{q^2} \), the last of the \( s \) nonzero values will be the one for which the sum is zero.)

As for the second term in (4), we may write \((n.z. = \text{nonzero})\)

\[
\Pr (E_2 \mid \overline{E}_0) = \Pr \left( \sum_{m=t}^{d_c-1} \binom{d_c-1}{m} \left(1-P\right)^m P^{d_c-1-m} \right) \tag{7}
\]

where \( P \) is obviously defined. Observe that this probability is identical to the probability that at least \( t \) of \( m \) people have the same birthday, where there are \( q-1 \) equiprobable
birthday choices. This birthday problem was solved in [6], although there is an error in that reference.

It is easiest to find \( P_{\geq t|m} \) via its complementary probability \( P_{<t|m} \). Thus, we are interested in the probability, \( P_{<t|m} \), that no \( t \) of the \( m \) nonzero messages are equal. Define the set of \( \{t-1\} \)-limited partitions of \( m \) as

\[
M = \left\{ m_1, m_2, \ldots, m_{t-1} : \sum_{j=1}^{t-1} jm_j = m \right\}
\]

where \( m_1 \) is the number of non-repeated message values, \( m_2 \) is the number of pairs of equal message values, ..., and \( m_{t-1} \) is the number of \( \{t-1\} \)-tuples of equal message values. It can then be shown that a particular pattern in \( M \) occurs with probability

\[
P(m_1, m_2, \ldots, m_{t-1}) = \frac{q-1}{(q-1)^m} \cdot \frac{m}{1!2!2!} \cdot \frac{1}{(t-1)!},
\]

where the first two terms are multinomial coefficients, and the notation \( a^{(m_a)} \) means \( a \) with a multiplicity of \( m_a \). Thus, for example, \( \left( \frac{5}{1, 2, 2, 2} \right) = \frac{5!}{1!2!2!2!} \). Also, we define \( m_0 = (q - 1) - \sum_{j=1}^{t-1} m_j \). Then, \( P_{<t|m} \) is the sum over the set \( M \) of the probabilities given in (8). Alternatively, since we are interested in \( P_{\geq t|m} \), we may write

\[
P_{\geq t|m} = 1 - \sum M P(m_1, m_2, \ldots, m_{t-1}).
\]

We now derive (8) in a manner that differs from [6]. The third factor follows because there are \( (q - 1)^m \) patterns of \( m \) nonzero \((q - 1)\)-ary messages, each pattern occurring equiprobably. As for the first factor, we imagine \( q-1 \) available slots (birthdays) in which there may occur \( m \) non-repeats, \( m_2 \) equal pairs, and so on, but also \( m_0 \) unoccupied slots. This first factor then gives the number of such possibilities. The second factor follows because, while there are \( m! \) possible orderings of the \( m \) messages (people), we must divide out the number of ways the \( m_2 \) pairs may be permuted (namely, \( 2!^{m_2} \)), the number of ways the \( m_3 \) triples may be permuted (\( 3!^{m_3} \)), and so on. This is so because we are simply interested in groups of equal value of size less than \( t \), irrespective of order.

In summary, the message error probability \( p_t+1 \) after the \( i \)th iteration is given by (4), where the right-hand side of (4) is given by equations (5) through (9), which depend on \( p_s \). Thus, with an initial message error rate of \( p_0 \) (the channel error rate) and given values of \( q, t, d_{w_0}, d_{w_{d}}, \) and \( d_{w_d} \), one may determine whether or not \( p_s \) converges to zero (that is, whether or not the decoder will correctly decode with probability near unity).

The table below presents several such computations in which the \( p_0 \) column contains the largest channel error rate for which \( p_s \) converges to zero. The values given for \( q = 2 \) and \( t = 2 \) (binary Algorithm A) are in agreement with those of Gallager up to the third decimal place. Also, observe that for the 256-ary examples examined, \( t = 3 \) is the optimum threshold. We have seen as well in our simulations that \( t = 3 \) is optimum (albeit, with 256-ary symbols over the binary symmetric channel).

<table>
<thead>
<tr>
<th>( q )</th>
<th>( d_{w_0} )</th>
<th>( d_{w_{d}} )</th>
<th>( d_{w_d} )</th>
<th>( t )</th>
<th>( p_0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>0.0618</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>6</td>
<td>2</td>
<td>0.0397</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
<td>2</td>
<td>0.0077</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
<td>3</td>
<td>0.0476</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>8</td>
<td>16</td>
<td>3</td>
<td>0.1169</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>8</td>
<td>16</td>
<td>4</td>
<td>0.0836</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>8</td>
<td>16</td>
<td>5</td>
<td>0.0571</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>16</td>
<td>16</td>
<td>3</td>
<td>0.1708</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>16</td>
<td>16</td>
<td>4</td>
<td>0.1451</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>16</td>
<td>16</td>
<td>5</td>
<td>0.1223</td>
<td></td>
</tr>
</tbody>
</table>

VI. WEIGHTED ALGORITHM B DECODING FOR THE BSC

The generalized Algorithm B that we presented and the computation of its optimum threshold both assume a \( q \)-ary symmetric channel. However, the transmission of \( q \)-ary LDPC codes over the binary symmetric channel (BSC) is of both practical and theoretical interest. In this section, we consider a modification of the gen-Alg-B decoder for the BSC which we call the weighted Algorithm B decoder and which we denote by wtd-Alg-B.

The motivation for this decoder is the fact that the equivalent channel model for the transmission of \( \log_2(q) \) bits over a BSC is manifestly not \((q \text{-} \text{ary})\) symmetric. Consider for example the transmission of the all-zeroes symbol \( \mathbf{0} \) over the BSC. Clearly, the probability of erroneously receiving a symbol that differs from \( \mathbf{0} \) in a single bit location is vastly larger than that of receiving a symbol that differs in two or more locations.

This fact can be exploited in a modified decoding algorithm. That is, rather than simply counting the number of like symbols at a variable node and comparing it to a threshold, the counting can be weighted to favor the more likely symbols. The more likely symbols are the ones that differ from the received symbol in zero locations or in one location. All other symbols are highly unlikely.

As an example, consider the computation of the VN-to-CN message \( M_{ji} \) from VN \( v_j \) to CN \( c_i \). Suppose that, among the incoming messages \( \{N_{v_j' i}, i' \neq i\} \) to VN \( v_j \), three of them equal \( \alpha \) and four of them equal \( \beta \). Gen-Alg-B would then set \( M_{ji} = \beta \) (assuming \( t = 3 \)). By contrast, wtd-Alg-B would first compute \( 3w_d(\alpha, z_j) \) and \( 4w_d(\beta, z_j) \), where \( d(\alpha, z_j) \) is the Hamming distance between the binary representations of \( \alpha \) and the received symbol \( z_j \), and \( w_d \) is a weighting factor corresponding to the Hamming distance \( d \). The message \( M_{ji} \) sent by the wtd-Alg-B would then be whichever of \( 3w_d(\alpha, z_j) \) and \( 4w_d(\beta, z_j) \) is greater than or equal to the threshold \( t \). If both of these numbers is greater than \( t \), then \( M_{ji} = \beta \) is set to the maximum of them. The wtd-Alg-B is described below.

We have found in our simulations that the algorithm is relatively insensitive to the choice of weights \( w_d \). For example, for our 256-ary code, we found that the performance for \( w_0 = 2.1, w_1 = 2.0, \) and \( w_d = 1.0 \) for \( d = 2, 3, \ldots, 8 \) had performance very similar to \( w_0 = 2.2, w_1 = 2.0, w_2 = 1.1, \) and \( w_d = 0.9 \) for \( d = 3, 4, \ldots, 8 \). Finding the optimal set of
weights \( \{w_d\} \) is a topic of future work. Lastly, note that when \( w_d = 1 \) for all \( d \), wtd-Alg-B reduces to gen-Alg-B.

VII. BSC SIMULATION RESULTS

We have simulated the various decoding algorithms presented in this paper using \((1000,500)\) 256-ary LDPC code introduced in Example 1. The binary symmetric channel was utilized in all cases so that each 256-ary symbol was represented by eight bits. The performance results are presented in Fig. 3 which plots the decoder output \( q \)-ary symbol rate \( P_s \) against the BSC bit-error rate \( \epsilon \). For the iterative algorithms, a maximum of 3 and 20 iterations were simulated.

We observe in the figure at least two orders of magnitude of improvement over the \( iMLgD-20 \) curve when 20 iterations are allowed. When moving rightward in the figure from the \( MLgD \) curve to all of the other decoders, the \( AlgB-20 \) decoder performs the best, and the wtd-AlgB-20 decoder is vastly superior to all of the other decoders.

For some context, we have also included in the figure the results for a randomly constructed \((256,128)\) Reed-Solomon code with column weight equal to 8 and no 4-cycles. The numbers “3” and “20” attached to the algorithm names give the maximum number of iterations.

![Figure 3](image-url) Simulations results on the binary-symmetric channel for a randomly constructed 256-ary \((1000,500)\) LDPC code with column weight equal to 8 and no 4-cycles. The numbers “3” and “20” attached to the algorithm names give the maximum number of iterations.

Algorithm 3 Weighted Algorithm B

0. Initialize: VN-to-CN message \( M_{ji} \) from VN \( v_j \) to CN \( c_i \) is set as \( M_{ji} = h_{ij}z_j \).
1. **CN-to-VN update:** CN-to-VN message \( N_{ij} \) from CN \( c_i \) to VN \( v_j \) is \( N_{ij} = h_{ij}^{-1} \sum_{j' \neq j} M_{ji} \).
2. **VN-to-CN update:** VN-to-CN message \( M_{ji} \) from VN \( v_j \) to CN \( c_i \) is set as follows: Let \( \Gamma \) be the subset of \( q \)-ary finite field elements that can be found among the incoming messages \( \{N_{i'j}, i' \neq i\} \). Count the number of occurrences of each element \( \gamma \in \Gamma \) and denote this number by \( n_{\gamma} \). Next, compute the set of products \( n_{\gamma}w_d(\gamma,z_j) \); else, \( M_{ji} \) is set as the scaled received value, \( h_{ij}z_j \).
3. **Decisions:** Compute the code symbol decision \( \hat{z}_j \) by considering all incoming messages, \( \{z_j, N_{ij}, \text{ all } i\} \), to VN \( v_j \). As in previous step, compute the set of products \( n_{\gamma}w_d(\gamma,z_j) \), but for the larger set \( \{z_j, \text{ all } i\} \), and compare these to \( t \). If the threshold is exceeded, \( \hat{z}_j \) is set to the received value, \( z_j \). (If the metrics of more than one finite field symbol exceed \( t \), choose in favor of the received symbol \( z_j \) for ties, or of the maximum metric otherwise.)
4. **Stopping criterion:** If a codeword is found or if reached the maximum number of iterations, stop; else, go to Step 1.
which uses 8-bit symbols and Berlekamp-Massey decoding. For the range of $P_s$ plotted, the Reed-Solomon code easily outperforms the 256-ary LDPC code with the wtd-AlgB-20 decoder. However, the latter code appears to have a steeper curve and it might be possible that the curves intersect down near the $P_s$ region of interest for data storage applications.

VIII. CONCLUSION

We have presented a number of hard-decision decoding algorithms for $q$-ary LDPC codes that are simultaneously related to Massey’s iterative threshold decoding algorithm and Gallager’s Algorithm B. The weighted Algorithm B decoder performs the best on binary symmetric channel, by a wide swath, but a $q$-ary LDPC code with this decoder appears to be inferior to a Reed-Solomon code of the same rate, at least for the range of decoder symbol error rates considered.

This paper leaves open several interesting problems, including the possible optimality of gen-Alg-B on the $q$-ary symmetric channel and the optimal choice of weights for the wtd-Alg-B. Moreover, with the vast jump in improvement in going from the gen-Alg-B to the wtd-Alg-B on the BSC (see Fig. 3), one wonders if there is an algorithm that does better still. Lastly, of interest is a complexity comparison between the wtd-Alg-B and the Berlekamp-Massey algorithm.

REFERENCES