List-Message Passing Achieves Capacity on the q-ary Symmetric Channel for Large q

Fan Zhang and Henry D. Pfister
Department of Electrical and Computer Engineering, Texas A&M University
{fanzhang,hpfister}@tamu.edu

Abstract

We discuss and analyze a list-message-passing decoder with verification for low-density parity-check (LDPC) codes on the q-ary symmetric channel (q-SC). Rather than passing messages consisting of symbol probabilities, this decoder passes lists of possible symbols and marks some messages as verified. The density evolution (DE) equations for this decoder are derived and used to compute decoding thresholds. If the maximum list size is unbounded, then we find that any capacity-achieving LDPC code for the binary erasure channel can be used to achieve capacity on the q-SC for large q. The decoding thresholds are also computed via DE for the case where each list is truncated to satisfy a maximum list size constraint. Simulation results are also presented to confirm the DE results. During the simulations, we observed differences between two verification-based decoding algorithms, introduced by Luby and Mitzenmacher, that were implicitly assumed to be identical. In this paper, we provide an analysis of the node-based algorithms from that paper and verify that it matches simulation results.

The probability of false verification (FV) is also considered and techniques are discussed to mitigate the FV. Optimization of the degree distribution is also used to improve the threshold for a fixed maximum list size. Finally, the proposed algorithm is compared with a variety of other algorithms using both density evolution thresholds and simulation results.

I. Introduction

Low-density parity-check (LDPC) codes are linear codes that were introduced by Gallager in 1962 [2] and re-discovered by MacKay in 1999 [3]. The ensemble of LDPC codes that we consider (e.g. see [4] and [5]) is defined by the edge degree distribution (d.d.) functions $\lambda(x) = \sum_{k\geq 2} \lambda_k x^{k-1}$ and $\rho(x) = \sum_{k\geq 2} \rho_k x^{k-1}$. The standard encoding and decoding algorithms are based on the bit-level operations. However, when applied to the transmission of data packets, it is natural to perform the encoding and decoding algorithm at the packet level rather than the bit level. For example, if we are going to transmit 32 bits as a packet, then we can use error-correcting codes over the, rather large, alphabet with 2^{32} elements.

The q-SC can be described as

$$p(y|x) = \begin{cases} 1-p & \text{if } x = y \\ p/(q-1) & \text{if } x \neq y \end{cases}$$

where x (resp. y) is the transmitted (resp. received) symbol and $x, y \in GF(q)$. The capacity of the q-SC is $1+(1-p)\log_q(1-p) + p\log_q p - p\log_q(q-1)$ which is approximately equal to $1-p$ for large q. This implies the capacity of the q-SC with large q is approximately equal to the capacity of the BEC with erasure probability p. Moreover, the behavior of the q-SC with large q is similar to the BEC in the sense that: i) incorrectly received symbols from the q-SC provides almost no information about the transmitted symbol and ii) error detection (e.g., a CRC) can be added to each symbol with negligible overhead [6].

Binary LDPC codes for the q-SC with moderate q are optimized based on EXIT charts in [7] and [8]. It is known that the complexity of the standard belief-propagation algorithm for q-ary LDPC codes scales like $q\log^2 q$. Even for moderate sizes of q, such as $q = 256$, this renders such algorithms ineffective in practice. However, when q is large, an interesting effect can be used to facilitate decoding: if a symbol is received in error, then it is essentially a randomly chosen element of the alphabet, and the parity-check equations involving this symbol is very unlikely to be valid.

In [1], Luby and Mitzenmacher developed an elegant algorithm for decoding LDPC codes on the q-SC for large q. However, their paper did not present simulation results and left capacity achieving ensembles as an interesting open problem. Metzner presented similar ideas earlier in [9] and [10], but the focus and analysis is quite different. Davey and MacKay develop and analyze a symbol-level message-passing decoder over small finite fields [11]. A number of approaches to the q-SC (for large q) based on interleaved Reed-Solomon codes are also possible [6] [12]. In [13], Shokrollahi and Wang discuss two ways of approaching capacity. The first uses a two-stage approach where the first stage uses a Tornado code and verification decoding. The second is, in fact, equivalent to one of the decoders we discuss in this paper.[1] When we discovered this, the authors were kind enough to send us an extended abstract [14] which contains more details. Still, the authors did not consider the theoretical performance with a maximum list size constraint, the actual performance of the decoder via simulation, or false verification (FV) due to cycles in the decoding graph. In this paper, we describe the algorithm in detail and consider those details.

This research was supported in part by the National Science Foundation under Grant No. 0747470.

[1]The description of the second method [13] is very brief and we believe its capacity-achieving nature deserves further attention.

Inspired by [1], we develop in this paper list-message-passing (LMP) decoding with verification for LDPC codes on the q-SC. Instead of passing a single value between bit and check nodes, we pass a list of candidates to improve the decoding threshold. This modification also increases the probability of FV. So, we analyze the causes of FV and discuss techniques to mitigate FV. It is worth noting that the LMP decoder we consider is somewhat different than the list extension suggested in [1]. Their approach uses a peeling-style decoder based on verification rather than erasures. In [1], the algorithms are proposed in a node-based (NB) style but analyzed using message-based (MB) decoders. It is implicitly assumed that the two approaches are equivalent. In fact, this is not always true. In this paper, we consider the differences between NB and MB decoders and derive an asymptotic analysis for NB decoders.

The paper is organized as follows. In Section II, we describe the LMP algorithm for bounded and unbounded list size and use density evolution (DE) [15] to analyze its performance. The difference between NB and MB decoders for the first (LM1) and second algorithm (LM2) in [1] is discussed and the NB decoder analysis is derived in Section III and V, respectively. The problem of FV is considered in Section V. In Section VI, we use differential evolution to optimize code ensembles. We describe the simulation of these codes and compare the results with the theoretical thresholds. We also compare our results with previously published results in this area [1] and [13]. In Section VII, simulation results are shown. Applications of the LMP algorithm are discussed and conclusions are given in Section VIII.

II. Description and Analysis

A. Description of the Decoding Algorithm

The LMP decoder we discuss is designed mainly for the q-SC and is based on local decoding operations applied to lists of possible codeword symbols. The messages passed in the graph have three types: *verified* (V), *unverified* (U) and *erasure* (E). Every V-message has a symbol value associated with it. Every U-message has a list of symbol values associated with it. Following [1], we will mark messages as verified when they are very likely to be correct. In particular, we will find that the probability of FV approaches zero as q goes to infinity.

The LMP decoder works by passing list-messages around the decoding graph. Instead of passing a single code symbol (e.g., Gallager A/B algorithm [2]) or a probability distribution over all possible code symbols (e.g., [11]), we pass a list of values that are more likely to be correct than the other messages. At a variable node, the output list contains all symbols which could satisfy the check constraint for the given input lists. At the check node, the output message will be verified if and only if all the incoming messages are verified. At a node of degree d, the associativity and commutativity of the node-processing operation allow it to be decomposed into $(d-1)$ binary[2] operations (e.g., $a{+}b{+}c{+}d{=}(a{+}b){+}(c{+}d)$). In such a scheme, the computational complexity of each binary-operation is proportional to s^2 at the check node and $s \ln s$ at the variable node[3], where s is the list size of the input list. The list size grows rapidly as the number of iterations increases. In order to make the algorithm practical, we have to truncate the list to keep the list size within some maximum value, denoted S_{max}. In the algorithm analysis, we also find that, after the number of iterations exceeds half the girth of the decoding graph, the probability of FV increases very rapidly. We analyze the reasons of FV and classify the FV's into two types. We find that the codes described in [1] and [13] both suffer from type-II FV. In Section V, we analyze these FV's and propose a scheme to reduce the probability of FV.

The message-passing decoding algorithm using list messages (or LMP) applies the following simple rules to calculate the output messages for a check node:

- If all the input messages are verified, then the output becomes verified with the value which makes all the incoming messages sum to zero.
- If any input message is an erasure, then the output message becomes an erasure.
- If there is no erasure on the input lists, then the output list contains all symbols which could satisfy the check constraint for the given input lists.
- If the output list size is larger than S_{max}, then the output message is an erasure.

It applies the following rules to calculate the output messages of a variable node:

- If all the input messages are erasures or there are multiple verified messages which disagree, then output message is the channel received value.
- If any of the input messages is verified (and there is no disagreement) or a symbol appears more than once, then the output message becomes verified with the same value as the verified input message or the symbol which appears more than once.
- If there is no verified message on the input lists and no symbol appears more than once, then the output list is the union of all input lists.
- If the output message has list size larger than S_{max}, then the output message is the received value from the channel.

[2]Here we use "binary" to emphasize that there are two inputs although the operation is over $GF(q)$.

[3]The binary-operation at the variable node can be done by s binary searches of length s and the complexity of a binary search of length s is $O(\ln s)$

B. DE for Unbounded List Size Decoding Algorithm

To apply DE to the LMP decoder with unbounded list sizes, denoted LMP-∞ (i.e., $S_{max} = \infty$), we consider three quantities which evolve with the iteration number i. Let x_i be the probability that the correct message symbol is not on the list passed from a variable node to a check node. Let y_i be the probability that the message passed from a variable node to a check node is not verified. Let z_i be the average list size passed from a variable node to a check node. The same variables are "marked" $(\tilde{x}_i, \tilde{y}_i, \tilde{z}_i)$ to represent the same values for messages passed from the check nodes to the variable nodes (i.e., the half-iteration value). We also assume all the messages are independent, that is, we assume the code length is infinite and there are no cycles in the bipartite graph.

First, we consider the probability, x_i, that the correct message symbol is not on the list. For any degree-d check node, the correct message symbol will only be on the edge output list if all of the other $d-1$ input lists contain their corresponding correct symbols. This implies that $\tilde{x}_i = 1 - \rho(1 - x_i)$. For any degree-$d$ variable node, the correct message symbol will not be on the edge output list only if it is on none of the other $d-1$ edge input lists. This implies that $x_{i+1} = p\lambda(\tilde{x}_i)$. This behavior is very similar to erasure decoding of LDPC codes on the BEC and gives the identical update equation

$$x_{i+1} = p\lambda\left(1 - \rho(1 - x_i)\right) \tag{1}$$

where p is the q-SC error probability. Next, we consider the probability, y_i, that the message is not verified. For any degree-d check node, an edge output message is verified only if all of the other $d-1$ edge input messages are verified. For any degree-d variable node, an edge output message is verified if any symbol on the other $d-1$ edge input lists is verified or occurs twice which implies $\tilde{y}_i = 1 - \rho(1 - y_i)$. The event that the output message is not verified can be broken into the union of two disjoint events: (i) the correct symbol is not on any of the input lists, and (ii) the symbol from the channel is incorrect and the correct symbol is on exactly one of the input lists and not verified. For a degree-d variable node, this implies that

$$\Pr(\text{not verified}) = (\tilde{x}_i)^{d-1} + p(d-1)\left(\tilde{y}_i - \tilde{x}_i\right)(\tilde{x}_i)^{d-2}. \tag{2}$$

Summing over the d.d. gives the update equation

$$y_{i+1} = \lambda\left(1 - \rho(1 - x_i)\right) + p\left(\rho(1 - x_i) - \rho(1 - y_i)\right)\lambda'\left(1 - \rho(1 - x_i)\right). \tag{3}$$

It is important to note that (1) and (3) were published first in [13, Thm. 2] (by mapping $x_i = p_i$ and $y_i = p_i + q_i$), but were derived independently by us.

Finally, we consider the average list size z_i. For any degree-d check node, the output list size is equal[4] to the product of the sizes of the other $d-1$ input lists. Since the mean of the product of i.i.d. random variables is equal to the product of the means, this implies that $\tilde{z}_i = \rho(z_i)$. For any degree-d variable node, the output list size is equal to one[5] plus the sum of the sizes of the other $d-1$ input lists if the output is not verified and one otherwise. Again, the mean of the sum of $d-1$ i.i.d. random variables is simply $d-1$ times the mean of the distribution, so the average output list size is given by

$$1 + \left((\tilde{x}_i)^{d-1} + p(d-1)\left(\tilde{y}_i - \tilde{x}_i\right)(\tilde{x}_i)^{d-2}\right)(d-1)\tilde{z}_i.$$

This gives the update equation

$$z_{i+1} = 1 + \left[\tilde{x}_i\lambda'\left(\tilde{x}_i\right) + p\left(\tilde{y}_i - \tilde{x}_i\right)\left(\lambda'\left(\tilde{x}_i\right) + \tilde{x}_i\lambda''\left(\tilde{x}_i\right)\right)\right]\rho(z_i).$$

For the LMP decoding algorithm, the threshold of an ensemble $(\lambda(x), \rho(x))$ is defined to be

$$p^* \triangleq \sup\left\{p \in (0,1] \middle| p\lambda(1 - \rho(1 - x)) < x \;\forall\; x \in (0,1]\right\}.$$

Next, we show that some codes can achieve channel capacity using this decoding algorithm.

Theorem 2.1: Let p^* be the threshold of the d.d. pair $(\lambda(x), \rho(x))$ and assume that the channel error rate p is less than p^*. In this case, the probability y_i that a message is not verified in the i-th decoding iteration satisfies $\lim_{i\to\infty} y_i \to 0$. Moreover, for any $\epsilon > 0$, there exists a $q < \infty$ such that LMP decoding of a long random (λ, ρ) LDPC code, on a q-SC with error probability p, results in a symbol error rate of less than ϵ.

Proof: See Appendix A. ∎

Remark 2.2: Note that the convergence condition, $p^*\lambda(1 - \rho(1 - x)) < x$ for $x \in (0, 1]$, is identical to the BEC case but that x has a different meaning. In the DE equation for the q-SC, x is the probability that correct value is not on the list. In the DE equation for the BEC, x is the probability that the message is an erasure. This tells us any capacity-achieving ensemble for the BEC is capacity-achieving for the q-SC with LMP-∞ algorithm and large q. This also gives some intuition about the behavior of the q-SC for large q. For example, when q is large, incorrectly received value behaves like an erasure [6].

[4]It is actually upper bounded because we ignore the possibility of collisions between incorrect entries, but the probability of this occurring is negligible as q goes to infinity.

[5]A single symbol is always received from the channel.

Corollary 2.3: The code with d.d. pair $\lambda(x) = x$ and $\rho(x) = (1-\epsilon)x + \epsilon x^2$ has a threshold of $1 - \frac{\epsilon}{1+\epsilon}$ and a rate of $r > \frac{\epsilon}{3(1+\epsilon)}$. Therefore, it achieves a rate of $\Theta(\delta)$ for a channel error rate of $p = 1 - \delta$.

Proof: Follows from $\left(1 - \frac{\epsilon}{1+\epsilon}\right) \lambda\left(1 - \rho(1-x)\right) < x$ for $x \in (0, 1]$ and Theorem 2.1. ■

Remark 2.4: We believe that Corollary 2.3 provides the first linear-time decodable construction of rate $\Theta(\delta)$ for a random-error model with error probability $1-\delta$. A discussion of linear-time encodable/decodable codes, for both random and adversarial errors, can be found in [16]. The complexity also depends on the required list size which may be quite large (though independent of block length). Unfortunately, we do not have explicit bounds on the required alphabet size or list size for this construction.

In practice, we cannot implement a list decoder with unbounded list size. Therefore, we also evaluate the LMP decoder under a bounded list size assumption.

C. DE for the Decoding Algorithm with Bounded List Size

First, we list some definitions and notation for the DE analysis with bounded list size decoding algorithm. Note that in the bounded list size list decoding algorithm, each list may contain at most S_{max} messages. For convenience, we classify the messages into four types:

(V) *Verified*: message is verified and has list size 1.
(E) *Erasure*: message is an erasure and has list size 0.
(L) *Correct on list*: message is not verified or erased and the correct message is on the list.
(N) *Correct not on list*: message is not verified or erased, and the correct message is not on the list.

For the first two message types, we only need to track the fraction, V_i and E_i, of message types in the i-th iteration. For the third and the fourth types of messages, we also need to track the list sizes. Therefore, we track the characteristic function of the list size for these messages, given by $L_i(x)$ and $N_i(x)$. The coefficient of x^j represents the probability that the message has list size j. Specifically, $L_i(x)$ is defined by

$$L_i(x) = \sum_{j=1}^{S_{max}} l_{i,j} x^j,$$

where $l_{i,j}$ is the probability that, in the i-th decoding iteration, the correct message is on the list and the message list has size j. The function $N_i(x)$ is defined similarly. This implies that $L_i(1)$ is the probability that the list contains the correct message and that it is not verified. For the same reason, $N_i(1)$ gives the probability that the list does not contain the correct message and that it is not verified. The same variables are "marked" ($\tilde{V}, \tilde{E}, \tilde{L}, \tilde{N}$ and $\tilde{P}$) to represent the same values for messages passed from the check nodes to the variable nodes (i.e., the half-iteration value). For compactness, we denote the overall density as $P_i = [V_i, E_i, L_i(x), N_i(x)]$.

Using these definitions, we find that DE can be computed efficiently by using arithmetic of polynomials. For the convenience of analysis and implementation, we use a sequence of binary-operations plus a separate truncation operator to represent a multiple-input multiple-output operation. We use $\boxplus$ to denote the check-node operator and $\otimes$ to denote the variable-node operator. Using this, the DE for the variable-node binary-operation $P^{(3)} = \tilde{P}^{(1)} \otimes \tilde{P}^{(2)}$ is given by

$$V^{(3)} = \tilde{V}^{(1)} + \tilde{V}^{(2)} - \tilde{V}^{(1)}\tilde{V}^{(2)} + \tilde{L}^{(1)}(1)\tilde{L}^{(2)}(1) \tag{4}$$

$$E^{(3)} = \tilde{E}^{(1)}\tilde{E}^{(2)} \tag{5}$$

$$L^{(3)}(x) = \tilde{L}^{(1)}(x)\left(\tilde{E}^{(2)} + \tilde{N}^{(2)}(x)\right) + \tilde{L}^{(2)}(x)\left(\tilde{E}^{(1)} + \tilde{N}^{(1)}(x)\right) \tag{6}$$

$$N^{(3)}(x) = \tilde{N}^{(1)}(x)\tilde{E}^{(2)} + \tilde{N}^{(2)}(x)\tilde{E}^{(1)} + \tilde{N}^{(1)}(x)\tilde{N}^{(2)}(x). \tag{7}$$

Note that Eq. (4) to Eq. (7) do not yet consider the list size truncation and the channel value. For the binary check-node operation $\tilde{P}^{(3)} = P^{(1)} \boxplus P^{(2)}$, the DE is given by

$$\tilde{V}^{(3)} = V^{(1)}V^{(2)} \tag{8}$$

$$\tilde{E}^{(3)} = E^{(1)} + E^{(2)} - E^{(1)}E^{(2)} \tag{9}$$

$$\tilde{L}^{(3)}(z) = \left[V^{(1)}L^{(2)}(z) + V^{(2)}L^{(1)}(z) + L^{(1)}(x)L^{(2)}(y)\right]_{x^j y^k \to z^{jk}} \tag{10}$$

$$\tilde{N}^{(3)}(z) = \left[N^{(1)}(x)N^{(2)}(y) + N^{(1)}(x)\left(V^{(2)}y + L^{(2)}(y)\right) + N^{(2)}(x)\left(V^{(1)}y + L^{(1)}(y)\right)\right]_{x^j y^k \to z^{jk}} \tag{11}$$

where the subscript $x^j y^k \to z^{jk}$ means the replacement of variables. Finally, the truncation of lists to size S_{max} is handled by truncation operators which map densities to densities. We use $\mathcal{T}$ and $\mathcal{T}'$ to denote the truncation operation at the check and variable nodes. Specifically, we truncate terms with degree higher than S_{max} in the polynomials $L(x)$ and $N(x)$. At check nodes, the truncated probability mass is moved to E.

TABLE I
BRIEF DESCRIPTION OF MESSAGE-PASSING ALGORITHMS FOR q-SC

Alg.	Description
LMP-S	LMP as described in Section II-A with $S_{max}=S$
LM1-MB	MP decoder that passes (value, U/V). [1, III.B] At VN's, output is V if any input is V or message matches channel value, otherwise pass channel value. At CN's, output is V if all inputs are V.
LM1-NB	Peeling decoder with VN state (value, U/V). [1, III.B] At CN's, if all neighbors sum to 0, then all neighbors get V. At CN's, if all neighbors but one are V, then last gets V.
LM2-MB	The same as LM1-MB with one additional rule. [1, IV.A]. At VN's, if two input messages match, then output V.
LM2-NB	The same as LM1-NB with one additional rule. [1, IV.A]. At VN's, if two neighbor values same, then VN gets V.
SW1	Identical to LM2-MB
SW2	Identical to LMP-∞. [13, Thm. 2]

At variable nodes, lists longer than S_{max} are replaced by the channel value. To analyze this, we separate $L(x)$ into two terms: $\tilde{A}(x)$ with degree less than S_{max} and $x^{S_{max}}\tilde{B}(x)$ with degree at least S_{max}. Likewise, we separate $\tilde{N}(x)$ into $\tilde{C}(x)$ and $x^{S_{max}}\tilde{D}(x)$. The inclusion of the channel symbol and the truncation are combined into a single operation

$$P^{(1)}=\mathcal{T}'\left(\left[\tilde{V},\tilde{E},\tilde{A}(x)+x^{S_{max}}\tilde{B}(x),\tilde{C}(x)+x^{S_{max}}\tilde{D}(x)\right]\right)$$

defined by

$$V^{(1)}=\tilde{V}+(1-p)\left(\tilde{A}(1)+\tilde{B}(1)\right) \tag{12}$$

$$E^{(1)}=0 \tag{13}$$

$$L^{(1)}(x)=(1-p)x\left(\tilde{E}+\tilde{C}(x)+\tilde{D}(1)\right)+px\tilde{A}(x) \tag{14}$$

$$N^{(1)}(x)=px\left(\tilde{E}+\tilde{B}(1)+\tilde{C}(x)+\tilde{D}(1)\right). \tag{15}$$

Note that in Eq. (12), the term $(1-p)\left(\tilde{A}(1)+\tilde{B}(1)\right)$ is due to the fact that messages are compared for possible verification before truncation.

The overall DE recursion is easily written in terms of the forward (bit to check) density P_i and the backward (check to bit) density $\tilde{P}_i$. The initial density is $P_0=[0,0,(1-p)x,px]$, where p is the error probability of the q-SC channel, and the recursion is given by

$$\tilde{P}_i=\sum_{k=2}^{d_c}\rho_k\,\mathcal{T}\left(P_i^{\boxplus k-1}\right) \tag{16}$$

$$P_{i+1}=\sum_{k=2}^{d_v}\lambda_k\,\mathcal{T}'\left(\tilde{P}_i^{\otimes k-1}\right). \tag{17}$$

Note that the DE recursion is not one-dimensional. This makes it difficult to optimize the ensemble analytically. It remains an open problem to find the closed-form expression of the threshold in terms of the maximum list size, d.d. pairs and q. In section VI, we will fix the maximum variable and check degrees, code rate, q and maximum list size and optimize the threshold over the d.d. pairs by using a numerical approach.

III. DIFFERENTIAL EQUATION ANALYSIS OF LM1-NB

A. Motivation

We refer to the first and second algorithm in [1] as LM1 and LM2, respectively. Each algorithm can be viewed either as message-based (MB) or node-based (NB). The first and second algorithm in [13] and [14] are referred to as SW1 and SW2. These algorithms are summarized in Table I. Note that if there is no verification occurring, the VN sends the ("channel value", U) and the CN sends the ("expected correct value",U). The algorithms SW1, SW2 and LMP are all MB algorithms, but can be modified to be NB algorithms.

In [1], the algorithms are proposed in the node-based (NB) style [1, Section III-A and IV] but analyzed in the message-based (MB) style [1, Section III-B and IV]. It is easy to see that the LM1-NB and LM1-MB are identical, but the NB and MB algorithms for LM2 are different. In this section, we will show the differences between the NB decoder and MB decoder and derive the correct analysis for LM1-NB.

First, we show the equivalence between LM1-MB and LM1-NB.

Theorem 3.1: Any verification that occurs in LM1-NB also occurs in LM1-MB and vice versa. Therefore, LM1-NB and LM1-MB are equivalent.

Proof: See Appendix B. ■

Remark 3.2: The theorem shows the equivalence between LM1-NB and LM1-MB. This also implies the error patterns or stopping sets of LM1-NB and LM1-MB are the same.

In the NB decoder, the verification status is associated with the node. Once a node is verified, all the outgoing messages are verified. In the MB decoder, the status is associated with the edge/message and the outgoing messages may have different verification status. NB algorithms cannot, in general, be analyzed using DE because the independence assumption between messages does not hold. Therefore, we develop peeling-style decoders which are equivalent to LM1-NB and LM2-NB and use differential equations to analyze them.

Following [4], we analyze the peeling-style decoder using differential equations to track the average number of edges (grouped into types) in the graph as decoding progresses. From the results from [17] and [4], we know that the actual number of edges (of any type), in any particular decoding realization is tightly concentrated around the average over the lifetime of the random process. In a peeling-style decoder for $GF(q)$, a variable node and its edges are removed after verification. The check node keeps track of the new parity constraint (i.e., the value to which the attached variables must sum) by subtracting values associated with the removed edges.

B. Analysis of Peeling-Style Decoding

First, we introduce some notation and definitions for the analysis. A variable node (VN) whose channel value is correctly received is called a correct variable node (CVN), otherwise it is called an incorrect variable node (IVN). A check node (CN) with i edges connected to the CVN's and j edges connected to the IVN's will be said to have C-degree i and I-degree j, or type (i, j).

We also define some quantities as follows:

- t: decoding time or the fraction of VNs removed from graph
- $L_i(t)$: the number of edges connected to CVN's with degree i at time t.
- $R_j(t)$: the number of edges connected to IVN's with degree j at time t.
- $N_{i,j}(t)$: the number of edges connected to CN's with C-degree i and I-degree j.
- $E_l(t)$: the remaining number of edges connected to CVN's at time t.
- $E_r(t)$: the remaining number of edges connected to IVN's at time t.
- $a(t)$: the average degree of CVN's,
$$a(t) = \sum_{i \geq 0} L_i(t) i / E_l(t)$$
- $b(t)$: the average degree of IVN's,
$$b(t) = \sum_{i \geq 0} R_i(t) i / E_r(t)$$
- E: number of edges in the original graph,
$$E = E_l(0) + E_r(0)$$

Counting edges in three ways gives the following equations:

$$\sum_{i \geq 0} L_i(t) + \sum_{i \geq 0} R_i(t) = E_l(t) + E_r(t) = \sum_{i \geq 0} \sum_{j \geq 0} N_{i,j}(t).$$

These r.v.'s represent a particular realization of the decoder. The differential equations are defined for the normalized (i.e., divided by E) expected values of these variables. We use lower-case notation (e.g., $l_i(t)$, $r_i(t)$, $n_{i,j}(t)$, etc.) for these deterministic trajectories. For a finite system, the decoder removes exactly one variable node in one time step of Δt.

The description of peeling-style decoder is as follows. The peeling-style decoder removes one CVN or IVN in each time step by the following rules:

CER: If any CN has its edges all connected to CVN's, pick one of the CVN's and remove it and all its edges.

IER1: If any IVN has at least one edge connected to the CN's of type $(0, 1)$, the value of the IVN is given by the attached CN and we remove the IVN and all its outgoing edges.

If both CER and IER1 can be applied, then one is chosen randomly as described below.

Since both rules remove exactly one VN, the decoding process either finishes in exactly N steps or stops early and cannot continue. The first case occurs only when either the IER1 or CER condition is satisfied in every time step. When the decoder stops early, the pattern of CVNs and IVNs remaining is known as a stopping set. We also note that the rules above, though described differently, are equivalent to the first node-based algorithm (LM1-NB) introduced in [1].

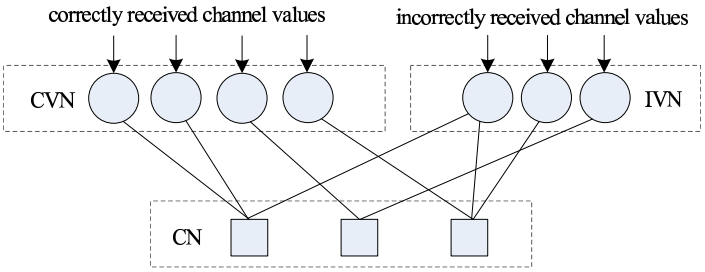

Fig. 1. Tanner graph for differential equation analysis.

C. Analysis

Recall that in the node-based algorithm for LM1 we have two verification rules. The first rule is that if all messages but one are verified at a CN, then all messages are verified. We call this type-1 incorrect-edge-removal (IER1) and this needs $n_{0,1}(t) > 0$ to be satisfied. The second rule is: if all messages sum to zero at a CN, all messages are verified. We call this as correct-edge-removal (CER) in the peeling-style decoder and this requires $n_{i,0} > 0$ for some $i \geq 1$. The peeling-style decoder performs one operation in time step. The operation is random and can be either CER or IER1. When both operations are possible, we choose smoothly between these two rules by picking CER with probability $c_1(t)$ and IER1 with probability $c_2(t)$, where

$$c_1(t) = \frac{\sum_{i\geq 0} n_{i,0}(t)}{\sum_{i\geq 0} n_{i,0}(t) + n_{0,1}(t)}$$
$$c_2(t) = \frac{n_{0,1}(t)}{\sum_{i\geq 0} n_{i,0}(t) + n_{0,1}(t)}.$$

Therefore, the differential equations can be written as

$$\frac{\mathrm{d}l_i(t)}{\mathrm{d}t} = c_1(t)\frac{\mathrm{d}l_i^{(1)}(t)}{\mathrm{d}t} + c_2(t)\frac{\mathrm{d}l_i^{(2)}(t)}{\mathrm{d}t}$$
$$\frac{\mathrm{d}r_i(t)}{\mathrm{d}t} = c_1(t)\frac{\mathrm{d}r_i^{(1)}(t)}{\mathrm{d}t} + c_2(t)\frac{\mathrm{d}r_i^{(2)}(t)}{\mathrm{d}t}$$
$$\frac{\mathrm{d}n_{i,j}(t)}{\mathrm{d}t} = c_1(t)\frac{\mathrm{d}n_{i,j}^{(1)}(t)}{\mathrm{d}t} + c_2(t)\frac{\mathrm{d}n_{i,j}^{(2)}(t)}{\mathrm{d}t},$$

where $^{(1)}$ and $^{(2)}$ denote, respectively, the effects of CER and IER1.

1) CER Analysis: If the CER operation is picked, then we choose randomly an edge attached to a CN of type $(i, 0)$ with $i \geq 1$. This VN endpoint of this edge is distributed uniformly across the CVN edge sockets. Therefore, it will be attached to a CVN of degree k with probability $\frac{l_k(t)}{e_l(t)}$. Therefore, one has the following differential equations for l_k and r_k

$$\frac{\mathrm{d}l_k^{(1)}(t)}{\mathrm{d}t} = \frac{l_k(t)}{e_l(t)}(-k), \text{ for } k \geq 1$$

and

$$\frac{\mathrm{d}r_k^{(1)}(t)}{\mathrm{d}t} = 0.$$

For the effect on check edges, we can think of removing a CVN with degree k as first randomly picking an edge of type $(k, 0)$ connected to that CVN and then removing all the other $k-1$ edges (called reflected edges) attached to the same CVN. The $k-1$ reflected edges are uniformly distributed over the $E_l(t)$ correct sockets of the CN's. Averaging over all graphs, the $k-1$ reflected edges hit $\frac{n_{i,j}(t)i(k-1)}{(i+j)e_l(t)}$ CN's of type (i, j). Averaging over the degree k shows that the reflected edges hit $\frac{n_{i,j}(t)i(a(t)-1)}{(i+j)e_l(t)}$ CN's of type (i, j).

If a CN of type (i, j) is hit by a reflected edge, we lose $i + j$ edges of type (i, j) and gain $i - 1 + j$ edges of type $(i - 1, j)$. Hence, one has the following differential equations for $j > 0$ and $i + j \leq d_c$

$$\frac{\mathrm{d}n_{i,j}^{(1)}(t)}{\mathrm{d}t} = \left(p_{i+1,j}^{(1)}(t) - p_{i,j}^{(1)}(t)\right)(i + j)$$

where

$$p_{i,j}^{(1)}(t) = \frac{n_{i,j}(t)i(a(t) - 1)}{(i + j)e_l(t)}.$$

One should keep in mind that $n_{i,j}(t) = 0$ for $i + j > d_c$.

For $n_{i,j}^{(1)}(t)$ with $j=0$, the effect from above must be combined with effect of the type-$(i,0)$ initial edge that was chosen. So the differential equation becomes

$$\frac{\mathrm{d}n_{i,0}^{(1)}(t)}{\mathrm{d}t} = \left(p_{i+1,0}^{(1)}(t) - p_{i,0}^{(1)}(t)\right) i + \left(q_{i+1}^{(1)}(t) - q_{i}^{(1)}(t)\right) i$$

where

$$q_i^{(1)}(t) = \frac{n_{i,0}(t)}{\sum_{m\geq 0} n_{m,0}(t)}.$$

2) IER1 Analysis: If the IER1 operation is picked, then we choose a random CN of type $(0,1)$ and follow its only edge to set of IVNs. This edge is attached uniformly to this set, so the differential equations for IER1 can be written as

$$\frac{\mathrm{d}l_k^{(2)}(t)}{\mathrm{d}t} = 0,$$

$$\frac{\mathrm{d}r_k^{(2)}(t)}{\mathrm{d}t} = \frac{r_k(t)}{e_r(t)}(-k), \text{ for } k \geq 1$$

and

$$\frac{\mathrm{d}n_{i,j}^{(2)}(t)}{\mathrm{d}t} = \left(p_{i,j+1}^{(2)}(t) - p_{i,j}^{(2)}(t)\right)(i+j), \text{ for } (i,j) \neq (0,1)$$

where

$$p_{i,j}^{(2)}(t) = \frac{n_{i,j}(t) j (b(t)-1)}{(i+j)e_r(t)}.$$

For $n_{i,j}(t)$ with $(i,j)=(0,1)$, the differential equation must also account for the initial edge and becomes

$$\frac{\mathrm{d}n_{0,1}^{(2)}(t)}{\mathrm{d}t} = \left(p_{0,2}^{(2)}(t) - p_{0,1}^{(2)}(t)\right) - 1.$$

Notice that even for (3,6) codes, there are 30 differential equations[6] to solve. So we solve the differential equations numerically and the threshold for (3,6) code with LM1 is $p^* = 0.169$. This coincides with the result from density evolution analysis for LM1-MB in [1] and hints at the equivalence between LM1-NB and LM1-MB. In the proof of Theorem 3.1 we make this equivalence precise by showing that the stopping sets of LM1-NB and LM1-MB are the same.

IV. Differential Equation Analysis of LM2-NB

We will first show the LM2-NB is equivalent to the following peeling-style decoder and then use differential equation to analysis the LM2-NB algorithm. The peeling-style decoder removes one CVN or IVN in each time unit by the following rules:

CER: If any CN has its edges all connected to CVN's, pick one of the CVN's and remove it.

IER1: If any IVN has messages from CN's with type $(0,1)$, the IVN and all its outgoing edges can be removed and we track the correct value by subtracting the value from the check node.

IER2: If any IVN is attached to more than one CN with I-degree 1, then it will be verified and all its outgoing edges can be removed.

If more than one of above are satisfied, then one is chosen randomly as described below.

Notice that if either CER, IER1, or IER2 is satisfied in each time unit, decoding finishes in at most N time units, where N is the total number of variable nodes. For the same reason with LM1 case, the peeling-style decoder above is equivalent to the second node-based algorithm (LM2-NB) introduced in [1] (Section IV).

The difference between LM2-NB and LM1-NB is that LM2-NB includes another verification rule: if two messages match at the same VN, the VN is verified and all messages connected to it are verified. In the peeling-style decoder, this can be interpreted as follows. If the VN is a CVN, then the case is covered in CER. If it is a IVN, this means that the IVN has more than 1 correct incoming message and IER2 applies. Since IVN only receives a correct message from type $(i,1)$ edges, this requires $n_{i,1} > 0$ for some $i > 0$. Since the unexposed edges in the graph are uniformly distributed over their respective VNs, the condition $n_{i,1} > 0$ for some i is sufficient to guarantee that a IVN satisfies IER2 with high probability.

[6]There are 28 for $n_{i,j}$ ($i,j \in [0,\cdots,6]$ such that $i+j \leq 6$), 1 for $r_k(t)$, and 1 for $l_k(t)$.

To smooth the choice between IER1, CER and IER2, we choose CER with probability $c_1(t)$, IER1 with probability $c_2(t)$ and IER2 with probability $c_3(t)$ where

$$c_1(t) = \frac{\sum_{i\geq 0} n_{i,0}(t)}{\sum_{i\geq 0} n_{i,0}(t) + \sum_{i\geq 0} n_{i,1}(t)}$$
$$c_2(t) = \frac{n_{0,1}(t)}{\sum_{i\geq 0} n_{i,0}(t) + \sum_{i\geq 0} n_{i,1}(t)}$$
$$c_3(t) = \frac{\sum_{i\geq 0} n_{i,1}(t)}{\sum_{i\geq 0} n_{i,0}(t) + \sum_{i\geq 0} n_{i,1}(t)}.$$

We use superscript $^{(3)}$ to denote the differential equations implied by IER2. The IER2 operation depends heavily on edges that connect IVNs to CNs of type $(i,1)$ with $i \geq 1$. For convenience, we refer to such edges as "IER2 edges".

The main complication in the analysis of the IER2 operation is that the IER2 rule can only apply to IVNs which have at least two IER2 edges. Therefore, choosing such a VN uniformly skews the variable degree distribution. So, we let $\eta(t) = \frac{1}{i+1}\sum_{i\geq 1} n_{i,1}(t)$ be the fraction of IVN edges that are IER2 edges and find that $\frac{\eta(t)}{e_r(t)}$ is the probability that a randomly chosen IVN edge is an IER2 edge. For a randomly chosen IVN at time t, let the r.v. K be its degree and the r.v. L be the number of IER2 edges attached to it. Given that $L \geq 2$, the joint distribution of K, L is given by

$$W_{k,l}(t) \triangleq \Pr\left(K = k, L = l | L \geq 2\right) = \frac{\frac{r_k(t)}{k}\binom{k}{l}\left(\frac{\eta(t)}{e_r(t)}\right)^l\left(1-\frac{\eta(t)}{e_r(t)}\right)^{k-l}}{\sum_{l=2}^{k}\frac{r_k(t)}{k}\binom{k}{l}\left(\frac{\eta(t)}{e_r(t)}\right)^l\left(1-\frac{\eta(t)}{e_r(t)}\right)^{k-l}}.$$

Therefore, the differential equations for l_k and r_k are

$$\frac{\mathrm{d}l_k^{(3)}(t)}{\mathrm{d}t} = 0$$

and

$$\frac{\mathrm{d}r_k^{(3)}(t)}{\mathrm{d}t} = \sum_{l=2}^{k}(-k)\, W_{k,l}(t), \text{ for } k \geq 1.$$

The $K - L$ edges, which are not IER2 edges, are called reflected edges and the average number of reflected edges that hit a check node of degree (i,j) is given by

$$p_{i,j}^{(3)}(t) = \frac{n_{i,j}(t)j}{(i+j)(e_r(t)-\eta(t))}\sum_{k\geq 2}\sum_{l=2}^{k}(k-l)W_{k,l}(t).$$

For $n_{i,j}^{(3)}(t)$ with $j \geq 1$ or $(i,j) = (0,1)$, the differential equation is therefore

$$\frac{\mathrm{d}n_{i,j}^{(3)}(t)}{\mathrm{d}t} = \left(p_{i,j+1}^{(3)}(t) - p_{i,j}^{(3)}(t)\right)(i+j),$$

We must also account for each of the L IER2 edges that, when removed, convert an $(i,1)$ CN into an $(i,0)$ CN for some $i \geq 1$. For $n_{i,1}^{(3)}(t)$ with $i > 0$, this gives

$$\frac{\mathrm{d}n_{i,1}^{(3)}(t)}{\mathrm{d}t} = -\frac{n_{i,1}(t)}{\eta(t)}\sum_{k\geq 2}\sum_{l=2}^{k} l\, W_{k,l}(t).$$

Likewise, for $n_{i,0}^{(3)}$ with $i > 0$, this gives

$$\frac{\mathrm{d}n_{i,0}^{(3)}(t)}{\mathrm{d}t} = \frac{i}{i+1}\frac{n_{i,1}(t)}{\eta(t)}\sum_{k\geq 2}\sum_{l=2}^{k} l\, W_{k,l}(t).$$

Averaging over the CER, IER1 and IER2 operations gives the differential equations

$$\frac{\mathrm{d}l_i(t)}{\mathrm{d}t} = c_1(t)\frac{\mathrm{d}l_i^{(1)}(t)}{\mathrm{d}t}$$
$$\frac{\mathrm{d}r_i(t)}{\mathrm{d}t} = c_2(t)\frac{\mathrm{d}r_i^{(2)}(t)}{\mathrm{d}t} + c_3(t)\frac{\mathrm{d}r_i^{(3)}(t)}{\mathrm{d}t}$$
$$\frac{\mathrm{d}n_{i,j}(t)}{\mathrm{d}t} = c_1(t)\frac{\mathrm{d}n_{i,j}^{(1)}(t)}{\mathrm{d}t} + c_2(t)\frac{\mathrm{d}n_{i,j}^{(2)}(t)}{\mathrm{d}t} + c_3(t)\frac{\mathrm{d}n_{i,j}^{(3)}(t)}{\mathrm{d}t}.$$

Solving the differential equations numerically, for the (3,6) code, gives the LM2-NB threshold $p^* = 0.259$. This is somewhat larger than the threshold $p^* = 0.21$ given by the message-based analysis in [1].

V. Error Floor Analysis of LMP Algorithms

A. The Union Bound for ML Decoding

During the simulation of the optimized ensembles of Table II, we observed that there is an error floor that can be explained. First, we derive the union bound on the probability of error with ML decoding for the q-SC. To match our simulations with the union bounds, we expurgate (i.e., ignore) all codeword weights that have an expected multiplicity less than 1.

First, we summarize a few results from [18, p. 497] that characterize the low-weight codewords of LDPC codes with degree-2 variable nodes. When the block length n is large, all of these low-weight codewords are caused, with high probability, by short cycles of degree-2 nodes. For binary codes, the number of codewords with weight k is a random variable which converges to a Poisson distribution with mean $\frac{\left(\lambda_2 \rho^{'}(1)\right)^k}{2k}$. When the channel quality is high (i.e., high SNR, low error/erasure rate), the probability of ML decoding error is mainly caused by low-weight codewords.

For non-binary $GF(q)$ codes, a codeword is supported on a cycle of degree-2 nodes only if the product of the edge weights is 1. This occurs with probability $1/(q-1)$ if we choose the i.i.d. uniform random edge weights for the code. Hence, the number of $GF(q)$ codewords of weight k is a random variable, denoted B_k, which converges to a Poisson distribution with mean $b_k = \frac{\left(\lambda_2 \rho^{'}(1)\right)^k}{2k(q-1)}$. After expurgating weights that have an expected multiplicity less than 1, $k_1 = \arg\min_{k \geq 1} b_k^{(n)} \geq 1$ becomes the minimum codeword weight.

The pairwise error probability (PEP) of the q-SC with error probability p is given by the following lemma.

Lemma 5.1: Let y be the received symbol sequence assuming the all-zero codeword was transmitted. Let u be any codeword with exactly k non-zero symbols. Then, the probability that the ML decoder chooses u over the all-zero codeword is upper bounded by

$$p_{2,k} \leq \left(p\frac{q-2}{q-1} + \sqrt{\frac{4p(1-p)}{q-1}} \right)^k.$$

Proof: See Appendix C. ∎

Remark 5.2: Notice that b_k is exponential in k and the PEP is also exponential in k. The union bound for the frame error rate, due to low-weight codewords, can be written as

$$P_B \leq \sum_{k=k_1}^{\infty} b_k p_{2,k}.$$

It is easy to see $k_1 = O(\log q)$ and the sum is dominated by the first term $b_{k_1} p_{2,k_1}$ which has the smallest exponent. When q is large, the PEP upper bound is on the order of $O\left(p^k\right)$. Therefore. the order of the union bound on frame error rate with ML decoding is

$$P_B = O\left(\frac{\left(\lambda_2 \rho^{'}(1) p\right)^{\log q}}{q \log q} \right)$$

and the expected number of symbols in error is

$$O\left(\frac{\left(\lambda_2 \rho^{'}(1) p\right)^{\log q}}{q} \right).$$

B. Error Analysis for LMP Algorithms

The error of LMP algorithm comes from two types of decoding failure. The first type of decoding failure is due to unverified symbols. The second one is caused by the false verification (FV). To understand the performance of LMP algorithms, we analyze these types of failure separately. Note that when we analyze the error caused by one reason, we do not consider the other for the simplicity of analysis.

The FV's can be classified into two types. The first type is, as [1] mentions, when the error magnitudes in a single check sum to zero; we call this type-I FV. For single-element lists, it occurs with probability roughly $1/q$ (i.e., the chance that two uniform random symbols are equal). For multiple lists with multiple entries, we analyze the FV probability under the assumption that no list contains the correct value. In this case, each list is uniform on the $q-1$ incorrect values. For m lists of size $s_1, \ldots, s_m$, the type-I FV probability is given by $1 - \binom{q-1}{s_1, s_2, \cdots, s_m} \Big/ \prod_{i=1}^{m} \binom{q-1}{s_i}$. In general, the Birthday paradox applies and the FV probability is roughly $s^2 \binom{m}{2}/q$ for large q and equal size lists.

The second type of FV is that messages become more and more correlated as the number of iterations grows, so that an incorrect message may go through different paths and return to the same node. We denote this kind of FV as a type-II FV.

Note that these are two different types of FV and one does not affect another. We cannot avoid type-II FV by increasing q and we cannot avoid type-I FV by constraining the number of decoding iterations to be within half of the girth (or increasing the girth). Fig. 2 shows an example of type-II FV. In Fig. 2, there is an 8-cycle in the graph and we assume the variable node on the right has an incorrect incoming message "a". Assume that the all-zero codeword is transmitted, all the incoming messages at each variable node are not verified, the list size is less than s_{max}, and each incoming message at each check node contains the correct message. In this case, the incorrect message will travel along the cycle and cause FV's at all variable nodes along the cycle. If the characteristic of the field is 2, there are a total of $c/2$ FV's occurring along the cycle, where c is the length of the cycle. This type of FV can be reduced significantly by choosing each non-zero entry in the parity-check matrix randomly from the non-zero elements of Galois field. In this case, a cycle causes a type-II FV only if the the product of the edge-weights along that cycle is 1. Therefore, we suggest choosing the non-zero entries of the parity-check matrix randomly to mitigate type-II FV. Recall that the idea to use non-binary elements in the parity-check matrix appears in the early works on the LDPC codes over $GF(q)$ [11].

C. An Upper Bound on the Probability of Type-II FV on Cycles

In this subsection, we analyze the probability of error caused by type-II FV. Note that type-II FV occurs only when the depth-$2k$ direct neighborhood of an edge (or a node) has cycles. But type-I FV occurs at every edge (or node). The order of the probability that type-I FV occurs is approximately $O(1/q)$ [1]. The probability of type-II FV is hard to analyze because it depends on q, s_{max} and k in a complicated way. But an upper bound of the probability of the type-II FV is derived in this section.

Since the probability of type-II FV is dominated by short cycles of degree-2 nodes, we only analyze type-II FV along cycles of degree-2 nodes. As we will soon see, the probability of type-II FV is exponential in the length of the cycle. So, the error caused by type-II FV on cycles is dominated by short cycles. We also assume s_{max} to be large enough such that an incorrectly received value can pass around a cycle without being truncated. This assumption makes our analysis an upper bound. Another condition required for an incorrectly received value to participate in a type-II FV is that the product of the edge weights along the cycle is 1. If we assume that almost all edges not on the cycle are verified, then once any edge on the cycle is verified, all edges will be verified in the next k iterations. So we also assume that nodes along a cycle are either all verified or all unverified.

We note that there are three possible patterns of verification on a cycle, depending on the received values. The first case is that all the nodes are received incorrectly. As mentioned above, the incorrect value passes around the cycle without being truncated, comes back the node again and falsely verifies the outgoing messages of the node. So all messages will be falsely verified if they are all received incorrectly after k iterations. Note that this happens with probability $\frac{1}{q-1}p^k$. The second case is that all messages are verified correctly, say, no false verification. Note that this does not require all the nodes to have correctly received values. For example, if any pair of adjacent nodes are received correctly, it is easy to see all messages will be correctly verified. The last case is, there is at least 1 incorrectly received node in any pair of adjacent nodes and there is at least 1 node with correctly received value on the cycle. In this case, all messages will be verified after k iterations, i.e., messages from correct nodes are verified correctly and those from incorrect nodes are falsely verified. Then the verified messages will propagate and half of the messages will be verified correctly and the other half will be falsely verified. Note that this happens with probability $\frac{1}{q-1}2\left(p^{k/2}-p^k\right) \approx \frac{2p^{k/2}}{q-1}$ and this approximation gives an upper bound even if we combine the previous $\frac{1}{q-1}p^k$ term.

Recall that the number of cycles with length k converges to a Poisson with mean $\frac{\left(\lambda_2\rho'(1)\right)^k}{2k}$. Using the union bound, we can upper bound on the ensemble average probability of any type-II FV event with

$$\Pr(\text{any type-II FV}) \leq \sum_{k=k_1}^{\infty} \frac{\left(\lambda_2\rho^{'}(1)\right)^k}{2k(q-1)} 2p^{\frac{k}{2}} = \sum_{k=k_1}^{\infty} \frac{\left(\lambda_2\rho^{'}(1)\sqrt{p}\right)^k}{k(q-1)}.$$

The ensemble average number of nodes involved in type-II FV events is given by

$$\mathrm{E}\left[\text{symbols in type-II FV}\right] \leq \sum_{k=k_1}^{\infty} \frac{\left(\lambda_2\rho^{'}(1)\right)^k}{2k(q-1)} 2kp^{\frac{k}{2}} = \sum_{k=k_1}^{\infty} \frac{\left(\lambda_2\rho^{'}(1)\sqrt{p}\right)^k}{(q-1)}.$$

The upper bound on the frame error rate of type-II FV is on the order of $O\left(\frac{\left(\lambda_2\rho^{'}(1)\sqrt{p}\right)^{\log q}}{(q-1)\log q}\right)$ and the upper bound on the ensemble average number of nodes in type-II FV symbol is on the order of $O\left(\frac{\left(\lambda_2\rho^{'}(1)\sqrt{p}\right)}{(q-1)}\right)$. Notice that both bounds are decreasing functions of q.

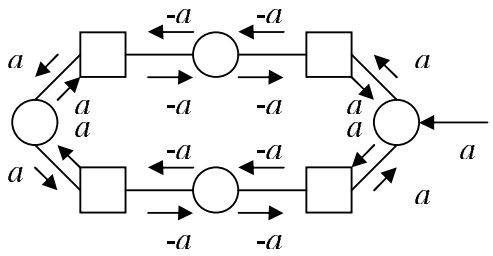

Fig. 2. An example of type-II FV's.

D. An Upper Bound on the Probability of Unverification on Cycles

In this subsection, We derive the union bound for the probability of decoder failure caused by the symbols on short cycles which never become verified. We call this event as *unverification*. As described above, we can always pick q large enough to have arbitrarily small probability of both type-I and type-II FV. In this case, the error is dominated by the unverified messages because the following analysis shows that the union bound on the probability of unverification is independent of q.

In contrast to type-II FV, unverification event does not require cycles, i.e., unverification occurs even on subgraphs without cycles. But in the low error-rate regime, the dominant unverification events occur on short cycles of degree-2 nodes. Therefore, we only analyze the probability of unverification caused by short cycles of degree-2 nodes.

Consider a degree-2 cycle of length k and assume that no FV occurs in the neighborhood of this cycle. Assuming the maximum list size is s_{max}, the condition, which is denoted as UV, for unverification is that there is at most one correctly received value along $s_{max}+1$ adjacent variable nodes. Note that we don't consider type-II FV since type-II FV occurs with probability $\frac{1}{q-1}$ and we can choose q to be arbitrarily large. On the other hand, unverification does not require the product of the edge weights on a cycle to be 1, so we cannot mitigate it by increasing q. So the union bound on the probability of unverification on a cycle with length k is

$$P_U \leq \sum_{k \geq k_2} \frac{\left(\lambda_2 \rho^{'}(1)\right)^k}{2k} \phi(s_{max}, p, k)$$

where $k_2 = \arg\min_{k \geq 1} \frac{\left(\lambda_2 \rho^{'}(1)\right)^k}{2k} \geq 1$ and $\phi(s_{max}, p, k)$ is the UV probability which is given by the following lemma.

Lemma 5.3: Let the cycle have length k, the maximum list size be s, and the channel error probability be p. Then, the probability of an unverification event on a degree-2 cycle of length-k is $\phi(s, p, k) = \mathrm{Tr}\left(B^k(p)\right)$ where $B(p)$ is the $(s+1)$ by $(s+1)$ matrix

$$B(p) = \begin{bmatrix} p & 1-p & 0 & 0 & \cdots & 0 \\ 0 & 0 & p & 0 & \cdots & 0 \\ 0 & 0 & 0 & p & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & p \\ p & 0 & 0 & 0 & \cdots & 0 \end{bmatrix}. \tag{18}$$

Proof: See Appendix D. ∎

Finally, the union bound on the average number of symbols involved in unverification events is

$$\mathrm{E}\left[\text{unverified symbols}\right] \leq \sum_{k \geq k_2}^{\infty} \frac{\left(\lambda_2 \rho^{'}(1)\right)^k}{2} \phi(s_{max}, p, k). \tag{19}$$

VI. Comparison and Optimization

In this section, we compare the proposed algorithm with maximum list size S (LMP-S) with other message-passing decoding algorithms for the q-SC. We note that the LM2-MB algorithm is identical to SW1 for any code ensemble because the decoding

TABLE II
Optimization Results for LMP Algorithms (rate 1/2)

Alg.	$\boldsymbol{\lambda}(\mathbf{x})$	$\boldsymbol{\rho}(\mathbf{x})$	$\mathbf{p}^*$
LMP-1	$.1200x+.3500x^2+.0400x^4+.4900x^{14}$	x^8	.2591
LMP-1	$.1650x+.3145x^2+.0085x^4+.2111x^{14}+.0265x^{24}+.0070x^{34}+.2674x^{49}$	$.0030x^2+.9970x^{10}$	.2593
LMP-8	$.32x+.24x^2+.26x^8+.19x^{14}$	$.02x^4+.82x^6+.16x^8$	.288
LMP-32	$.40x+.20x^3+.13x^5+.04x^8+.23x^{14}$	$.04x^4+.96x^6$	.303
LMP-∞	$.34x+.16x^2+.21x^4+.29x^{14}$	x^7	.480
LM2-MB	$.2x+.3x^3+.05x^5+.45x^{11}$	x^8	.289

TABLE III
THRESHOLD VS. ALGORITHM FOR THE (3,6) REGULAR LDPC ENSEMBLE

LMP-1	LMP-8	LMP-32	LMP-∞	LM1	LM2-MB	LM2-NB
.210	.217	.232	.429	.169	.210	.259

rules are the same. LM2-MB, SW1 and LMP-1 are identical for (3,6) regular LDPC codes because the list size is always 1 and erasure never happens in LMP-1 for (3,6) regular LDPC codes. and the LMP-∞ algorithm is identical to SW2.

There are two important differences between the LMP algorithm and previous algorithms: (i) erasures and (ii) FV recovery. The LMP algorithm passes erasures because, with a limited list size, it is better to pass an erasure than to keep unlikely symbols on the list. The LMP algorithm also detects FV events and passes an erasure if they cause disagreement between verified symbols later in decoding, and can sometimes recover from a FV event. LM1-NB and LM2-NB fix the status of a variable node once it is verified and pass the verified value in all following iterations.

The results in [1] and [14] also do not consider the effects of type-II FV. These FV events degrade the performance in practical systems with moderate block lengths, and therefore we use random entries in the parity-check matrix to mitigate these effects.

Using the DE analysis of the LMP-S algorithm, we can improve the threshold by optimizing the degree distribution pair (λ, ρ). Since the DE recursion is not one-dimensional, we use differential evolution to optimize the code ensembles [19]. In Table II, we show the results of optimizing rate-$\frac{1}{2}$ ensembles for LMP with a maximum list size of 1, 8, 32, and ∞. Thresholds for LM1 and LM2-NB/MB with rate 1/2 are also shown. In all but one case, the maximum variable-node degree is 15 and the maximum check-node degree is 9. From Table II, one can see that the resulting code ensembles have concentrated check-node degrees. The second table entry allowed for larger degrees (in order to improve performance) but very little gain was observed. We can also see that there is a gain of between 0.05 and 0.07 over the thresholds of (3,6) regular ensemble with the same decoder.

VII. SIMULATION RESULTS

In this part, we show the simulation results for (3,6) regular LDPC codes using various decoding algorithms as well as the simulation results for the optimized ensembles shown in Table II with LMP algorithms in Fig. VIII. In the simulation of optimized ensembles, we try different maximum list sizes and different finite fields. We use notation "LMPs,q,ensemble" to denote the simulation result of LMP algorithm with maximum list size s, finite field $GF(q)$ and the simulated ensemble. We choose block length to be 100000. The parity-check matrices are chosen randomly without 4-cycles. Each non-zero entry in the parity-check matrix is chosen uniformly from GF$(q) \setminus 0$. This allows us to keep the FV probability low. The maximum number of decoding iterations is fixed to be 200 and more than 1000 blocks are run for each point. These results are compared with the theoretical thresholds. Table III shows the theoretical thresholds of $(3, 6)$ regular codes on the q-SC for different algorithms and Table II shows the thresholds for the optimized ensembles. The numerical results match the theoretical thresholds very well.

In the results of (3,6) regular codes simulation, we cannot see any error floor because there is almost no FV in the simulation. The LM2-NB performs much better than other algorithms with list size 1 for (3,6) regular ensemble. But in the results of the optimized ensembles, the error floors occur because the number of degree-2 variable nodes the maximum variable/check degrees are significantly larger than the (3,6) regular ensemble. By evaluating Eq. (19), the predicted error floor caused by unverification is 1.6×10^{-5} for the optimized $s_{max} = 1$ ensemble, 8.3×10^{-7} for the optimized $s_{max} = 8$ ensemble, and 1.5×10^{-6} for the optimized $s_{max} = 32$ ensemble. From the results, we see the analysis of unverification events matches the numerical results very well.

VIII. CONCLUSIONS

In this paper, we discuss list-message-passing (LMP) decoding algorithms for the q-ary symmetric channel (q-SC). It is shown that capacity-achieving ensembles for the BEC achieve capacity on the q-SC when the list size is unbounded. Decoding thresholds are also calculated by density evolution (DE). We also derive a new analysis for the node-based algorithms described in [1]. The causes of false verification (FV) are analyzed and random entries in the parity-check matrix are used to mitigate avoid type-II FV. Degree profiles are optimized for the LMP decoder and reasonable gains are obtained. Finally, simulations show that, with list size larger than 8, the proposed LMP algorithm outperforms previously proposed algorithms.

While we focus on the q-SC in this work, there are a number of other applications of LMP decoding that are also quite interesting. For example, the iterative decoding algorithm described in [20] for compressed sensing is actually the natural extension of LM1 to continuous alphabets. For this reason, the LMP decoder may also be used to improve the threshold of compressed sensing. This is, in some sense, more valuable because there are a number of good coding schemes for the q-SC, but few low-complexity near-optimal decoders for compressed sensing.

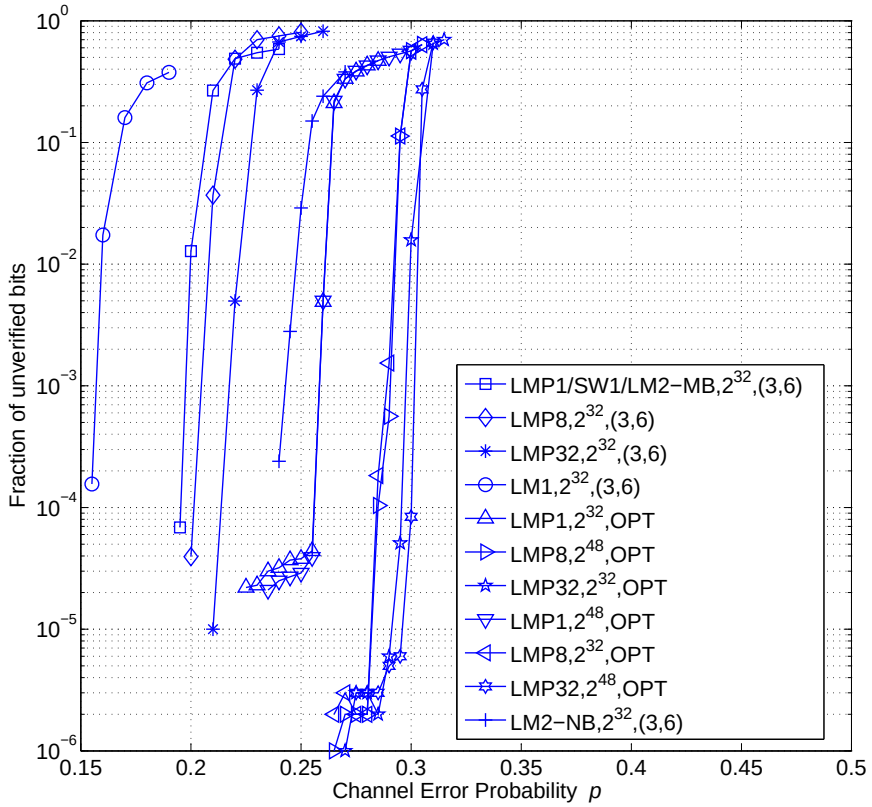

Fig. 3. Simulation results for (3,6) regular codes with block length 100000.

APPENDIX I
PROOF OF THEOREM 2.1

Proof: Given $p\lambda(1-\rho(1-x)) < x$ for $x \in (0,1]$, we start by showing that both x_i and y_i go to zero as i goes to infinity. To do this, we let $\alpha = \sup_{x\in(0,1)} \frac{1}{x} p\lambda(1-\rho(1-x))$ and note that $\alpha < 1$ because $p < p^*$. It is also easy to see that, starting from $x_0 = 1$, we have $x_i \le \alpha^i$ and $x_i \to 0$. Next, we rewrite Eq. (3) as

$$\begin{aligned} y_{i+1} &= \frac{1}{p} x_{i+1} + p\left(\rho(1-x_i) - \rho(1-y_i)\right)\lambda'(1-\rho(1-x_i)) \\ &\overset{(a)}{\le} \frac{1}{p}\alpha^{i+1} + p\left(1-\rho'(1)\alpha^i - \rho(1-y_i)\right)\left(\lambda_2 + O(\alpha^i)\right) \\ &\overset{(b)}{\le} \frac{1}{p}\alpha^{i+1} + p\lambda(1-\rho(1-y_i))\left(1+O(\alpha^i)\right) \\ &\overset{(c)}{\le} \frac{1}{p}\alpha^{i+1} + \alpha y_i\left(1+O(\alpha^i)\right), \end{aligned}$$

where (a) follows from $\rho(1-x) \le 1-\rho'(1)x$, (b) follows from $\lambda_2(1-\rho(1-y)) \le \lambda(1-\rho(1-y))$, and (c) follows from $p\lambda(1-\rho(1-y)) \le \alpha y$. It is easy to verify that $y_{i+1} < y_i$ as long as $y_i > \frac{\alpha^{i+1}}{p(1-\alpha(1+O(\alpha^i)))}$. Therefore, we find that $y_i \to 0$ because the recursion does not have any positive fixed points as $i \to \infty$. Moreover, one can show that y_i eventually decreases exponentially at a rate arbitrarily close to α.

Note that the decoding error comes from two reasons, one is the event that message is not verified and the other one is the event that the message is falsely verified. Next, we are going to show that the actual performance of a code converges to the ensemble average exponentially which means almost every code in a capacity-achieving ensemble has capacity-achieving performance. Note that the concentration effect and the decay of FV probability hold regardless the error probability of the decoder converges to zero or not.

We can prove that the performance of a particular code converges to the threshold which is the average performance of a tree-like ensemble in a similar way in [15], where the average is over the graph ensemble $(\lambda(x), \rho(x))$ and all the channel inputs. There are two difference between our scenario and [15], i.e, our algorithm passes a list of values with unbounded list size, the second difference is the graph may be irregular in our case. Here we only mention the brief procedure of the proof. We can let $Z^{(l)}/E$ denote the fraction of *unverified* messages at the l-th iteration, where E is the number of edges in the graph. Note that $Z^{(l)}$ denotes the number of *incorrect* and *erasure* messages in [15]. Following [15], we can break failure the

probability into a tree-like neighborhood term and a Martingale concentration term to get

$$\Pr\left(\left|\frac{Z^{(l)}(\mathbf{s})}{E}-y_l\right|\geq\epsilon\right)\leq$$
$$\Pr\left(\left|\frac{Z^{(l)}(\mathbf{s})}{E}-\frac{\mathrm{E}\left[Z^{(l)}(\mathbf{s})\right]}{E}\right|\geq\epsilon/2\right)+$$
$$\Pr\left(\left|\frac{\mathrm{E}\left[Z^{(l)}(\mathbf{s})\right]}{E}-y_l\right|\geq\epsilon/2\right)$$

where $\mathbf{s}$ is an arbitrary codeword chosen from ensemble (λ,ρ), $Z^{(l)}(\mathbf{s})$ is the random variable that denotes the number of erroneous variable-to-check messages after l decoding iterations. E is the number of edges in the graph. This means the concentration theorem can be think of consisting of two parts of concentration, i.e., concentration from a particular code to the ensemble with cycles and concentration from a ensemble with cycles to the tree-like ensemble. Notice that the proof of the later concentration and the proof of the probability of a tree-like neighborhood are not limited to the specific decoding algorithm and the definition of $Z^{(l)}$, the proof is omitted here. By forming a Doob's martingale on the edge-exposure and applying Azuma's inequality, we can prove the concentration from a particular code to the ensemble in the same manner as [15]. In our scenario, the proof of bounded difference of the martingale, the right hand side of in [15, Eq. (16)] should be the cardinality of depth $2l$ directed neighbor of e, $\frac{\left|\vec{\mathcal{N}}_e^{(2l)}\right|}{2}$. The right hand side of in [15, Eq. (17)] should be $4\left|\vec{\mathcal{N}}_e^{(2l)}\right|$. The β in applying Azuma's inequality is $\sum_{k=1}^{E}\left(4\left|\vec{\mathcal{N}}_e^{(2l)}\right|\right)^2+\sum_{k=1}^{n}\left(4\left|\vec{\mathcal{N}}_e^{(2l)}\right|\right)^2$. So far, we prove that, for an arbitrary small constant $\epsilon/2$, there exist positive numbers β and γ, such that if $n>\frac{2\gamma}{\epsilon}$, then

$$\Pr\left(\left|\frac{Z^{(l)}(\mathbf{s})}{E}-y_l\right|\geq\epsilon\right)\leq e^{-\beta\epsilon^2 n}$$

Note that the similar proof can be found in [15] (the proof of Theorem 2) and [21] (the proof of Theorem 1). Note that [15] proves for the regular code ensemble and [21] extends the proof to the irregular code ensemble. So, for an arbitrary code $\mathbf{s}$ and an arbitrary small quantity ϵ, the fraction of unverified message is less than $\epsilon/2$ as n goes to infinity.

In [15] and [21], it is proved that, when a code graph is chosen uniformly at random from all possible graphs with degree distribution pair $(\lambda(x),\rho(x))$,

$$\Pr\left(\text{neighborhood of depth } 2l \text{ is not tree-like}\right)\leq\frac{\gamma}{n}$$

where γ is a constant independent of n. So, given ϵ, we can choose n large enough such that the number of variable nodes which are involved in cycles of length less than $2l$ is less than $n\epsilon/2$ with probability arbitrarily close to one as n goes to infinity. So the probability of error caused by type-II FV's is upper bounded by $\epsilon/2$ (for the notation of type-I and type-II FV, please refer to Section V-B). Here, we don't consider the type-I FV's because the probability of type-I FV's can be forced arbitrarily close to zero by choosing a large enough q. ■

APPENDIX II
PROOF OF THEOREM 3.1

The verification occurs in LM1-NB also occurs in LM1-MB and vice versa. So LM1-NB and LM1-MB are equivalent.

Proof: The operations of LM1-MB and LM1-NB are different because they have different verification rules (see Table. I). We can prove they are equivalent by showing the verification occurs in LM1-MB also occurs in LM1-NB and vice versa, but in different decoding steps. Let's first look at the check node when the summation of all messages equals to zero but there are more than 1 messages are unverified. In this case, LM1-NB will verify all the messages. In LM1-MB none of them will be verified but all the values will be correct. In the following iteration, all these messages will be verified on their variable nodes. Notice that this is the only case verification occurs in LM1-NB but not in LM1-MB. So verification in LM1-NB also occurs in LM1-MB. Let's then look at the variable node when any incoming message is correct and the channel value is correct. In LM1-MB, the outgoing message will be verified. In LM1-NB the message will be correct but not verified. Notice that the incoming is correct means all the other messages are correct at the check node, so the unverified correct message will be verified in the next step on check node. Notice that this is the only case verification occurs in LM1-MB but not in LM1-NB. So verification in LM1-MB also occurs in LM1-NB. ■

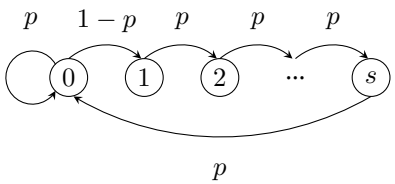

Fig. 4. Finite-state machine for Lemma 5.3.

APPENDIX III
PROOF OF LEMMA 5.1

Let y be the received symbol sequence assuming the all-zero codeword was transmitted. Let u be any codeword with exactly k non-zero symbols. It is easy to verify that the probability that ML decoder chooses u over the all-zero codeword is given by

$$p_{2,k} = \sum_{j=0}^{k} \sum_{i=0}^{j} \binom{k}{i, j, k-i-j} (1-p)^i \left(\frac{p}{q-1}\right)^j \left(\frac{p(q-2)}{q-1}\right)^{k-i-j}.$$

Using the multinomial theorem, it is also easy to verify that

$$\begin{aligned} A(x) &= \left((1-p) + \frac{p}{q-1}x^2 + \frac{p(q-2)}{q-1}x\right)^k \\ &= \sum_{j=0}^{k} \sum_{i=0}^{k-j} \binom{k}{i, j, k-i-j} (1-p)^i \left(\frac{p}{q-1}\right)^j \left(\frac{p(q-2)}{q-1}\right)^{k-i-j} x^{k-i+j} \\ &\triangleq \sum_{l=0}^{2k} A_l x^l, \end{aligned}$$

where A_l is the coefficient of x^l in $A(x)$. Finally, we observe that $p_{2,k} = \sum_{l=k}^{2k} A_l$ is simply an unweighted sum of a subset of terms in $A(x)$ (namely, those where $k - i + j \geq k$).

This implies that

$$x^k p_{2,k} = \sum_{l=k}^{2k} A_l x^k \leq A(x)$$

for any $x \geq 1$. Therefore, we can compute the Chernoff-type bound

$$p_{2,k} \leq \inf_{x \geq 1} x^{-k} A(x).$$

By taking derivative of $x^{-k}A(x)$ over x and setting it to zero, we arrive at the bound

$$p_{2,k} \leq \left(p\frac{q-2}{q-1} + \sqrt{\frac{4p(1-p)}{q-1}}\right)^k.$$

APPENDIX IV
PROOF OF LEMMA 5.3

Proof: An unverification event occurs on a degree-2 cycle of length-k when there is at most one correct variable node in any adjacent set of $s+1$ nodes. Let the set of all error patterns (i.e., 0 means correct and 1 means error) of length-k which satisfy the UV condition be $\Phi(s, p, k) \subseteq \{0, 1\}^k$. Using the Hamming weight $w(z)$, of an error pattern as z, to count the number of errors, we can write the probability of UV as

$$\phi(s, p, k) = \sum_{z \in \Phi(s,p,k)} p^{w(z)} (1-p)^{k-w(z)}.$$

This expression can be evaluated using the transfer matrix method to enumerate all weighted walks through a particular digraph. If we walk through the nodes along the cycle by picking an arbitrary node as the starting node, the UV constraint can be seen as k-steps of a particular finite-state machine. Since we are walking on a cycle, the initial state must equal to the final state.

The finite-state machine, which is shown in Fig. 4, has $s+1$ states $\{0, 1, \ldots, s\}$. Let state 0 be the state where we are free to choose either a correct or incorrect symbol (i.e., the previous s symbols are all incorrect). This state has a self-loop associated with the next symbol also being incorrect. Let state $i > 0$ be the state where the past i values consist of one correct symbol followed by $i-1$ incorrect symbols. Notice that only state 0 may generate correct symbols. By defining the transfer matrix with (18), the probability that the UV condition holds is therefore $\phi(s, p, k) = \mathrm{Tr}\left(B^k(p)\right)$. ■

REFERENCES

[1] M. Luby and M. Mitzenmacher, "Verification-based decoding for packet-based low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 51, pp. 120–127, Jan. 2005.

[2] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, vol. 18, pp. 21–28, Jan. 1962.

[3] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, vol. 45, pp. 399–431, Mar. 1999.

[4] M. Luby, M. Mitzenmacher, M. Shokrollahi, and D. Spielman, "Efficient erasure correcting codes," *IEEE Trans. Inform. Theory*, vol. 47, pp. 569–584, Feb. 2001.

[5] T. Richardson, M. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 47, pp. 619–637, Feb. 2001.

[6] A. Shokrollahi, "Capacity-approaching codes on the q-ary symmetric channel for large q," in *Proc. IEEE Inform. Theory Workshop*, (San Antonio, TX), pp. 204–208, Oct. 2004.

[7] G. Lechner and C. Weidmann, "Optimization of binary LDPC codes for the q-ary symmetric channel with moderate q," in *Proc. IEEE Int. Symp. Information Theory*, (Lausanne, Switzerland), Aug. 2008.

[8] G. Lechner and C. Weidmann, "Optimization of binary ldpc codes for the q-ary symmetric channel with moderate q," in *Proc. 5th International Symposium on Turbo Codes and Related Topics*, pp. Lausanne, Switzerland, 2008.

[9] J. Metzner, "Majority-logic-like decoding of vector symbols," *IEEE Trans. Commun.*, vol. 44, pp. 1227–1230, Oct. 1996.

[10] J. Metzner, "Majority-logic-like vector symbol decoding with alternative symbol value lists," *IEEE Trans. Commun.*, vol. 48, pp. 2005–2013, Dec. 2000.

[11] M. Davey and D. MacKay, "Low density parity check codes over GF(q)," *IEEE Commun. Lett.*, vol. 2, pp. 58–60, 1998.

[12] D. Bleichenbacher, A. Kiyayias, and M. Yung, "Decoding of interleaved Reed-Solomon codes over noisy data," in *Proc. of ICALP*, pp. 97–108, 2003.

[13] A. Shokrollahi and W. Wang, "Low-density parity-check codes with rates very close to the capacity of the q-ary symmetric channel for large q," in *Proc. IEEE Int. Symp. Information Theory*, (Chicago, IL), p. 275, June 2004.

[14] A. Shokrollahi and W. Wang, "Low-density parity-check codes with rates very close to the capacity of the q-ary symmetric channel for large q." Unpublished extended abstract, 2004.

[15] T. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inform. Theory*, vol. 47, pp. 599–618, Feb. 2001.

[16] V. Guruswami and P. Indyk, "Linear time encodable and list decodable codes," in *Proc. of the 35th Annual ACM Symp. on Theory of Comp.*, pp. 126–135, 2003.

[17] N. Wormald, "Differential equations for random processes and random graphs," *Annals of Applied Probability*, vol. 5, pp. 1217–1235, 1995.

[18] T. J. Richardson and R. L. Urbanke, *Modern Coding Theory*. Cambridge, 2008.

[19] R. Storn and K. Price, "Differential evolution–A simple and efficient heuristic for global optimization over continuous spaces," *J. Global Optim.*, vol. 11, no. 4, pp. 341–359, 1997.

[20] S. Sarvotham, D. Baron, and R. Baraniuk, "Sudocodes–Fast measurement and reconstruction of sparse signals," in *Proc. IEEE Int. Symp. Information Theory*, (Seattle, WA), pp. 2804–2808, July 2006.

[21] A. Kavcic, X. Ma, and M. Mitzenmacher, "Binary intersymbol interference channels: Gallager codes, density evolution, and code performance bounds," *IEEE Trans. Inform. Theory*, vol. 49, pp. 1636–1652, July 2003.