A Fresh Look at Coding for q-ary Symmetric Channels
Claudio Weidmann, Member, IEEE, and Gottfried Lechner Member, IEEE

Abstract—This paper studies coding schemes for the q-ary symmetric channel based on binary low-density parity-check (LDPC) codes that work for any alphabet size \( q = 2^m \), \( m \in \mathbb{N} \), thus complementing some recently proposed packet-based schemes requiring large \( q \). First, theoretical optimality of a simple layered scheme is shown, then a practical coding scheme based on a simple modification of standard binary LDPC decoding is proposed. The decoder is derived from first principles and using a factor-graph representation of a front-end that maps q-ary symbols to groups of \( m \) bits connected to a binary code. The front-end can be processed with a complexity that is linear in \( m = \log_q q \). An extrinsic information transfer chart analysis is carried out and used for code optimization. Finally, it is shown how the same decoder structure can also be applied to a larger class of q-ary channels.

Index Terms—q-ary symmetric channel, low-density parity-check (LDPC) codes, decoder front-end.

I. INTRODUCTION

The q-ary symmetric channel (q-SC) with error probability \( \epsilon \) takes a q-ary symbol at its input and outputs either the unchanged input symbol, with probability \( 1 - \epsilon \), or one of the other \( q - 1 \) symbols, with probability \( \epsilon / (q - 1) \). It has attracted some attention recently as a more general channel model for packet-based error correction. For very large \( q \), its appropriately normalized capacity approaches that of an erasure (packet loss) channel. In the following, we will only consider channel alphabets of size \( q = 2^m \) with \( m \in \mathbb{N} \).

The capacity of the q-SC with error probability \( \epsilon \) is

\[
C_{q-SC} = m - h(\epsilon) - \epsilon \log_2(2^m - 1)
\]

bits per channel use, where \( h(x) = -x \log_2 x - (1-x) \log_2(1-x) \) is the binary entropy function. Asymptotically in \( m \), the normalized capacity \( C_{q-SC}/m \) thus approaches \( 1 - \epsilon \), which is the capacity of the binary erasure channel (BEC) with erasure probability \( \epsilon \).

Recent work [1]–[5] has shown that it is possible to devise coding schemes that approach \( C_{q-SC} \) for large alphabet sizes \( q = 2^m \), with symbols of hundreds to thousands of bits, and complexity \( O(\log q) \) per code symbol. The focus of the present work is on smaller \( q \), with symbols of tens of bits at most, although the presented coding techniques will work for any \( q = 2^m \).

The q-ary channel input and output symbols will be represented by binary vectors of length \( m \). Hence a simplistic coding approach consists in decomposing the q-SC into \( m \) binary symmetric channels (BSCs) with crossover probability

\[
\epsilon_{BSC} = \frac{q \epsilon}{2(q - 1)} = \frac{\epsilon}{2(1 - 2^{-m})},
\]

which have capacity \( C_{BSC} = 1 - h(\epsilon_{BSC}) \) each.

We briefly study the normalized capacity loss, \( \Delta = C_{q-SC}/m - C_{BSC} \), which results from (wrongly) assuming that the q-SC is composed of independent BSCs. For fixed \( m \), we have \( \lim_{\epsilon \to 0} \Delta = 0 \); so using binary codes with independent decoders on the \( m \)-fold BSC decomposition might be good enough for small \( \epsilon \) (e.g., \( \epsilon < 10^{-3} \)). However, Fig. 1 shows that the relative capacity loss \( m \Delta / C_{q-SC} = 1 - m C_{BSC}/C_{q-SC} \) increases close to linearly in \( \epsilon \). For fixed \( \epsilon \), we have \( \lim_{m \to \infty} \Delta = h(\epsilon/2) - \epsilon \), which can be a substantial fraction of the normalized q-SC capacity (e.g., for \( \epsilon = 0.1 \), \( h(\epsilon/2) - \epsilon = 0.19 \)). Fig. 2 shows that already for small \( m \), the q-SC capacity is substantially larger than what can be achieved with the BSC decomposition. Clearly, there is a need for coding schemes targeted at "large" \( \epsilon \) (say \( \epsilon > 10^{-1} \)), but moderate \( m \) (say \( 2 \leq m < 20 \)), which are not that well handled by methods for large \( m \). For example, to use verification-based decoding, the symbols should have between \( m = 32 \) bits for short codes (block length \( 10^2 \ldots 10^3 \)) and \( m = 64 \) bits for longer codes (block length \( 10^4 \ldots 10^5 \)) [1].

One example application, which also motivated this work, is Slepian-Wolf coding of q-ary sources with a discrete impulse-noise correlation model, using q-SC channel code syndromes as fixed-rate (block) source code. This can then be used as a building block for a Wyner-Ziv coding scheme, where \( q \) is the number of scalar quantizer levels, or for fixed-rate quantization of sparse signals with a Bernoulli-\( \epsilon \) prior on being nonzero [6]. Clearly, \( q \) will be only moderately large in such a scenario.

The capacity loss of the binary decomposition is due to the fact that the correlation between errors on the binary sub-channels is not taken into account, i.e., since an error on one sub-channel of the q-SC implies a symbol error, the error probabilities on the other sub-channels will change.
from first principles, by factorizing the posterior probabilities. The binary code and develops the new variable node decoding rules in rate. Section III-A then proposes a scheme using a single bits of the symbol may be treated as erasures without loss at least \(q\)-ary decoder of complexity \(O_m\) that maps \(2^m\) such binary vectors \(y \neq x\), out of a total \(2^m - 1\). The \(i\)-th binary sub-channel is thus characterized by

\[
\epsilon_i = \frac{2^{m-i}}{2^m - 1} \epsilon_i
\]

\[
\delta_i = \sum_{j=1}^{m-i} \epsilon_j = \frac{2^m - 2^{m-i+1}}{2^m - 1} \epsilon_i.
\]

**Theorem 1** The layered scheme achieves q-SC capacity if the constituent BSEC codes achieve capacity.

**Proof:** We may assume an ideal scheme, in which all layers operate at their respective BSEC capacities and correct all errors and erasures. The BSEC(\(\delta_i, \epsilon_i\)) capacity is

\[
C_{\text{BSEC}} = (1 - \delta_i) \left(1 - h \left(\frac{\epsilon_i}{1 - \delta_i}\right)\right).
\]
Hence the sum of the layer rates becomes

$$\sum_{i=1}^{m} R_i = m \sum_{i=1}^{m} \left( 1 - \delta_i \left( 1 - h \left( \frac{\epsilon_i}{1 - \delta_i} \right) \right) \right)$$

$$= m + \sum_{i=1}^{m} \left\{ - \delta_i (1 - \delta_i) \log_2 (1 - \delta_i) + \epsilon_i \log_2 \epsilon_i + (1 - \delta_{i+1}) \log_2 (1 - \delta_{i+1}) \right\}$$

$$= m + \sum_{i=1}^{m} \left\{ - (m - i) \epsilon_i + \epsilon_i \log_2 \epsilon_i \right\} + (1 - p) \log_2 (1 - p)$$

$$= m + \sum_{i=1}^{m} \left\{ - (m - i) \epsilon_i + \epsilon_i \right\} + p \log_2 p - p \log_2 (2m - 1) + (1 - p) \log_2 (1 - p)$$

$$= m - h(p) - p \log_2 (2m - 1) = C_{qSC},$$

where (5) follows from (4) and the definition of the layered scheme, (6) follows from $\delta_i + \epsilon_i = \delta_{i+1}$ (which holds up to $i = m$, when $\delta_{m+1} = \epsilon$), (7) follows from the evaluation of the telescoping sum and $\sum_{i=1}^{m} \delta_i = \sum_{i=1}^{m} (m - i) \epsilon_i$, and (8) follows from substituting (2) for $\epsilon_i$.

As can be seen from inserting (2) and (3) into (4), the layer rate quickly approaches $1 - \epsilon$ for increasing $i$, that is, for large $m$ the last layers all operate at rates close to $1 - \epsilon$. Thus an interesting variant of the layered scheme is to use $\mu < m$ BSEC layers as above, followed by a single “thicker” layer, which sends the remaining $m - \mu$ bits (per symbol) over the same BSEC($\delta_{\mu+1}, \epsilon_{\mu+1}$). In particular, this could even be beneficial in practical implementations, since combining layers leads to longer codewords and thus better codes, which might outweigh the theoretical rate loss. The next section will show that it is actually possible to reap the benefits of a single large binary code, without any layering, by using a decoder that exploits the dependencies among the bits in a symbol.

The layered scheme has several disadvantages compared to the symmetric scheme presented below. The main disadvantage is that in practical implementations, each layer will need a different BSEC code that is tuned to the effective erasure and error probabilities resulting from the layers preceding it. Besides causing problems with error propagation, this is also impractical for hardware implementation, since the required silicon area would necessarily grow with the number of layers. Another disadvantage is that for a fixed symbol block length $n$, the layered scheme uses binary codes of length $n$, while the symmetric scheme uses block length $mn$. The latter will provide a performance advantage, especially for short block lengths.

III. BIT-SYMMETRIC CODING SCHEME

Ideally, a coding scheme for the $q$-SC should be symmetric in the bits composing a symbol, that is, no artificial hierarchy among bit layers should be introduced (notice that the order of the bit layers may be chosen arbitrarily). We propose to encode all bits composing the symbols with one “big” binary code, which needs to satisfy just slightly stricter constraints than an ordinary code, while the decoder alone will exploit the knowledge about the underlying $q$-SC. The key concept that should carry over from the layered scheme is that the decoder is able to declare erasures at certain symbol positions and thus needs less error correction capability (for part of the bits in erased symbols).

We present two approaches to describe the decoder for the proposed scheme: a bit-level APP approach, which clearly displays the probabilities being estimated (and is more intuitive to generalize, see Sec. V), and a decoder front-end approach, which facilitates the use of EXIT chart tools. The model underlying both approaches is the same: the decoder will use estimates of the extrinsic symbol error probabilities to estimate the marginal bit error probabilities, thus “adjusting” the individual error probabilities of the binary sub-channels.

A. Bit-Level APP Approach

Our proposal for a practical symmetric $q$-SC coding scheme relies on an LDPC code with information block length $K = mn$ bits and channel block length $N = mn$ bits. We assume that the variable nodes (VNs) in the decoder receive independent extrinsic soft estimates of $X^j_i$ (that is, bit $i$ of code symbol/vector $X^j$, for $i = 1, \ldots, m$, $j = 1, \ldots, n$) from the check nodes (CNs). These amount to estimates of $P(X^j_i | Y^{|j|} = y^{|j|})$ or the corresponding log-likelihood ratio (LLR), $L(X^j) = \log(Pr(X^j = 0)/Pr(X^j = 1))$. (As usual, the notation $y^{|j|}$ denotes the block consisting of all symbols/vectors except the $j$-th.) In particular, the extrinsic estimate of $X^j_i$ is assumed to be independent of the other bits $X^j_{i'}$ of the same symbol, so that we may write

$$P(X^j_i | X^j_{i+1}, \ldots, X^j_{i-1}, Y^{|j|} = y^{|j|}) = P(X^j_i | Y^{|j|} = y^{|j|}).$$

In the standard case, these independence assumptions are justified by the fact that asymptotically in the block length, the neighborhood of a VN in the LDPC decoder computation graph becomes a tree [9, Chap. 3]. Unfortunately, the $q$-SC VN message computation rule has to depend on the other bits in the same symbol, in order to account for the bit error correlation, and thus will introduce cycles. However, this problem can be alleviated by imposing an additional constraint on the code, namely that the parity checks containing $X^j_i$ do not involve any of the bits $X^j_{i'}$. This is a necessary condition for the above intra-symbol independence assumption and may be achieved by using an appropriate edge interleaver in the LDPC construction. Then the cycles introduced by the VN message rule will grow asymptotically and are thus not expected to lead to problems in practice.

As suggested above, the properties of the $q$-SC can be taken into account via a simple modification of the VN computation in the message-passing decoding algorithm for binary LDPC codes. We factor the a posteriori probability (APP) of symbol $X^j$ as follows:

$$P(X^j | Y = y) \doteq P(Y^j = y^{|j|} | X^j)P(X^j | Y^{|j|} = y^{|j|})$$

$$= P(Y^j = y^{|j|} | X^j) \prod_{i=1}^{m} P(X^j_i | Y^{|j|} = y^{|j|}), \quad (9)$$
where the factorization in (9) is made possible by the above independence assumption (the symbol \( \oplus \) denotes equality up to a positive normalization constant). Using the definition of the q-SC, this becomes

\[
P(X^j = x^j|Y = y) = \frac{1}{\mathcal{Q}^m} \sum_{i=1}^{m} P(X^j = x^j|Y^i = y^i, x^j = y^i), \quad x^j \neq y^j. \quad (10)
\]

We define the extrinsic probability that \( X^j = y^j \) as

\[
\beta^j = \frac{P(X^j = y^j|Y^i = y^i)}{P(y^i)} = \prod_{i=1}^{m} p_i^j,
\]

where we introduced

\[
p_i^j = P(X^j = y^i|Y^i = y^i) \quad (11)
\]

for notational convenience. The normalization constant in (10) thus becomes

\[
\gamma^j = (1 - \epsilon)\beta^j + \frac{\epsilon}{q - 1}(1 - \beta^j).
\]

Then the bit APP may be obtained by the marginalization

\[
P(X^j = x^j|Y = y) = \sum_{x^j \in \{0, 1\}^n} P(X^j = x^j|Y = y), \quad (12)
\]

which may be written as

\[
P(X^j = x^j|Y = y) = \frac{(1 - \epsilon)\beta^j + \frac{\epsilon}{q - 1}(1 - \beta^j)}{1 + \epsilon \frac{q - 1}{q} \prod_{i=1}^{m} p_i^j}, \quad x^j = y^j, \quad x^j \neq y^j, \quad (13)
\]

where \( \beta^j_i = \beta^j/p_i^j \) is the intra-symbol extrinsic probability that \( X^j = y^j \), using no information on bit \( X^j_i \). Finally, we may express the \textit{a posteriori} bit-level LLR as

\[
L_{\text{app}}(X^j_i = y^j_i) = \log \left( \frac{(q - 1)\beta^j_i - q\epsilon\beta^j_i + \epsilon p_i^j}{\epsilon(1 - p_i^j)} \right) = \log \left( \frac{(q - 1)\beta^j_i - q\epsilon\beta^j_i + \epsilon}{\epsilon} \cdot \frac{1}{1 - p_i^j} \right) = \log \left( 1 + \frac{q - 1 - \epsilon}{\epsilon} \beta^j_i \right) + L_{\text{extr}}(X^j_i = y^j_i). \quad (14)
\]

The usual \( L(X) \) is obtained from \( L(X = y) = \log(\Pr(X = y)/\Pr(X = 1 - y)) \) via a sign flip, \( L(X) = (1 - 2y)L(X = y) \).

The second term in (14) corresponds to the extrinsic information from the CNs that is processed at the VNs in order to compute the bit APP in standard binary LDPC decoding. The difference lies in the first term in (14), which corresponds to the channel LLR (which would be \( L_{\text{ch}}(X) = \log(P(y_i|X = 0)/P(y_i|X = 1)) \) in the binary case). When the extrinsic information on the bits \( X^j_i \) favors the hypothesis \( X^j \neq y^j \), the product \( \beta^j_i \) will be small and therefore \( L_{\text{ch}} \) in (14) will be close to zero, which is equivalent to declaring a bit erasure. This shows that the symmetric LDPC scheme relies on “distributed” soft bit erasure estimates, while in the layered scheme the erasures are declared in a \textit{hard} “top-down” fashion.

Equation (14) describes the modification of the VN computation that turns a message-passing binary LDPC decoder into one for the q-ary symmetric channel. The outgoing VN messages are computed as usual by subtracting the incoming edge message from \( L_{\text{app}}(X^j_i) \); also the CN messages are the same as in the binary case. For practical implementation purposes, (14) should probably be modified (approximated) in order to avoid switching back and forth between probabilities and LLRs when computing \( \beta^j \).

A final detail is the specification of the initial channel LLR \( L_{\text{ch}}^{(0)} \) in (14), which is needed to start the decoder iterations. By inserting the memoryless worst-case estimate \( \beta^j_i = 2^{-m+1} \) into (14), we obtain

\[
L_{\text{ch}}^{(0)} = \log \left( \frac{2(1 - 2^{-m}) - \epsilon}{\epsilon} \right), \quad (15)
\]

which is exactly the channel LLR for the marginal BSC with crossover probability \( \epsilon_{\text{BSC}} \) given in (1).

Notice that the decoder iterations are exclusively between VN (14) and CN computations, like in the binary case. However, computing (14) at the VNs requires the extrinsic information (the CN messages) for all bits within a symbol; this might be considered an additional level of message exchanges (specifically, plain copying of messages), but it does not involve iterations of any kind. The complexity increase compared to binary LDPC decoding is on the order of at most \( m \) operations per variable node, depending on the scheduling. In fact, the marginalization (12) is reminiscent of a combined detector and VN decoder for binary LDPC codes that are directly mapped to larger signal constellations [10]. Thanks to the symmetry of the q-SC, here it is not necessary to actually sum over all \( q \) symbol values.

### B. q-SC Front-End Approach

The similarity of the bit-APP approach to combined detection and decoding mentioned above points to a different view on the bit-symmetric coding scheme, which is to consider a q-SC front-end for a binary LDPC decoder, similar to approaches for iterative demapping and decoding [11], [12]. These techniques involve proper iterations between the demapper and the LDPC decoder, being treated as separate functional blocks.

The q-SC front-end takes into account the correlation between the errors on the \( m \) bit layers. Its factor graph representation allows to formulate a message passing algorithm that computes essentially the same quantities as the approach in Sec. III-A, but opens the way to EXIT analysis and displays more clearly the opportunity for further complexity reduction by appropriate message scheduling.

As before, let \( x \) denote the vector of q-ary channel input symbols \( x^j \) (\( j = 1, \ldots, n \)) and let \( x^j_i \) denote bit \( i \) of symbol \( j \). In the same way, the output of the channel is represented by \( y^j \) and \( y^j_i \), and the errors are denoted by \( e \), \( e^j \) and \( e^j_i \), respectively. We assume w.l.o.g. that \( e^j_i = x^j_i \oplus y^j_i \), where \( \oplus \) denotes addition in GF(2), and \( e^j = x^j \oplus y^j \), by extension.
of the transmitted codeword. In the following we will describe and all bits $i$
f by characteristic function $\mu$ and vice versa as 

$$f_{X \mid Y}(x \mid y) \equiv \chi_C(x) \prod_{j=1}^{n} f_{Y \mid x}(y \mid x^{j})$$

where we used the fact that $x^{j}$, $e^{j}$ and $y^{j}$ are related in a deterministic way.

Let $\chi_C(x)$ be the characteristic function of the code, which evaluates to one if $x$ is a codeword and to zero otherwise. Furthermore assuming that the transmitted codewords are equally likely, the a posteriori probability satisfies the proportionality relation 

$$f_{X \mid Y}(x \mid y) = \chi_C(x) \prod_{j=1}^{n} f_{Y \mid x}(y \mid x^{j})$$

where $\chi_C(x)$ is the characteristic function of the code.

The errors on the channels are assumed to be memoryless (on the symbol level), leading to a factorization of $f_{Y \mid x}$ as

$$f_{X \mid Y}(x \mid y) = \chi_C(x) \prod_{j=1}^{n} f_{Y \mid x}(y \mid x^{j})$$

where we used the fact that $x^{j}$, $e^{j}$ and $y^{j}$ are related in a deterministic way.

Given an error symbol $e^{j}$, the bit-layers are independent of each other, leading to

$$f_{X \mid Y}(x \mid y) = \chi_C(x) \prod_{j=1}^{n} f_{Y \mid x}(y \mid x^{j}, e^{j})$$

where $f_{X \mid Y}(x \mid y) = f_{Y \mid x}(y \mid x^{j}, e^{j})$.

This factorization is shown for one symbol $x^{j}$ in Fig. 3, where the edges on the right side are connected to the check nodes of the LDPC code, i.e. the factorization of the characteristic function $\chi_C(x)$. In the following, we will denote a message sent from a variable node $x$ to a function node $f$ and vice versa as $\mu_{x \rightarrow f}(x)$ and $\mu_{f \rightarrow x}(x)$, respectively [13].

1) Message Passing Rules: We apply the sum-product algorithm to compute the marginals $f_{X \mid Y}(x^{j} \mid y)$ for all symbols $j$ and all bits $i$, i.e. to perform a posteriori decoding for every bit of the transmitted codeword. In the following we will describe the operations for a single $q$-ary symbol and therefore omit the symbol index $j$ for convenience.

First, we define the local functions and derive the message passing rules for the function nodes $f_{ch}$ and $f_{E}$. Since the variable nodes $e_i$ have degree two, they just forward the incoming messages, e.g. $\mu_{e_i \rightarrow f_{ch}}(e_i) = \mu_{f_{E} \rightarrow e_i}(e_i)$ and $\mu_{e_i \rightarrow f_{E}}(e_i) = \mu_{f_{ch} \rightarrow e_i}(e_i)$.

a) Bit Layer Channels: In the binary sub-channels, $x_i$, $y_i$, and $e_i$ have to sum up to zero in GF(2), therefore the local functions $f_{ch}$ are given by 

$$f_{ch}(y_i \mid x_i, e_i) = [y_i = x_i \oplus e_i],$$

where $[P]$ evaluates to 1 if the expression $P$ is true and to 0 otherwise.

Having defined the local function, we can derive the message passing algorithm by applying the rules of the sum-product algorithm. Given a message $\mu_{e_i \rightarrow f_{ch}}(e_i)$ and a received bit $y_i$, the channel function node computes

$$\mu_{f_{ch} \rightarrow e_i}(e_i) = \sum_{\sim\{e_i\}} (f_{ch}(y_i \mid x_i, e_i) \cdot \mu_{x_i \rightarrow f_{ch}}(x_i))$$

where $\sum_{\sim\{a\}}$ denotes the marginalization over all variables except $a$. In the same way, the message sent back to variable node $x_i$ is computed as

$$\mu_{f_{ch} \rightarrow x_i}(x_i) = \sum_{\sim\{x_i\}} (f_{ch}(y_i \mid x_i, e_i) \cdot \mu_{e_i \rightarrow f_{ch}}(e_i))$$

b) Error Patterns: The function node $f_{E}$ represents the probability that a certain binary error pattern occurs. In the $q$-ary symmetric channel, every error pattern has the same probability $\frac{1}{q^m}$, except the all-zero pattern that has probability $1 - e$, thus

$$f_{E}(e_1, \ldots, e_m) = \begin{cases} 1 - e, & \text{if } e_1 = \ldots = e_m = 0 \\ \frac{e}{q-1}, & \text{otherwise.} \end{cases}$$

The derivation of the outgoing messages of $f_{E}$ leads to

$$\mu_{f_{E} \rightarrow e_i}(e_i) = \sum_{\sim\{e_i\}} (f_{E}(e_1, \ldots, e_m) \cdot \prod_{i' \neq i} \mu_{e_{i'} \rightarrow f_{E}}(e_{i'}))$$

$$= \left\{ \frac{1 - p}{q-1} \cdot \beta_{i} + \frac{1}{q-1} (1 - \beta_{i}) \right\}$$

if $e_i = 0$

$$= \frac{e}{q-1}$$

if $e_i = 1$, 

where $\beta_{i}$ is defined as

$$\beta_{i} = \prod_{i' \neq i} \mu_{e_{i'} \rightarrow f_{E}}(e_{i'})$$

Appendix A outlines the simplification of the different messages using scalar quantities and highlights the fact that these are essentially the same quantities as in the bit-level APP approach in Sec. III-A.
2) Complexity: Clearly, the processing complexity of the front-end is dominated by (16) and is thus \(O(\log q)\) per symbol, as in verification decoding [1]. This means that the overall decoding complexity scales linearly in the number of transmitted bits, independently of the symbol alphabet size \(q\). Complexity may be further reduced by a constant factor (i.e. not its order) if the front-end messages are not recomputed on every LDPC decoder iteration; this is often sufficient in practice.

C. EXIT Analysis of the \(q\)-SC Front-end

In this section, we analyze the \(q\)-SC front-end using extrinsic information transfer (EXIT) charts [14], which will also be used later for the design (optimization) of LDPC codes.

Let the a priori and extrinsic messages for the front-end denote the messages between the bit layer sub-channels \(I_{ch}\) and the bit nodes \(x_i\) of the LDPC code, that is, \(\mu_{a \rightarrow x}(x_i)\) and \(\mu_{x_i \rightarrow a}(x_i)\), respectively. For the EXIT chart analysis, the front-end is characterized by the transfer function \(I_e(I_a)\), where \(I_a\) and \(I_e\) denote the a priori and the extrinsic mutual information, respectively, between the messages and the corresponding bits.

First, we derive the EXIT function of the front-end for the case when the a priori messages are modeled as coming from a binary erasure channel (BEC). In this special case, the EXIT functions have important properties (e.g. the area-property), which were shown in [14].

In the case of a BEC, the variables \(x_i\) are either known to be error-free or erased, thus simplifying the computation of \(\beta_q\) defined in (16) as follows. The product in (16) has \(m - 1\) factors, taking on the value 1/2 if the bit \(x_i\) is erased, 0 if the bit \(x_i\) is not in agreement with the received bit \(y_i\) and 1 otherwise. If at least one message is not in agreement with the received bit, a symbol error occurred and \(\beta_q = 0\). Otherwise, \(\beta_q = 2^{-t}\), where \(t\) denotes the number of erased messages.

Since \(\beta_q = 0\) corresponds to \(\epsilon_t = 1/2\) through (22) in Appendix A. this case is equivalently described by an erasure. Therefore, the extrinsic messages can be modeled as being transmitted over a binary symmetric erasure channel (BSEC), with parameters \(\epsilon_{m-t}\) and \(\delta_{m-t}\) given by (2) and (3), respectively, since \(t\) erasures correspond to the situation in layer \(i = m - t\) in the layered scheme of Section II. Its capacity is thus

\[
I_t = (1 - \delta_{m-t}) \left(1 - h\left(\frac{\epsilon_{m-t}}{1 - \delta_{m-t}}\right)\right). \tag{17}
\]

In order to compute the extrinsic information \(I_e(I_a)\) at the output of the front-end as

\[
I_e(I_a) = \sum_{t=0}^{m-1} I_t \lambda_t(I_a), \tag{18}
\]

we have to compute the probability \(\lambda_t(I_a)\) that \(t\) messages are erased given a priori mutual information \(I_a\). Let \(\Delta_a = 1 - I_a\) denote the probability that an a priori message is erased. The probability that \(t = 0, \ldots, m - 1\) out of \(m - 1\) messages are erased is computed as

\[
\lambda_t(I_a) = \binom{m - 1}{t} \Delta_a^t (1 - \Delta_a)^{m-1-t} = \binom{m - 1}{t} (1 - I_a)^t I_a^{m-1-t}. \tag{19}
\]

The EXIT function for BEC a priori messages can therefore be computed using (2), (3), (17) and (19) in (18).

Theorem 2 An iterative decoder for the bit-symmetric scheme using a \(q\)-SC front-end can attain \(q\)-SC capacity.

**Proof:** According to the area theorem for EXIT charts [14], the capacity of each bit layer is given by the area below the EXIT function, if the a priori messages are modeled as coming from a BEC. Therefore, the capacity of the overall channel is given by

\[
C_{\text{total}} = m \int_0^1 \sum_{t=0}^{m-1} I_t \lambda_t(I_a) \, dI_a
\]

where we used the fact that the integral corresponds to the definition of the beta function. The last sum in (20) is indeed

\[
\sum_{t=0}^{m-1} I_t = C_{q-SC}, \tag{20}
\]
equal to the capacity of the $q$-SC, as was already shown in the proof of Theorem 1.

Without iterative processing, the maximum achievable rate is given by $I_a(0)$. In that case, all a priori messages are erased and one has

$$I_{m-1} = 1 - h(\epsilon_1) = 1 - h\left(\frac{q}{2(q-1)}\right) = C_{\text{BSC}},$$

which corresponds to the simplistic coding approach where the $q$-SC is decomposed into $m$ BSCs.

Fig. 4 shows an EXIT chart for BEC a priori messages. If the $q$-SC would be decomposed into BSCs (without iterating between the front-end and the LDPC code and without using the layered scheme) then the value at $I_a = 0$ corresponds to the capacity of these BSCs. In contrast, the area below the EXIT functions corresponds to the normalized capacity of the $q$-SC.

When using this front-end with an LDPC code, the a priori messages can not be modeled as coming from a BEC. For code design, we will make the approximation that the messages are Gaussian distributed, which is a common assumption when using EXIT charts. The simulated EXIT function using Gaussian a priori messages is also shown in Fig. 4.

IV. LDPC DESIGN AND CONSTRUCTION

After obtaining the EXIT function of the $q$-SC front-end, we can design an LDPC code. Code design is an optimization problem that selects the degree distributions of the code in order to maximize the rate, under the constraint that the EXIT function of the overall LDPC code does not intersect the EXIT function of the $q$-SC front-end.

Joint optimization of the variable and check node degree distributions is a nonlinear problem, but it was shown in [12] that the optimization of the check node degree distribution given a fixed variable node degree distribution is a linear programming problem, which can be solved efficiently.

To reduce the optimization complexity and to obtain practical degree distributions, we choose three non-zero variable node degrees (compare e.g., the approach in [15]). This allows to perform an exhaustive search over the variable node degree distribution, and for each variable node degree distribution we optimize the check node degree distribution as described in [12] with a maximum check node degree $d_{c,\text{max}} = 50$.

Fig. 5 shows the normalized capacity of the $q$-SC and the obtained optimized code rates (under the Gaussian approximation) versus the error probability $\epsilon$ of the $q$-SC for $m = 4$ and $m = 8$. Using this code optimization, we are able to design codes that perform close to capacity over a wide range of error probabilities for moderate $q$.

For the actual code construction, we used a modified version of the PEG algorithm [16], [17], which allows us to construct codes with a specific variable and check node degree distribution. To avoid (short) cycles in the overall factor graph, one has to ensure that the bits of a given $q$-ary symbol do not participate in the same parity check. This can be achieved by using an appropriate interleaver between the $q$-SC front-end and the LDPC decoder. Another way would be to use the multi-edge type framework for LDPC code where such constraints can be explicitly specified [9, Chap. 7].

To verify our derivations, we performed bit error rate simulations for an optimized binary LDPC code of rate $R = 1/2$ and length $N = 12000$, shown in Fig. 6. This code is optimized for a $q$-SC with $m = 4$, thus $N = 12000$ bits correspond to $k = 1500$ $q$-ary symbols. It has a decoding threshold of $\epsilon = 0.28$ (the Shannon limit for $R = 1/2$ is at $\epsilon = 0.29$). As a comparison we also show the bit error rate for a regular LDPC code with $d_v = 3$ and $d_c = 6$ which has a decoding threshold of $\epsilon = 0.25$.

V. GENERALIZATION TO CONDITIONALLY INDEPENDENT BINARY SUB-CHANNELS

The decoder in Section III is seen to easily extend to $q$-ary channels with modulo-additive noise, where the $m$ bits of the noise (error pattern) are independent if an error occurs, i.e.
\( L_{\text{app}}(X_i^j = y_i^j) = \log \left( \frac{(1 - \epsilon)^{\beta_j} + \alpha(1 - \epsilon_{i^*}) \prod_{k \neq i^*} \left( \epsilon_{k^*} + p_k^j - 2 \epsilon_{k^*} p_k^j \right) - \prod_{k \neq i^*} (1 - \epsilon_{k^*}) p_k^j}{\alpha \epsilon \prod_{k \neq i^*} \left( \epsilon_{k^*} + p_k^j - 2 \epsilon_{k^*} p_k^j \right)} \right) \)

\[ \begin{align*}
&= \log \left( \frac{(1 - \epsilon)^{\beta_j} + \alpha(1 - \epsilon_{i^*}) \prod_{k \neq i^*} \left( \epsilon_{k^*} + p_k^j - 2 \epsilon_{k^*} p_k^j \right) + \log \left( \frac{1}{1 - p_i^j} \right)}{\alpha \epsilon \prod_{k \neq i^*} \left( \epsilon_{k^*} + p_k^j - 2 \epsilon_{k^*} p_k^j \right)} \right) + L_{\text{ext}}(X_i^j = y_i^j). \\
&= \log \left( \frac{1 - \epsilon_{i^*}}{\alpha \epsilon_{i^*}} \right) + \alpha \epsilon_{i^*} \prod_{k \neq i^*} \left( \epsilon_{k^*} + p_k^j - 2 \epsilon_{k^*} p_k^j \right) + L_{\text{ch}}(X_i^j = y_i^j),
\end{align*} \tag{21} \]

at least one bit is nonzero. This assumption leads to define the \( q \)-ary symmetric channel with conditionally independent binary sub-channels (\( q \)-SC\textsuperscript{*}), with transition probabilities

\[ P_Y(x|y) = \begin{cases} 1 - \epsilon, & y = x, \\ \alpha \prod_{i=1}^{m} (1 - \epsilon_{i^*}) (1 - \epsilon_{i^*}) & y \neq x, \end{cases} \]

where \( \alpha = \frac{1}{1 - \prod_{i=1}^{m} (1 - \epsilon_{i^*})} \)

and \( \oplus \) denotes GF(2) addition (this definition encompasses all channels satisfying the above conditional independence assumption, from which it may be derived axiomatically). The defining parameters are the symbol error probability \( \epsilon \) and the \( m \) conditional probabilities \( \epsilon_{i^*} \), which are the bit error probabilities conditioned on the event that at least one of the other \( m - 1 \) bits is in error. The marginal bit error probabilities will be \( \epsilon_{\text{BSC}, i} = \alpha \epsilon_{i^*} \). Since the \( q \)-SC\textsuperscript{*} is strongly symmetric, the uniform input distribution achieves its capacity

\[ C_{q\text{-SC}^*} = m - \alpha \sum_{i=1}^{m} h(\epsilon_{i^*}) + (1 - \epsilon) \log_2(1 - \epsilon) \]

\[ + \alpha \log_2 \alpha - (\alpha - \epsilon) \log_2(\alpha - \epsilon). \]

When \( \epsilon \) is such that \( \alpha = 1 \), the \( q \)-SC\textsuperscript{*} reduces to \( m \) independent BSCs, while for \( \epsilon_{i^*} = \frac{1}{2}, \ i = 1, \ldots, m \), it becomes an ordinary \( q \)-SC.

The unnormalized bit APP (compare (13)) is obtained as

\[ P(X_i^j = x_i^j | Y = y) = \begin{cases} (1 - \epsilon) \prod_{k=1}^{n} p_k^i & \\
\alpha(1 - \epsilon_{i^*}) \prod_{k \neq i^*} \left( \epsilon_{k^*} + p_k^j - 2 \epsilon_{k^*} p_k^j \right) - \prod_{k \neq i^*} (1 - \epsilon_{k^*}) p_k^j, & x_i^j = y_i^j, \\
\alpha \epsilon_{i^*} \prod_{k \neq i^*} \left( \epsilon_{k^*} + p_k^j - 2 \epsilon_{k^*} p_k^j \right), & x_i^j \neq y_i^j, \end{cases} \]

leading to the LLR expression (21). The quantities \( p_k^i \), \( \beta_k^i \) and \( \beta_k^{i^*} \) are as defined in Section III. In the special case of the \( q \)-SC, for \( \beta_k^{i^*} \rightarrow 0 \) the channel LLR \( L_{\text{ch}} \) tends to a constant, which is equal to the LLR of the binary sub-channel conditioned on the occurrence of an error on one or more of the other sub-channels. The initial value of \( L_{\text{ch}} \) can again be obtained by assuming a worst-case of \( p_k^j = \frac{1}{2}, \beta_k^{i^*} = 2^{-m+1} \) in (21), yielding

\[ L_{\text{ch}}(X_i^j = y_i^j) = \log \left( \frac{1 - \epsilon_{i^*}}{\alpha \epsilon_{i^*}} \right) + \alpha \epsilon_{i^*} \prod_{k \neq i^*} \left( \epsilon_{k^*} + p_k^j - 2 \epsilon_{k^*} p_k^j \right) + L_{\text{ext}}(X_i^j = y_i^j), \]

which is the channel LLR of the \( i \)-th marginal BSC.

**APPENDIX**

**A. Simplification of the Messages**

Since all involved variables are binary, we can represent the messages by scalar quantities. For the messages from and to the variable nodes \( x_i^j \) of the LDPC code, we use log-likelihood ratios of the corresponding messages

\[ L_{a,i} = \log \frac{\mu_{x_i^j \rightarrow f_a}(0)}{\mu_{x_i^j \rightarrow f_a}(1)}, \]

\[ L_{ch,i} = \log \frac{\mu_{f_a \rightarrow x_i^j}(0)}{\mu_{f_a \rightarrow x_i^j}(1)}. \]

The connection with the bit-level APP approach is made by equating \( \mu_{x_i^j \rightarrow f_a}(0) = P(X_i^j = 0 | Y = y) = y^{(j)}(b) \) in the expression for \( L_{a,i} \); the correspondence for \( L_{ch,i} \) will be given below.

The messages from the function nodes \( f_a \) to the node \( f_E \) are defined as the probability of no error

\[ p_i = \mu_{f_a \rightarrow e_i}(0) = \mu_{e_i \rightarrow f_a}(0), \]

and the messages from the function node \( f_E \) are defined as the error probability

\[ \epsilon_i = \frac{\mu_{f_E \rightarrow e_i}(1)}{\mu_{f_E \rightarrow e_i}(0) + \mu_{f_E \rightarrow e_i}(1)}. \]

Using these definitions, we can reformulate the computation rules at the function nodes as

\[ p_i = \frac{e^{|b_i|}}{e^{|b_i|} + e^{-|b_i|}}, \]

which corresponds to (11), and

\[ \epsilon_i = \frac{(1 - \epsilon) \cdot \beta_{i|} + \frac{\beta_{i|}}{\epsilon}}{(1 - \beta_{i|}) + \frac{\beta_{i|}}{\epsilon}} = \frac{2 \epsilon + \beta_{i|}(q - \epsilon q - 1)}{q - 1} \tag{22} \]
For the initialization, we set all a-priori L-values $L_{a,i}$ to zero, leading to $\beta_i = 2^{-(m-1)} = 2/q$. Inserting this in (22) leads to

$$\epsilon_i = \frac{eq}{2(q-1)} = \epsilon_{BSC}.$$ 

Finally, the messages sent from the function nodes $f_{ch}$ to the variable nodes $x_i$ are computed as

$$L_{ch,i} = (1 - 2y_i) \cdot \log \frac{1 - \epsilon_i}{\epsilon_i}.$$

Using (22), the logarithmic term in (23) can be expanded as

$$\log \frac{1 - \epsilon_i}{\epsilon_i} = \log \left(2 + \frac{q - qe - 1}{\epsilon} \cdot \beta_i - 1\right),$$

which is equivalent to $L_{ch}(X^j_i = y^j_i)$ defined in (14) in the bit-level APP approach.

ACKNOWLEDGMENTS

The authors would like to thank Jossy Sayir for helpful discussions and for pointing out the similarity with iterative demapping and decoding.

REFERENCES


