IPN Progress Report 42-194 • August 15, 2013

Decoding Complexity and Performance of Short-Block LDPC Codes Over *GF*(*q*)

Bruce Moision*

ABSTRACT. — We examine the power efficiency and decoding complexity of short-block, low-density parity-check (LDPC) codes being considered for inclusion in a deep-space telecommand standard. The codes have rate 1/2, blocklengths $k \in \{64, 128, 256\}$, and operate over Galois fields (GF) of order $q \in \{2, 16, 256\}$. We show that codes over GF(16) are ≈ 4 times as complex to decode in return for a 0.6 dB gain in performance and codes over GF(256) are ≈ 44 times as complex to decode in return for a ≈ 1.0 dB gain in performance, all relative to codes over GF(2) at the same blocklength. This provides a quantitative trade-off in selecting a class of codes for a standard.

I. Introduction

The power efficiency of error-correction codes increases with their blocklength. However, certain channels are constrained to use short-block codes due to latency or memory constraints. For example, a deep-space communications uplink channel, transmitting at relatively low data rates and with short packets, may be constrained to have blocklengths on the order of hundreds of bits. Among short-block codes, low-density parity-check (LDPC) codes are very power efficient and have been proposed for this application, see, e.g., [1]. In this article, we examine a class of short-block LDPC codes proposed for this use.

LDPC codes are linear codes, and the collection of codewords belonging to a code are the vectors of symbols that satisfy a set of parity-check equations. The parity checks, and symbols, may be over a binary alphabet, or from a Galois field of order q, denoted GF(q), where q is typically chosen to be a power of 2. It has been shown that the power efficiency of LDPC codes of a given blocklength can be made to increase with q [2].

The commonly used decoding algorithm for an LDPC code is iterative, requiring a random number of iterations that is a function of the channel signal-to-noise ratio (SNR) and code design. We'll see that LDPC codes over GF(q) require on the order of $q \log_2(q) |\mathcal{E}|$ operations per decoding iteration, where $|\mathcal{E}|$ is the number of edges in a graph describing the code. A binary LDPC code requires on the order of $|\mathcal{E}|$ operations per iteration, and the

* Communications Architectures and Research Section.

The research described in this publication was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

average number of iterations at a specified error rate generally increases for more power efficient codes. The number of edges for a code over GF(q) typically decreases with q (for comparable performance and blocklength). Taken together, the complexity increases with q, and one trades performance for complexity when varying q. A quantitative comparison depends on the details of the code design. In this article, we quantify the performance/complexity trade-off for several short-blocklength codes that are candidates for a Consultative Committee for Space Data Systems (CCSDS) uplink standard.

The article is organized as follows. In Section II, we briefly describe the channel model; in Section III, we review a decoding algorithm for q-ary codes, treating the binary (q = 2) as a special case, and tabulate its complexity. In Section IV, we illustrate the simulated performance of the candidate codes. In Section V, we illustrate the performance/complexity trade-off for the codes. In Section VI, we draw some conclusions. Two Appendices contain derivations and descriptions of the decoding algorithms.

Our notation is as follows. We use $\mathbf{x}, \mathbf{y}$ to denote vectors and x, x_i to denote scalars. The collection of scalars with indices in a set $\mathcal{N}$ is denoted $\mathbf{x}_{\mathcal{N}} = \{x_i \mid i \in \mathcal{N}\}$. We use α, β to denote elements of GF(q). $p_{Y|X}(y_i \mid x_i)$ denotes the conditional density of random variable Y given $X = x_i$ evaluated at y_i. To simplify notation, we simply write this as $p(y_i \mid x_i)$. Uppercase $P(A)$ denotes the probability of the event A.

II. Channel Model

Figure 1 illustrates our channel model. An $(n,k)_q$ code is a collection of q^k vectors of length n with elements from GF(q). Given a mapping from GF(q) to $\log_2(q)$-vectors over GF(2) maps the $(n,k)_q$ code to an $(n',k')_2$ code. As our channel modulation is fundamentally binary, we represent the encoder and decoder with their binary equivalents.

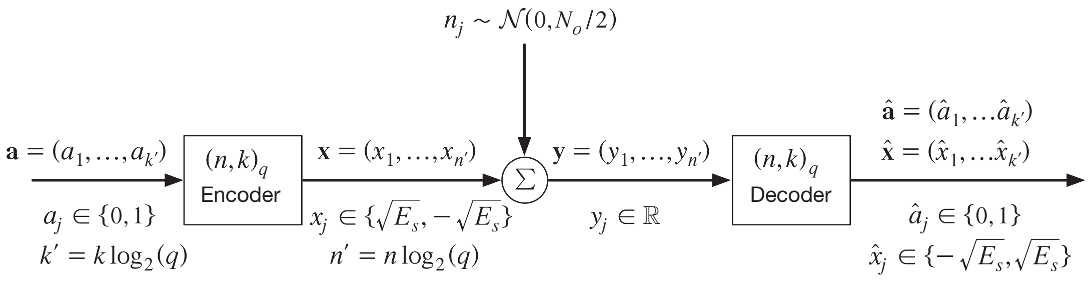
$n_j \sim \mathcal{N}(0, N_o/2)$
$\mathbf{a} = (a_1, \ldots, a_{k'})$
$a_j \in \{0,1\}$
$k' = k \log_2(q)$
$(n,k)_q$ Encoder
$\mathbf{x} = (x_1, \ldots, x_{n'})$
$x_j \in \{\sqrt{E_s}, -\sqrt{E_s}\}$
$n' = n \log_2(q)$
Σ
$\mathbf{y} = (y_1, \ldots, y_{n'})$
$y_j \in \mathbb{R}$
$(n,k)_q$ Decoder
$\hat{\mathbf{a}} = (\hat{a}_1, \ldots \hat{a}_{k'})$
$\hat{\mathbf{x}} = (\hat{x}_1, \ldots \hat{x}_{k'})$
$\hat{a}_j \in \{0,1\}$
$\hat{x}_j \in \{-\sqrt{E_s}, \sqrt{E_s}\}$

Figure 1. Block diagram of LDPC-encoded additive white Gaussian noise (AWGN) binary phase-shift keyed (BPSK) channel model.

A block of $k' = k\log_2(q)$ bits $\mathbf{a} = (a_1, \ldots, a_{k'})$ are mapped by an $(n,k)_q$ LDPC encoder to a block of $n' = n\log_2(q)$ binary symbols $\mathbf{x} = (x_1, \ldots, x_{n'})$, with each $x_i \in \{\sqrt{E_s}, -\sqrt{E_s}\}$, where E_s denotes the energy of the transmitted symbol. Independent Gaussian noise samples n_j with mean 0 and variance $\sigma^2 = N_0/2$ are added to each symbol, such that we receive the vector $\mathbf{y} = (y_1, \ldots, y_{n'})$ with $y_j = x_j + n_j$. The decoder maps $\mathbf{y}$ to estimates $\mathbf{a} = (\hat{a}_1, \ldots, \hat{a}_{k'})$ and $\mathbf{x} = (\hat{x}_1, \ldots, \hat{x}_{n'})$ of the information and coded sequences, respectively. A decoding word error occurs if $\hat{\mathbf{x}} \neq \mathbf{x}$. We characterize the performance of the code by determining its word error rate $P_\omega = P(\hat{\mathbf{x}} \neq \mathbf{x})$ as a function of the SNR E_b/N_0, where $E_b = E_s/(k/n)$ is the energy per information bit. In this article, we focus on the complexity of the decoding algorithm and its performance, and do not address the encoding algorithms.

III. Decoding Algorithms

Decoding algorithms for LDPC codes over binary and higher order fields are well known [3,4,5]. We repeat them briefly here for completeness of presentation, and to establish assumptions in tabulating decoding complexity. As a measure of complexity, we determine the average number of binary operations performed per decoded information bit (we exclude certain operations such as bit shifts that are considered to have negligible cost, as described later). This does not take into account the fact that not all binary operations have the same complexity, e.g., an operation stored as a precomputed table lookup may not have the same complexity as an addition. It also does not delve into the details of a hardware implementation of the algorithms, which may take advantage of certain structures to reduce complexity. Nonetheless, it is sufficient for our purpose, which is to compare the complexity of essentially similar algorithms.

The decoding algorithms are instances of solving a marginalize-product-of-functions (MPF) problem by use of the sum-product algorithm (SPA) [6]. We briefly review the SPA and its application to decoding parity-check codes. Let $\mathbf{x} = (x_1, \ldots, x_n)$, an element of a domain S, $g(\mathbf{x})$ a global function acting on $\mathbf{x}$, and

$$g_i(\alpha) = \sum_{\mathbf{x} \in S | x_i = \alpha} g(\mathbf{x}) \tag{1}$$

the ith marginal function of g. Suppose the global function factors as

$$g(\mathbf{x}) = \prod_{j \in J} f_j(\mathbf{x}_{\mathcal{N}(j)}), \tag{2}$$

where $\mathbf{x}_{\mathcal{N}(j)} \subseteq \{x_1, \ldots, x_n\}$ is the argument of f_j. The SPA provides an efficient way to compute (or estimate) the marginals in a way that exploits the factorization of the global function to efficiently avoid repetitive computation of intermediate values. The SPA can be visualized as acting on a factor graph that represents Equation (2) with a variable node for each x_i, a factor node for each f_j and an edge connecting x_i to f_j if x_i is an argument of f_j, in which case we say x_i and f_j are neighbors. The algorithm proceeds by passing messages between nodes in the graph as follows.

Let $\mu_{x_i \to f_j}$ be the message transmitted from variable node x_i to function node f_j and $\mu_{f_j \to x_i}$ the message sent in the reverse direction. Let $\mathcal{N}(j)$ denote the neighbors of f_j, and $\mathcal{M}(i)$ denote the neighbors of x_i. One message is sent for each realization of x_i (simplifications are used when x_i is binary-valued, as described in Section III.B). The message computations in the SPA may be expressed as follows:

$$\mu_{x_i \to f_j}(\alpha) = \prod_{l \in \mathcal{M}(i) \setminus j} \mu_{f_l \to x_i}(\alpha)$$

$$\mu_{f_j \to x_i}(\alpha) = \sum_{\mathbf{x}_{\mathcal{N}(j)} \mid x_i = \alpha} f(\mathbf{x}_{\mathcal{N}(j)}) \prod_{l \in \mathcal{N}(j) \setminus i} \mu_{x_l \to f_j}(x_l).$$

A cycle on the factor graph is a path obtained by traversing edges that closes on itself. The SPA solves the MPF problem exactly when the underlying factor graph has no cycles — i.e, it finds all the marginal functions exactly. We'll see that maximum a posteriori (MAP) decoding a code may be cast as an MPF problem. In this case, the factor graph contains cycles. Nonetheless, the SPA provides an efficient approximation to MAP decoding.

A. Decoding a Linear Code

Let $\mathcal{C}$ be an n-symbol code and χ the characteristic function for $\mathcal{C}$, that is,

$$\chi(\mathbf{x}) = \begin{cases} 1 & , x \in \mathcal{C} \\ 0 & , \text{otherwise} \end{cases}$$

Suppose χ factors as

$$\chi(\mathbf{x}) = \prod_j f_j(\mathbf{x}_{\mathcal{N}(j)})$$

where each f_j is an indicator function on its arguments, evaluating to 1 if satisfied, 0 otherwise. A codeword $\mathbf{x}$ is transmitted over a memoryless channel $p(\mathbf{y} \mid \mathbf{x}) = \prod_i p(y_i \mid x_i)$, and we receive the noisy version $\mathbf{y}$. We desire to compute the probability, for each $i \in \{1, \ldots, n\}$ and $\alpha \in \mathbf{GF}(q)$,

$$P(x_i = \alpha, \mathbf{y}) = \sum_{\mathbf{x} \mid x_i = \alpha} p(\mathbf{x}) p(\mathbf{y} \mid \mathbf{x})$$

$$= \sum_{\mathbf{x} \mid x_i = \alpha} \frac{1}{|\mathcal{C}|} \chi(\mathbf{x}) \prod_{i=1}^{n} p(y_i \mid x_i)$$

$$= \sum_{\mathbf{x} \mid x_i = \alpha} \frac{1}{|\mathcal{C}|} \prod_j f_j(\mathbf{x}_{\mathcal{N}(j)}) \prod_{i=1}^{n} p(y_i \mid x_i).$$

We see that this is an instance of Equations (1) and (2), and hence may be solved, or approximated, with the SPA. A factor graph representing the global function has a variable node for each x_i, and function nodes for each parity check f_j and each channel likelihood $p(y_i \mid x_i)$.

When applying the SPA to decoding a linear code, we will use the shorthand

$$r_{ji} = \mu_{f_j \to x_i}$$
$$q_{ij} = \mu_{x_i \to f_j}$$

and let $\mathcal{N}(j)$ denote the neighbors of f_j, and $\mathcal{M}(i)$ denote the neighbors of x_i excluding the channel information $p(y_i | x_i)$. The SPA is implemented as follows:

1. Initialize

$$r_{ji}(\alpha) = 1 \quad \forall i, j \in \mathcal{M}(i)$$
$$\mu_{p(y_i|x_i) \to i}(\alpha) = p(y_i | x_i = \alpha)$$

2. Data to parity

$$q_{ij}(\alpha) = p(y_i | x_i = \alpha) \prod_{l \in \mathcal{M}(i) \setminus j} r_{li}(\alpha) \tag{3}$$

3. Parity to data

$$r_{ji}(\alpha) = \sum_{\mathbf{x}_{\mathcal{N}(j)} | x_i = \alpha} f_j(\mathbf{x}_{\mathcal{N}(j)}) \prod_{l \in \mathcal{N}(j) \setminus i} q_{lj}(x_l) \tag{4}$$

4. Parity check

$$\hat{x}_i = \max_{\alpha} p(y_i | x_i = \alpha) \prod_{l \in \mathcal{M}(i)} r_{li}(\alpha) \tag{5}$$

terminate if $\chi(\hat{\mathbf{x}}) = 1$, else repeat from step 2.

Each step is evaluated for each $\alpha \in \mathbf{GF}(q)$. Note that in Equation (5), the estimates are invariant to a scaling of the collection of messages $\{r_{lj}(\alpha), \alpha \in \mathbf{GF}(q)\}$ by a non-negative multiplicative constant. It follows that scaling each message $q_{ij}(\alpha)$ in Equation (3) by a multiplicative constant does not change the outcome of the algorithm. Hence, we may, without loss of generality, assume $\sum_\alpha q_{ij}(\alpha) = 1$. Under this assumption, we may interpret the q_{ij} as assigning probabilities $q_{ij}(\alpha) = P(x_i = \alpha)$, in which case the parity-to-data message represents $r_{ji}(\alpha) = P(f_j | x_i = \alpha)$, where $P(f_j)$ denotes the probability that parity check f_j is satisfied. This is a convenient intuitive interpretation.

B. Codes Over GF(2)

When the code is over the binary field, the algorithm may be simplified by passing messages corresponding to log-likelihood ratios (LLRs). This simplification is described in Appendix A. The resulting algorithm is as follows.

1. Initialize

$$L(r_{ji}) = 0$$
$$L(y_i) = \log\left(\frac{p(y_i \mid x_i = 0)}{p(y_i \mid x_i = 1)}\right)$$

2. Data to parity

$$L(q_{ij}) = L(y_i) + \sum_{l \in \mathcal{M}(i) \backslash j} L(r_{li}) \tag{6}$$

3. Parity to data

$$L(r_{ji}) = \prod_{k \in \mathcal{N}(j) \backslash i} \operatorname{sgn}(L(q_{kj}))\phi^{-1}\left(\sum_{k \in \mathcal{N}(j) \backslash i} \phi(|L(q_{kj})|)\right) \tag{7}$$

4. Parity check

$$\hat{x}_i = \begin{cases} 1 & , L(y_i) + \sum_{l \in \mathcal{M}(i)} L(r_{li}) > 0 \\ 0 & , \text{otherwise} \end{cases}$$

terminate if $\chi(\hat{\mathbf{x}}) = 1$, else repeat from step 2.

Let $\mathcal{E}$ denote the collection of edges between variable nodes and check nodes (excluding channel likelihood nodes). Table 1 summarizes the decoding complexity of the GF(2) algorithm. We assess no cost to evaluating the parity check used as a stopping rule, as this duplicates computations in the data-to-parity step. We also assess no cost to multiplications by a power of two, that require a simple bit-shift, or to evaluating the product of the sign bits in the parity-to-data binary step.

Table 1. GF(2) decoding algorithm complexity, operations per iteration.

Step	Operation Count	
	Table Lookup	+
(6) Data to parity		$2\lvert\mathcal{E}\rvert$
(7) Parity to data	$2\lvert\mathcal{E}\rvert$	$2\lvert\mathcal{E}\rvert - (n-k)$
	$2\lvert\mathcal{E}\rvert$	$4\lvert\mathcal{E}\rvert - (n-k)$

C. Codes Over GF(*q*)

The application of the SPA to decoding a code over $\mathrm{GF}(q)$ was described in [2]. However, this straightforward application had complexity that grows as $O(q^2)$. An alternative algorithm, based on using a fast-Hadamard transform (FHT), was later developed, see, e.g., [4], reducing the complexity to $O(q\log(q))$. More recent developments in [5] allow a wider range of trade-offs in the performance/complexity space. As these do not gain in both performance and complexity over the FHT algorithm, we take the FHT algorithm as our baseline and use it to tabulate decoding complexity. Let $H_{jl} \in \mathbf{GF}(q)$ be the element of the jth row and lth column of the parity check matrix. The FHT algorithm is described in detail in Appendix B, and summarized as follows:

1. Initialize

$$\begin{aligned} &r_{ji}(\alpha) = 1 \\ &r_{p(y_i|x_i) \to i}(\alpha) = p(y_i | x_i = \alpha) \end{aligned}$$

2. Data to parity

$$q_{ij}(\alpha) = p(y_i | x_i = \alpha) \prod_{l \in \mathcal{M}(i) \backslash j} r_{li}(\alpha) \tag{8}$$

3. Permute

$$q'_{lj}(\alpha) = q_{lj}\left(H_{jl}^{-1} \alpha\right)$$

4. Transform

$$Q'_{lj}(\alpha) = \sum_{\beta \in GF(q)} q'_{lj}(\beta)(-1)^{\alpha \cdot \beta^T} \tag{9}$$

5. Parity to data

$$R'_{ji}(\beta) = \prod_{l \in \mathcal{N}(j) \backslash i} Q'_{lj}(\beta) \tag{10}$$

6. Transform

$$r'_{ji}(-\alpha) = \frac{1}{2^m} \sum_{\beta \in GF(q)} R'_{ji}(\beta)(-1)^{\alpha \cdot \beta^T} \tag{11}$$

7. Permute

$$r_{ji}(\alpha) = r'_{ji}(H_{ji} \alpha)$$

8. Parity check

$$\hat{x}_i = \max_{\alpha} p(y_i | x_i = \alpha) \prod_{l \in \mathcal{M}(i)} r_{li}(\alpha)$$

terminate if $\chi(\hat{\mathbf{x}}) = 1$, else 2.

Table 2 summarizes the decoding complexity of the FHT $\mathrm{GF}(q)$ algorithm. We assess no cost to the parity check or multiplications by a power of two that require a simple bit-shift. We see that the $\mathrm{GF}(q)$ algorithm costs on the order of $q\log_2(q)$ more operations per iteration than the binary algorithm.

Table 2. FHT GF(*q*) decoding algorithm complexity, operations per iteration.

Step	Operation Count	
	$\times$ and $\div$	+
(8) Data to parity	$2q\,\lvert\mathcal{E}\rvert$	
(9) Transform		$q(\log_2(q)-1)\lvert\mathcal{E}\rvert$
(10) Parity to data	$q(2\lvert\mathcal{E}\rvert-(n-k))$	
(11) Transform		$q(\log_2(q)-1)\lvert\mathcal{E}\rvert$
	$(4\lvert\mathcal{E}\rvert-(n-k))q$	$2\lvert\mathcal{E}\rvert q(\log_2(q)-1)$

IV. Performance

We consider nine candidate rate $k/n = 1/2$ short-block LDPC codes: the nine combinations of blocklengths $k' \in \{64, 128, 256\}$ and fields $q \in \{2, 16, 256\}$. The design and description of these codes is addressed in [7]. Figure 2 illustrates P_w as a function of E_b/N_0 for the codes and Table 3 summarizes the parameters for the codes relevant to our discussion. In all cases, decoding iterations were terminated after 50 iterations (when not terminated by the stop-

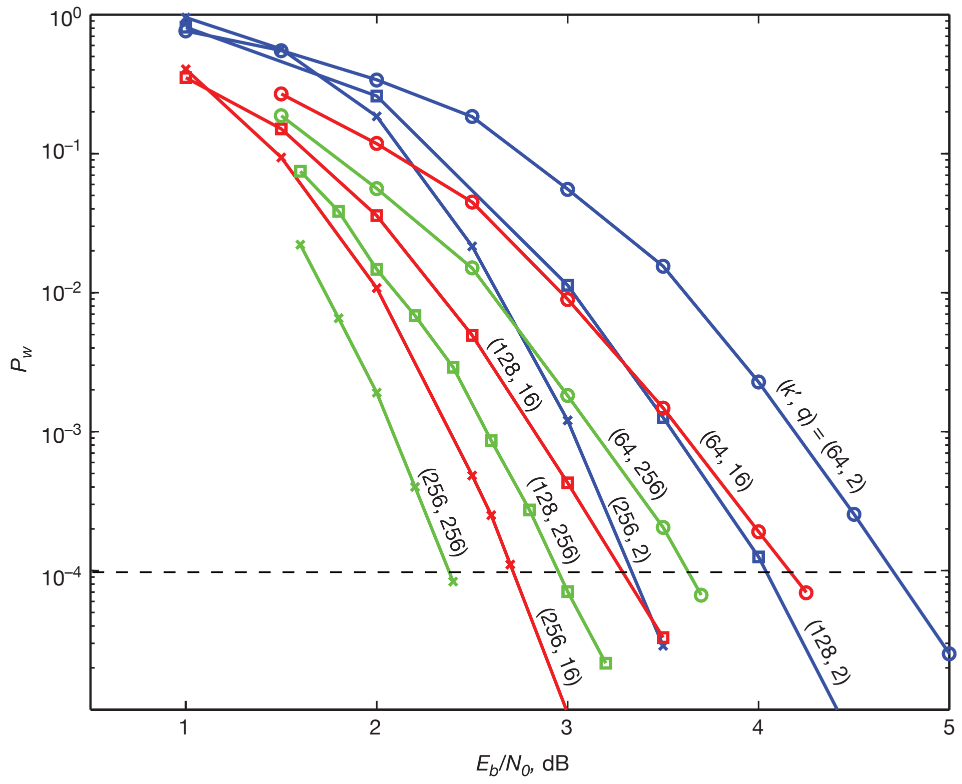

Figure 2. Word-error rate P_w versus E_b/N_0 for rate $k/n = 1/2$ codes with $k' \in \{64,128,256\}$ and $q \in \{2,16,256\}$. Each curve is labeled (k', q).

Table 3. Code comparisons: $|\mathcal{E}|$ = number of edges in factor graph of the code, $(E_b/N_0)^* = E_b/N_0$ @ $P_w = 10^{-4}$, Iterations = average number of decoding iterations at $(E_b/N_0)^*$, Operations/bit = average operations per bit at $(E_b/N_0)^*$.

q	$(n,k)_q$	(n',k')	$\lvert\mathcal{E}\rvert/k'$	$(E_b/N_0)^*$	Iterations	Operations/bit
2	(128,64)	·	8	4.7	1.8	85
2	(256,128)	·	8	4.0	2.7	127
2	(512,256)	·	8	3.3	4.1	193
16	(32,16)	(128,64)	1.25	4.1	1.9	372
16	(64,32)	(256,128)	1.125	3.3	2.7	475
16	(128,64)	(512,256)	1.1875	2.7	4.1	763
256	(16,8)	(128,64)	0.5	3.6	1.6	3635
256	(32,16)	(256,128)	0.5	2.9	2.6	5907
256	(64,32)	(512,256)	0.5	2.4	3.7	8406

ping rule). The value of E_b/N_0 at which the codes achieve $P_w = 10^{-4}$, which we take to be our target word-error rate (WER), is listed. Throughout, we refer to this as the code *threshold*, denoted $(E_b/N_0)^*$.

Figure 3 illustrates the code family performances with respect to the sphere-packing bound (SPB) [8]. We see that for a fixed blocklength, the codes become more power efficient with increasing q. For comparison, we also illustrate the thresholds for the current CCSDS

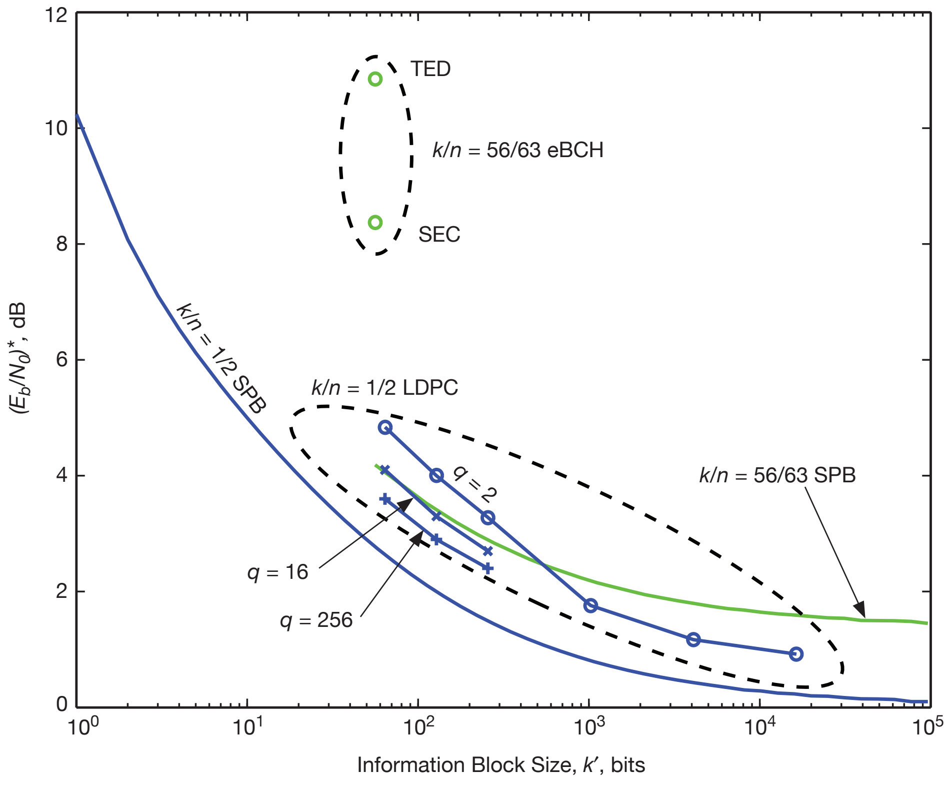

Figure 3. The sphere-packing bound on the minimum required E_b/N_0, in dB, to achieve $P_w = 10^{-4}$ for $k/n = 1/2$ and $k/n = 56/63$ codes, compared to the E_b/N_0 at which specific codes achieve $P_w = 10^{-4}$. Illustrated are the three classes of short-block LDPC codes considered in this article, as well as several CCSDS standards: the eBCH code in triple-error-detect (TED) and single-error-correct (SEC) modes, and the AR4JA LDPC codes of length k = 1024, 4096, and 16384.

standard short-blocklength code for uplink, an extended Bose–Chaudhuri–Hocquenghem (eBCH) code with $(n,k)_2 = (63,56)$ [9], and the thresholds for the CCSDS standard AR4JA rate = 1/2 binary LDPC codes of length $k' = 1024, 4096, 16384$ [10]. There are two thresholds for the eBCH code corresponding to two manners of decoding: single-error correction (SEC) and triple-error detection (TED). We see that large gains over the eBCH code are available at comparable blocklengths, ranging from ≈ 3 to > 9 dB, depending on the pair of codes and decoders selected. These gains come at a cost in complexity. We examine the complexity/performance trade-off in the next section.

V. Complexity/Performance Trade-Offs

Figure 4 illustrates the code threshold versus complexity, measured in operations/bit, grouping codes according to GF(q) (we also include the $k' = 1024$ AR4JA code for comparison). We see that for a given threshold, the complexity increases with order q. With no blocklength constraint, the codes over GF(2) are the most power efficient at a fixed complexity. As we see from the SPB in Figure 3, the binary AR4JA codes are within ≈ 1.0 dB of the SPB at blocklengths greater than ≈ 1024. For $k \geq 1024$, we expect the performance/complexity trade-off to favor binary codes.

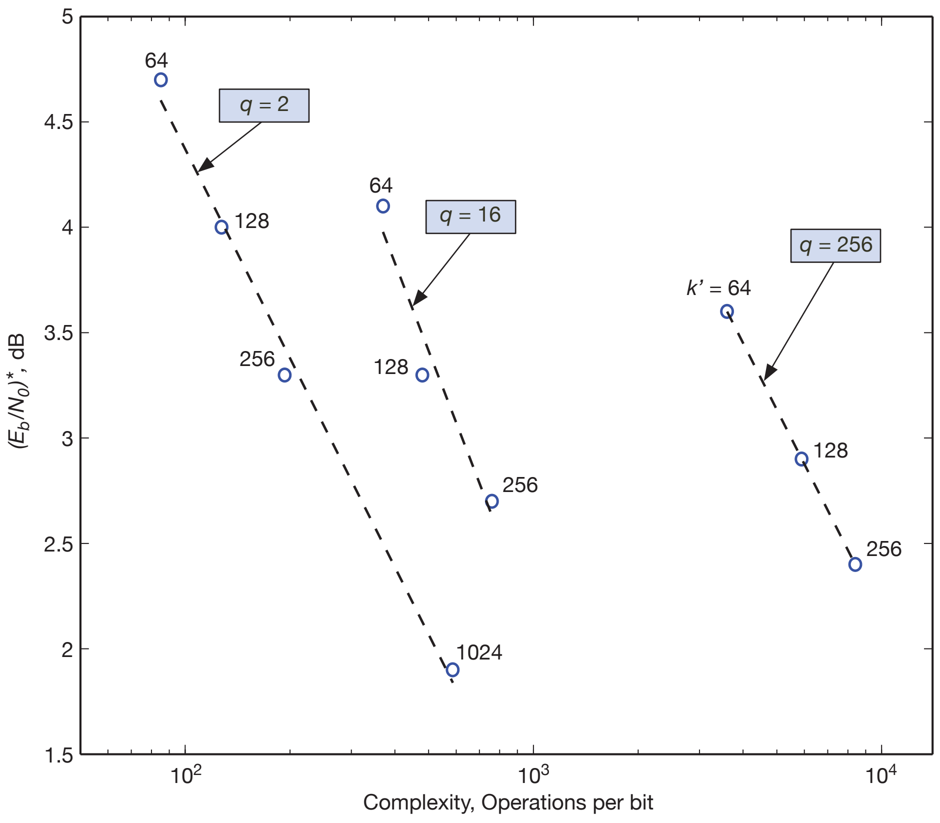

Figure 4. Complexity, measured as average operations/bit, versus decoding threshold in dB. All codes are $k/n = 1/2$, codes grouped by order q, and labeled with k'. A linear fit to each group is shown.

At short blocklengths, a linear fit in the log-log domain among each family GF(q) gives slopes of $\Delta\,\text{threshold, dB}/\Delta\,\text{ops./bit, dB} \approx \{-0.3, -0.4, -0.3\}$ for $q \in \{2, 16, 256\}$ in the plotted range. Hence, it costs roughly an increase in complexity by a factor of 2.0 for each 1 dB decrease in threshold. The threshold is bounded by 0.0 dB for any k, so we do not mean to imply this behavior extends indefinitely, we merely capture the trade-off at small k. This increase in complexity is due almost entirely to an increased number of iterations, which may be seen as the inevitable cost of approaching the SPB. Table 4 summarizes the performance/complexity trade-offs for the codes over GF(16) and GF(256) relative to binary codes. For these codes, the codes over GF(16) are ≈ 4 times as complex to decode and provide ≈ 0.6 dB gain relative to binary codes of the same blocklength. The codes over GF(256) are ≈ 44 times as complex to decode and provide ≈ 1.0 dB gain relative to binary codes of the same blocklength.

Table 4. Increase in complexity (multiplicative increase in operations/bit) and threshold gain (dB shift in threshold) at $P_w = 10^{-4}$ for GF(16) and GF(256) codes relative to GF(2) codes.

	GF(16)		GF(256)	
k'	Complexity Increase	Threshold Gain, dB	Complexity Increase	Threshold Gain, dB
64	4.4	0.6	43	1.0
128	3.7	0.7	47	1.1
256	4.0	0.6	44	0.9

VI. Conclusions

LDPC codes are promising candidates for applications with short blocklength constraints. The code thresholds may be improved by constructing codes over nonbinary fields, but at a cost in complexity. For a set of candidate codes with $k' \in \{64, 128, 256\}$, we see that codes over GF(16) are ≈ 4 times as complex to decode in return for a 0.6 dB gain in performance and codes over GF(256) are ≈ 44 times as complex to decode in return for ≈ 1.0 dB gain in performance, all relative to codes over GF(2) at the same blocklength. This provides a quantitative trade-off in selecting a class of codes for a standard.

Acknowledgments

Thanks to Dariush Divsalar for providing the code design and description, and to Kenneth Andrews for discussions on decoding algorithms and a careful review of the article.

References

[1] L. Costantini, B. Matuz, G. Liva, E. Paolini, and M. Chiani, "On the Performance of Moderate-Length Non-Binary LDPC Codes for Space Communications," *Proceedings of the IEEE Fifth Advanced Satellite Multimedia Systems Conference (ASMA) and 11th Signal Processing for Space Communications Workshop (SPSC)*, pp. 122–126, Cagliari, Italy, September 13–15, 2010.

[2] M. Davey and D. MacKay, "Low-Density Parity Check Codes Over GF(q)," *IEEE Communications Letters*, vol. 2, no. 6, pp. 165–167, June 1998.

[3] T. K. Moon, *Error Correction Coding*, New Jersey: John Wiley & Sons, 2005.

[4] L. Barnault and D. Declercq, "Fast Decoding Algorithms for LDPC Codes Over GF(2^q)," *Proceedings of the IEEE 2003 Information Theory Workshop*, pp. 70–73, Paris, France, March 2003.

[5] A. Voicila, D. Declercq, F. Verdier, M. Fossorier, and P. Urard, "Low-Complexity Decoding for Non-Binary LDPC Codes in High Order Fields," *IEEE Transactions on Communications*, vol. 58, no. 5, pp. 1365–1375, May 2010.

[6] F. Kschischang, B. Frey, and H.-A. Loeliger, "Factor Graphs and the Sum-Product Algorithm," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 498–519, February 2001.

[7] B.-Y. Chang, D. Divsalar, and L. Dolecek, "Non-Binary Protograph-Based LDPC Codes for Short Block-Lengths," *2012 IEEE Information Theory Workshop (ITW)*, pp. 282–286, Lausanne, Switzerland, September 3–7, 2012.

[8] S. Dolinar, D. Divsalar, and F. Pollara, "Code Performance as a Function of Block Size," *The Telecommunications and Mission Operations Progress Report*, vol. 42-133, Jet Propulsion Laboratory, Pasadena, California, pp. 1–23, May 15, 1998.
http://ipnpr.jpl.nasa.gov/progress_report/42-133/133K.pdf

[9] Consultative Committee for Space Data Systems (CCSDS), *Telecommand Synchronization and Channel Coding*, 231.0-B-2, Blue Book, Issue 2, September 2010.
http://public.ccsds.org/publications/BlueBooks.aspx

[10] Consultative Committee for Space Data Systems (CCSDS), *Synchronization and Channel Coding*, 131.0-B-2, Blue Book, Issue 2, August 2011.
http://public.ccsds.org/publications/BlueBooks.aspx

Appendix A

Decoding Codes Over GF(2)

When the functions f_j represent binary parity checks, the parity-to-data message may be simplified. Let $\mathbf{z} = (z_1, \ldots, z_m)$ be m independent bits, and $p_i(1) = P(z_i = 1)$. Then

$$P(\mathbf{z}\text{ has even parity}) = \frac{1}{2} + \frac{1}{2}\prod_{i=1}^{m}\left(1 - 2p_i(1)\right).$$

The proof is straightforward by induction on m. Recall that if we interpret $q_{ij}(\alpha)$ as assigning probabilities $P(x_i = \alpha)$, we can write our parity-to-data message as

$$\begin{aligned} r_{ji}(1) &= P\left(f_j \mid x_i = 1\right) \\ &= 1 - P\left(\mathbf{x}_{\mathcal{N}(j)\setminus i}\text{ has even parity}\right) \\ &= \frac{1}{2} - \frac{1}{2}\prod_{k\in\mathcal{N}(j)\setminus i}\left(1 - 2q_{kj}(1)\right). \end{aligned}$$

This representation facilitates transforming the algorithm to operate in the log-domain, which replaces the product involved in computing r_{ji} and yields an algorithm more amenable to numerical computation. Define the log-likelihood ratios (LLRs):

$$\begin{aligned} L(y_i) &= \log\frac{p(y_i \mid x_i = 0)}{p(y_i \mid x_i = 1)} \\ L(r_{ji}) &= \log\frac{r_{ji}(0)}{r_{ji}(1)} \\ L(q_{ij}) &= \log\frac{q_{ij}(0)}{q_{ij}(1)}. \end{aligned}$$

Note that $1 - 2p = \tanh\left(\frac{1}{2}\log\frac{1-p}{p}\right)$, and, since tanh is an odd function, $\tanh(\operatorname{sgn}(x)\,|x|/2) = \operatorname{sgn}(x)\tanh(|x|/2)$. Hence,

$$\begin{aligned} \tanh\left(L(r_{ji})/2\right) &= \prod_{k\in\mathcal{N}(j)\setminus i}\tanh\left(L(q_{kj})/2\right) \\ L(r_{ji}) &= 2\tanh^{-1}\left(\prod_{k\in\mathcal{N}(j)\setminus i}(\operatorname{sgn}(L(q_{kj})\,|\,L(q_{kj}|)/2)\right) \\ &= 2\prod_{k\in\mathcal{N}(j)\setminus i}\operatorname{sgn}\left(L(q_{kj})\right)\tanh^{-1}\left(\prod_{k\in\mathcal{N}(j)\setminus i}\tanh\left(|L(q_{kj})|/2\right)\right) \\ &= \prod_{k\in\mathcal{N}(j)\setminus i}\operatorname{sgn}\left(L(q_{kj})\right)2\tanh^{-1}\circ\log^{-1}\left(\sum_{k\in\mathcal{N}(j)\setminus i}\log\circ\tanh(|L(q_{kj})|/2)\right). \end{aligned}$$

Defining

$$\phi(x) = \log\circ\tanh\left(\frac{x}{2}\right),$$

we have

$$L(r_{ji}) = \prod_{k \in \mathcal{N}(j) \setminus i} \operatorname{sgn}\big(L(q_{kj})\big)\phi^{-1}\left(\sum_{k \in \mathcal{N}(j) \setminus i} \phi(|L(q_{kj})|)\right)$$

The function ϕ may be precomputed and tabulated in numerical implementation. Since $\phi^{-1}(x) = \phi(x)$, a single table suffices for ϕ and its inverse.

Transforming the data-to-checks message to a form amenable to computation requires no such contortions:

$$L\big(q_{ij}\big) = L\big(y_i\big) + \sum_{l \in \mathcal{M}(i)/j} L\big(r_{li}\big),$$

The resulting log-domain algorithm is summarized in Section III.B.

Appendix B

Decoding Codes over GF(q)

In applying the SPA algorithm to decoding codes over $\mathrm{GF}(q)$, Equation (3) may be evaluated efficiently with no modifications. However, an efficient implementation of Equation (4) is not as straightforward. Efficient methods to evaluate this step have been explored and we assume the use of the fast-Hadamard-transform (FHT) decoding algorithm, see, e.g., [4], which we review here for completeness.

We can think of the edge between parity node j and variable node i as carrying the label H_{ji} that defines a permutation of the messages along that edge. Define the permuted messages:

$$\begin{aligned} q'_{lj}(\alpha) &= P(H_{jl}x_l = \alpha) \\ &= P\left(x_l = H_{jl}^{-1}\alpha\right) \\ &= q_{lj}\left(H_{jl}^{-1}\alpha\right) \\ r'_{ji}(\alpha) &= r_{ji}\left(H_{jl}^{-1}\alpha\right). \end{aligned}$$

Consider parity check j. Let $w_i = H_{ji}x_i$. Then the parity-to-data message may be expressed as

$$\begin{aligned} r'_{ji}(\alpha) &= P\Big(\alpha + \sum_{l \in \mathcal{N}(j)\setminus i} w_l = 0\Big) \\ &= \sum_{w_{\mathcal{N}(j)} | w_i = \alpha} \mathbf{1}\Big(\sum_{k \in \mathcal{N}(j)} w_k = 0\Big) \prod_{l \in \mathcal{N}(j)\setminus i} q'_{lj}(w_l), \end{aligned}$$

where $\mathbf{1}(\cdot)$ evaluates to 1 when its argument is satisfied, and 0 otherwise. For $|\mathcal{N}(j)| \in \{1,2\}$, the computation is trivial. Suppose $|\mathcal{N}(j) = 3|$, and the neighbors are $1, 2$. Then

$$\begin{aligned} r'_{ji}(\alpha) &= \sum_{w_1, w_2} \mathbf{1}(\alpha + w_1 + w_2 = 0) q'_{1j}(w_1) q'_{2j}(w_2) \\ &= \sum_{w_1} q'_{1j}(w_1) q'_{2j}(-w_1 - \alpha) \\ &= q'_{1j} \circledast q'_{2j}(-\alpha) \end{aligned}$$

a convolution of the input messages. For $\mathcal{N}(j) > 3$, the convolutions nest, yielding

$$r'_{ji}(\alpha) = \Big(\underset{l \in \mathcal{N}(j)\setminus i}{\circledast} q'_{lj}\Big)(-\alpha)$$

which may be computed more efficiently in the transform domain. Define the transform pair:

$$Q_{lj}(\alpha) = \sum_{\beta} q_{lj}(\beta)(-1)^{\alpha \cdot \beta^T}$$
$$q_{lj}(\alpha) = \frac{1}{2^m}\sum_{\beta} Q_{lj}(\beta)(-1)^{\alpha \cdot \beta^T},$$

where $\alpha \cdot \beta^T$ is the dot-product of the binary-vector representations of the elements in $GF(2^m)$. Then

$$R'_{ji}(\beta) = \prod_{l \in \mathcal{N}(j)\setminus i} Q'_{lj}(\beta)$$
$$r'_{ji}(-\alpha) = \frac{1}{2^m}\sum_{\beta} R'_{ji}(\beta)(-1)^{\alpha \cdot \beta^T}.$$

Finally, permute to obtain $r_{ji}(\alpha) = r'_{ji}(H_{ji}\alpha)$. A summary of the resulting FHT-LDPC GF(q) is given in Section III.C.