IPN Progress Report 42-184 • February 15, 2011

Performance of Low-Density Parity-Check Coded Modulation

Jon Hamkins*

This article presents the simulated performance of a family of nine AR4JA low-density parity-check (LDPC) codes when used with each of five modulations. In each case, the decoder inputs are codebit log-likelihood ratios computed from the received (noisy) modulation symbols using a general formula which applies to arbitrary modulations. Suboptimal soft-decision and hard-decision demodulators are also explored. Bit-interleaving and various mappings of bits to modulation symbols are considered. A number of subtle decoder algorithm details are shown to affect performance, especially in the error floor region. Among these are quantization dynamic range and step size, clipping degree-one variable nodes, "Jones clipping" of variable nodes, approximations of the min function, and partial hard-limiting messages from check nodes. Using these decoder optimizations, all coded modulations simulated here are free of error floors down to codeword error rates below* 10^{-6}.

The purpose of generating this performance data is to aid system engineers in determining an appropriate code and modulation to use under specific power and bandwidth constraints, and to provide information needed to design a variable/adaptive coded modulation (VCM/ACM) system using the AR4JA codes.

I. Introduction

Forward error correction using Low-Density Parity-Check (LDPC) codes is rapidly gaining acceptance in the aerospace community [1]. A set of LDPC codes is in the final stages of approval as an international standardization by the Consultative Committee for Space Data Systems (CCSDS) [2]. The standard LDPC codes include a family of nine accumulate repeat-4 jagged accumulate (AR4JA) LDPC codes, available in any combination of three code rates (1/2, 2/3, and 4/5) and three input block lengths (1024, 4096, and 16384).

---

*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. © 2011 California Institute of Technology. Government sponsorship acknowledged.

Table 1. Codes, modulations, bit-to-symbol mappings, and demodulator structures considered by this article.

Code Rates	Code Lengths	Modulations	Bit Mappings	Demodulator Types
1/2	1024	BPSK	Natural	LLR
2/3	4096	QPSK	Gray	Approximate LLR
4/5	16384	8-PSK	Anti-Gray	Hard decision LLR
		16-APSK	DVB-S2	
		32-APSK		

The performance of the AR4JA LDPC codes on a binary-input additive white Gaussian noise (AWGN) channel is well-documented [1,2]. Such published performance results apply to binary phase-shift keying (BPSK) or quadrature PSK (QPSK) modulation, as is typically used in deep space missions. When bandwidth is constrained, however, system engineers may also desire to know the performance of LDPC codes when used with higher order modulations, in order to most effectively trade off power efficiency, bandwidth efficiency, and complexity. The need for bandwidth-efficient higher order modulations will become more pressing in the future as NASA and other space agencies utilize higher data rates and more simultaneous missions in the same limited spectrum. Modern variable coded modulation (VCM) or adaptive coded modulation (ACM) schemes will be able to switch between the different coded modulations as power and bandwidth resources vary.

Therefore, it is helpful to assess the performance of the standard LDPC codes when used with higher order modulations such as 8-PSK, 16-ary amplitude PSK (16-APSK), and 32-APSK. The performance of rate 4/5 AR4JA codes used with BPSK, 8-PSK, and 16-APSK has been previously reported [3]. For other combinations of codes and modulations, performance may be estimated based on the concept of code imperfectness. First, the code imperfectness of the code when used with BPSK is determined by measuring the difference between the code's required bit signal to noise ratio E_b/N_0 to attain a given codeword error rate (CWER) and the minimum possible E_b/N_0 required to attain the same CWER as implied by the sphere-packing bounds for codes with the same block size k and code rate r [4]. This same imperfectness is then applied with respect to the capacity of the higher order modulation to arrive at an approximated performance of the code when used with the higher order modulation. The imperfectness approximation has generally been found to be fairly accurate, to within about 0.5 dB, over a wide variety of codes and modulations.

Despite the results mentioned above, it remains helpful to have a comprehensive suite of simulation results for all combinations of the nine AR4JA LDPC codes and five modulations, as this is the most accurate estimate of performance. This article provides this suite of results. Along the way, we also provide a semi-tutorial presentation of the complex baseband representation of various modulation types, various bit-to-modulation-symbol mappings, a derivation of associated log likelihood ratios (LLRs), and a description of decoder imple-

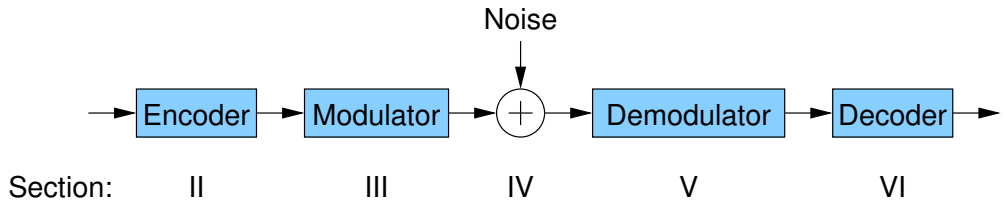


Figure 1. The signal flow considered in this article.

mentation details to optimize performance. One of the simple and well-performing LLR approximations can be expressed in a general equation that applies to all of the modulation types.

In particular, this article reports the simulated performance of the parameters shown in Table 1, for all combinations of codes and modulations, along with some combinations of mappings, demodulator structures, and number of decoder iterations. As shown in Figure 1, Section II discusses the LDPC encoder, Section III discusses the modulator, Section IV discusses the channel model, Section V discusses the demodulator, and Section VI discusses the decoder. Numerical results of the coded modulation are given in Section VII.

II. Encoding

Since the AR4JA LDPC codes are binary, linear codes, encoding is accomplished by multiplying, in GF(2), an information vector by a generator matrix. The AR4JA codes have a number of features that simplify the encoding process, and a more thorough treatment of the encoding process can be found in [5]. First, they are systematic, which means the information bits appear unchanged in the encoded codeword. Therefore, only the final $n-k$ columns of the $k \times n$ generator matrix need be stored by the encoder. The codes are also quasi-cyclic, which is a result of using circulants to permute edges of the protograph copies [2]. An encoder storing only rows $1, m+1, 2m+1, \ldots$, where m is the circulant size, may generate the other rows on the fly using shift registers.

In a software implementation, it may remain most convenient and efficient to store the last $n-k$ columns of the generator matrix in their entirety, not making use of the quasi-cyclic property, and performing the encoding operation using standard matrix multiplication. In a high-level language such as C, individual bit operations are not as efficient as operations that are applied on registers that are 32 or 64 bits wide. Therefore, in C it is efficient to break each of the $n-k$ columns into 64-bit segments, and store each segment in a 64-bit wide "long int" data structure. In this way, in one operation, 64-bits of information can be XORed with a 64-bit portion of the generator column, and the final codebit determined from the parity of all such 64-bit operations of the column. Since the input lengths of each of the AR4JA codes are a multiple of 64, this approach makes efficient use of the 64-bit data structures.

Table 2 shows the encoding speeds achieved using a software encoder in C on a standard

Table 2. Speed of encoders and decoders in C.

Input length	Code rate	E_b/N_0 (dB)	Average iterations	Enc. speed (Mbps)	Dec. speed (Mbps)
1024	1/2	1.80	16.44	14.0	0.597
1024	2/3	2.60	12.86	25.9	0.928
1024	4/5	3.70	9.20	49.5	1.410
4096	1/2	1.25	27.75	8.23	0.357
4096	2/3	2.00	22.94	14.4	0.537
4096	4/5	3.00	16.74	33.8	0.789
16384	1/2	0.95	46.03	1.57	0.219
16384	2/3	1.75	35.11	3.53	0.347
16384	4/5	2.75	23.97	6.45	0.541

desktop. Encoding speeds ranged from 1.5 to 50 Mbps.

III. Modulation

A. Modulation types

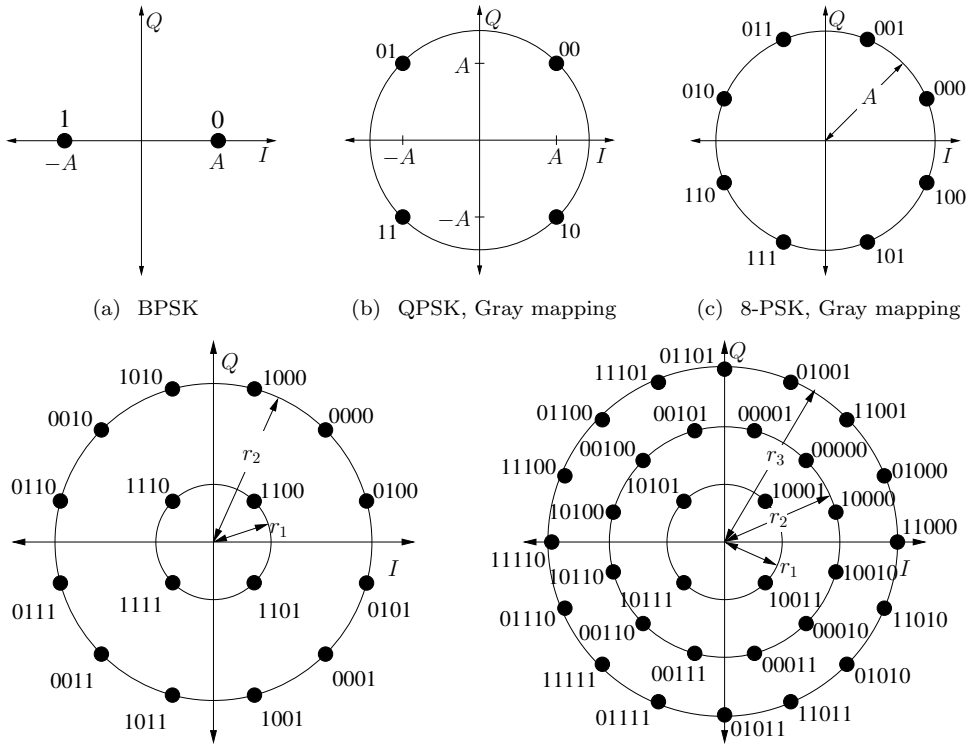
We now enumerate the modulation types considered in this article, along with their associated complex signal constellations, default indexing, and average complex baseband energy. The signal constellations are shown in Figure 2.

1. *BPSK* is a real-valued constellation with two signal points: $c(0) = A$ and $c(1) = -A$, where A is a scaling factor. This is shown in Figure 2(a). The average complex baseband symbol energy is $E_s = E[c(i)^2] = A^2$.

2. *QPSK* is a complex constellation with four signal points, $c(i) = \sqrt{2}A \exp\left[j\frac{\pi}{2}\left(i + \frac{1}{2}\right)\right]$, for $i = 0, 1, 2, 3$. It is convenient to include the $\sqrt{2}$ factor so that the average symbol energy is $E_s = E[\|c(i)\|^2] = 2A^2$, double that of BPSK, but with the same energy per transmitted bit as BPSK.

3. *8-PSK* has constellation points $c(i) = A \exp\left[j\frac{\pi}{4}\left(i + \frac{1}{2}\right)\right]$, for $i = 0, 1, \ldots, 7$. In general, M-PSK has constellation points $c(i) = A \exp\left[j\frac{2\pi}{M}\left(i + \frac{1}{2}\right)\right]$, for $i = 0, 1, \ldots M-1$. The average symbol energy is $E_s = E[\|c(i)\|^2] = A^2$.

4. *16-APSK* is a standard of the second generation Digital Video Broadcast for Satellites [6]. It is also referred to as 12/4 APSK or 12/4 QAM. It consists of the union of



(a) BPSK (b) QPSK, Gray mapping (c) 8-PSK, Gray mapping

(d) 16-APSK, DVB-S2 mapping (e) 32-APSK, DVB-S2 mapping

Figure 2. Signal constellations of various modulations.

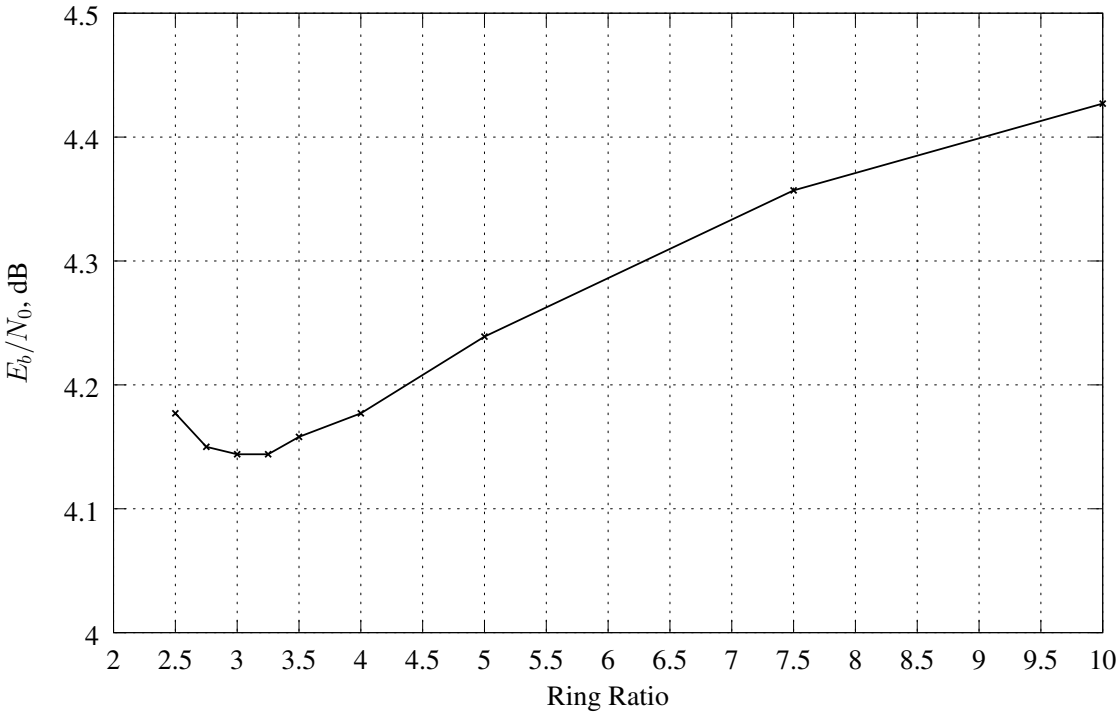


Figure 3. Required E_b/N_0 to achieve CWER $= 10^{-3}$ with $r = 1/2$, $k = 1024$ AR4JA LDPC coded 16-APSK, as a function of ring ratio.

amplitude-scaled QPSK and 12-PSK signal constellations

$$c(i) = \begin{cases} r_1 \exp\left[j\frac{\pi}{2}\left(i+\frac{1}{2}\right)\right] & i = 0, 1, 2, 3 \\ r_2 \exp\left[j\frac{\pi}{6}\left(i+\frac{1}{2}\right)\right] & i = 4, 5, \ldots, 15 \end{cases} \tag{1}$$

The DVB-S2 standard defines the ratio $r_2/r_1 = 3.15$, 2.85, 2.75, 2.70, 2.60, and 2.57 for code rates 2/3, 3/4, 4/5, 5/6, 8/9, and 9/10, respectively. The DVB-S2 standard does not specify use of a rate 1/2 code with 16-APSK; for our simulations, we set $r_2/r_1 = 3.15$ when a rate 1/2 code is used. The average symbol energy is $E_s = E[\|c(i)\|^2] = (r_1^2 + 3r_2^2)/4$.

Figure 3 shows the required E_b/N_0 to achieve CWER $= 10^{-3}$ for $r = 1/2$, $k = 1024$ AR4JA coded 16-APSK, as a function of the outer-to-inner ring ratio r_2/r_1. Although there is variation, the sensitivity is quite small. The optimal ratio for this coded modulation combination is about 3.15. For code-modulation combinations specified by DVB-S2, the simulations reported in this article used the standard ratios. For rate-modulation combinations not in the DVB-S2 standard, we first optimized the ratios using data as in Figure 3, and then ran all subsequent simulations with the optimized ratios.

5. *32-APSK* is also a DVB-S2 standard. It is the union of three PSK constellations

$$c(i) = \begin{cases} r_1 \exp\left[j\frac{\pi}{2}\left(i+\frac{1}{2}\right)\right] & i = 0, 1, 2, 3 \\ r_2 \exp\left[j\frac{\pi}{6}\left(i-4+\frac{1}{2}\right)\right] & i = 4, 5, \ldots, 15 \\ r_3 \exp\left[j\frac{\pi}{8}i\right] & i = 16, 17, \ldots, 31 \end{cases} \tag{2}$$

The DVB-S2 standard defines the ratios $r_2/r_1 = 2.84$, 2.72, 2.64, 2.54, and 2.53, and $r_3/r_1 = 5.27$, 4.87, 4.64, 4.33, and 4.30 for for code rates 3/4, 4/5, 5/6, 8/9, and 9/10, respectively. The DVB-S2 standard does not specify use of rate 1/2 or 2/3 codes with 32-APSK; for our simulations, we set $r_2/r_1 = 4.0$ and 3.15 and $r_3/r_1 = 8.0$ and 6.25 when rate 1/2 and 2/3 codes, respectively, are used. The average symbol energy is $E_s = E[\|c(i)\|^2] = (r_1^2 + 3r_2^2 + 4r_3^2)/8$.

B. Mapping bits to symbols

Encoded bits are assigned to a sequence of corresponding complex constellation points, or modulation symbols. Each of the modulations considered in this article has a number of constellation points that is a power of two, which makes such bit-to-symbol mappings straightforward.

The signal constellations in the previous section define a natural binary ordering. For example, the 8-PSK constellation points indexed by $i = 0$, 1, 2, 3, 4, 5, 6, and 7 correspond to the 3-bit patterns 000, 001, 010, 011, 100, 101, 110, and 111, respectively. We refer to this as the natural bit-to-symbol mapping for the modulation. Note that the natural ordering, or any other, is dependent on the way the constellation points happen to be indexed which, in principle, is arbitrary. In part, this is why we were explicit in the previous section in defining how each modulation is indexed with respect to i.

Other mappings, such as Gray codes,[2] can often give better performance. There are many Gray codes with the defining property that adjacent members in the list differ in exactly one bit in their binary representation, some with slightly different performance than others. In our simulations, we use the binary reflected Gray code, which has recently been proven to be the optimal Gray code for M-PSK modulations [7]. The binary reflected Gray code of length M is obtained from the binary reflected Gray code of length $M/2$ by listing the members $0, 1, \ldots, M-1$, each preceded by a zero, followed by the members $M-1, M-2, \ldots, 0$, each preceded by a one.

The binary reflected Gray code has the prefix property, i.e., a length M' Gray code's members are equal to the first M' members of a Gray code of length M, $M > M'$. Thus, when conducting simulations of Gray codes of various lengths, only the longest Gray code need be stored.

An anti-Gray code has the property that adjacent members in the list differ either in all their bits or in all but one of their bits. An anti-Gray code of length M can be obtained from a binary reflected Gray code of length M by removing the last $M/2$ entries and inserting after each of the remaining $M/2$ entries the ones complement of that entry. Anti-Gray codes do not have a prefix property, meaning a separate mapping should be stored for each length.

---

[2]Technically, a Gray code is more properly referred to as a Gray *labeling*. A code's word error rate performance is not dependent on the order of indexing, whereas with a Gray labeling, the whole point is that it is defined in a particular order. Nevertheless, we use the term Gray codes here, for consistency with common usage.

For modulations in which constellation points have more than two near neighbors, a specialized bit to symbol mapping is needed. The DVB-S2 standard specifies such a mapping to use with 16-APSK and 32-APSK.

The bit representations of the constellation points under the natural, Gray, anti-Gray, and DVB mappings are given in Table 3, for lengths 2, 4, 8, 16, and 32. Note that in the Gray column, 0, 1, 3, 2, ... in binary is 00000, 00001, 00011, 00010, ..., and each subsequent constellation point has a binary representation that differs in exactly one bit, including wrapping around to the beginning. The anti-Gray column has a separate specification for each length and, for example, 0, 7, 1, 6, ..., in binary is 000, 111, 001, 110, ..., with each entry differing in either two or all three bits. In Figure 2, the BPSK, QPSK, and 8-PSK modulations are shown with the Gray code, and the 16-APSK and 32-APSK modulations are shown with the DVB-S2 standard mapping.

Table 3 gives a mapping from the constellation index i to the bit representation $map(i)$, but at the modulator we need the inverse operation, to map bits to a constellation point. The inverse is defined by $c_m[map(i)] = c(i)$ for each i, where the subscript m indicates that the constellation has been mapped to a new ordering. For example, to map "1000" to a constellation point using the Gray code, we note that "1000" is 8 in decimal, and $c_m[8] = c(15)$ is the corresponding constellation point.

Figure 4 shows the performance of the $r = 1/2$, $k = 1024$ AR4JA code with 8-PSK when the bit-to-symbol mapping is Gray, natural, and anti-Gray. At BER $= 10^{-6}$, a natural mapping incurs a loss of 2.8 dB compared to the Gray code, and an anti-Gray code incurs a loss of 4.1 dB compared to the Gray code. It is important for system designers, therefore, to use a Gray mapping when using LDPC codes and higher order modulations.

IV. Channel Model

To isolate the coded modulation performance from other effects, this article assumes an additive white Gaussian noise (AWGN) channel with no Doppler, fading, or other channel impairments, no amplifier distortions, and perfect receiver synchronization of carrier frequency, phase, and timing.

The passband signal is assumed to be of the form

$$s(t) = a(t)\cos(2\pi f_c t + \theta(t)) \tag{3}$$

where f_c is the carrier frequency in Hz, and $a(t)$ and $\theta(t)$ are arbitrary modulation-dependent signals. We may rewrite this as

$$s(t) = \mathrm{Re}\left\{\tilde{s}(t)e^{j2\pi f_c t}\right\} \tag{4}$$

where $\tilde{s}(t) = a(t)e^{j\theta(t)}$ is the complex baseband representation of $s(t)$. We may also write $\tilde{s}(t)$ as

$$\tilde{s}(t) = \sqrt{P_c} + \tilde{m}(t) \tag{5}$$

Table 3. Bit representations of constellation points. Table entries have been converted from binary to decimal.

Natural	Gray	Anti-Gray					DVB-16	DVB-32
0	0	0	0	0	0	0	12	17
1	1	1	3	7	15	31	14	21
2	3		1	1	1	1	15	23
3	2		2	6	14	30	13	19
4	6			3	3	3	4	16
5	7			4	12	28	0	0
6	5			2	2	2	8	1
7	4			5	13	29	10	5
8	12				6	6	2	4
9	13				9	25	6	20
10	15				7	7	7	22
11	14				8	24	3	6
12	10				5	5	11	7
13	11				10	26	9	3
14	9				4	4	1	2
15	8				11	27	5	18
16	24					12		24
17	25					19		8
18	27					13		25
19	26					18		9
20	30					15		13
21	31					16		29
22	29					14		12
23	28					17		28
24	20					10		30
25	21					21		14
26	23					11		31
27	22					20		15
28	18					9		11
29	19					22		27
30	17					8		10
31	16					23		26

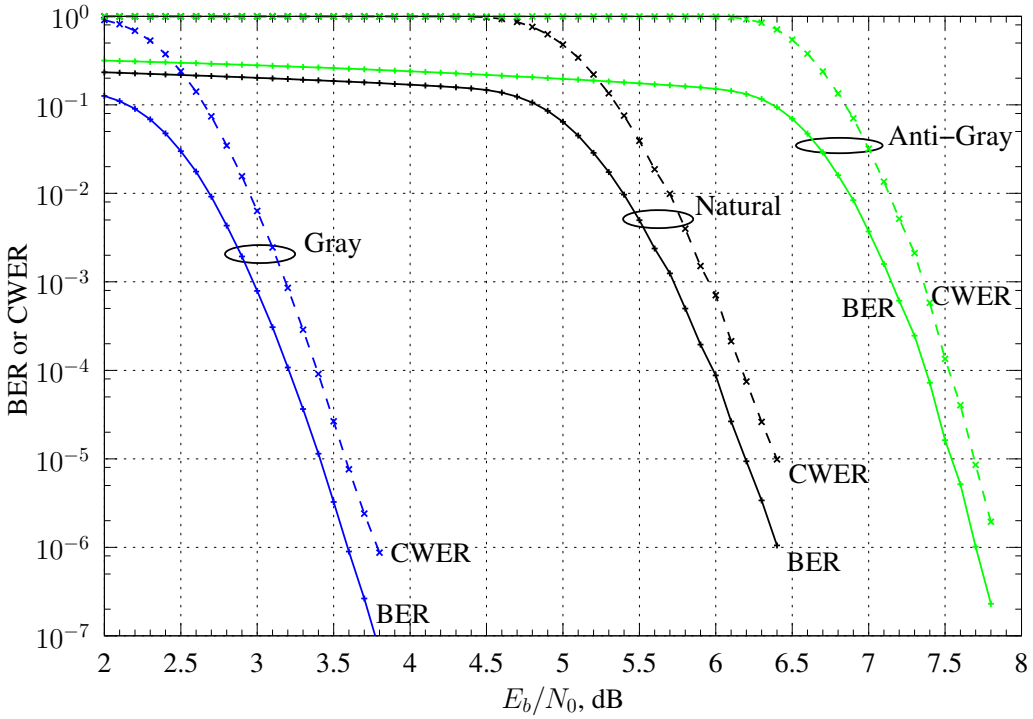


Figure 4. Performance of $r = 1/2$, $k = 1024$ AR4JA LDPC coded 8-PSK using various bit-to-symbol mappings.

where $\sqrt{P_c}$ is an unmodulated residual carrier signal with complex baseband power P_c, and $\tilde{m}(t)$ is a complex baseband modulation with complex baseband power $P_d = \lim_{T\to\infty} \frac{1}{T} \int_0^T \tilde{m}^2(t)dt$. This can be put back in passband notation using Equation (4), from which the residual carrier signal term $\sqrt{P_c}\cos(2\pi f_c t)$ is readily apparent. The modulations considered in this article have the form

$$\tilde{m}(t) = \sum_{i=-\infty}^{\infty} m[i]p(t - iT) \tag{6}$$

where $m[i]$ is a member of a signal constellation $m[i] \in C = \{c(0), c(1), \ldots, c(M-1)\}$ in the complex plane, and where $p(t)$ is square pulse shape of symbol duration T

$$p(t) = \begin{cases} 1 & \text{if } 0 \le t < T \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

For our purposes, the residual carrier signal can be assumed to have been filtered out of the modulated received signal or, equivalently, $P_c = 0$. Thus, the received modulated complex baseband signal is of the form

$$\tilde{r}(t) = \tilde{m}(t) + \tilde{n}(t) \tag{8}$$

where $\tilde{n}(t)$ is a complex baseband Gaussian noise process with one-sided power-spectral density N_0 in each dimension. At the receiver, $\tilde{r}(t)$ is put through a perfect matched filter, which results in complex soft symbols

$$r[i] = m[i] + n[i] \tag{9}$$

where $n[i]$ is a complex Gaussian random variable with variance variance σ^2 in each of its real and imaginary components.

V. Demodulation– forming the Log Likelihood Ratio (LLR)

Soft decision decoders take as input the log likelihood ratio (LLR) for each code bit [3]. Suppose bits $\mathbf{b} = b_{m-1}b_{m-2}\cdots b_0$ are mapped to the complex constellation point $c = c(\mathbf{b})$. Here, we have dropped the subscript m for notational convenience, and assume $c(\cdot)$ itself specifies the correct order of symbols for the desired mapping. Let $r = c + n$ denote the noisy received symbol.

In this section, we derive the exact LLR expression for an arbitrary constellation, provide a lower-complexity approximate LLR expression based on nearest neighbors to the received point, and provide the LLR expressions specific to BPSK, QPSK, 8-PSK, 16-APSK, and 32-APSK.

A. LLR for an arbitrary constellation

1. Exact LLR

The LLR for the jth bit of the symbol is

$$\begin{aligned}\lambda_j &\triangleq \ln\left[\frac{P(b_j=0|r)}{P(b_j=1|r)}\right] \\ &= \ln\left[\frac{p(r|b_j=0)P(b_j=0)/p(r)}{p(r|b_j=1)P(b_j=1)/p(r)}\right] \\ &= \ln\left[\frac{p(r|b_j=0)}{p(r|b_j=1)}\right] \end{aligned} \tag{10}$$

where we use P to indicate a probability and p to indicate a probability density function (pdf), we applied Bayes' rule for a mixture of probabilities and pdfs, and in the last step we assume $P(b_j=0) = P(b_j=1) = 1/2$. For $i \in \{0,1\}$, we have

$$p(r|b_j=i) = \sum_{\mathbf{b}:b_j=i} p(r|\mathbf{b}) \tag{11}$$

$$= \sum_{\mathbf{b}:b_j=i} p(r-c(\mathbf{b})) \tag{12}$$

$$= \sum_{\mathbf{b}:b_j=i} \frac{\exp\left(-\frac{\|r-c(\mathbf{b})\|^2}{2\sigma^2}\right)}{2\pi\sigma^2} \tag{13}$$

where Equation (11) follows because it is a sum of disjoint events, and Equation (13) is the pdf of a complex Gaussian random variable with variance σ^2 in each of its real and imaginary components. Substituting into Equation (10), we have

$$\lambda_j = \ln\left[\frac{\sum_{\mathbf{b}:b_j=0}\exp\left(\frac{-\|r-c(\mathbf{b})\|^2}{2\sigma^2}\right)}{\sum_{\mathbf{b}:b_j=1}\exp\left(\frac{-\|r-c(\mathbf{b})\|^2}{2\sigma^2}\right)}\right] \tag{14}$$

Thus, to compute the jth bit LLR from r, one may compute the squared distance to *each* of the constellation points, separating those constellation points that have a 0 in bit j from those that have a 1, and using Equation (14).

We may use the relation

$$\|r - c\|^2 = \|r\|^2 - 2\langle r, c\rangle + \|c\|^2 \tag{15}$$

in Equation (14), where the inner product is

$$\langle r, c\rangle \stackrel{\triangle}{=} \mathrm{Re}\{r\} \times \mathrm{Re}\{c\} + \mathrm{Im}\{r\} \times \mathrm{Im}\{c\}$$

When the modulation has symbols each of the same energy, as is the case for PSK modulations, the $\|r\|^2$ and $\|c\|^2$ terms in the numerator and denominator cancel we arrive at the simpler form

$$\lambda_j = \ln\left[\frac{\sum_{\mathbf{b}:b_j=0} \exp\left(\frac{\langle r, c(\mathbf{b})\rangle}{\sigma^2}\right)}{\sum_{\mathbf{b}:b_j=1} \exp\left(\frac{\langle r, c(\mathbf{b})\rangle}{\sigma^2}\right)}\right] \tag{16}$$

2. Approximate LLR

A common approximation to the LLR is to replace each sum in Equation (14) by its largest term, i.e., by using only the nearest constellation point that has $b_j = 0$ in the numerator, and the nearest neighbor that has $b_j = 1$ in the denominator. If we denote these nearest neighbor constellation points by

$$c^*(j, i) \stackrel{\triangle}{=} c\left(\underset{\mathbf{b}:b_j=i}{\mathrm{argmin}} \|r - c(\mathbf{b})\|^2\right) \tag{17}$$

$i \in \{0, 1\}$, we may write

$$\begin{aligned}
\lambda_j &\approx \ln\left[\frac{\exp\left(\frac{-\|r-c^*(j,0)\|^2}{2\sigma^2}\right)}{\exp\left(\frac{-\|r-c^*(j,1)\|^2}{2\sigma^2}\right)}\right] && (18)\\
&= \frac{1}{2\sigma^2}\left(\|r - c^*(j,1)\|^2 - \|r - c^*(j,0)\|^2\right) \\
&= \frac{1}{2\sigma^2}\left(2\langle r, c^*(j,0) - c^*(j,1)\rangle + \|c^*(j,1)\|^2 - \|c^*(j,0)\|^2\right) && (19)
\end{aligned}$$

or, for equal energy signal constellations

$$\lambda_j \approx \frac{\langle r, c^*(j,0) - c^*(j,1)\rangle}{\sigma^2} \tag{20}$$

This requires one subtraction and two multiplications. The step of dividing by σ^2 can be eliminated if σ remains constant over many symbols, by precomputing $c(i)/\sigma^2$ for each i.

Figure 5 shows the codeword error rate (CWER) performance of the decoder when using the exact LLR in Equation (16) and the nearest neighbor approximation in Equation (19). The results shown are for 32-APSK with AR4JA LDPC codes of length $k = 1024$ and rates $r = 1/2$, $2/3$, and $4/5$. As can be seen, the approximate LLR leads to about 0.1 dB of loss for

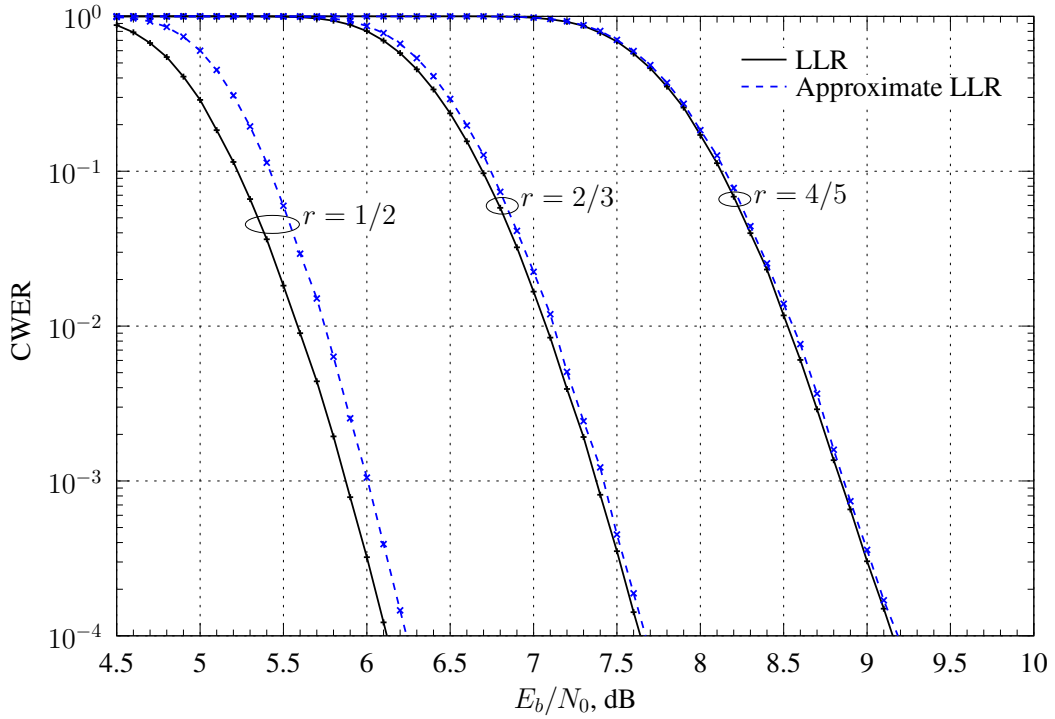


Figure 5. Comparison of LLR and approximate LLR decoder performance for AR4JA LDPC coded 32-APSK with $k = 1024$, and $r = 1/2, 2/3,$ and $4/5$.

$r = 1/2$, and 0 to 0.05 dB of loss for rates 2/3 and 4/5. This justifies using the approximate LLR in an implementation. Nevertheless, in all other simulation results reported in this article the exact LLR is used because the demodulator complexity is small compared to the decoder complexity, and thus the simulation time is not substantially increased by using the exact demodulator.

3. LLR for Hard Decision Decoding

When the demodulator produces hard decisions, the decoder does not have access to r, and therefore cannot compute λ_j as in Equation (14). Instead, the decoder only is told whether b_j is more probably a 1 or a 0, i.e., whether $\lambda_j \leq 0$ or $\lambda_j > 0$, respectively. That is, the hard decision decoder is given $\mathrm{sgn}(\lambda_j)$.

Because the decoder operates on LLRs, we may proceed as before to define a hard decision LLR, given by

$$\begin{aligned} \lambda_j^{(H)} &\triangleq \ln\left[\frac{P(b_j=0|\mathrm{sgn}(\lambda_j))}{P(b_j=1|\mathrm{sgn}(\lambda_j))}\right] \\ &= \ln\left[\frac{p(\mathrm{sgn}(\lambda_j)|b_j=0)}{p(\mathrm{sgn}(\lambda_j)|b_j=1)}\right] \\ &= \mathrm{sgn}(\lambda_j)\cdot\ln\left[\frac{1-p}{p}\right] \end{aligned} \tag{21}$$

where p is the probability that the hard decision is incorrect. For BPSK, $p = Q\left(\sqrt{2E_s/N_0}\right)$, where

$$Q(x) = \int_x^\infty \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx$$

Note that computation of $\lambda_j^{(H)}$ requires knowledge of E_s/N_0. The receiver typically makes an estimation of this, but if this estimate is not available, there would be an additional decoder implementation loss.

B. LLR for BPSK

For BPSK modulation there are only two constellation points, and so the expression in Equation (18), and hence Equation (20), is exact. There is only one bit LLR to compute, namely, λ_0, with $c^*(0,0) = A$ and $c^*(0,1) = -A$, and the LLR is given by

$$\lambda_0 = \frac{\langle r, c^*(j,0) - c^*(j,1)\rangle}{\sigma^2} = \frac{\langle r, 2A\rangle}{\sigma^2} = \frac{2A\mathrm{Re}\{r\}}{\sigma^2} \tag{22}$$

When a code is used with BPSK, the LLRs of the codebits are independent and identically distributed (i.i.d.), because each codebit gets mapped to its own modulation symbol, and each modulation symbol is corrupted by i.i.d. noise.

C. LLR for QPSK

As can be seen from Figure 2(b), the least significant bit (LSB) of a Gray coded QPSK modulation depends on $\mathrm{Re}\{r\}$ in exactly the same way as for BPSK. This can be seen mathematically by noting

$$\begin{aligned} c(0) &= A(1+j) \\ c(1) &= A(-1+j) \\ c(2) &= A(1-j) \\ c(3) &= A(-1-j) \end{aligned}$$

and then plugging these into Equation (16), which becomes

$$\lambda_0 = \ln\left[\frac{\exp\left(\frac{\langle r,c(0)\rangle}{\sigma^2}\right) + \exp\left(\frac{\langle r,c(2)\rangle}{\sigma^2}\right)}{\exp\left(\frac{\langle r,c(1)\rangle}{\sigma^2}\right) + \exp\left(\frac{\langle r,c(3)\rangle}{\sigma^2}\right)}\right] \tag{23}$$

Using

$$\begin{aligned} \langle r, c(0)\rangle &= A(\mathrm{Re}\{r\} + \mathrm{Im}\{r\}) \\ \langle r, c(1)\rangle &= A(-\mathrm{Re}\{r\} + \mathrm{Im}\{r\}) \\ \langle r, c(2)\rangle &= A(\mathrm{Re}\{r\} - \mathrm{Im}\{r\}) \\ \langle r, c(3)\rangle &= A(-\mathrm{Re}\{r\} - \mathrm{Im}\{r\}) \end{aligned}$$

and plugging into Equation (23) and simplifying, we have

$$\lambda_0 = \frac{2A\mathrm{Re}\{r\}}{\sigma^2}$$

which is identical to Equation (22). Following the same procedure for the most significant bit, where now $c(0)$ and $c(1)$ are in the numerator and $c(2)$ and $c(3)$ are in the denominator, the LLR is given by

$$\lambda_1 = \frac{2A\text{Im}\{r\}}{\sigma^2} \tag{24}$$

As was the case for BPSK, with coded QPSK using a Gray bit-to-symbol mapping, the LLRs of the codebits are independent and identically distributed (i.i.d.). Note, when the bit-to-symbol mapping is not a Gray code, the LLR expressions will not simplify to the expressions above, and the LLR's will not be i.i.d.

D. LLR for 8-PSK

The three bit LLRs for each 8-PSK symbol can be computed using Equation (16), with four terms each in the numerator and denominator. As there is no apparent simplification of this exact LLR expression, the approximate LLR computation of Equation (20) can be used when a lower complexity computation is needed.

To identify the closest constellation point with a 0 or a 1 in the bit position of interest, one could compute the distances to all eight constellation points. This is unnecessary, however. As can be seen from Figure 2(c), if we express r in polar coordinates as $r = \|r\|e^{j\phi}$, the closest constellation point with LSB equal to zero is given by

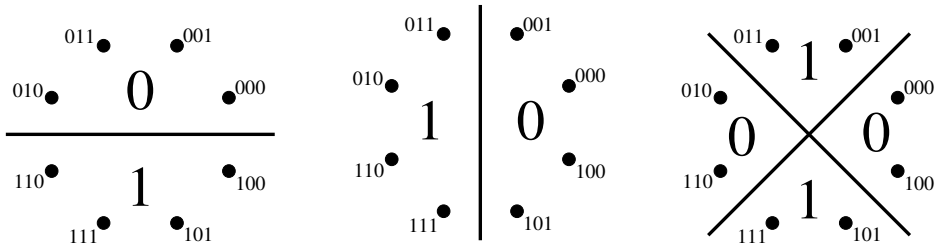
$$c^*(0,0) = \begin{cases} c(0) & \text{if } 0 \le \phi < \pi/4 \\ c(3) & \text{if } 3\pi/4 \le \phi < \pi \\ c(4) & \text{if } \pi \le \phi < 5\pi/4 \\ c(7) & \text{if } 7\pi/4 \le \phi < 2\pi \end{cases} \tag{25}$$

This computation requires only comparisons to constants, and no computation of distances. Similarly

$$c^*(0,1) = \begin{cases} c(1) & \text{if } \pi/4 \le \phi < \pi/2 \\ c(2) & \text{if } \pi/2 \le \phi < 3\pi/4 \\ c(5) & \text{if } 5\pi/4 \le \phi < 3\pi/2 \\ c(6) & \text{if } 3\pi/2 \le \phi < 7\pi/4 \end{cases} \tag{26}$$

These can then be plugged into Equation (20). The LLRs for the other two bits can be computed in a similar fashion.

Unlike BPSK and QPSK, when higher order modulations are used, the codebit LLRs are neither independent nor identically distributed. They are not independent because noise affecting reception of an 8-PSK constellation point affects the LLRs of the three associated codebits in a correlated manner. They are not identically distributed because the distance properties are not the same with respect to each bit. For example, with Gray-coded 8-PSK as shown in Figure 2(c), the most significant bit (MSB) is '1' if the point is above the I axis and '0' otherwise. Figure 6 shows this partition, and the partitions for the middle bit and least significant bit (LSB).



(a) MSB (b) Middle bit (c) LSB

Figure 6. Bit to symbol mapping regions for Gray-coded 8-PSK.

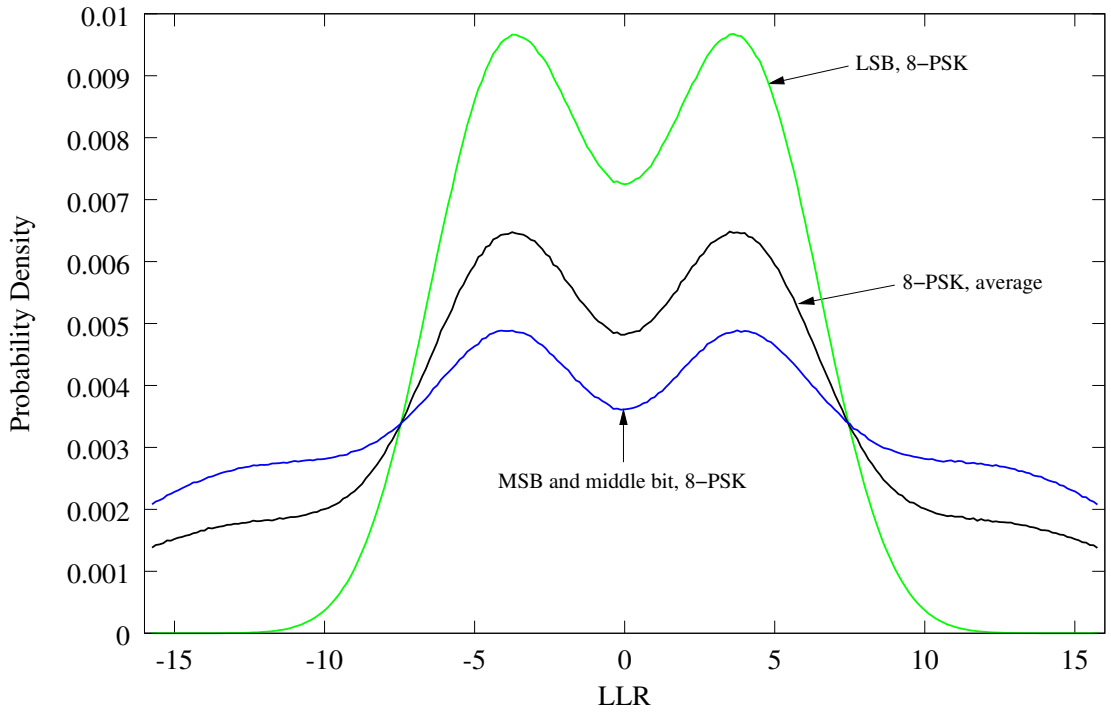


Figure 7. LLR Distribution for the individual bits of 8-PSK.

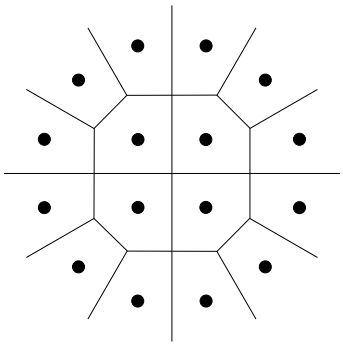


Figure 8. Voronoi regions of 16-APSK.

The distance properties of the LSB are worse than those of the other two bits. As a result, the MSB and middle bit of Gray-coded 8-PSK are received, on average, with a higher absolute LLR than the LSB is. Figure 7 shows this for $k = 1024$, $r = 2/3$ coded 8-PSK at $E_b/N_0 = 5$ dB. This SNR corresponds to CWER $\approx 10^{-5}$. As can be seen, the LSB is more likely to have a lower absolute LLR than the MSB or middle bits. The aggregate LLR distribution for 8-PSK is shown as well. This effect is important when considering an implementation of interleavers, which is discussed in Section VIII.

E. LLR for 16-APSK

The four bit LLRs for each 16-APSK symbol can be computed using Equation (16), with eight terms each in the numerator and denominator. As there is no apparent simplification of this exact LLR expression, the approximate LLR computation of Equation (20) can be used when a lower complexity computation is needed.

To identify the closest constellation point with a 0 or a 1 in the bit position of interest, one could compute the distances to all sixteen constellation points. As was the case for 8-PSK, this is unnecessary. Since 16-APSK is simply the union of two PSK modulations, the angle comparison approach used for 8-PSK can be used to identify the closest inner-ring constellation point with a 0 in the bit position of interest, and separately, to identify the closest outer-ring constellation point. Then $\langle r, c\rangle$ can be computed for each of the two candidate constellation points to find the closer point. This requires computation of a total of four inner products, or eight multiplications, to compute an approximate bit LLR.

A more careful approach can be even more efficient. The Voronoi regions of 16-APSK are shown in Figure 8. As can be seen, the Voronoi region boundaries between the inner and outer constellation points are either horizontal, vertical, or at a 45 degree angle. Thus, a carefully crafted series of comparisons involving $\mathrm{Re}\{r\}$, $\mathrm{Im}\{r\}$, $\mathrm{Re}\{r\} \pm \mathrm{Im}\{r\}$, and ϕ can identify $c^*(j, i)$ without multiplications. In this way, only comparisons and the one inner product in Equation (20) would need to be computed.

F. LLR for 32-APSK

The five bit LLRs for each 32-APSK symbol can be computed using Equation (16), with sixteen terms each in the numerator and denominator. As there is no apparent simplification of this exact LLR expression, the approximate LLR computation of Equation (20) can be used when a lower complexity computation is needed. Since 32-APSK is the union of three PSK modulations, the angle comparison approach used for 8-PSK can be used to identify the closest constellation point with a 0 in the bit position of interest, on each ring. Then $\langle r, c \rangle$ can be computed for each of the three candidate constellation points to find the closest point. The same type of calculation is made for constellation points with a 1 in the bit position of interest. This requires computation of a total of six inner products, or twelve multiplications, to compute an approximate bit LLR.

The Voronoi boundaries of 32-APSK are not all horizontal, vertical, or at a 45 degree angle, so the more efficient method detailed above for 16-APSK could not be used for 32-APSK.

VI. Decoding

An LDPC code is decoded with an iterative message passing algorithm on a bipartite graph. A summary description (e.g., [1]) and full derivation (e.g., [8]) of the decoding algorithm is available in several places in the literature. Such descriptions address the computation of appropriate conditional probabilities of maximum a posteriori (MAP) bit estimates, however, they do not typically address some of the practical aspects of decoder design, such as the quantization of the input LLRs, the finite-precision of the computations and messages being passed, complexity-reducing approximations, and subtle decoder variations. These details can have a significant impact on performance. We will discuss some of these details here.

Figure 9 is representative of the type of performance differences observed in independently developed decoders. The code illustrated is the $k = 1024$, $r = 4/5$ AR4JA code, with BPSK modulation. Shiva Planjery produced perhaps the largest set of decoder variations for this code[3], although those results are not included here. Among the CCSDS AR4JA LDPC codes, the highest error floor is usually seen on this code, so it is an instructive code to study.

As can be seen in Figure 9, the location of the floor is dependent on the decoder. The three decoders share several salient features – they all used 8-bit quantization and a similar $\min^*$ implementation, for example – but small differences in the decoders led to significant differences in the error floor performance. The JH2009 curve[4] has an error floor beginning

[3]Shiva Planjery, Fall 2008 CCSDS presentation, and unpublished manuscript.

[4]The blue curve in Figure 9, labeled JH2009, is from a software simulation made by Jon Hamkins in 2009. That decoder is an 8-bit decoder with dynamic range (-15.875, 15.875). It uses an approximation of $\min^*$ based on min minus one log correction term (with the difference not allowed to flip the sign), no special clipping of channel symbols for degree-1 variable nodes, and no Jones clipping of variable nodes.

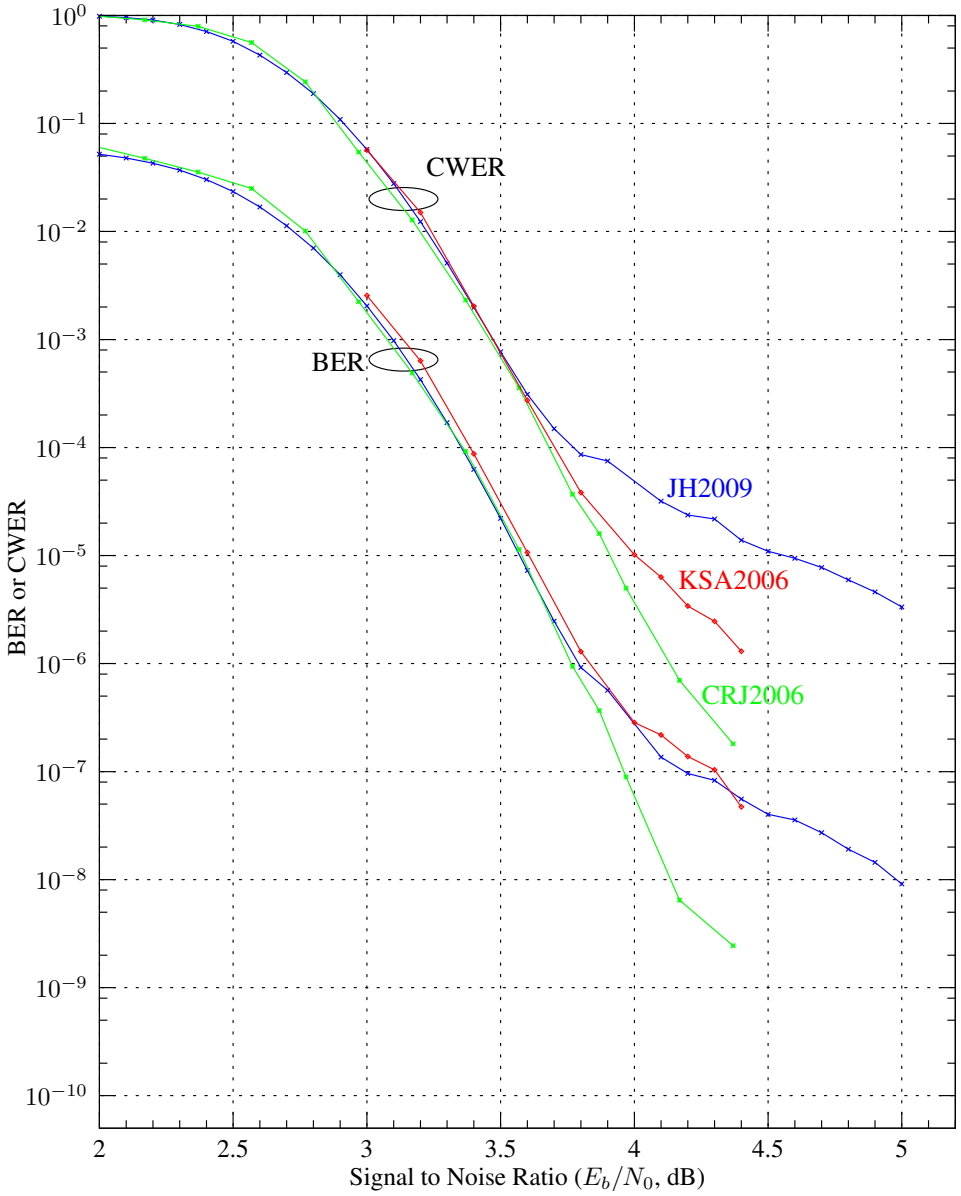


Figure 9. Previously reported performance of (1024,4/5) AR4JA decoders.

at about CWER= 10^{-4} and BER=10^{-6}, the KSA2006 curve[5] has a floor beginning at about CWER=10^{-5} and BER=3×10^{-7}, and the CRJ2006 curve[6] has no indication of a floor except possibly in its last simulated point, at about CWER=10^{-6} and BER=10^{-8}. Shiva's multiple-decoders approach showed an error floor near about CWER=10^{-7} and BER=10^{-10}.

After optimization, the performance can be improved to that shown in Figure 10. This performance was the result of a simulation of more than 8×10^{12} bits. We now discuss the various optimizations used to achieve this performance. (The use of partial hard-limiting discussed in Section VI-D.4 was the key to the dramatically lower floor.)

A. Number of iterations

Figure 11 shows the bit error rate (BER) performance of a decoder as a function of the number of iterations. The results shown are for the $k = 1024$, $r = 1/2$ AR4JA code used with BPSK on an AWGN channel, demodulated with an exact LLR computation quantized to 8 bits, and with a decoder limited to a maximum of 2, 5, 10, 20, 50, 100, and 200 iterations. As indicated in the figure, there is not much performance improvement beyond about 50 iterations for this code. The $k = 4096$ and $k = 16384$ results show slightly larger performance improvement beyond 50 iterations than is the case for $k = 1024$, and this has led us to conduct the remainder of simulations reported in this article with a maximum of 200 iterations. When a codeword takes significantly longer than the average number of iterations to decode, incoming codewords may be buffered, and generally a buffer of 2 or 3 codewords reduces the probability of buffer overflow (or equivalently, implementation loss) to near zero. In a deployed implementation, a system engineer may trade off the implementation loss with the maximum number of iterations supported.

B. Quantization

In a practical decoder, LLRs are represented by digital quantities. This quantization limits both the dynamic range and the resolution of the LLRs. In early experiments, it has been determined that 8 bits of quantization for the LLRs leads to a negligible loss in performance[7].

---

[5]The red curve in Figure 9, labeled KSA2006, is from a simulation by Ken Andrews in 2006. This was an integers-only decoder using 8 bits for channel LLRs and all messages, uniform quantization between -127/8 and +127/8, and clipping of degree-1 variable nodes to maximum magnitude 116/8.

[6]The green curve in Figure 9, labeled CRJ2006 is from an FPGA simulation by Chris Jones in 2006. This performance was reported in the FY2006 annual review of the IND Technology Program and in the AR4JA CCSDS Orange Book. This also was an 8-bit decoder with dynamic range (-15.875, 15.875) and degree-1 clipping, and in addition it incorporated "Jones clipping" at variable nodes, in which the sum of all messages into a variable node is clipped (e.g., to ± 127, for an 8-bit decoder) prior to forming an outgoing message by subtracting off one of the incoming messages. It also included a number of other differences in check node processing, such as at most 2 unique outgoing messages at each iteration.

[7]Kenneth Andrews, personal communication.

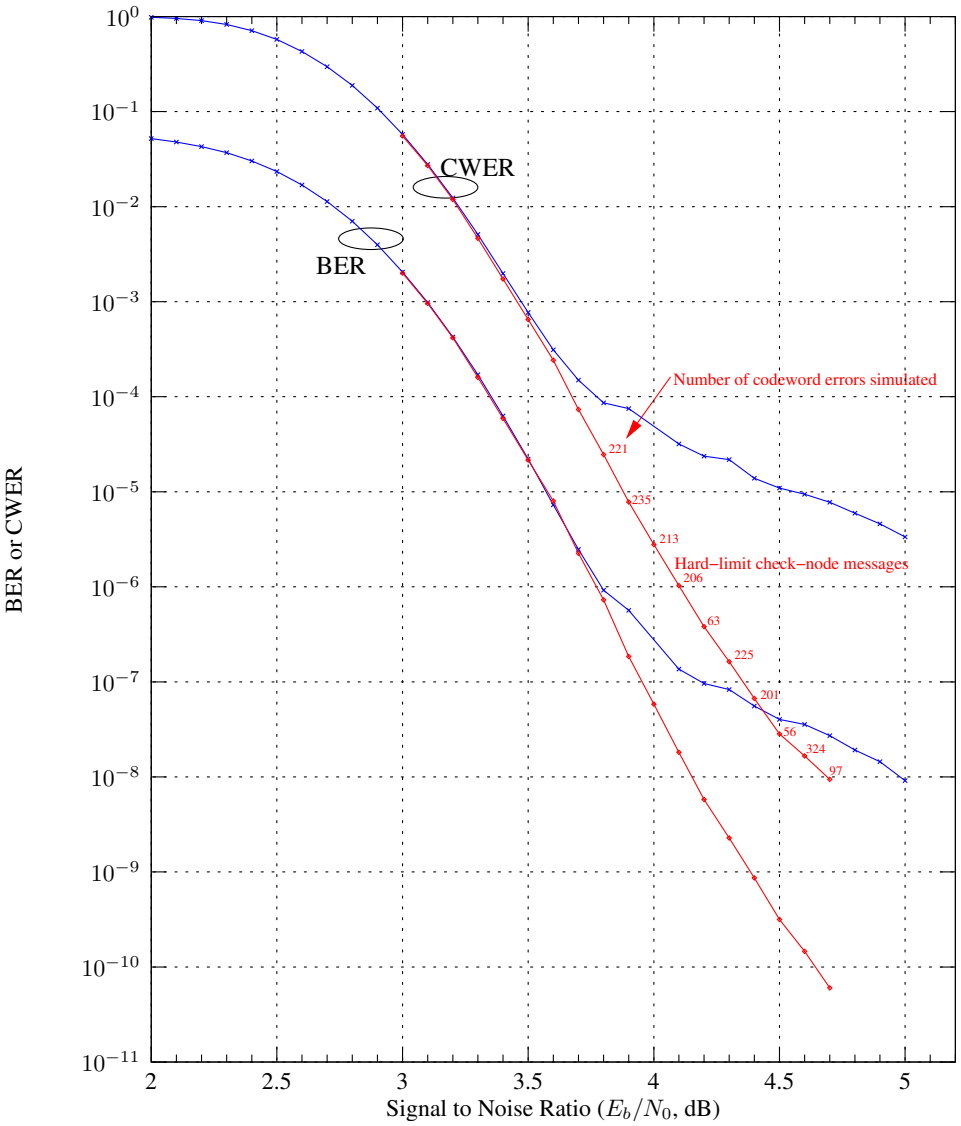


Figure 10. A (1024,4/5) AR4JA decoder with a lower error floor.

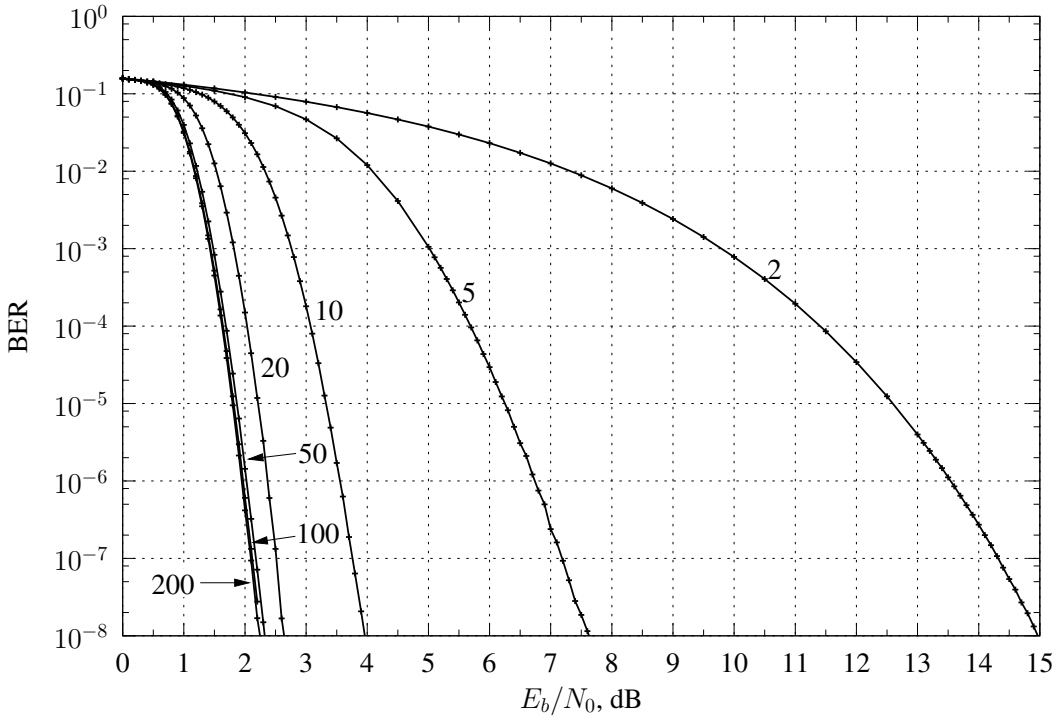


Figure 11. Performance of $k = 1024$, rate 1/2 AR4JA LDPC coded BPSK/QPSK, when decoded with a maximum of 2, 5, 10, 20, 50, 100, and 200 iterations.

A quantizer of the form

$$Q(x) = \begin{cases} 127 & \text{if } Cx \geq 127 \\ -127 & \text{if } Cx \leq -127 \\ \text{round}(Cx) & \text{otherwise} \end{cases} \tag{27}$$

is convenient, where C is a scale factor. In this way, $Q(x)$ takes on the integer values -127, -126, $\ldots, 126, 127$, and can be stored in an 8-bit register. This is a symmetric, uniform (equal step-size) quantizer, and for x in the grunular region, $Q(x) \approx Cx$. In the decoding algorithm, the value $Q(x)/C$ can be used wherever x would normally be used. Note that the quantizer represents zero exactly, which is helpful to represent the LLRs of untransmitted variable nodes. It also is symmetric about zero, so that a decoder will not be biased toward either positive or negative LLRs.

Since the quantizer output has maximum magnitude 127, it represents LLRs in the dynamic range $(-127/C, +127/C)$. Smaller values of C correspond to a larger dynamic range, which could aid the performance of a decoder. Given the fixed number (255) of quantizer levels, however, a larger dynamic range also means larger, coarser step size between quantizer levels. These two effects may be traded off to optimize performance. Figure 12 shows the performance of the $r = 4/5$, $k = 1024$ AR4JA code operating at $E_b/N_0 = 4$ dB, as a function of C. As can be seen, a value of $C = 8$ approximately optimizes performance. Hence, in the following numerical results, we use $C = 8$, which corresponds to a step-size of 1/8 and an LLR dynamic range of $\left(-15\frac{7}{8}, +15\frac{7}{8}\right)$.

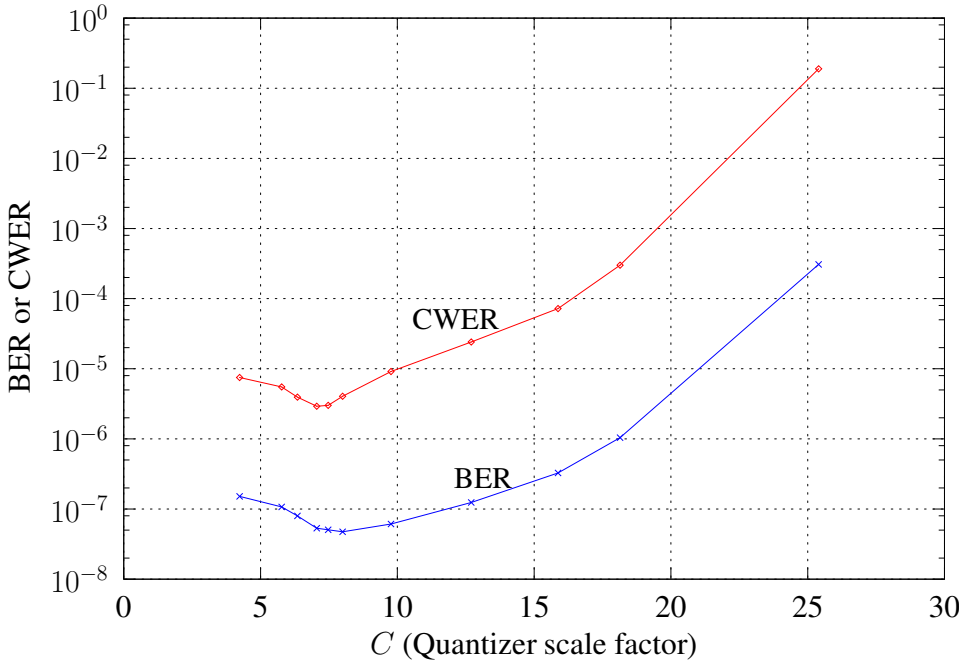


Figure 12. Performance of 8-bit decoder for $r = 4/5$, $k = 1024$ AR4JA code operating at $E_b/N_0 = 4$ dB, as a function of dynamic range of quantized LLRs.

C. Variable node processing

A given variable node receives LLR messages u_1, u_2, ..., u_d from d check nodes, where d is the degree of the variable node, along with an LLR λ from the demodulator. The message the variable node sends back to the jth of the d check nodes connected to it is given by

$$v_j = \lambda + \sum_{\substack{i=1 \\ i \neq j}}^{d} u_i \tag{28}$$

Given quantized inputs $Q(\lambda)$ and $Q(u_i)$, which as described above are about 8 times their true LLR values and are clipped to ± 127, the outgoing quantized message may be computed as

$$Q(v_j) = \text{clip}\left(Q(\lambda) + \sum_{\substack{i=1 \\ i \neq j}}^{d} Q(u_i) \right) \tag{29}$$

where

$$\text{clip}(x) = \begin{cases} 127 & \text{if } x \geq 127 \\ -127 & \text{if } x \leq -127 \\ x & \text{otherwise} \end{cases} \tag{30}$$

This can also be written as

$$Q(v_j) = \text{clip}\,(U - u_j) \tag{31}$$

where $U \triangleq Q(\lambda) + \sum_{i=1}^{d} Q(u_i)$. This form is convenient because each of the outgoing messages $v_1, \ldots, v_d$ can be computed from U with a single subtraction.

1. Jones clipping

In an early FPGA LDPC decoder implementation by Chris Jones[8], U was clipped prior to the subtraction

$$Q(v_j) = \text{clip}\left(\text{clip}(U) - u_j\right) \tag{32}$$

Intuitively, this clipping seems undesirable because, for example, if all of the incoming messages are large, including u_j, then the outgoing message will be near zero. Without the clipping of U, the message $Q(v_j)$ would be large, as is intuitively desirable.

Despite the intuition about the detrimental effect of this "Jones clipping," it turns out that the overall effect is to improve performance because such clipping apparently helps the decoder dig itself out of trapping sets in which it otherwise would get stuck. The effect may be analogous to simulated annealing, in which the algorithm occasionally moves in the opposite direction of the gradient in order to dig itself out of a local minimum. A solid theoretical understanding of this is lacking, however.

The performance improvement can be seen in the green curve labeled "with Jones clipping" in Figure 13. The blue curve is a nominal 8-bit decoder, and shows an error floor beginning at about CWER= 10^{-4}. Introducing Jones clipping reduced the error floor by one decade, to about CWER=10^{-5}. As we shall see below, this reduced-floor performance can be improved even more by carefully utilizing additional optimizations.

2. Clipping degree-1 variable nodes.

When channel symbol LLRs for degree-1 variable nodes are not clipped to levels below the maximum magnitude of check node messages, an error floor results[9]. The reason for the floor is that a strong but wrong channel symbol LLR is not able to be overcome by the single message from the check node. For the (1024,4/5) code with 128 degree-1 variable nodes, channel symbol LLRs clipped to ± 15.875, and a decoder with maximum check node message 15.125, the theoretical floor[10], $128Q((4E_s/N_0 + 15.125)/\sqrt{8E_s/N_0})$, is shown in light blue Figure 13. The theoretical floor reaches a maximum of approximately 2.4×10^{-6} at $E_b/N_0 \approx 6.7$ dB, and then trends lower at higher SNR.

Altering the decoder to clip degree-1 variable nodes to to 116/8=14.5 made little difference in the error floor, as seen in the red curve labeled "degree-1 clipping" in Figure 13, because the degree-1 problem was not the dominant flooring effect in this decoder in the region simulated.

---

[8]Chris Jones, personal communication.

[9]Chris Jones and Sam Dolinar, Monthly Management Review for the IND Technology Program, October 2004.

[10]Sam Dolinar, personal communication.

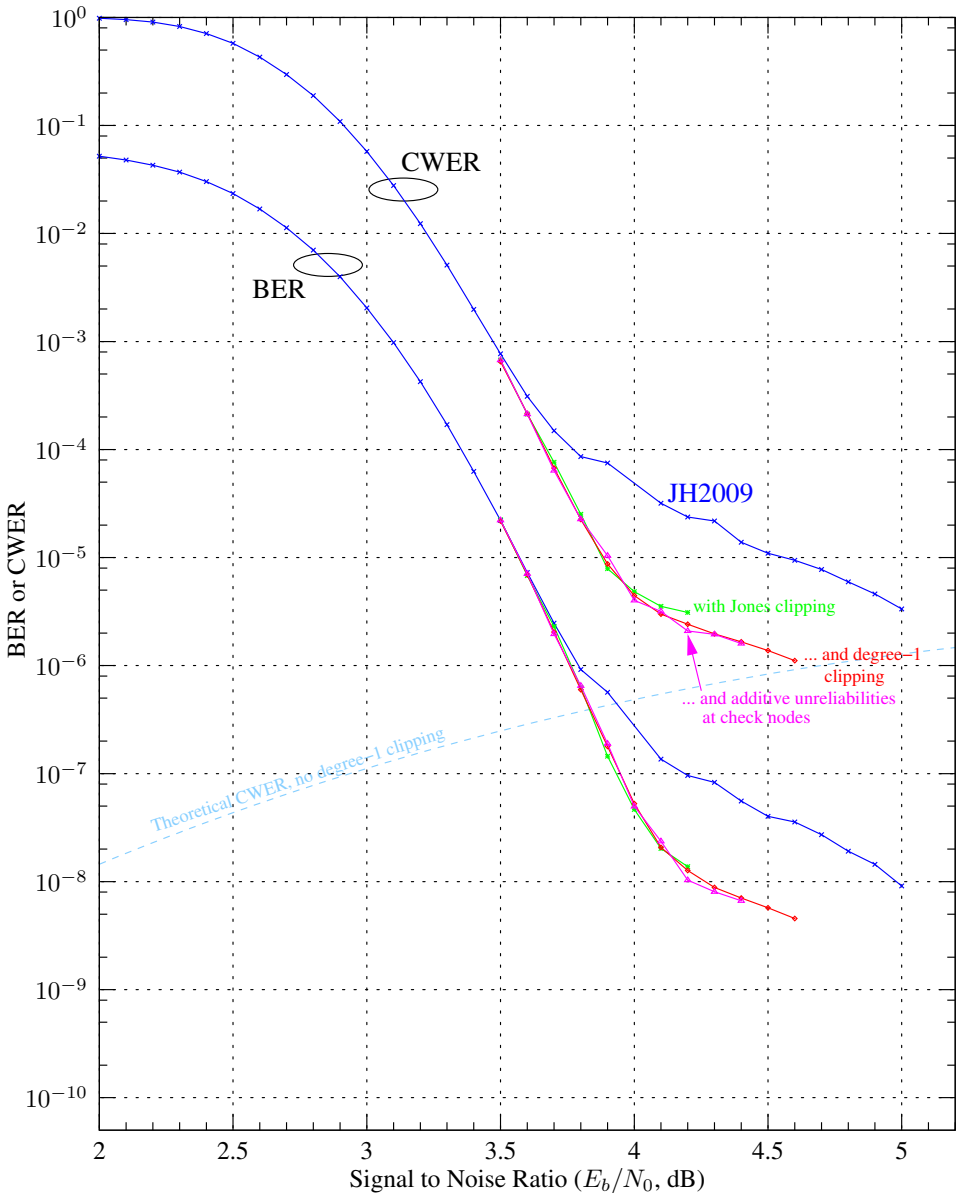


Figure 13. A few (1024,4/5) AR4JA decoder variants.

D. Check node processing

A given check node receives messages v_1, v_2, ..., v_d from d variable nodes, where d is the degree of the check node. The message the check node sends back to the jth of the d variable nodes connected to it is given by

$$u_j = 2\tanh^{-1}\left(\prod_{\substack{i=1\\ i\neq j}}^{d}\tanh\frac{v_i}{2}\right) \tag{33}$$

This can be computed by repetitively applying the function

$$\min{}^*(x,y) \stackrel{\triangle}{=} 2\tanh^{-1}\left[\tanh\left(\frac{x}{2}\right)\tanh\left(\frac{y}{2}\right)\right] \tag{34}$$

$$= \mathrm{sgn}(xy)\left[\min(|x|,|y|) - \ln\left(1+e^{-(||x|-|y||)}\right) + \ln\left(1+e^{-(|x|+|y|)}\right)\right] \tag{35}$$

1. Quantized min*

The second ln term of min* is smaller than the first, and can be ignored. The first ln term can be quantized using the approximation

$$\ln\left(1+e^{-||x|-|y||}\right) \approx \frac{1}{8}\mathrm{round}\left[8\ln\left(1+e^{-||x|-|y||}\right)\right] \tag{36}$$

With quantized inputs $Q(x)/8$ and $Q(y)/8$ in place of x and y, this is nonzero only when $||Q(x)|-|Q(y)|| \leq 21$, so a length 22 look-up table can implement this approximation. Thus, the entire min* approximation can be computed with a few comparisons, one subtraction, and no multiplications, logarithms, or exponentials.

In some implementations, such as a software decoder on a standard desktop, it is efficient to replace the comparisons, small look-up table, and subtraction with a single look-up table. With the 8-bit quantized values, an unsigned min* table has $128 \times 128 = 16384$ 1-byte entries, and a signed min* table has $256 \times 256 = 65536$ 1-byte entries, which is within the reach of typical computing platforms.

2. Exact min*.

When a full look-up table is used for min*, there is no need to use an approximation as in Equation (36). Instead the table can simply contain the entries

$$Q(\min{}^*(Q(x),Q(y)) = Q\left\{2\tanh^{-1}\left[\tanh\left(\frac{Q(x)}{2C}\right)\tanh\left(\frac{Q(y)}{2C}\right)\right]\right\} \tag{37}$$

which can be conveniently computed once, ahead of time. This is equivalent to Equation (34), using quantized inputs. Note, using the approximation (36) for both log terms of Equation (35) is not equivalent to Equation (37), because Equation (36) quantizes the log term separately, introducing quantization noise twice, whereas Equation (37) does not quantize until the end of the full computation.

Nevertheless, this more exact min* computation made no discernible difference in the simulated error floor.

3. Additive unreliability at check node.

The rate 4/5 AR4JA codes have degree-18 check nodes. To compute a min* function of 17 variables, multiple 2-input min* functions are repeatedly computed, using a tree-structure. Since each min* involves quantization noise, the total quantization noise for the min* with 17 variables could be significant. As an alternative, each reliability message v_i from a variable node can be transformed to an unreliability $\Psi(v_i) = \ln(\tanh(v_i))$, so that the product in Equation (33) becomes a summation

$$u_j = \Psi\left(\sum_{\substack{i=1 \\ i\neq j}}^{d} \Psi(v_i)\right) \tag{38}$$

Here, we have made use of the fact that $\Psi(\cdot)$ is a self-inverse function. With quantized inputs and outputs this becomes

$$Q(u_j) = Q\left[\Psi\left(\sum_{\substack{i=1 \\ i\neq j}}^{d} \Psi\left(\frac{Q(v_i)}{C}\right)\right)\right] \tag{39}$$

In this form, the addition can be performed without introducing quantization noise beyond that present in the inputs, and the result is transformed back to a reliability and re-quantized only at the end of the computation. The overall quantization noise is less using this method. This alteration had no discernible effect on error-floor performance, as seen in the magenta curve in Figure 13. Since this optimization also led to a slower software, it was not used in the numerical in the remainder of this article.

4. Partial hard limiting check node messages

One additional decoder variation made a big difference in the error floor performance. Messages from each check node were *partially hard-limited*, so that every message from a check node which would otherwise have a quantized magnitude at least 100 was re-assigned to have maximum magnitude (127). This resulted in the performance in the red curve in Figure 10. As can be seen, the floor was reduced to about CWER=3×10^{-8} and BER=3×10^{-10}, with no loss in the waterfall region. The average number of iterations in the waterfall region is the same as for the JH2009 decoder, so this decoder seems to be a promising candidate for low-complexity error-floor mitigation.

We may offer some limited reasoning for why the check-node hard-limiter helps improve performance. The lower floor means that the decoder is handling trapping sets better than the JH2009 decoder. Consider a trapping set V of incorrectly converged variable nodes, with a set C of neighboring check nodes, each connected to V an odd number of times (i.e., a $(|V|, |C|)$ trapping set). The check nodes in C are unsatisfied. In general, a node of V may receive messages from nodes in C and nodes not in C. If the decoder is stuck in the trapping set, the (correct) messages from nodes in C are not powerful enough to overcome the (incorrect) messages from nodes not in C. Because of how C is connected to V, the messages from check nodes in C tend to start converging slightly faster than those not in

C. By hard-limiting the messages from all check nodes above 100, the unsatisfied checks are able to more quickly correct incorrect nodes in V. The interaction of Jones clipping with the partial hard-limiter may also be important.

5. Other variations

Various other damping, amplifying, optimal processing of cycles, and iterative demodulation-decoding may also be incorporated. These may lead to additional performance improvements.

E. Software Implementation

Software was written in C to implement the encoder, bit-mapper, modulator, noise generator, demodulator, LLR computation, and decoder for each combination of code, modulation, bit-mapping type, and demodulation type shown in Table 1. Additional support for random message generation, noise generation, and gathering performance statistics was also included. The decoder uses LLRs quantized to eight bits.

The same encoder/decoder software is used for all nine codes. Prior to simulating the coded modulation, the software reads an initialization file that defines the protograph LDPC code's input and output length, circulant size, number of check and variable nodes in the protograph, number of edges in the protograph, a compact representation of the generator matrix, and an edgelist describing the parity check protograph and circulant offsets.

Table 2 shows the encoding and decoding speed of the C simulations, when compiled with a GNU C compiler on a typical desktop PC (a 3 GHz Intel Xeon processor running linux). The decoder is an 8-bit message passing decoder that stops iterating when a codeword is found. Because more iterations are needed at lower signal-to-noise ratios (SNRs), the speed of such a variable iterations decoder is sensitive to the SNR. The speeds reported in the table refer to a simulation with BPSK modulation, soft decisions, and operation at the E_b/N_0 shown, which in each case corresponds to operation at a codeword error rate of about 10^{-4} and represents a reasonable lower limit on the E_b/N_0 at which the decoder would be operated in practice. The software simulation was found to spend only a small fraction of its running time computing LLRs. Most of the time is spent performing decoder iterations. This is true even with the high order modulations such as 16-APSK and 32-APSK, where exact LLR computations amounted to only about 5 percent of the overall simulation time. As a result, the numerical results reported in this article used the exact LLR expression of Equation (14), and not the lower-complexity approximate LLR expressions developed in Section V.

We also developed a separate MATLAB implementation of equivalent functionality. The MATLAB implementation was found to run about 50 times slower. Simulation results reported in the article were collected with the C software.

VII. Numerical Results

We are now ready to present the main numerical results of the article: the performance of AR4JA codes when used with a variety of modulations, an optimized bit-mapping, an optimum demodulator (LLR computation), and the optimized decoder algorithms described in the sections above.

A. Performance of AR4JA coded BPSK, QPSK, 8-PSK, 16-APSK, and 32-APSK

Figure 14 shows the performance of AR4JA coded BPSK or QPSK on an AWGN channel, demodulated with an exact LLR computation and quantized to 8 bits, and decoded using up to a maximum of 200 iterations. BERs and CWERs are shown for codes of input codeword lengths $k = 1024$, $k = 4096$, and $k = 16384$ and rates 1/2, 2/3, and 4/5. These simulation results are in agreement with those reported elsewhere [1], except that the error floors have been eliminated.

Figure 15 shows the performance of AR4JA LDPC codes as before except that 8-PSK with a Gray mapping is used. BERs and CWERs are shown for codes of input codeword lengths $k = 1024$, $k = 4096$, and $k = 16384$ and rates 1/2, 2/3, and 4/5.

Figures 16 and 17 show the performance of AR4JA as before, except that 16-APSK and 32-APSK, respectively, with the DVB-S2 mapping is used. BERs and CWERs are shown for codes of input codeword lengths $k = 1024$, $k = 4096$, and $k = 16384$ and rates 1/2, 2/3, and 4/5.

B. Hard decision Decoding

Figures 18, 19, and 20 show the loss when the demodulator uses hard decision decoding. When taking a hard-decision input, the decoder uses Equation (21) as its LLR. The results shown are for the nine AR4JA codes used with BPSK on an AWGN channel. For all nine codes, the loss due to hard decision decoding is seen to be about 1.6 dB at CWER = 10^{-4}.

VIII. Conclusions and Future Work

This article provides a large set of simulation results for LDPC codes in combination with several modulations. The numerical results are consistent with previous results [1,3], except that a new partial hard-limiter for check node messages has been introduced to eliminate error floors. The simulation results provide a foundation for the design of variable coded modulation (VCM) or adaptive coded modulation (ACM) schemes.

Performance depends on optimization of bit-to-symbol mapping in the modulator, LLR computation by the demodulator, and on the decoder's quantization dynamic range and step-size, variable node clipping strategy, check node partial hard-limiting, and number of iterations. With careful optimizations, error floors can be avoided down to below CWER=10^{-6}.

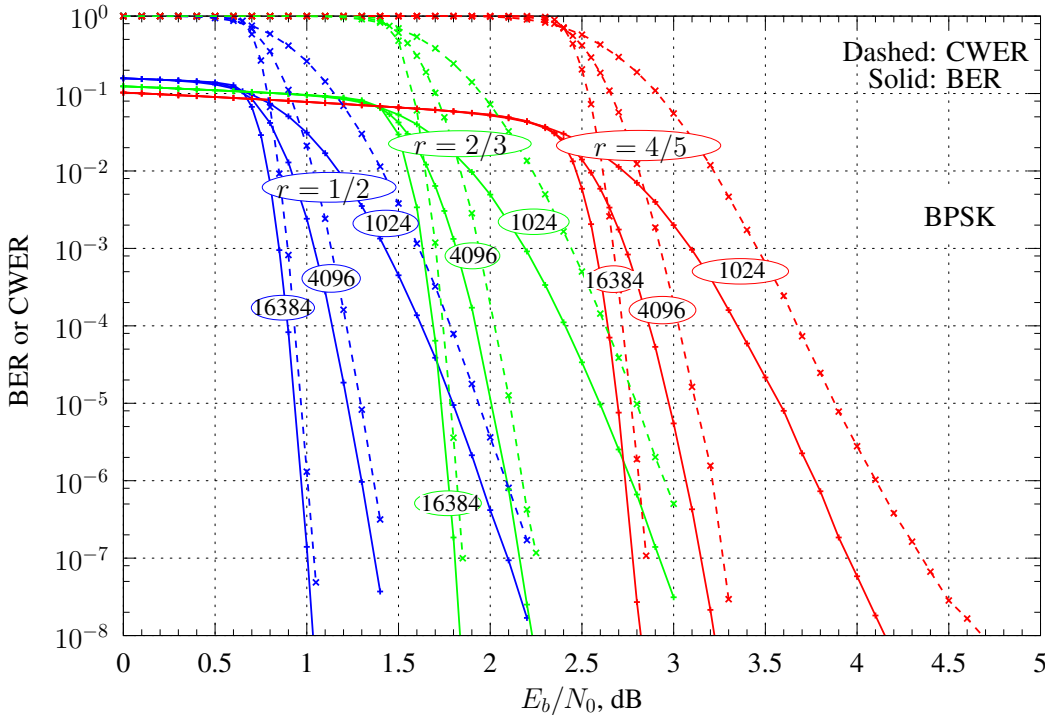


Figure 14. Performance of AR4JA LDPC coded BPSK/QPSK.

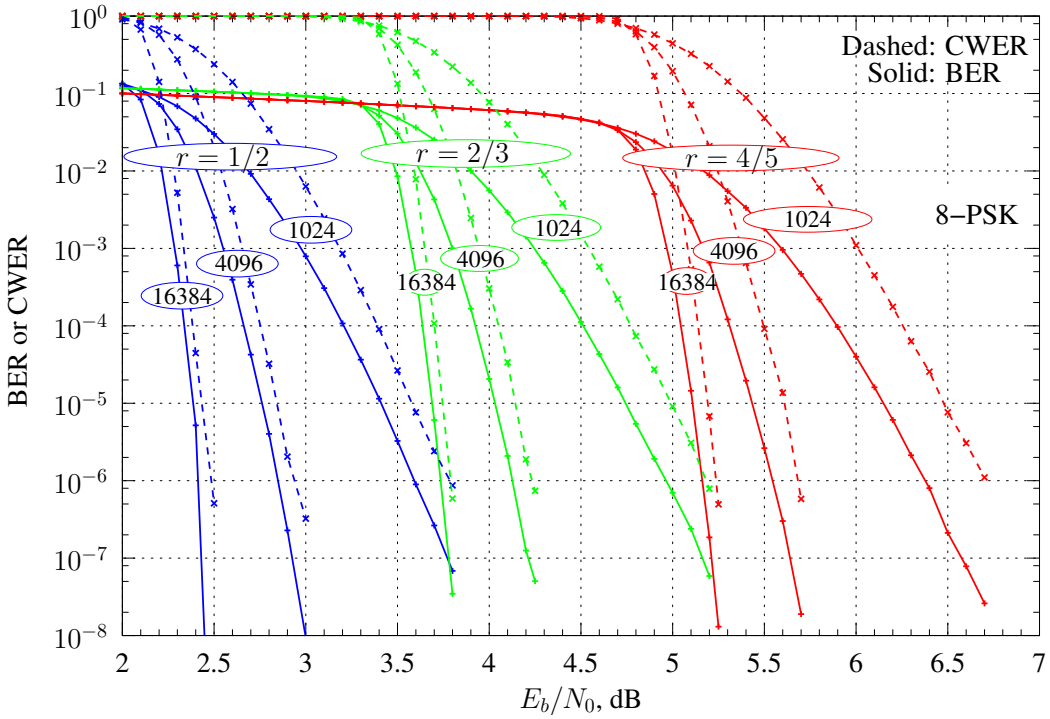


Figure 15. Performance of AR4JA LDPC coded 8-PSK.

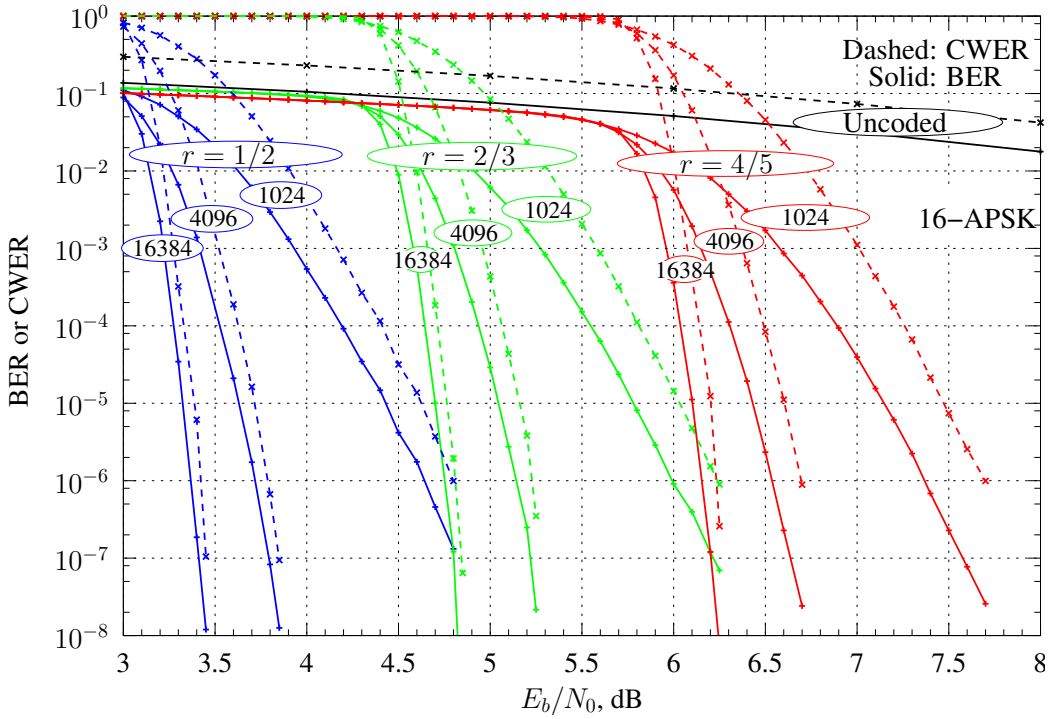


Figure 16. Performance of AR4JA LDPC coded 16-APSK.

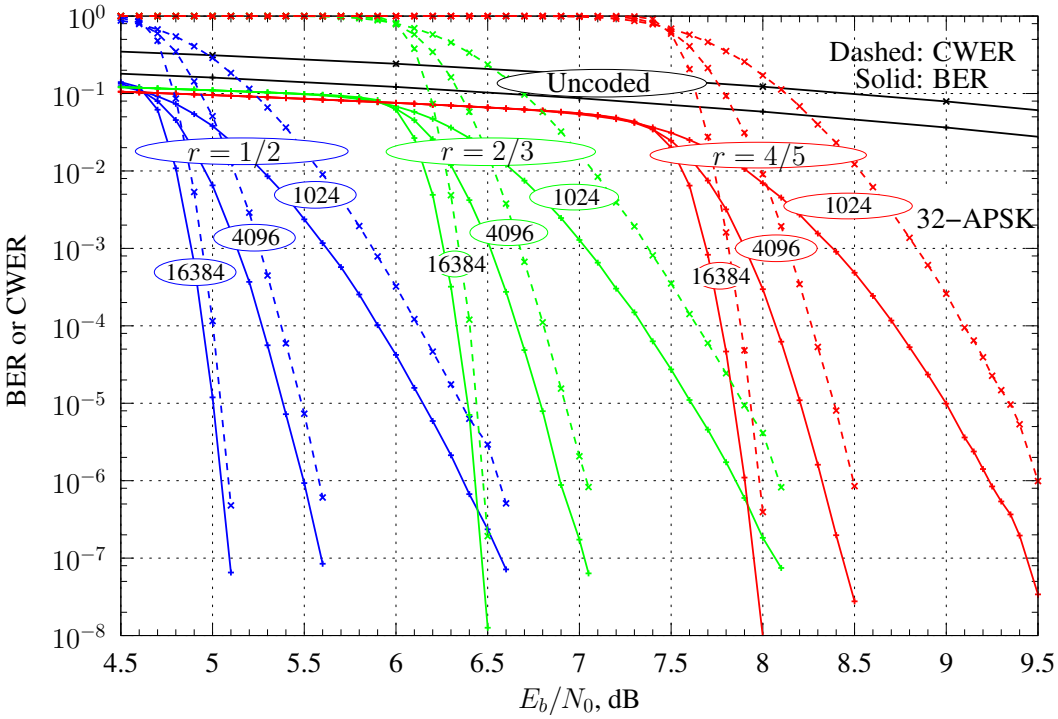


Figure 17. Performance of AR4JA LDPC coded 32-APSK.

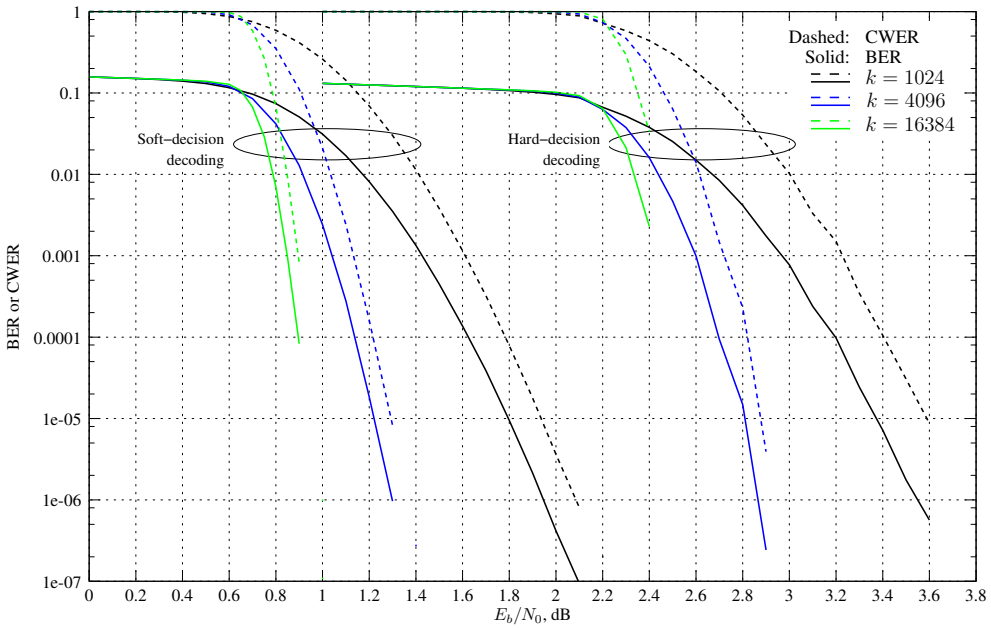


Figure 18. Performance of rate 1/2 AR4JA LDPC coded BPSK/QPSK using hard decision demodulator.

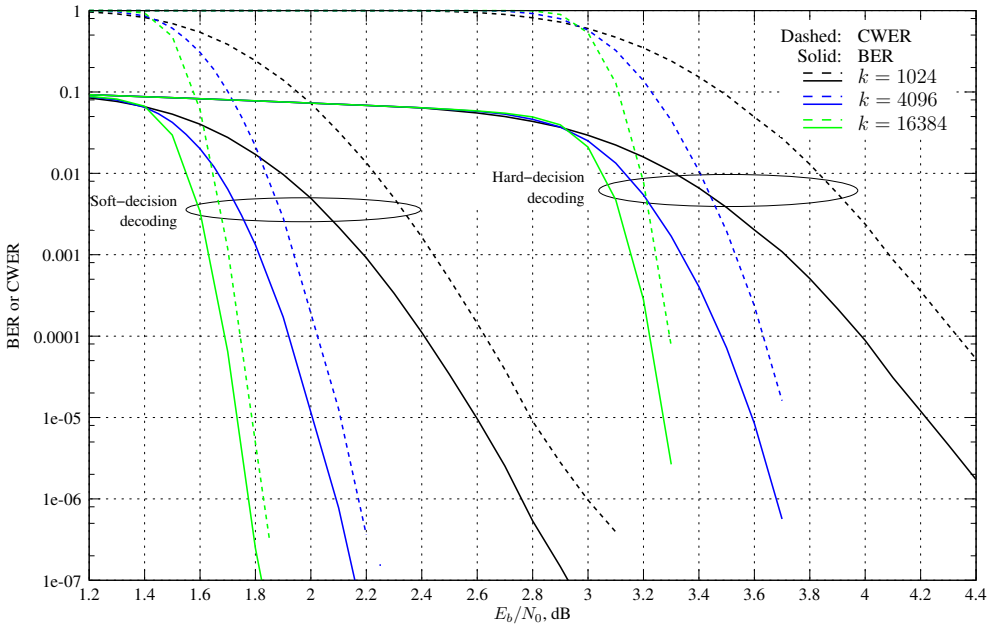


Figure 19. Performance of rate 2/3 AR4JA LDPC coded BPSK/QPSK using hard decision demodulator.

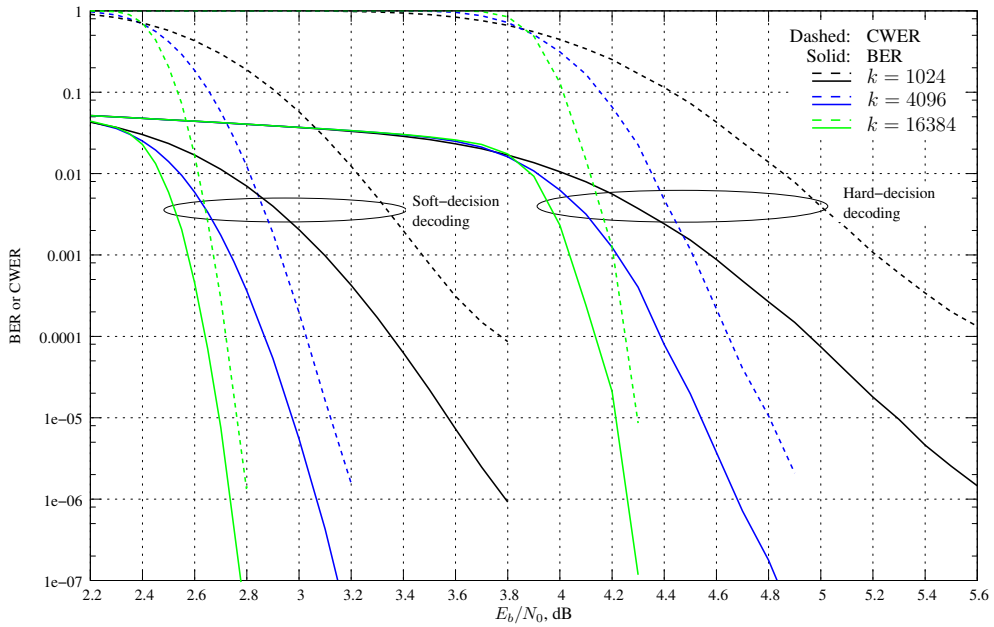


Figure 20. Performance of rate 4/5 AR4JA LDPC coded BPSK/QPSK using hard decision demodulator.

Error floors may be lower, as they were not reached with the simulations conducted here. Performance is not sensitive to ring ratios used in 16-APSK and 32-APSK, nearest neighbor approximations to the LLR, and maximum iterations beyond about 200. Use of an interleaver may be avoided without performance degradation.

Iterative demodulating and decoding was not attempted here and may improve performance further.

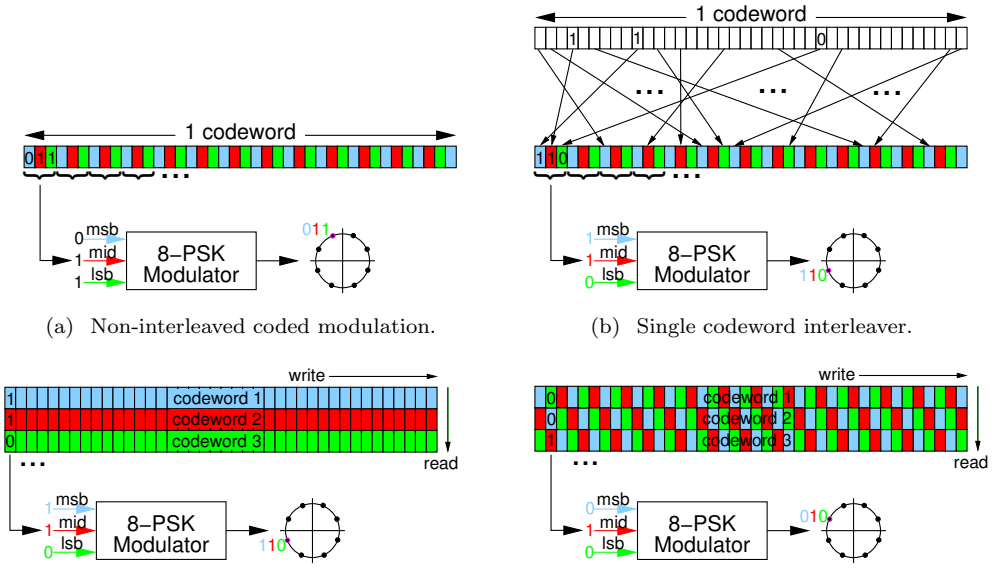
Acknowledgments The author thanks Kenneth Andrews, Sam Dolinar, Dariush Divsalar, and Chris Jones for teaching me most of what I know about AR4JA LDPC codes, and especially Kenneth Andrews for providing MATLAB code for the $k = 1024$, $r = 1/2$ AR4JA LDPC decoder, and Christopher Jones for providing edge lists for the AR4JA codes.

Appendix: A word about interleavers

This article did not make use of an interleaver – each set of adjacent codebits was grouped and used as input to the modulator, as shown in Figure 21(a) for 8-PSK. The blue, red, and green colors correspond the most significant bit, middle bit, and least significant bit shown in the 8-PSK signal constellation in Figure 2(c). When a codeword is not a multiple of the number of bits per modulation symbol, the modulator input can be padded with zeros to generate the final symbol, or combined with the first bits of the following codeword.

Not using an interleaver may make a code vulnerable to losses when used with higher order modulations, because a weakly received modulation symbol may give rise to multiple poor codebit LLRs. An interleaver helps distribute these bursts of poor LLRs across multiple codewords, instead of bunching them in a single codeword. Codebits are passed through an interleaver, π, prior to modulation, and a de-interleaver, π^{-1}, after demodulation, as shown in Figure 22.

In the *single codeword interleaver*, the bits within a codeword are re-ordered arbitrarily, as shown in Figure 21(b), prior to being mapped to modulation symbols. In principle, any



(a) Non-interleaved coded modulation.

(b) Single codeword interleaver.

(c) Block interleaver (not recommended).

(d) Block interleaver with bit-reordering.

Figure 21. Interleavers for coded 8-PSK.

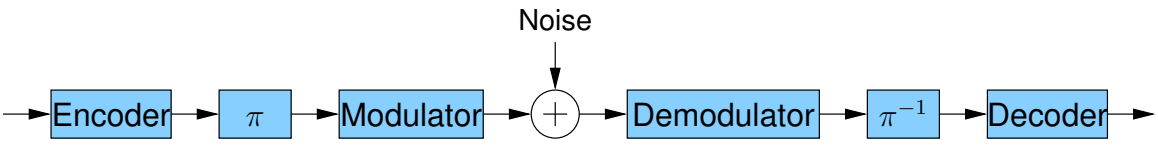


Figure 22. Signal flow when an interleaver is used.

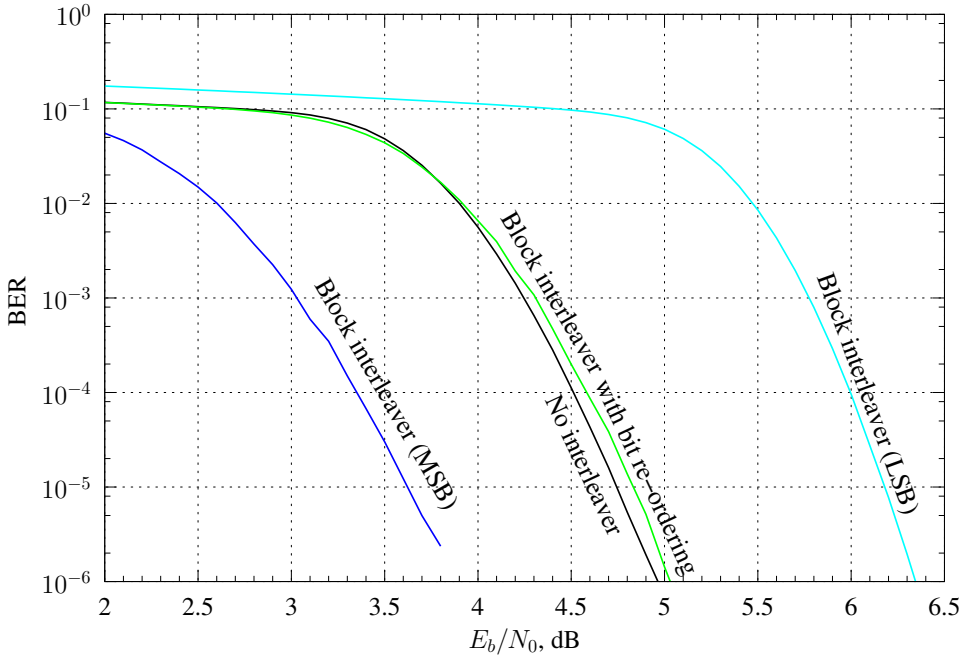


Figure 23. Performance of coded modulation when not interleaved, block interleaved, and block interleaved with bit re-ordering.

interleaver of this type may simply be incorporated into the definition of the LDPC code, with no need to refer to an additional interleaver. However, it was convenient to define the AR4JA codes in the way they were because they have the quasi-cyclic property, which simplifies the encoding process. Changing the definition of the code to reorder the bits would destroy this property.

In a *block interleaver*, codewords are written in rows and read out in columns, as shown in Figure 21(c), again for 8-PSK. In the usual type of block interleaver, the first codeword would always correspond to the msb, the second codeword to the middle bit, and the third codeword to the lsb. As noted in Figures 6 and 7, the lsb of Gray-coded 8-PSK has worse distance properties, which means that the error rate for the codeword 3 using the lsbs will be much worse. This is shown in Figure 23. The performance of codewords mapped to the MSB is very good, while those mapped to the LSB are quite poor, and the average performance would be dominated by the poor LSB performance. As a result, a block interleaver of this type should not be used with modulations whose bits have different distance properties.

Additionally, Figure 23 indicates that a block interleaver with bit re-ordering does not offer an advantage over the non-interleaved coded modulation. This implies that the AR4JA codes are inherently resilient to the bursts of poor LLRs that result from the use of a higher order modulation. This may be because the number of bits per modulation symbol, five or less, is small compared to the codeword length, which is 1280 or longer.

References

[1] K. S. Andrews, D. Divsalar, S. Dolinar, J. Hamkins, C. R. Jones, and F. Pollara, "The development of turbo and LDPC codes for deep-space applications," *Proceedings of the IEEE*, vol. 95, no. 11, pp. 2142–2156, Nov. 2007.

[2] "Low density parity check codes for use in near-Earth and deep space. CCSDS 131.1-O-2. Orange Book, Issue 2," Sept. 2007, http://public.ccsds.org/publications/archive/131x1o2e2.pdf.

[3] M. Cheng, D. Divsalar, and S. Duy, "Structured low-density parity-check codes with bandwidth efficient modulation," in *Proceedings of SPIE Conference on Defense Security and Sensing*, Apr. 2009.

[4] S. Dolinar, D. Divsalar, and F. Pollara, "Code performance as a function of block size," *TDA Progress Report*, vol. 42, no. 133, May 1998.

[5] Kenneth Andrews, Sam Dolinar, and Jeremy Thorpe, "Encoders for block-circulant LDPC codes," in *Proc. IEEE Int. Symp. Inf. Theory*, Sept. 2005, pp. 2300–2304.

[6] "ETSI EN 302 307 v1.1.2, "Digital Video Broadcasting (DVB): Second generation framing structure, channel coding and modulation systems for broadcasting, interactive services, news gathering and other broadband satellite applications"," June 2006.

[7] E. Agrell, J. Lassing, E.G. Strom, and T. Ottosson, "On the optimality of the binary reflected Gray code," *IEEE Trans. Inform. Theory*, vol. 50, no. 12, pp. 3170–3182, 2004.

[8] Tom Richardson and Ruediger Urbanke, *Modern Coding Theory*, Cambridge University Press, 2008.