Digital Video Broadcasting

Matthew C. Valenti, Shi Cheng, and Rohit Iyer Seshadri

West Virginia University
Morgantown, WV 26506-6109 USA
{mvalenti,shic,iyerr}@csee.wvu.edu

The Digital Video Broadcasting (DVB) Project was founded in 1993 by the European Telecommunications Standards Institute (ETSI) with the goal of standardizing digital television services. Its initial standard for satellite delivery of digital television, dubbed DVB-S, used a concatenation of an outer (204,188) byte shortened Reed Solomon code and an inner constraint length 7, variable rate (r ranges from 1/2 to 7/8) convolutional code [1].

The same infrastructure used to deliver television via satellite can also be used to deliver Internet and data services to the subscriber. Internet over DVB-S is a natural competitor against cable modem and DSL technology, and its universal coverage allows even the most remote areas to be served. Because DVB-S only provides a downlink, an uplink is also needed to enable interactive applications such as web browsing. The uplink and downlink need not be symmetric, since many Internet services require a faster downlink.

One alternative for the uplink is to use a telephone modem, but this does not allow for always-on service, has modest data rates, and can be costly in remote areas. A more attractive alternative is for the subscriber equipment to transmit an uplink signal back to the satellite over the same antenna used for receiving the downlink signal. However, given the small antenna aperture and requirement for a low-cost, low-power amplifier, there is very little margin on the uplink. Therefore, strong FEC coding is desired. For this reason, the DVB Project has adopted turbo codes for the satellite return channel in its DVB-RCS (Return Channel via Satellite) standard [2].

At the same time that the DVB Project was developing turbo coding technology for the return channel, it was updating the downlink with modern coding technology. The latest standard, called DVB-S2, replaces the concatenated Reed-Solomon/convolutional coding approach of DVB-S with a concatenation of an outer BCH code and inner low density parity check (LDPC) code [3]. The result is a 30% increase in capacity over DVB-S. In this chapter, the coding strategies used by both DVB-RCS and DVB-S2 are discussed.

1 DVB-RCS

The DVB-RCS turbo code was optimized for short frame sizes and high data rates. Twelve frame sizes are supported raging from 12 bytes to 216 bytes, including a 53 byte frame compatible with ATM and a 188 byte frame compatible with both MPEG-2 and the original DVB-S standard. The return link supports data rates from 144 kbps to 2 Mbps and is shared among terminals by using muti-frequency time-division multiple-access (MF-TDMA) and demand-assigned multiple-access (DAMA) techniques. Eight code rates are supported, ranging from $r = 1/3$ to $r = 6/7$.

Like the turbo codes used in other standards, a pair of constituent RSC encoders is used along with log-MAP or max-log-MAP decoding [4]. The decoder for each constituent code performs best if the encoder begins and ends in a known state, such as the all-zeros state. This can be accomplished by independently terminating the trellis of each encoder with a tail which forces the encoder back to the all-zeros state. However, for the small frame lengths supported by DVB-RCS, such a tail imposes a non-negligible reduction in code rate and is therefore undesirable. As an alternative to terminating the trellis of the code, DVB-RCS uses circular recursive systematic convolutional (CRSC) encoding [5], which is based on the concept of *tailbiting* [6]. CRSC codes do not use tails, but rather are encoded in such a way that the ending state matches the starting state.

Most turbo codes use binary encoders defined over GF(2). However, to facilitate faster decoding in hardware, the DVB-RCS code uses *duobinary* constituent encoders defined over GF(4) [7]. During each clock cycle, the encoder takes in two data bits and outputs two parity bits so that, when the systematic bits are included, the code rate is $r = 2/4$. In order to avoid parallel transitions in the code trellis, the memory of the encoder must exceed the number of input bits, and so DVB-RCS uses constituent encoders with memory three (a constraint length of four).

There are several benefits to using duobinary encoders. First, the trellis contains half as many states as a binary code of identical constraint length (but the same number of edges) and therefore needs half as much memory and the decoding hardware can be clocked at half the rate as a binary code. Second, the duobinary code can be decoded with the suboptimal but efficient max-log-MAP algorithm at a cost of only about 0.1-0.2 dB relative to the optimal log-MAP algorithm. This is in contrast with binary codes, which lose about 0.3-0.4 dB when decoded with the max-log-MAP algorithm [8]. Additionally, duobinary codes are less impacted by the uncertainty of the starting and ending states when using tailbiting and perform better than their binary counterparts when punctured to higher rates.

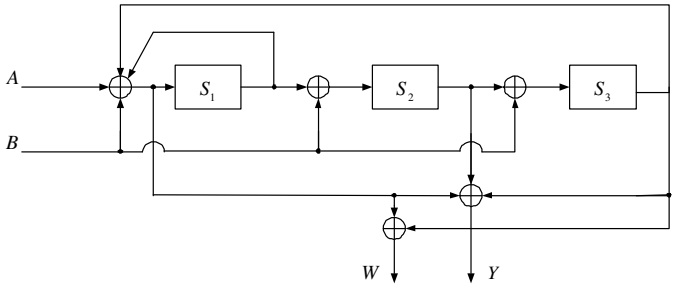

Fig. 1. Duobinary CRSC constituent encoder used by DVB-RCS.

1.1 Encoding

The CRSC constituent encoder used by DVB-RCS is shown in Fig. 1. The encoder is fed blocks of k message bits which are grouped into $N = k/2$ *couples*. The number of couples per block can be $N \in \{48, 64, 212, 220, 228, 424, 432, 440, 752, 848, 856, 864\}$. The number of bytes per block is $N/4$. In Fig. 1, A represents the the first bit of the couple, and B represents the second bit. The two parity bits are denoted W and Y. For ease of exposition, subscripts are left off the figure, but below a single subscript is used to denote the time index $k \in \{0, ..., N-1\}$ and an optional second index is used on the parity bits W and Y to indicate which of the two constituent encoders produced them.

Let the vector $\mathbf{S}_k = [S_{k,1}\ S_{k,2}\ S_{k,3}]^T, S_{k,m} \in \{0, 1\}$ denote the state of the encoder at time k. Note that although the inputs and outputs of the encoder are defined over GF(4), only binary values are stored within the shift register and thus the encoder has just eight states. The encoder state at time k is related to the state at time $k-1$ by

$$\mathbf{S}_{k+1} = \mathbf{G}\mathbf{S}_k + \mathbf{X}_k \tag{1}$$

where

$$\mathbf{X}_k = \begin{bmatrix} A_k + B_k \\ B_k \\ B_k \end{bmatrix} \tag{2}$$

and

$$\mathbf{G} = \begin{bmatrix} 1\ 0\ 1 \\ 1\ 0\ 0 \\ 0\ 1\ 0 \end{bmatrix} \tag{3}$$

Because of the tailbiting nature of the code, the block must be encoded twice by each constituent encoder. During the first pass at encoding, the encoder is initialized to the all-zeros state, $\mathbf{S}_0 = [0\ 0\ 0]^T$. After the block is

encoded, the final state of the encoder $\mathbf{S}_N$ is used to derive the *circulation state*

$$\mathbf{S}_c = \left(\mathbf{I} + \mathbf{G}^N\right)^{-1} \mathbf{S}_N \tag{4}$$

Where the above operations are over GF(2). Note that the matrix $\mathbf{I} + \mathbf{G}^N$ is not invertible if N is a multiple of the period of the encoder's impulse response (which is seven for this encoder). However, this is not a problem because none of the permitted values of N are multiples of seven. In practice, the circulation state $\mathbf{S}_c$ can be found from $\mathbf{S}_N$ by using a lookup table (which is specified in the standard). Once the circulation state is found, the data is encoded again. This time, the encoder is set to start in state $\mathbf{S}_c$ and will be guaranteed to also end in state $\mathbf{S}_c$.

The first encoder operates on the data in its natural order, yielding parity couples $\{W_{k,1}, Y_{k,1}\}$. The second encoder operates on the data after it has been interleaved. Interleaving is performed on two levels. First, interleaving is performed within the couples, and second, interleaving is performed between couples. Let $\{A'_k, B'_k\}$ denote the sequence after the first level of interleaving and $\{A''_k, B''_k\}$ denote the sequence after the second level of interleaving. In the first level of interleaving, every other couple is reversed in order, i.e. $(A'_k, B'_k) = (B_k, A_k)$ if k is even, otherwise $(A'_k, B'_k) = (A_k, B_k)$. In the second level of interleaving, couples are permuted in a pseudorandom fashion. The exact details of the second level permutation can be found in the standard [2].

After the two levels of interleaving, the second encoder (which is identical to the first) encodes the sequence $\{A''_k, B''_k\}$ to produce the sequence of parity couples $\{W_{k,2}, Y_{k,2}\}$. As with the first encoder, two passes of encoding must be performed, and the second encoder will have its own independent circulation state. To create a rate $r = 1/3$ turbo code, a codeword is formed by first transmitting all the uninterleaved data couples $\{A_k, B_k\}$, then transmitting $\{Y_{k,1}, Y_{k,2}\}$ and finally transmitting $\{W_{k,1}, W_{k,2}\}$. The bits are transmitted using QPSK modulation, so there is a one-to-one correspondence between couples and QPSK symbols. Alternatively, the code word can be transmitted by exchanging the parity and systematic bits, i.e. $\{Y_{k,1}, Y_{k,2}\}$, followed by $\{W_{k,1}, W_{k,2}\}$ and finally $\{A_k, B_k\}$.

Code rates higher than $r = 1/3$ are supported through the puncturing of parity bits. To achieve $r = 2/5$, both encoders maintain all the Y_k but delete odd-indexed W_k. For rate $1/2$ and above, the encoders delete all W_k. For rate $r = 1/2$, all the Y_k bits are maintained, while for rate $r = 2/3$ only the even-indexed Y_k are maintained, and for rate $r = 4/5$ only every fourth Y_k is maintained. Rates $r = 3/4$ and $6/7$ maintain every third and sixth Y_k respectively, but are only exact rates if N is a multiple of three (otherwise the rates are slightly lower).

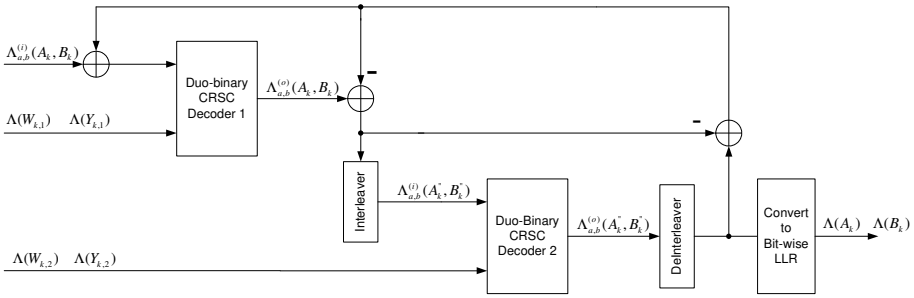

Fig. 2. A decoder for the DVB-RCS code.

1.2 Decoding

Decoding of the DVB-RCS code is complicated by the fact that the constituent codes are duobinary and circular. As with conventional turbo codes, decoding involves the iterative exchange of extrinsic information between the two component decoders. While decoding can be performed in the probability domain, the log-domain is preferred since the low complexity max-log-MAP algorithm can then be applied [4]. Unlike the decoder for a binary turbo code, which can represent each binary symbol as a single log-likelihood ratio, the decoder for a duobinary code requires *three* log-likelihood ratios. For example, the likelihood ratios for message couple (A_k, B_k) can be represented in the form

$$\Lambda_{a,b}(A_k, B_k) = \log \frac{P(A_k = a, B_k = b)}{P(A_k = 0, B_k = 0)} \tag{5}$$

where (a, b) can be $(0, 1)$, $(1, 0)$, or $(1, 1)$.

An iterative decoder that can be used to decode the DVB-RCS turbo code is shown in Fig. 2. The goal of each of the two constituent decoders is to update the set of log-likelihood ratios associated with each message couple. In the figure and in the following discussion, $\{\Lambda^{(i)}_{a,b}(A_k, B_k)\}$ denotes the set of LLRs corresponding to the message couple at the input of the decoder and $\{\Lambda^{(o)}_{a,b}(A_k, B_k)\}$ is the set of LLRs at the output of the decoder. Each decoder is provided with $\{\Lambda^{(i)}_{a,b}(A_k, B_k)\}$ along with the received values of the parity bits generated by the corresponding encoder (in LLR form). Using these inputs and knowledge of the code constraints, it is able to produce the updated LLRs $\{\Lambda^{(o)}_{a,b}(A_k, B_k)\}$ at its output.

As with binary turbo codes, extrinsic information is passed to the other constituent decoder instead of the raw LLRs. This prevents the positive feedback of previously resolved information. Extrinsic information is found by simply subtracting the appropriate input LLR from each output LLR, as indicated in Fig. 2.

The extrinsic information that is passed between the two decoders must be interleaved or deinterleaved so that it is in the proper sequence at the

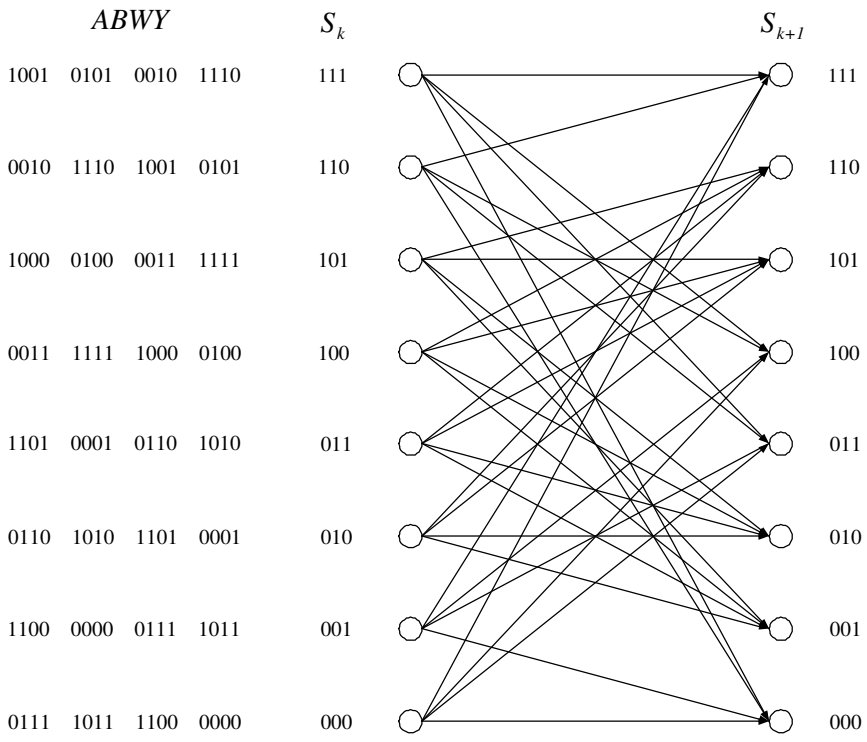

Fig. 3. Trellis associated with the duobinary CRSC constituent encoder used by DVB-RCS. The numbers on the left indicate the labels (A, B, W, Y) of the branches exiting each state. From left to right, the groups of numbers correspond to the exiting branches from top to bottom.

input of the other decoder. Interleaving and deinterleaving between the two constituent decoders must be done on a symbol-wise basis by assuring that the three likelihood ratios $\{\Lambda_{0,1}(A_k, B_k), \Lambda_{1,0}(A_k, B_k), \Lambda_{1,1}(A_k, B_k)\}$ belonging to the same couple are not separated.

The trellis for the duobinary constituent code is as shown in Fig. 3. The trellis contains eight states, with four branches entering and exiting each state. Note that this is in contrast with a conventional binary code which only has a pair of branches entering and exiting each state. The trellis contains two 4 by 4 butterflies, and because these two butterflies are independent, they can be processed in parallel. In the following, the i^{th} state is denoted by $\mathbf{S}_i$ where $i \in \{0, ..., 7\}$ for DVB-RCS. Note that the subscript i takes on a slightly different connotation depending on whether encoding or decoding is being discussed. When discussing encoding, the subscript was used to indicate a time step, but when discussing decoding the subscript indicates a particular state.

The extension of the log-MAP and max-log-MAP algorithms [4] to the duobinary case is fairly straightforward. Each branch must be labeled with the log-likelihood ratios corresponding to the systematic and parity couples associated with that branch. Because QPSK modulation is orthogonal, the LLR of message couple (A, B) can be initialized prior to being fed into the first decoder as $\Lambda_{a,b}^{(i)}(A_k, B_k) = a\Lambda(A_k) + b\Lambda(B_k)$, where $\Lambda(C) = \log[P(C = 1)/P(C = 0)]$. Because extrinsic information about the parity bits is not exchanged, the parity bits can always be decomposed in a similar manner. However, for the systematic bits, the three likelihood ratios defined in (5) must be calculated during each iteration and exchanged between the decoders.

Let $\gamma_k(\mathbf{S}_i \rightarrow \mathbf{S}_j)$ denote the branch metric corresponding to state transition $\mathbf{S}_i \rightarrow \mathbf{S}_j$ at time k. The branch metric depends on the message and parity couples that label the branch along with the channel observation and extrinsic information at the decoder input. In particular, if transition $\mathbf{S}_i \rightarrow \mathbf{S}_j$ is labelled by $(A_k, B_k, W_k, Y_k) = (a, b, w, y)$ then

$$\gamma_k(\mathbf{S}_i \rightarrow \mathbf{S}_j) = \Lambda_{a,b}^{(i)}(A_k, B_k) + w\Lambda(W_k) + y\Lambda(Y_k) \tag{6}$$

As with binary codes, the constituent decoder must perform a forward and a backward recursion. Let $\alpha_k(\mathbf{S}_i)$ denote the normalized forward metric at trellis stage k and state $\mathbf{S}_i$, while $\alpha'_{k+1}(\mathbf{S}_j)$ is the forward metric at trellis stage $k+1$ and state $\mathbf{S}_j$ prior to normalization. The forward recursion is

$$\alpha'_{k+1}(\mathbf{S}_j) = \underset{\mathbf{S}_i \rightarrow \mathbf{S}_j}{\max*} \{\alpha_k(\mathbf{S}_i) + \gamma_k(\mathbf{S}_i \rightarrow \mathbf{S}_j)\} \tag{7}$$

where the max$*$ operation[1] is performed over the four branches $\mathbf{S}_i \rightarrow \mathbf{S}_j$ leading into state $\mathbf{S}_j$ at time $k+1$. While the log-MAP algorithm uses the exact definition of max$*$, the max-log-MAP algorithm uses the approximation $\max*(x, y) \approx \max(x, y)$.

After computing $\alpha'_{k+1}(\mathbf{S}_j)$ for all $\mathbf{S}_j$ at time $k+1$, the forward metrics are normalized with respect to the metric stored in state zero

$$\alpha_{k+1}(\mathbf{S}_j) = \alpha'_{k+1}(\mathbf{S}_j) - \alpha'_{k+1}(\mathbf{S}_0) \tag{8}$$

Similarly, let $\beta_{k+1}(\mathbf{S}_j)$ denote the normalized backward metric at trellis state $k+1$ and state $\mathbf{S}_j$ and $\beta'_k(\mathbf{S}_i)$ denote the backward metric at trellis state k and state $\mathbf{S}_i$ prior to normalization. The backward recursion is

$$\beta'_k(\mathbf{S}_i) = \underset{\mathbf{S}_i \rightarrow \mathbf{S}_j}{\max*} \{\beta_{k+1}(\mathbf{S}_j) + \gamma_k(\mathbf{S}_i \rightarrow \mathbf{S}_j)\} \tag{9}$$

where max$*$ is over the four branches $\mathbf{S}_i \rightarrow \mathbf{S}_j$ exiting state $\mathbf{S}_i$ at time k. As with α, the β's are normalized with respect to the metric stored in state zero

[1] The max$*$ operation is defined in [9] as $\max*(x, y) = \max(x, y) + \log(1 + e^{-|x-y|})$. Multiple arguments imply a recursion of pairwise operations, i.e. $\max*(x, y, z) = \max*(x, \max*(y, z))$.

$$\beta_k(\mathbf{S}_i) = \beta'_k(\mathbf{S}_i) - \beta'_k(\mathbf{S}_0) \tag{10}$$

Because the encoders are circular, special care must be taken to initialize the forward and backward recursions. Since the starting and stopping states are identical, the code trellis can be visualized as a cylinder (see, for example, Fig. 1 in [10]). The forward recursion can be interpreted as going around the cylinder in the clockwise direction and the backward recursion as going around the cylinder in the counter-clockwise direction.

Several algorithms are presented in [10] for decoding circular/tailbiting convolutional codes. The most practical algorithm, called algorithm A3 in [10], begins by initializing the decoder so that all initial states are equally likely. The forward recursion is initialized so that $\alpha_0(\mathbf{S}_j) = 0, \forall \mathbf{S}_j$. The forward recursion then cycles through the entire trellis in the clockwise direction. If the encoder was terminated in a known state, then the forward recursion could halt once it reaches the end of the trellis. However, since the starting and ending states are not known, the forward recursion continues around the cylinder a second time. During the second cycle, the new value of the α_k's are compared against the same value computed during the first cycle, and the new α_k's are used to replace the ones from the last cycle. Once all the α_k's are close to the value from the last cycle, the forward recursion halts. The number of extra trellis sections beyond the first cycle around the cylinder is called the *wrap depth*. For long frames, the wrap depth is typically smaller than the frame length (so an entire second cycle does not need to be run). However, for short frames, a third or fourth cycle around the trellis cylinder could be required, i.e. the wrap depth could exceed N.

The backward recursion is executed in similar manner, with $\beta_N(\mathbf{S}_i) = 0, \forall \mathbf{S}_i$ and the decoder cycling around the cylinder in the counter-clockwise direction. After making one lap around the cylinder, the algorithm continues until the β_k's closely match the values computed during the previous lap.

After the forward and backward recursions have been completed, a full set of $\{\alpha_k\}$ and $\{\beta_k\}$ metrics will be stored in memory. The next step is for the decoder to use these metrics to compute the LLRs given by (5). This is accomplished by first computing the likelihood of each branch

$$Z_k(\mathbf{S}_i \to \mathbf{S}_j) = \alpha_k(\mathbf{S}_i) + \gamma_k(\mathbf{S}_i \to \mathbf{S}_j) + \beta_{k+1}(\mathbf{S}_j) \tag{11}$$

Next, the likelihood that message pair $(A_k, B_k) = (a, b)$ is calculated using

$$t_k(a,b) = \max_{\mathbf{S}_i \to \mathbf{S}_j : (a,b)} * \{Z_k\} \tag{12}$$

where the max$*$ operator is over the eight branches labelled by message couple (a, b). Finally, the LLR at the output of the decoder is found as

$$\Lambda^{(o)}_{a,b}(A_k, B_k) = t_k(a,b) - t_k(0,0) \tag{13}$$

where $(a, b) \in \{(0, 1), (1, 0), (1, 1)\}$.

After the turbo decoder has completed a fixed number of iterations or met some other convergence criterion, a final decision on the bits must be made. This is accomplished by computing the LLR of each bit in the couple (A_k, B_k) according to

$$\begin{aligned}\Lambda(A_k) &= \max\!*\left\{\Lambda_{1,0}^{(o)}(A_k, B_k), \Lambda_{1,1}^{(o)}(A_k, B_k)\right\} \\ &\quad - \max\!*\left\{\Lambda_{0,0}^{(o)}(A_k, B_k), \Lambda_{0,1}^{(o)}(A_k, B_k)\right\} \\ \Lambda(B_k) &= \max\!*\left\{\Lambda_{0,1}^{(o)}(A_k, B_k), \Lambda_{1,1}^{(o)}(A_k, B_k)\right\} \\ &\quad - \max\!*\left\{\Lambda_{0,0}^{(o)}(A_k, B_k), \Lambda_{1,0}^{(o)}(A_k, B_k)\right\}, \end{aligned} \tag{14}$$

where $\Lambda_{0,0}^{(o)}(A_k, B_k) = 0$. The hard bit decisions can be found by comparing each of these likelihood ratios to a threshold.

1.3 Simulation Results

In this section, simulation results are presented that illustrate the performance of the DVB-RCS turbo code. Fig. 4 shows the frame error rate (FER) of several decoding algorithms when using blocks of $N = 212$ message couples (53 bytes) and code rate $r = 1/3$. One problem with using circular constituent codes is that the circulation state is unknown at the decoder. The two lowermost curves in Fig. 4 show the impact of the unknown circulation state when using eight iterations of log-MAP decoding. The lowermost curve was created by using a genie-aided decoder that knows the exact circulation state of the encoder. While this decoder is not feasible in practice, it serves as a bound for more practical decoders. The second curve shows the performance when the circulation state is not known and algorithm A3 from [10] is used to compensate for the unknown circulation state. Note that the loss due to imperfect knowledge of the circulation state is only about 0.02 dB at a FER of 10^{-4}.

The uppermost curve in Fig. 4 shows the performance when using eight iterations of the max-log-MAP algorithm along with algorithm A3 from [10] to handle the unknown circulation state. At a frame error rate (FER) of 10^{-4}, the loss due to using max-log-MAP is only about 0.16 dB. This is in contrast with the $0.3 - 0.4$ dB losses that are incurred when decoding binary turbo codes with the max-log-MAP algorithm, and for this reason many DVB-RCS decoder implementations use max-log-MAP [11].

Fig. 5 shows the influence of the block size. Frame error rate results are shown for blocks of $N = \{48, 64, 212, 432, 752\}$ message couples, or correspondingly $\{12, 16, 53, 108, 188\}$ bytes. In each case, the code rate is $r = 1/3$, the circulation state is unknown at the decoder, and eight iterations of max-log-MAP decoding are performed. The SNR required to achieve a FER of 10^{-4} is $\mathcal{E}_b/N_o = \{3.02, 2.77, 1.86, 1.65, 1.44\}$ dB for $N = \{48, 64, 212, 432, 752\}$, respectively.

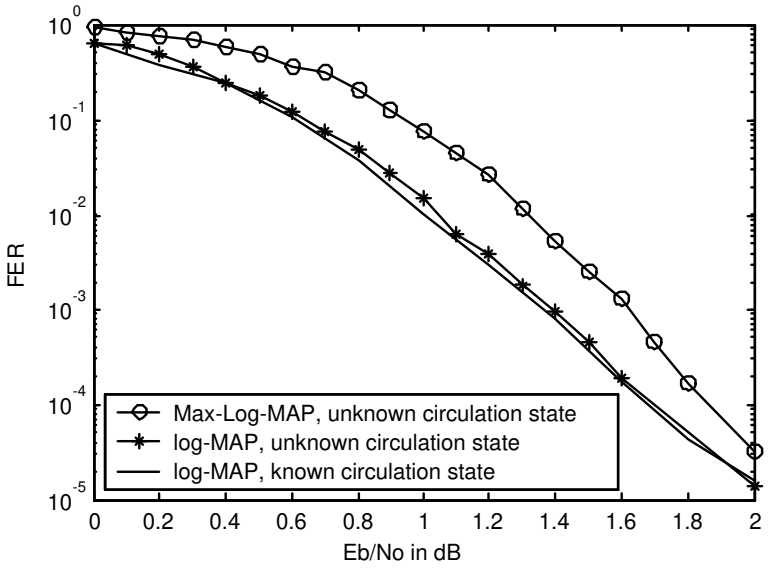

Fig. 4. Influence of decoding algorithm on the performance of the DVB-RCS turbo code. The frame length is $K = 212$ (53 bytes) and code rate is $r = 1/3$. Eight iterations of decoding are performed. The curve with the best performance shows the performance of the log-MAP algorithm if the circulation state used by the encoder were to be known by the decoder. The other two curves show the performance of log-MAP and max-log-MAP decoding when the decoder does not know the circulation state and uses algorithm A3 from [10].

Fig. 6 shows the influence of the code rate. Frame error rate results are shown for all seven code rates when the block size is $N = 212$ message couples. As with Fig. 5, eight iterations of max-log-MAP decoding are performed and the circulation state is unknown at the decoder. The SNR required to achieve a FER of 10^{-4} is $\mathcal{E}_b/N_o = \{1.86, 2.03, 2.37, 3.29, 3.96, 4.57, 5.21\}$ dB for $r = \{1/3, 2/5, 1/2, 2/3, 3/4, 4/5, 6/7\}$, respectively.

2 DVB-S2

The DVB-S2 standard uses a serial concatenation of two binary linear codes: an outer BCH code and an inner low density parity check (LDPC) code [3]. With binary linear block codes, a k bit message $\mathbf{d}$ is encoded into a n bit codeword $\mathbf{c}$ according to

$$\mathbf{c} = \mathbf{dG}, \tag{15}$$

where $\mathbf{G}$ is the k by n *generator matrix* and the matrix multiplication is modulo-2. The *parity-check matrix* $\mathbf{H}$ is a $m = n - k$ by n matrix that spans the null space of $\mathbf{G}$ and therefore satisfies $\mathbf{GH}^T = \mathbf{0}$.

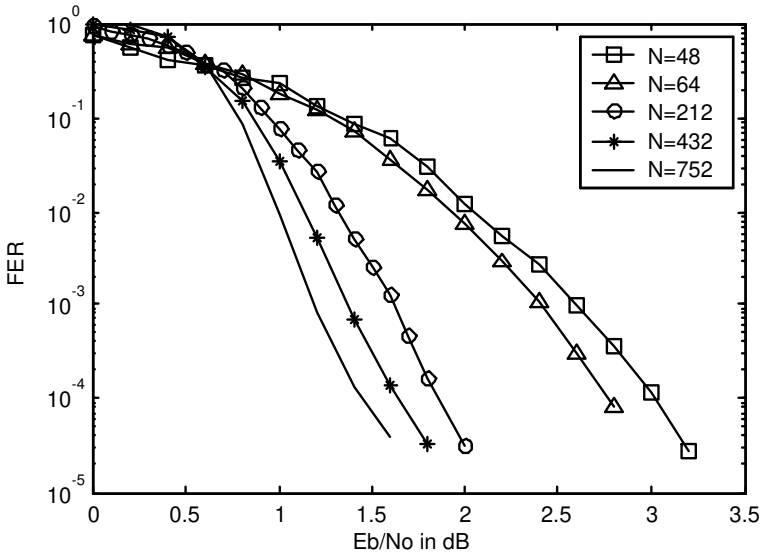

Fig. 5. Influence of block size on the performance of the DVB-RCS turbo code. The code rate is $r = 1/3$, block size is N message couples, and eight iterations of max-log-MAP decoding are performed.

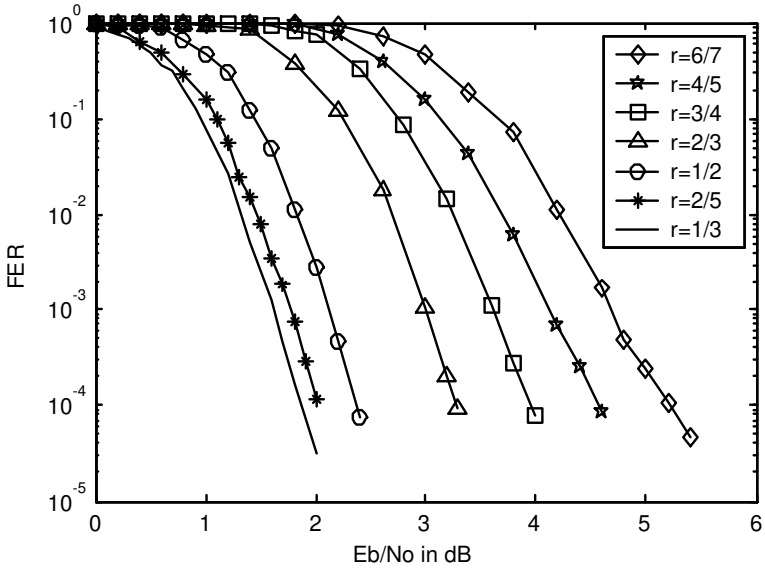

Fig. 6. Influence of code rate on the performance of the DVB-RCS turbo code. The code rate is r, block size is $N = 212$ message couples, and eight iterations of max-log-MAP decoding are performed.

LDPC codes are characterized by having very sparse parity check matrices. An LDPC code is said to be *regular* if all rows in **H** have the same weight (number of ones) and all columns in **H** have the same weight; otherwise the code is *irregular*. LDPC codes were originally proposed by Gallager in his 1960 dissertation [12] along with an iterative process for decoding. Although Gallager proved that the codes were good in theory, they were largely ignored until the advent of turbo codes because the decoder was thought to be too complex. However, after turbo codes showed the practicality of iterative decoding, interest in LDPC codes was soon renewed. In the mid-1990's, MacKay rediscovered LDPC codes [13, 14] and showed that they are capable of approaching the Shannon limit. Soon afterwards, Richardson and Urbanke [15] and Luby et al [16] showed that long irregular LDPC codes can be superior to turbo codes of the same length and can approach the Shannon capacity by a fraction of a decibel [17].

2.1 Encoding

The DVB-S2 channel encoder begins by first encoding a length k' binary message into a n' bit systematic BCH codeword. The $k = n'$ BCH codeword is then encoded into a n bit systematic LDPC codeword. The codeword length n can be either 64,800 or 16,200 bits long, producing *normal* and *short* frames, respectively. Note that unlike DVB-RCS, which fixes the value of the encoder *input*, DVB-S2 fixes the length of the encoder *output*. Because of this, the length k' of the input to the BCH encoder and the length k of the input to the LDPC encoder (which equals the length n' of the output of the BCH encoder) are variable and depend on the rate r of the LDPC code. Normal frames can be encoded at eleven different code rates, as shown in Table 1. Short frames can be encoded at all the same code rates except for rate $r = 9/10$, which is not supported, as shown in Table 2. For short frames, the "rates"

rate	k'	n'	t
9/10	58192	58320	8
8/9	57472	57600	8
5/6	53840	54000	10
4/5	51648	51840	12
3/4	48408	48600	12
2/3	43040	43200	12
3/5	38688	38880	12
1/2	32208	32400	12
2/5	25728	25920	12
1/3	21408	21600	12
1/4	16008	16200	12

Table 1. The input k' and output n' word sizes of the outer BCH code used by normal DVB-S2 frames. Also listed is the error correcting capability t of the BCH code.

rate	k'	n'	actual r
8/9	14232	14400	8/9
5/6	13152	13320	37/45
4/5	12432	12600	7/9
3/4	11712	11880	11/15
2/3	10632	10800	2/3
3/5	9552	9720	3/5
1/2	7032	7200	4/9
2/5	6312	6480	2/5
1/3	5232	5400	1/3
1/4	3072	3240	1/5

Table 2. The input k' and output n' word sizes of the outer BCH code used by short DVB-S2 frames. Each of these BCH codes has error correcting capability $t = 12$. Also listed is the actual rate $r = k/16200$ of the inner LDPC code.

$1/4, 1/2, 3/4, 4/5$, and $5/6$ are misnomers because the actual rates, as defined by $r = k/n = k/16200$ are $1/5, 4/9, 11/15, 7/9$ and $37/45$, respectively.

For short frames, the BCH code can correct $t = 12$ errors at the output of the LDPC decoder, while for normal frames it can correct between $t = 8$ and $t = 12$ errors, depending on rate of the LDPC code. The BCH code requires 168 parity bits for short frames and between 128 and 192 parity bits for long frames. Thus the overall rate r' of the concatenated BCH/LDPC code is slightly lower than that of the LDPC code alone. For instance, short frames encode messages of length $k' = k - 168 = 16200r - 168$ and thus have overall rate $r' = r - 168/16200$.

Unlike with turbo codes, the encoding of LDPC codes can be very complex if the code is not designed with encoding in mind [18]. This is especially true for systematic LDPC codes, because a sparse $\mathbf{H}$ matrix could require a dense $\mathbf{G}$ matrix. In order to facilitate systematic encoding and produce irregular LDPC that are especially suitable for high code rates, DVB-S2 uses a class of LDPC codes called *extended irregular repeat accumulate* (eIRA) codes [19]. With eIRA codes, the parity check matrix is constrained to be in the form

$$\mathbf{H} = [\mathbf{H}_1 \; \mathbf{H}_2] \tag{16}$$

where $\mathbf{H}_1$ is a sparse m by k matrix and $\mathbf{H}_2$ has the form

$$\mathbf{H}_2 = \begin{bmatrix} 1 & & & & \\ 1 & 1 & & & \\ & 1 & 1 & & \\ & & \cdots & & \\ & & & 1 & 1 \\ & & & & 1 \end{bmatrix} \tag{17}$$

Given the constraint on the $\mathbf{H}$ matrix, the generator matrix can be expressed in systematic form as

$$\mathbf{G} = [\mathbf{I} \; \mathbf{P}] \tag{18}$$

where $\mathbf{P} = \mathbf{H}_1^T \mathbf{H}_2^{-T}$ and the form of $\mathbf{H}_2^{-T}$ is

$$\mathbf{H}_2^{-T} = \begin{bmatrix} 1 & 1 & \cdots & 1 & 1 \\ & 1 & \cdots & 1 & 1 \\ & & \ddots & \vdots & \vdots \\ & & & 1 & 1 \\ & & & & 1 \end{bmatrix} \tag{19}$$

The matrix $\mathbf{H}_2^{-T}$ is actually the generator matrix for a differential encoder (also called an accumulator). Thus the encoding of the DVB-S2 LDPC code can be accomplished in two stages. First the output of the BCH encoder $\mathbf{d}$ is multiplied by the sparse matrix $\mathbf{H}_1^T$, yielding the intermediate result $\mathbf{p}' = \mathbf{d}\mathbf{H}_1^T$. The standard specifies the matrix $\mathbf{H}_1^T$ in the form of a table that lists the locations of the ones in the sparse matrix $\mathbf{H}_1^T$ for each code rate and length. Next, the intermediate result is differentially encoded yielding the set of parity bits $\mathbf{p} = \mathbf{p}'\mathbf{H}_2^{-T}$. Finally the parity bits and message are combined into the systematic codeword as $\mathbf{c} = [\mathbf{d}\ \mathbf{p}]$.

The modulation can be either QPSK, 8PSK, 16APSK, or 32APSK. The 16APSK constellation consists of two concentric rings with 4 uniformly spaced symbols on the inner ring and 12 uniformly spaced symbols on the outer ring. The 32APSK adds a third ring outside the 16APSK constellation, with 16 equally spaced symbols along the third ring. For the higher order modulations (everything except QPSK), a bit interleaver is placed between the channel encoder and the modulator, and thus the system uses bit interleaved coded modulation (BICM) [20].

2.2 Decoding

An LDPC code can be decoded iteratively using a message passing algorithm [14] over a graphical representation of the code's parity check matrix called a Tanner graph [21]. A Tanner graph is a bipartite graph consisting of n *variable* nodes (v-nodes) and m *check* nodes (c-nodes). Variable node y_j is connected to check node f_i if and only if the $(i,j)^{th}$ entry of $\mathbf{H}$ is equal to one. As an example, consider the systematic $(7,4)$ Hamming code with parity check matrix:

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{20}$$

The corresponding Tanner graph for this code is shown in Fig. 7.

In the message passing algorithm, messages (in the form of extrinsic information) flow up from the variable nodes to the check nodes and down from the check nodes to the variable nodes. A full description of the decoding algorithm is presented in [22], so here only the main results are presented. The

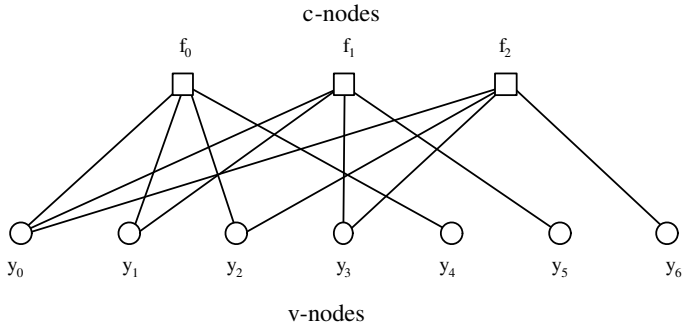

Fig. 7. Tanner graph for the $(7, 4)$ systematic Hamming code. Each row of $\mathbf{H}$ is represented by a check node and each column of $\mathbf{H}$ is represented by a function node.

goal of the decoder is to compute the LLR of the i^{th} code bit, $\Lambda^{(o)}(c_i)$, from which the systematic bits can be extracted and a hard decision made.

In the following, let $q_{i,j}$ represent the message passed from v-node i to c-node j and $r_{j,i}$ represent the message passed from c-node j to v-node i. Let $C_i = \{j : h_{j,i} = 1\}$ denote the row locations of the 1's in the i^{th} column of $\mathbf{H}$ and likewise $R_j = \{j : h_{j,i} = 1\}$ the column locations of the 1's in the j^{th} row. The set $R_{j\setminus i}$ is equal to R_j with i excluded.

Initially, the messages passed from the v-nodes to the c-nodes are set to the channel likelihood values, i.e. $q_{i,j} = \Lambda^{(i)}(c_i)$. Then each c-node computes the messages that it sends to every v-node it is attached to according to:

$$r_{j,i} = \left(\prod_{i' \in R_{j\setminus i}} \text{sign}(q_{i',j}) \right) \phi \left(\sum_{i' \in R_{j\setminus i}} \phi \left(|q_{i',j}| \right) \right) \tag{21}$$

where the nonlinear $\phi(x)$ function is defined as

$$\begin{aligned} \phi(x) &= -\log \tanh \left(\frac{x}{2} \right) \\ &= \log \left(\frac{e^x + 1}{e^x - 1} \right). \end{aligned} \tag{22}$$

Next, each v-node updates the LLR of its corresponding code bit according to

$$\Lambda^{(o)}(c_i) = \Lambda^{(i)}(c_i) + \sum_{j \in C_i} r_{j,i} \tag{23}$$

and then generates the output message sent to every c-node that it is connected to

$$q_{i,j} = \Lambda^{(o)}(c_i) - r_{j,i} \tag{24}$$

A hard decision can be made by simply comparing $\Lambda^{(o)}(c_i)$ to a threshold. Since LDPC codes are linear block codes, they have a built in error detecting mechanism. When the estimated codeword $\hat{\mathbf{c}}$ is a valid codeword, then $\hat{\mathbf{c}}\mathbf{H}^T = \mathbf{0}$ and thus the decoder can halt. If $\hat{\mathbf{c}}\mathbf{H}^T \neq \mathbf{0}$, then the decoder can iterate by having each c-node compute new messages to send to the v-nodes according to (21). Once a maximum number of iterations is reached, the decoder will quit.

2.3 Simulation Results

In this section, simulation results are presented that illustrate the performance of the DVB-S2 LDPC code. Fig. 8 shows the frame error rate (FER) performance of the short frame size and Fig. 9 shows the FER performance of normal frame size. In each case, up to 100 iterations of the log-domain sum-product algorithm [14] described in the last subsection are executed. Table 3 shows the $\mathcal{E}_b/N_o$ required to achieve a FER of 10^{-3} for each rate and frame size. Because of the large size of the normal frame size code and the steepness of the corresponding FER curve, results could not be simulated all the way down to a FER 10^{-3} for every code rate. Thus, extrapolated results are given for rates $r = 1/3, 1/2, 2/3$ and $5/6$. Note that the results presented here are only for the LDPC code. The outer BCH code used by DVB-S2 helps to clean up additional errors at the output of the LDPC decoder and will improve the overall performance (mainly be reducing an error floor that begins to emerge below the shown error rates).

rate	short	normal
9/10	N/A	3.78 dB
8/9	3.78 dB	3.68 dB
5/6	3.06 dB	3.03* dB
4/5	2.72 dB	2.68 dB
3/4	2.33 dB	2.18 dB
2/3	1.95 dB	1.86* dB
3/5	1.59 dB	1.36 dB
1/2	0.93 dB	0.85* dB
2/5	0.55 dB	0.54 dB
1/3	0.37 dB	0.22* dB
1/4	0.25 dB	0.13 dB

Table 3. The $\mathcal{E}_b/N_o$ required to achieve $FER = 10^{-3}$ for the LDPC codes used in DVB-S2. Values marked with an asterisk (*) are extrapolated from Fig. 9.

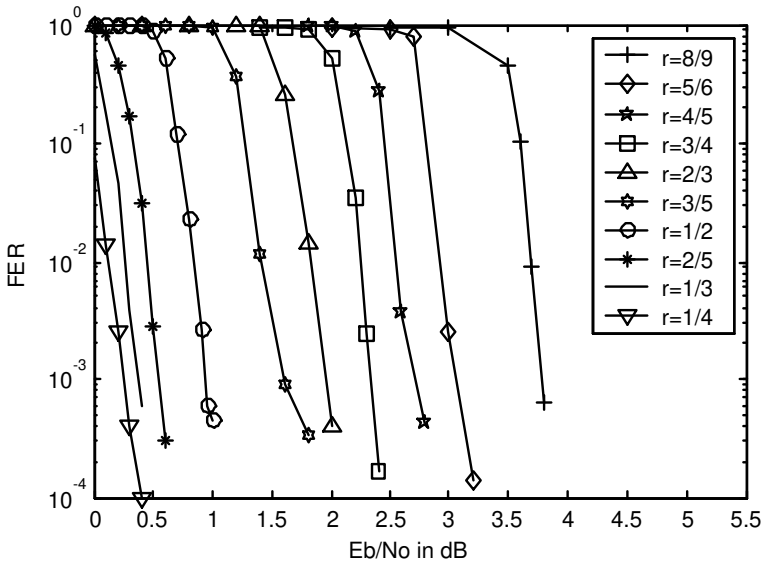

Fig. 8. Frame error rate performance of the $n = 16,200$ bit (short frame) LDPC code used in DVB-S2. The decoder uses 100 iterations of the log-domain sum-product algorithm.

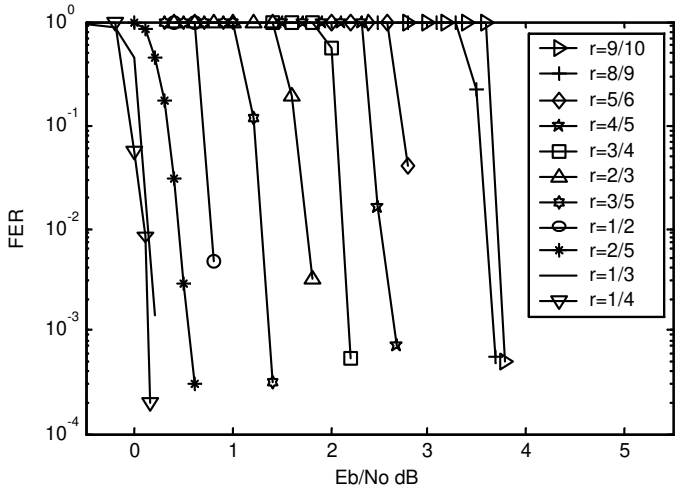

Fig. 9. Frame error rate performance of the $n = 64,800$ bit (normal frame) LDPC code used in DVB-S2. The decoder uses 100 iterations of the log-domain sum-product algorithm.

3 Putting It All Together

The turbo principle stands to revolutionize the delivery of digital content via satellite. The future of the DVB project hinges upon turbo-like coding techniques. The DVB-RCS standard, which uses a circular duobinary turbo code, provides a return channel for Internet services, thereby instantly making satellite a serious competitor to cable modems and DSL. The DVB-S2 standard, which uses LDPC codes, represents a significant improvement in the satellite downlink. However, for these technological improvements to be a complete success several hurdles remain. Turbo and LDPC codes are still more complex than their convolutional and Reed Solomon brethren, and therefore significant advances in implementation must still come to fruition. In addition, iteratively decodable codes are more sensitive to channel estimation and synchronization errors and therefore these issues must be dealt with carefully.

4 About the Simulations

The software to generate the plots in this chapter has been made available to the public at the http://www.iterativesolutions.com website. The software runs within matlab, but the key encoding and decoding functions are written in c for rapid execution and called as c-mex functions from matlab.

References

1. European Telecommunications Standards Institute. Digital broadcasting system for television, sound, and data services. *ETS 200 421*, 1994.
2. European Telecommunications Standards Institute. Digital video broadcasting (DVB); interaction channel for satellite distribution systems;. *ETSI EN 301 790 V1.2.2 (2000-12)*, 2000.
3. European Telecommunications Standards Institute. Digital video broadcasting (DVB) second generation framing structure, channel coding and modulation systems for broadcasting, interactive services, news gathering and other broadband satellite applications. *DRAFT EN 302 307 DVBS2-74r15*, 2003.
4. P. Robertson, P. Hoeher, and E. Villebrun. Optimal and sub-optimal maximum a posteriori algorithms suitable for turbo decoding. *European Trans. on Telecommun.*, 8(2):119–125, Mar./Apr. 1997.
5. C. Berrou, C. Douillard, and M. Jezequel. Multiple parallel concatenation of circular recursive convolutional (CRSC) codes. *Annals of Telecommunication*, 54(3-4):166–172, Mar.-Apr. 1999.
6. H. H. Ma and J. K. Wolf. On tail biting convolutional codes. *IEEE Trans. Commun.*, 34:104–111, May 1986.
7. C. Berrou and M. Jezequel. Non binary convolutional codes for turbo coding. *IEE Electronics Letters*, 35(1):39–40, Jan. 1999.

8. M. C. Valenti and J. Sun. The UMTS turbo code and an efficient decoder implementation suitable for software defined radios. *Int. J. Wireless Info. Networks*, 8:203–216, Oct. 2001.
9. A. J. Viterbi. An intuitive justification and a simplified implemetation of the MAP decoder for convolutional codes. *IEEE J. Select. Areas Commun.*, 16(2):260–264, Feb. 1998.
10. J. B. Anderson and S. M. Hladik. Tailbiting MAP decoders. *IEEE J. Select. Areas Commun.*, 16:297–302, Feb. 1998.
11. M. R. Soleymani, Y. Gao, and Y. Vilaipornsawai. *Turbo Coding for Satellite and Wireless Communications*. Kluwer Academic Publishers, Dordrecht, the Netherlands, 2002.
12. R. G. Gallager. *Low-Density Parity-Check Codes*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1960.
13. D. J. C. MacKay and R. M. Neal. Near Shannon limit performance of low density parity check codes. *IEE Electronics Letters*, 32:1645–1646, Aug. 1996.
14. D. J. C. MacKay. Good error correcting codes based on very sparse matrices. *IEEE Trans. Inform. Theory*, 45:399–431, Mar. 1999.
15. T. Richardson, A. Shokrollahi, and R. Urbanke. Design of capacity approaching irregular low density parity check codes. *IEEE Trans. Inform. Theory*, 47:618–637, Feb. 2001.
16. M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman. Improved low-density parity-check codes using irregular graphs. *IEEE Trans. Inform. Theory*, 47:585–598, Feb. 2001.
17. S. Y. Chung, G. D. Forney, T. J. Richardson, and R. Urbanke. On the design of low-density parity-check codes within 0.0045 dB of the shannon limit. *IEEE Commun. Letters*, 5:58–60, Feb. 2001.
18. T. Richardson and R. Urbanke. Efficient encoding of low-density parity-check codes. *IEEE Trans. Inform. Theory*, 47:638–656, Feb. 2001.
19. M. Yang, W. E. Ryan, and Y. Li. Design of efficiently encodable moderate-length high-rate irregular LDPC codes. *IEEE Trans. Commun.*, 52:564–571, Apr. 2004.
20. G. Caire, G. Taricco, and E. Biglieri. Bit-interleaved coded modulation. *IEEE Trans. Inform. Theory*, 44:927–946, May 1998.
21. R. M. Tanner. A recursive approach to low complexity codes. *IEEE Trans. Inform. Theory*, 27:533–547, Sept. 1981.
22. W. E. Ryan. An introduction to LDPC codes. In B. Vasic, editor, *Handbook for Coding and Signal Processing for Recording Systems*. CRC, New York, 2004.