Chapter 2

Soft-Output Algorithms for Detection and Correction

This chapter reviews two soft-output algorithms which are used in error correction and detection. In Section 2.1, partial-response channels and convolutional codes are described, then the BCJR algorithm is given. In Section 2.2, low-density parity check codes are reviewed, including the graphical representation of the decoder and the attendant message-passing decoding algorithm. Also, in Section 2.3, Wiberg's graph-based description of codes on trellises is reviewed.

2.1 BCJR Algorithm for Decoding and Detection

The solution to the problem of finding the *a posteriori* probabilities of a hidden Markov model is an important problem in a variety of applications, such as communications and speech processing. In 1966, Chang and Hancock gave an algorithm that computed the *a posteriori* probabilities for the states of a channel with memory. Hard decisions on these probabilities give a state sequence estimate, which can be used to determine the channel inputs. However, if the estimated state

sequence included invalid transitions, the authors suggested that it may be more efficient to request retransmission of the data [1].

In 1974, Bahl, Cocke, Jelinek and Raviv generalized this algorithm to estimating the *a posteriori* probabilities of the transitions (as well as of the states) for a Markov source observed through a discrete memoryless channel. This algorithm is applicable to convolutional codes and partial-response channels, and is known as the BCJR algorithm. Because it produces *a posteriori* probabilities, or soft information, the algorithm is important as a component in turbo decoding and turbo equalization [2].

2.1.1 Partial-Response Channels

Practical communication channels are noisy, bandlimited, and have intersymbol interference. A moderately realistic longitudinal magnetic recording channel can be modeled by a sequence of Lorenzian pulses with alternating sign, corresponding to magnetic transitions. The sequence of pulses, corrupted by noise, is filtered, sampled and then equalized to a target response. This target response, rather than being free of intersymbol interference, has a controlled amount of interference. This output from the equalizer is the input to a sequence detector, such as the Viterbi algorithm.

As an alternative to the above system, it is convenient to take the communication channel to be the target intersymbol-interference model itself. The receiver observes the output of the intersymbol-interference model plus additive white Gaussian noise. In this way, a continuous-time, sampled system is modeled by a discrete-time system. This discrete-time model is more amenable to analysis and simulation. Although it is not as representative of real channels as the continuous-time model, it is a reasonable surrogate, and permits the efficient evaluation of algorithms.

The discrete-time intersymbol interference is represented by a linear filter $h(D) = \sum_{i=0}^{\nu} h_i D^i$, for a channel model of degree ν. The input to this filter is a binary

Partial-response channel	Special name	$h(D)$
PR1	duobinary	$1+D$
—	dicode	$1-D$
PR2	class 2	$(1+D)^2 = 1+2D+D^2$
PR3	class 3	$(1+D)(2-D) = 2+D-D^2$
PR4	class 4, modified duobinary	$(1+D)(1-D) = 1-D^2$
PR5	class 5	$(1+D)^2(1-D)^2 = 1-2D^2+D^4$
EPR4	extended PR4	$(1-D)(1+D)^2 = 1+D-D^2-D^3$
E^2PR4	extended EPR4	$(1-D)(1+D)^3 = 1+2D-2D^3-D^4$
ME^2PR4	modified E^2PR4	$(1-D)(5+9D+6D^2+2D^2) = 5+4D-3D^2-4D^3+2D^4$

Table 2.1: Some well-known partial-response channel models.

sequence $x(D) = x_0 + x_1D + x_2D^2 + \cdots$, where x_i are equally likely zeros and ones, independent and identically distributed. The output is $y(D) = h(D)x(D)$, where arithmetic is over the real numbers. The output symbols y_t are drawn from the output alphabet $\mathcal{Y}$. For example, for the PR2 channel ($h(D) = (1+D)^2$), $\mathcal{Y} = \{-2, -1, 0, 1, 2\}$. The notation $\mathbf{x} = (x_0, x_1, \ldots)$, etc. will be used interchangeably for $x(D)$, etc.

Some of the well-known, named partial-response channels are shown in Table 2.1. Note that partial-response channels are not linear, in the sense that for two distinct inputs $x_a(D)$ and $x_b(D)$, $x_a(D)h(D)+x_b(D)h(D) \neq (x_a(D)+x_b(D))h(D)$, because of the restriction that the inputs are binary.

2.1.2 Convolutional Codes

Convolutional codes are a class of error-correcting codes. A systematic, binary, rate 1/2-convolutional code has input $u(D) = u_0 + u_1 D + u_2 D^2 + \cdots$. The systematic output is also $u(D)$. The convolutional code's parity is generated by a polynomial $g(D) = g_0 + g_1 D + \cdots + g_\nu D^\nu$, where $g_i \in \{0, 1\}$. The parity output of the convolutional code is $c(D) = g(D)u(D)$, where arithmetic is over the binary field. This definition of convolutional codes has been simplified for clarity, and can be generalized to non-systematic convolutional codes, to recursive convolutional codes, and to codes with rates other than 1/2. The output alphabet for convolutional codes is $\{0, 1\}$.

2.1.3 Trellis Representations

Both partial-response channels and convolutional codes can be represented by a finite state machine, with state $s_t \in \{0, \ldots, M-1\}$ at time t. The finite state machine can be drawn as a trellis, with $M = 2^\nu$ states at time t, and M states at time $t+1$. Fig. 2.1(a) shows the trellis for the PR2 partial-response channel. Each edge e in the trellis has a starting (left-hand) state $s^S(e)$ and an ending (right-hand) state $s^E(e)$. Each $s^S(e) \in \{0, \ldots, M-1\}$ and likewise for $s^E(e)$. For the partial-response trellis, the edge e has an input label $x(e)$ and output label $y(e)$. Fig. 2.1(b) shows a trellis fragment with labeled edges. For binary inputs, $x(e) \in \{0, 1\}$, and for this example, $y(e) \in \mathcal{Y} = \{-2, -1, 0, 1, 2\}$.

Both the partial-response channel's and the convolutional code's trellis transitions have a single input label (x and u, respectively) and a single output label (y and c, respectively). From here, x and y shall be used to refer to the inputs and outputs of the finite state machine generally, even in the context of convolutional codes.

There are N output symbols $y_1, \ldots, y_N$ Unless otherwise stated, it is assumed that the initial and final states are zero, that is $s_1 = 0$ and $s_{N+1} = 0$. Because the trellis is terminated, there are $N - \nu$ independent information bits $x_1, \ldots, x_{N-\nu}$.

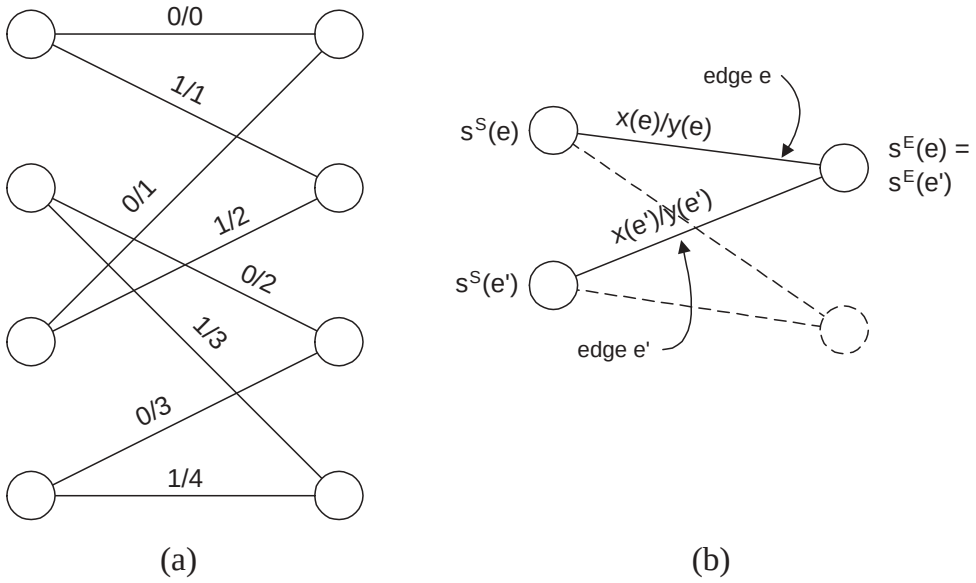

Figure 2.1: (a) The trellis for the PR2 channel. In labels are input/output. (b) The edge view of a generic trellis fragment.

2.1.4 The Channel

Information about the symbols x_t and y_t comes from either observation of corresponding channel outputs r_t^x and r_t, or from another decoder in a turbo decoding or turbo equalization scheme. The soft information about the input x_t and output y_t is represented by $X_t^I(i)$ and $Y_t^I(i)$, respectively. The superscript I indicates these are inputs (to the BCJR algorithm, next subsection), and the index i is from the symbol alphabet, $\{0, 1\}$ for $X_t^I(i)$ and $\mathcal{Y}$ for $Y_t^I(i)$. For symbols which were transmitted through the channel, the channel observations r_t^x, r_t can be used to find the BCJR input:

$$X_t^I(i) = P[r_t^x | x_t = i], \tag{2.1}$$

$$Y_t^I(i) = P[r_t | y_t = i]. \tag{2.2}$$

For the AWGN channel with noise variance σ^2, this function may be computed using the probability density function of the Gaussian:

$$\begin{aligned} X_t^I(i) &= e^{-\frac{(r_t^x - i)^2}{2\sigma^2}}, \\ Y_t^I(i) &= e^{-\frac{(r_t - i)^2}{2\sigma^2}}. \end{aligned}$$

In turbo decoding and turbo equalization, the soft information is *a priori* probabilities about that symbol, as provided by another decoder:

$$\begin{aligned} X_t^I(i) &= P[x_t = i], \\ Y_t^I(i) &= P[y_t = i]. \end{aligned}$$

2.1.5 The BCJR Algorithm

When the noise is memoryless, the BCJR algorithm computes the *a posteriori* probability (APP) for bit x_t, given the entire received sequence. It is often convenient to represent the output of the BCJR algorithm as a log probability ratio Λ_t:

$$\Lambda_t = \log \frac{P(x_t = 1 | r_1^N)}{P(x_t = 0 | r_1^N)}. \tag{2.3}$$

Using Λ_t to choose the bit which has the greatest *a posteriori* probability produces the maximum *a posteriori* (MAP) probability decision on that bit:

$$\widehat{x}_t = \begin{cases} 0 & \Lambda_t < 0 \\ 1 & \Lambda_t \geq 0 \end{cases}$$

Alternatively in the probability domain, the BCJR algorithm computes $P[x_t = i | r_1^N]$ and the corresponding MAP estimate is $\widehat{x}_t = \arg\max_i P[x_t = i | r_1^N]$, for $i = 0, 1$.

The derivation of the BCJR algorithm is elegant in that it begins with the *a posteriori* probability to be calculated, (2.3) and finds the recursive state metric computations. The derivation can be found in Schlegel's book [3], Souvignier's dissertation [4] and the original BCJR paper [2].

Benedetto, *et. al.* describe a variation of the BCJR algorithm which is suitable for turbo decoding and turbo equalization [5], called the soft-input, soft-output algorithm. Although there are various algorithms with soft inputs and soft outputs (for example, the soft-output Viterbi, algorithm, SOVA [6]), the algorithm of Benedetto *et al.* shall be referred to as *the* SISO algorithm. The SISO algorithm has two inputs, $X_t^I(i)$ and $Y_t^I(i)$, and two outputs, $X_t^O(i)$ and $Y_t^O(i)$. Both outputs are not always required, for example, for turbo equalization on partial-response channels, the output $Y_t^O(i)$ is not used.

Block diagrams for the partial-response and convolutional code systems are shown in Fig. 2.2. For partial-response channel, the estimates Y_t^I are found using knowledge that the channel is Gaussian. The *a priori* information X_t^I is provided by an outer code in a turbo equalization scheme. For the convolutional code, a generic channel is assumed, from which both inputs X_t^I and Y_t^I are computed.

The SISO algorithm computes:

$$\Lambda_t^{\text{SISO}} \quad = \quad \log \frac{P(x_t = 1|r_1^N)}{P(x_t = 0|r_1^N)} - \log \frac{P(x_t = 1)}{P(x_t = 0)}$$

Recall that the output of the BCJR algorithm is Λ_t. If we let $\Lambda_t^X = \log X_t^I(1)/X_t^I(0)$, then the relationship between the BCJR algorithm and the SISO algorithm is:

$$\Lambda_t^{\text{SISO}} \quad = \quad \Lambda_t - \Lambda_t^X.$$

Both the SISO and BCJR algorithms compute forward state metrics $A_t(m)$ and backward state metrics $B_t(m)$, where state m is one of $0, \ldots, M-1$, and time t is one of $1, \ldots, N+1$. Storage of these metrics requires an array of size M-by-N (A_{N+1} and B_1 are not used and do not need to be computed). Corresponding to the assumption that the finite-state machine is in the all-zeros state at time $t = 1$, the forward state metric recursion is initialized with $A_1 = (1, 0, \ldots, 0)$. The forward state metrics are then computed recursively, that is $A_2(\cdot)$ is computed from $A_1(\cdot)$, $A_3(\cdot)$ is computed from $A_2(\cdot)$, etc. Similarly, the backward state metric recursion is initialized with $B_{N+1}(\cdot) = (1, 0, \ldots, 0)$, and then $B_N(\cdot)$ is computed from $B_{N+1}(\cdot)$, $B_{N-1}(\cdot)$ is computed from $B_N(\cdot)$, etc.

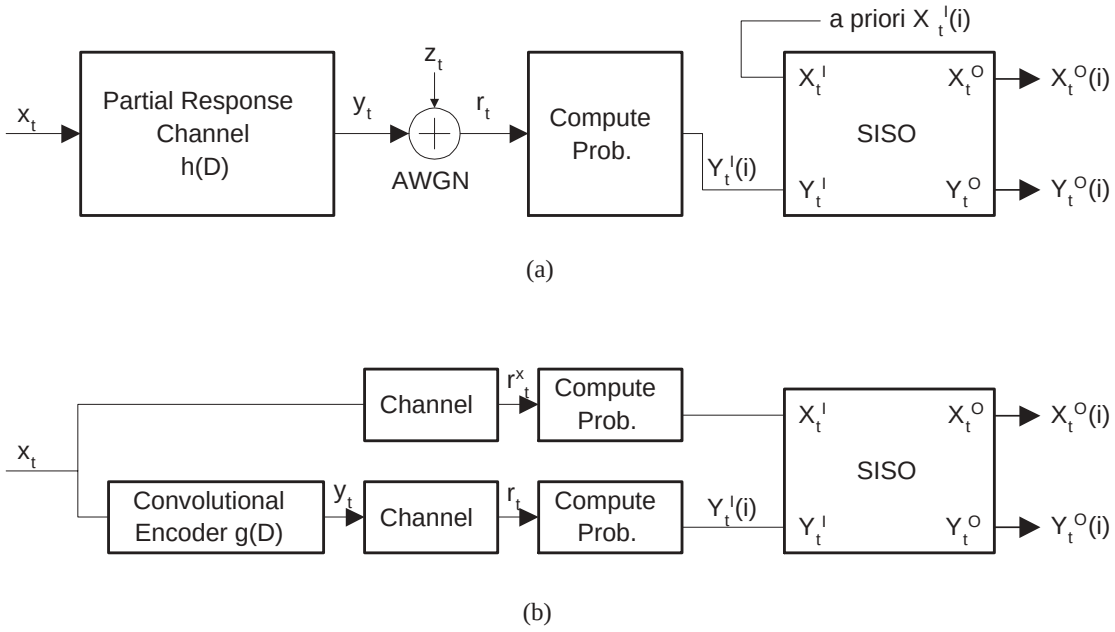

Figure 2.2: A system diagram using the SISO as a decoder for the (a) partial-response channel and (b) Convolutional code transmitted over a memoryless channel.

In the algorithm description, sums are performed over edges, for example for the forward recursion (see below), $e : s^E(e) = m$ indicates that the sum is over all of the edges e which end in state m. Let $\tilde{A}(m)$ and $\tilde{B}(m)$ be temporary storage used to normalize the state metrics so that $\sum_m A_t(m) = 1$ and $\sum_m B_t(m) = 1$. Normalization keeps the state metrics within the range of floating-point numbers used in computers. Without normalization, the state metrics would vanish to zero after a few recursion steps. Similarly, $\tilde{X}_t^O(m)$ and $\tilde{Y}_t^O(m)$ are temporary storage used for normalization such that $\sum_i X_t(i) = 1$ and $\sum_i B_t(i) = 1$, because X_t^O and Y_t^O are probabilities.

The SISO algorithm:

- Algorithm inputs: $X_t^I(i)$, $Y_t^I(i)$.

- The forward recursion:
  - Initialize $A_1 = (1, 0, \ldots, 0)$.
  - For each $t = 2, 3, \ldots, N$:
    - For each $m = 0, \ldots, M-1$, compute and normalize:

$$\begin{aligned} \tilde{A}(m) &= \sum_{e:s^E(e)=m} A_{t-1}(s^S(e)) X_{t-1}(x(e)) Y_{t-1}(y(e)), \quad (2.4) \\ A_t(m) &= \frac{\tilde{A}(m)}{\sum_m \tilde{A}(m)}. \end{aligned}$$

- The backward recursion:
  - Initialize $B_{N+1} = (1, 0, \ldots, 0)$.
  - For each $t = N, N-1, \ldots, 2$:
    - For each $m = 0, \ldots, M-1$, compute and normalize:

$$\begin{aligned} \tilde{B}(m) &= \sum_{e:s^S(e)=m} B_{t+1}(s^S(e)) X_t(x(e)) Y_t(y(e)), \quad (2.5) \\ B_t(m) &= \frac{\tilde{B}(m)}{\sum_m \tilde{B}(m)}. \end{aligned}$$

- *SISO output.* The SISO outputs are X_t^O and Y_t^O:

$$\begin{aligned} \tilde{X}_t^O(i) &= \sum_{e:x(e)=i} A_t(s^S(e)) Y_t^I(c(e)) B_{t+1}(s^E(e)), \\ X_t^O(i) &= \frac{\tilde{X}_t^O(i)}{\sum_i \tilde{X}_t^O(i)}, \quad (2.6) \\ \tilde{Y}_t^O(i) &= \sum_{e:y(e)=i} A_t(s^S(e)) X_t^I(c(e)) B_{t+1}(s^E(e)), \\ Y_t^O(i) &= \frac{\tilde{Y}_t^O(i)}{\sum_i \tilde{Y}_t^O(i)}. \quad (2.7) \end{aligned}$$

The log-probability ratio output of the SISO algorithm is:

$$\Lambda_t^{\text{SISO}} = \log \frac{X_t^O(1)}{X_t^O(0)}.$$

Then, the BCJR algorithm's output is:

$$\Lambda_t \;=\; \Lambda_t^{\text{SISO}} + \log \frac{X_t^I(1)}{X_t^I(0)}.$$

If there is no *a priori* information, that is, $\log \frac{X_t^I(1)}{X_t^I(0)} = 0$, then the BCJR and SISO algorithms produce the same result.

For clarity of notation, the SISO algorithm has been presented here for partial response channels and systematic, rate-1/2 convolutional codes. However, the presentation can be generalized to convolutional codes of arbitrary rate, and non-systematic codes, as in [5].

2.2 Low Density Parity Check Codes

Low Density Parity Check (LDPC) codes are a class of error-correcting codes defined by a sparse parity check matrix, and were originally proposed by Gallager in 1963. They have good minimum distance at longer block lengths and guaranteed reliability when decoding below a fixed noise threshold. LDPC codes have an efficient decoding algorithm which Gallager termed probabilistic decoding. Because of the limitations of computing power at that time, these codes could not be simulated for the block lengths at which their benefits were evident [7].

Until the mid-1990's, these codes were largely ignored. An exception was in 1981, when Tanner studied codes on graphs, including LDPC codes, and introduced the idea of using bipartite graphs to describe code constructions, particularly codes which have good minimum distance properties. However, the code constructions did not reveal the capacity-approaching performance of randomly constructed codes at long block lengths [8].

Wiberg, Loeliger and Koetter expanded on Tanner's work on graphs in 1995, and introduced hidden nodes, which permitted a description of trellis-based codes using a message-passing graph description [9], [10]. This led to a graphical description of turbo codes, which had already been introduced by Berrou, Galvieux

and Thitimajshima in 1993 [11]. By 1996, the significance of LDPC codes was becoming clear, when MacKay and Neal showed that LDPC codes could perform as well as turbo codes on the AWGN channel [12].

There is now good understanding of LDPC codes, particularly for the case of asymptotically long block lengths. Gallager's probabilistic decoding method has been generalized, described as instances of the sum-product algorithm on factor graphs [13], belief propagation [14] and message-passing decoding. It has been shown that asymptotically long LDPC codes provide for reliable communications on binary-input memoryless channels at a signal-to-noise ratio above a fixed threshold [15]. For irregular LDPC codes [16], it has been shown that this threshold is close to channel capacity [17].

2.2.1 LDPC Code Construction

A linear, binary error correcting code can be defined by a M-by-N parity check matrix, $\mathbf{H}$. The block length of the error correcting code is N, the number of parity checks is M, and so the number of information bits K, is $N - M$, if $\mathbf{H}$ is full rank. The entries in the matrix are zeros and ones, and the code is the set of row vectors $\mathbf{x}$ which satisfy:

$$\mathbf{x}\mathbf{H}^t = 0, \tag{2.8}$$

where superscript t represents matrix transposition and matrix multiplication is performed over the binary field (*i.e.* modulo 2).

LDPC codes are defined by a sparse parity check matrix. The parity check matrix for a regular LDPC code has a constant number j ones in each column, and k ones in each row. The total number of ones in the parity check matrix is kM or jN. The rate $R = K/N$ of a regular LDPC code can be written $1 - j/k$.

Gallager described a simple technique for constructing a regular low-density parity check matrix. Begin with a M/j-by-N submatrix $\mathbf{H}'$ which has one 1 per column, and k ones per row. This can be formed by the horizontal concatenation

of Nj/M M/j-by-M/j identity matrices $\mathbf{I}_{M/j}$:

$$\begin{aligned}\mathbf{H}' &= [\mathbf{I}_{M/j} \cdots \mathbf{I}_{M/j}] \\ &= \left[\begin{array}{cccccccccccc} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right].\end{aligned}$$

Let $\pi(\mathbf{H}')$ be a column permutation of $\mathbf{H}'$. Form the parity check matrix $\mathbf{H}$ from the vertical concatenation of $\mathbf{H}'$ and $j-1$ distinct permutations $\pi_2, \ldots, \pi_j$ of $\mathbf{H}'$:

$$\mathbf{H} = \left[\begin{array}{c} \mathbf{H}' \\ \pi_2(\mathbf{H}') \\ \vdots \\ \pi_j(\mathbf{H}') \end{array}\right] = \left[\begin{array}{cccccccccccc} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{array}\right]$$

This technique cannot create all possible regular LDPC matrices $\mathbf{H}$ with j 1's per column and k 1's per row. However, it is an efficient construction technique.

Tanner introduced the idea of representing the parity check matrix using a bipartite graph. The graph has two types of nodes, bit nodes represented by circles, and check nodes represented by squares. In a bipartite graph, nodes are connected by edges, but a bit node is only connected to check nodes and a check node is only connected to bit nodes. Let there be an edge $e_{m,n}$ between bit node n and parity check node m if the entry in the m^{th} row and n^{th} column of $\mathbf{H}$, $H_{m,n}$, is a one. The columns of the parity check matrix correspond to bits, and the rows correspond to checks.

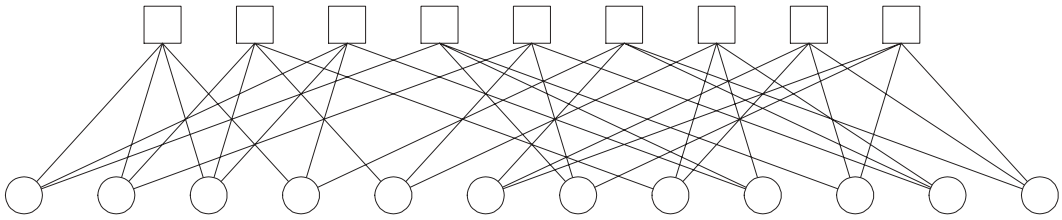

Figure 2.3: The bipartite graph, or Tanner graph, for a sparse parity check matrix.

For example, the Tanner graph for this regular LDPC matrix with $j = 3, k = 4$:

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix},$$

is shown in Fig. 2.3.

2.2.2 Message-Passing Decoding Algorithm

The message-passing algorithm is a decoding algorithm for LDPC codes. Although it is sub-optimal, this low-complexity decoder is effective at decoding LDPC codes on a variety of channels. Under the message-passing algorithm, messages are passed between nodes along edges. Each edge e connecting nodes n and m has two messages. Messages q pass from bit nodes to check nodes. Messages r pass from check nodes to bit nodes. Each message is a number, the probability that bit x_n is a zero, $P[x_n = 0]$ (logarithms of probability ratios can also be used). Each node computes a local function of input messages, generating outputs using its inputs. All bit nodes can perform their operations in parallel; likewise for the check nodes. Messages pass back and forth iteratively, between bit and check nodes, until a stopping condition is reached, and the algorithm produces output messages.

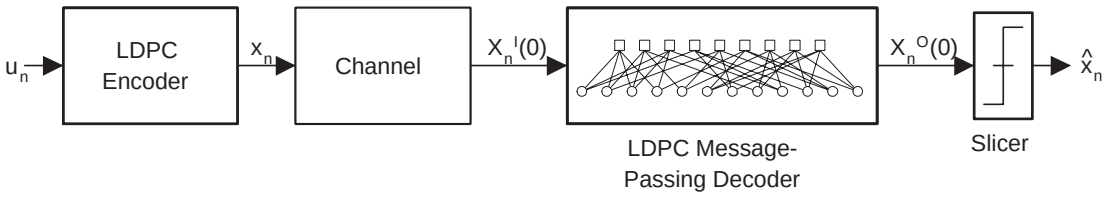

Figure 2.4: Block diagram for an LDPC decoder.

Message-passing decoding of LDPC codes works in the following manner. The system shown in Fig. 2.4 is assumed. Initially, only the channel information, $X_n^I(0)$, computed according to (2.1) is available at the bit nodes. The other inputs to the bit nodes, the check-to-bit messages, are all initialized to an unknown state, that is all messages are $r = 0.5$.

The bit node generates outgoing messages from the incoming messages. The computation of messages follows what Kschischang, *et al.* call the *sum-product update rule*, that is, the output message from a node along edge e is based upon all inputs to that node except the message from edge e [13]. To illustrate, suppose a node has four edges connected to it, labeled e_1, e_2, e_3 and e_x with corresponding incoming messages r_1, r_2, r_3 and $X^I(0)$; see Fig. 2.5.

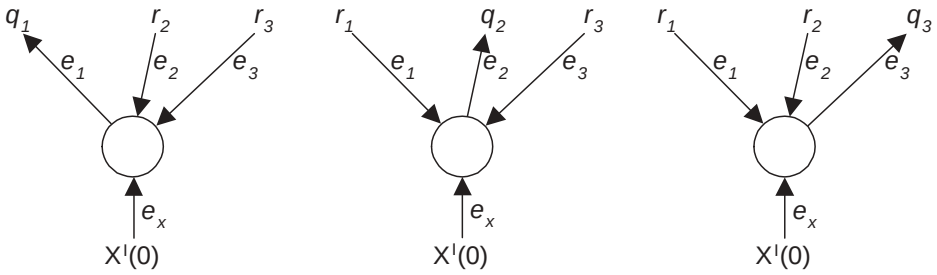

Figure 2.5: The sum-product update rule.

The node generates three outgoing messages q_1, q_2 and q_3 (the generation of the outgoing message on edge e_x as the last step in decoding will be discussed

later). The incoming messages are different estimates about that bit, combined to generate the outgoing message q_1 according to:

$$q_1 = \frac{X^I(0) r_2 r_3}{X^I(0) r_2 r_3 + (1 - X^I(0))(1 - r_2)(1 - r_3)},$$

and more generally, for a bit node with j edges, the output on the l^{th} edge is:

$$q_l = \frac{X^I(0) \prod_{i=1}^{j \setminus l} r_i}{X^I(0) \prod_{i=1}^{j \setminus l} r_i + (1 - X^I(0)) \prod_{i=1}^{j \setminus l} (1 - r_i)} \tag{2.9}$$

where "\" indicates that the product is over all elements $i = 1, \ldots, j$ except for $i = l$. Each of the N bit nodes generate j outgoing messages. It is convenient to think of a matrix $\mathbf{Q}$, the same size as $\mathbf{H}$, which is storage for the bit-to-check messages. Let $q_{m,n}$ be the message passed from bit node n to check node m. In the positions where $\mathbf{H}$ has a one, $\mathbf{Q}$ has $q_{m,n}$, and where $\mathbf{H}$ has a zero, $\mathbf{Q}$ has a zero.

These messages q are the input to the check nodes. The computation at the check node also follows the sum-product update rule, which is illustrated with a check node that has four edges e_1, e_2, e_3 and e_4, and corresponding incoming messages q_1, q_2, q_3 and q_4 in Fig. 2.6.

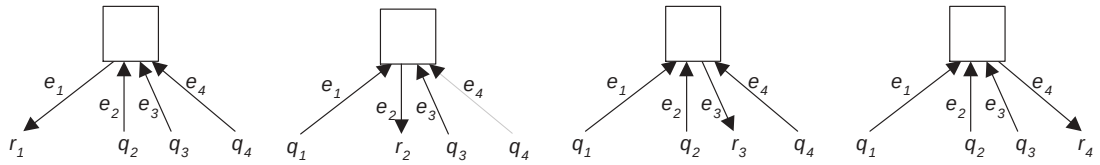

Figure 2.6: The sum-product rule applied at the check node.

The code bits associated with the incoming edges satisfy the parity check equation $x_1 + x_2 + x_3 + x_3 = 0$ (modulo 2), since the codeword satisfies (2.8). Each outgoing edge computes the local *a posteriori* probability for each bit. For example, the output message r_1 is:

$$r_1 = \frac{1 + (2q_2 - 1)(2q_3 - 1)(2q_4 - 1)}{2},$$

and more generally, for a check node with k edges, the output on the l^{th} edge is:

$$r_l \;=\; \frac{1+\prod_{i=1}^{k\backslash l}(2q_i-1)}{2}. \tag{2.10}$$

Each of the M check nodes generates k outgoing messages. It is convenient to think of a matrix $\mathbf{R}$, the same size as $\mathbf{H}$ which stores the check-to-bit messages. Let $r_{m,n}$ be the message passed from check node m to bit node n. In the positions where $\mathbf{H}$ has a one, $\mathbf{R}$ has $r_{m,n}$, and where $\mathbf{H}$ has a zero, $\mathbf{R}$ has a zero.

The messages are now passed back to the bit nodes, for the second iteration. Every time a computation is performed at the bit nodes, the channel input $X_n^I(0)$ is used. Iterative decoding proceeds in this way, with messages passed between the bit nodes and check nodes.

No outputs are generated until iterative decoding stops. When the algorithm finally stops, each bit node produces an estimate:

$$X^O(0) \;=\; \frac{\prod_{i=1}^{j} r_i}{\prod_{i=1}^{j} r_i + \prod_{i=1}^{j}(1-r_i)}. \tag{2.11}$$

This output $X^0(0)$ excludes the original input $X^I(0)$, and is appropriate to use in turbo equalization. However, to find a hard decision estimate $\widehat{x}_n$ of the input bits x_n, combine the message-passing decoder's estimate $X_n^O(0)$ with the original channel soft information $X_n^I(0)$:

$$\widehat{x}_n \;=\; \begin{cases} 0, & \frac{X_n^O(0)X_n^I(0)}{X_n^O(0)X_n^I(0)+(1-X_n^O(0))(1-X_n^I(0))} \geq 0.5 \\ 1, & \frac{X_n^O(0)X_t^I(0)}{X_t^O(0)X_t^I(0)+(1-X_t^O(0))(1-X_t^I(0))} < 0.5 \end{cases}$$

A stopping condition is required to keep the decoder from iterating forever. The decoder can check if the estimate $\widehat{\mathbf{x}} = \widehat{x}_1, \ldots, \widehat{x}_N$ is a codeword. After each iteration, the decoder computes $\widehat{\mathbf{x}}\mathbf{H}^t$. If it is a zero, $\widehat{\mathbf{x}}$ is a codeword, then the decoder is done, and outputs $\widehat{\mathbf{x}}$. If not, it performs another iteration. It is possible that the decoder will never converge to a codeword. When the decoder reaches some maximum number of iterations, it stops, and outputs the information which it has.

2.3 Wiberg Graphs

In his dissertation Wiberg builds upon Tanner's and Gallager's work, and introduces a new type of message-passing graph, one with "hidden" nodes. Hidden nodes are distinguished from bit nodes which are visible. Wiberg's graph is well-suited for describing the behavior of a finite state machine, where the hidden nodes represent state, and are indicated by a double circle. Small solid circles represent "check" sites, which could be thought of as analogous to the check nodes of the LDPC message-passing graph. An example is illustrated in Fig. 2.7, which comes from Wiberg's dissertation. The graph is for a (6,3) block code, and that code's trellis representation is also shown for comparison [10, p.8].

Hidden states have an alphabet size equal to the number of states in the trellis, one, two or four in this example. For a cycle-free graph, it is possible to use a sequential update schedule for computation of the new messages—this is what forward-backward algorithms do. Thus, the BCJR algorithm can be described as a sequential schedule applied to this graph.

Wiberg went on to describe how this idea could be used to draw a message-passing graph for turbo codes, which consist of two convolutional codes, separated by an interleaver. Thus, Wiberg showed that turbo codes, like LDPC codes, could be described by a graph. In this way, the subtraction of *a priori* information described by Berrou *et al.* can be viewed in terms of the sum-product update rule.

2.4 Conclusion

In this chapter, partial-response channels and convolutional codes were defined, and the BCJR decoding algorithm was given. LDPC codes were described and the message-passing decoding algorithm was given. Wiberg's graph-based description of codes on trellises was given as well. These ideas will be used in the remaining chapters.

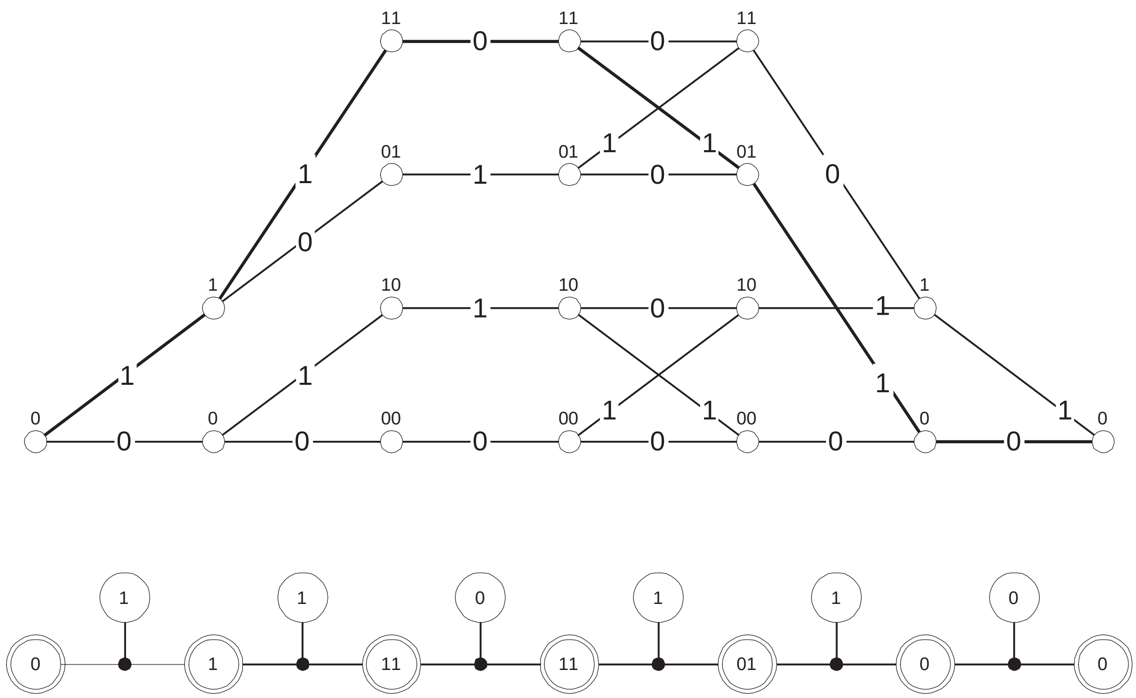

Figure 2.7: The edge view of a trellis fragment.

References

[1] R. W. Chang and J. C. Hancock, "On receiver structures for channels having memory," *IEEE Transactions on Information Theory*, vol. 12, pp. 463–468, October 1966.

[2] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Transactions on Information Theory*, vol. 20, pp. 284–287, March 1974.

[3] C. Schlegel, *Trellis Coding*. Piscataway, NJ, USA: IEEE Press, 1997.

[4] T. V. Souvignier, *Turbo Decoding for Partial Response Channels*. PhD thesis, University of California, San Diego, 1999.

[5] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "A soft-input soft-output APP module for iterative decoding of concatenated codes," *IEEE Communications Letters*, vol. 1, pp. 22–24, January 1997.

[6] J. Hagenauer and P. Hoeher, "A Viterbi algorithm with soft-decision outputs and its applications," in *Proceedings IEEE Global Telecommunications Conference*, (Dallas, TX, USA), pp. 1680–1686, IEEE, November 1989.

[7] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA, USA: The M.I.T. Press, 1963.

[8] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on Information Theory*, no. 5, pp. 533–547, 1981.

[9] N. Wiberg, H.-A. Loeliger, and R. Kötter, "Codes and iterative decoding on general graphs," *European Transactions on Telecommunications and Related Technologies*, vol. 6, pp. 513–525, September – October 1995.

[10] N. Wiberg, *Codes and Decoding on General Graphs*. PhD thesis, Linköping University, S-581 83 Linköping, Sweden, 1996.

[11] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," in *Proceedings IEEE International Conference on Communications*, vol. 2, (Geneva, Switzerland), pp. 1064–1070, IEEE, May 1993.

[12] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electronic Letters*, vol. 32, pp. 1645–1646, August 1996.

[13] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on Information Theory*, vol. 47, pp. 498–519, February 2001.

[14] R. J. McEliece, D. J. C. MacKay, and J.-F. Cheng, "Turbo decoding as an instance of pearl's 'belief propagation' algorithm," *IEEE Journal on Selected Areas in Communications*, vol. 16, pp. 140–152, February 1998.

[15] T. J. Richardson and R. L. Urbanke, "The capacity of low-density parity check codes under message-passing decoding," *IEEE Transactions on Information Theory*, vol. 47, pp. 599–618, February 2001.

[16] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Efficient erasure correcting codes," *IEEE Transactions on Information Theory*, vol. 47, pp. 569–598, February 2001.

[17] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, pp. 619–637, February 2001.

[18] R. Akella and J. K. Wolf, "On the parallel MAP algorithm," in *Proceedings of IEEE International Symposium on Information Theory*, (Cannes, France), pp. 371–376, IEEE, October 2001.

[19] A. J. Viterbi, "An intuitive justification and a simplified implementation of the MAP decoder for convolutional codes," *IEEE Journal on Selected Areas in Communications*, vol. 16, pp. 260–264, February 1998.

[20] S. S. Pietrobon, "Efficient implementation of continuous map decoders and a synchronization technique for turbo decoders," in *International Symposium on Information Theory and its Applications*, (Victoria, BC, Canada), pp. 586–589, IEEE, September 1996.

[21] S. A. Barbulescu, *Iterative Decoding of Turbo Codes and Other Concatenated Codes*. PhD thesis, University of South Australia, Australia, February 1996.

[22] S. Benedetto, D. Divsalar, F. Pollara, and G. Montorsi, "A soft-input soft-output maximum a posteriori (MAP) module to decode parallel and serial concatenated codes," *The Telecommunications and Data Acquisition Progress Report*, vol. 42, pp. 1–20, November 1996.

[23] J. Pearl, *Probabilistic Reasoning in Intelligent Systems*. San Mateo, CA, USA: Morgan Kaufmann, 1988.

[24] A. D. Weathers, S. A. Altekar, and J. K. Wolf, "Distance spectra for PRML channels," *IEEE Transactions on Magnetics*, vol. 33, pp. 2809–2811, September 1997.

[25] B. M. Kurkoski, P. H. Siegel, and J. K. Wolf, "Joint message-passing decoding of LDPC codes and partial-response channels," *IEEE Transactions on Information Theory*, vol. 48, pp. 1410–1422, June 2002.

[26] G. D. Forney, Jr. and A. R. Calderbank, "Coset codes for partial response channels; or, coset codes with spectral nulls," *IEEE Transactions on Information Theory*, vol. 35, pp. 925–943, September 1989.

[27] S. A. Altekar, M. Berggren, B. E. Moision, P. H. Siegel, and J. K. Wolf, "Error-event characterization on partial-response channels," *IEEE Transactions on Information Theory*, vol. 45, pp. 241–247, January 1999.

[28] C. Douillard, M. Jézéquel, C. Berrou, A. Picart, P. Didier, and A. Glavieux, "Iterative correction of intersymbol interference: Turbo equalization," *European Transactions on Telecommunications and Related Technologies*, vol. 6, pp. 507–511, September – October 1995.

[29] W. E. Ryan, L. L. McPheters, and S. W. McLaughlin, "Combined turbo coding and turbo equalization for PR4-equalized Lorentzian channels," in *Proceedings of the Conference on Information Sciences and Systems*, March 1998.

[30] W. E. Ryan, "Performance of high rate turbo codes on a PR4-equalized magnetic recording channel," in *Proceedings IEEE International Conference on Communications*, (Atlanta, GA, USA), pp. 947–951, IEEE, June 1998.

[31] T. Souvignier, A. Friedman, M. Öberg, P. H. Siegel, R. E. Swanson, and J. K. Wolf, "Turbo codes for PR4: Parallel versus serial concatenation," in *Proceedings IEEE International Conference on Communications*, (Vancouver, BC, Canada), pp. 1638–1642, IEEE, June 1999.

[32] J. Li, K. R. Narayanan, E. Kurtas, and C. N. Georghiades, "On the performance of high-rate TPC/SPC codes and LDPC codes over partial response channels," *IEEE Transactions on Communications*, vol. 50, pp. 723–734, May 2002.

[33] J. L. Fan, A. Friedmann, E. Kurtas, and S. W. McLaughlin, "Low density parity check codes for magnetic recording," in *Proceedings 37th Annual Allerton Conference on Communication, Control, and Computing*, (Monticello, IL, USA), pp. 1314–1323, September 1999.

[34] H. Song, R. M. Todd, and J. R. Cruz, "Low density parity check codes for magnetic recording channels," *IEEE Transactions on Magnetics*, vol. 36, no. 5, pp. 2183–2186, 2000.

[35] A. Thangaraj and S. W. McLaughlin, "Threshold and scheduling for LDPC-coded partial response channels," *IEEE Transactions on Magnetics*, vol. 38, pp. 2307–2309, September 2002.

[36] S. ten Brink, "Convergence behavior of iteratively decoded parallel concatenated codes," *IEEE Transactions on Communications*, vol. 49, pp. 1727–1737, October 2001.

[37] D. MacKay. `http://wol.ra.phy.cam.ac.uk/mackay/codes/data.html`. Online.

[38] C. Méasson and R. Urbanke, "Further analytic properties of EXIT-like curves and applications," in *Proceedings of IEEE International Symposium on Information Theory*, (Yokohama, Japan), p. 266, IEEE, July 2003.

[39] H. D. Pfister, *On the Capacity of Finite State Channels and the Analysis of Convolutional Accumulate-m Codes*. PhD thesis, University of California, San Diego, La Jolla, CA, USA, March 2003.

[40] A. Thangaraj and S. W. McLaughlin, "Threshold for regular LDPC codes over PR channels," in *Proceedings of IEEE International Symposium on Information Theory*, (Washington, D.C., USA), IEEE, June 2001.

[41] M. Tüchler, S. ten Brink, and J. Hagenauer, "Measures for tracing convergence of iterative decoding algorithms," No. 4, (Berlin, Germany), pp. 53–60, January 2002.

[42] P. Elias, "Coding for noisy channels," in *IRE Conf. Rec.*, pt. 4, pp. 37–46, 1955.

[43] S. M. Aji and R. J. McEliece, "The generalized distributive law," *IEEE Transactions on Information Theory*, vol. 46, pp. 325–343, March 2000.

[44] G. D. Forney, Jr., "The Viterbi algorithm," *Proceedings of the IEEE*, vol. 61, pp. 268–277, March 1973.

[45] M. R. Best, M. V. Burnashev, Y. Levy, A. Rabinovich, P. C. Fishburn, A. R. Calderbank, and D. J. Costello, "On a technique to calculate the exact performance of a convolutional code," *IEEE Transactions on Information Theory*, vol. 41, pp. 441–447, March 1995.

[46] M. Lentmaier, D. Truhachev, and K. S. Zigangirov, "Analytic expression for the exact bit error probability of the (7,5) convolutional code," in *Proceedings of IEEE International Symposium on Information Theory*, (Lausanne, Switzerland), p. 7, IEEE, June 2002.

[47] M. Lentmaier, D. Truhachev, and K. S. Zigangirov, "Analytic expression for the exact bit error probabilities of rate 1/2 convolutional encoders," *Submitted to IEEE Trans. on Information Theory*.

[48] J. P. M. Schalkwijk and A. J. Vinck, "Syndrome decoding of convolutional codes," *IEEE Transactions on Communications*, vol. 23, pp. 789–792, July 1975.

[49] A. Yamaguchi, M. Arai, S. Fukumoto, and K. Iwasaki, "Analytical evaluation of internet packet loss recovery using convolutional codes," *IEICE Trans. Fundamentals*.

[50] A. Yamaguchi, M. Arai, H. Kurosu, S. Fukumoto, and K. Iwasaki, "Fault-tolerance design for multicast using convolutional-code-based FEC and its analytical evalutation," *IEICE Trans. Fundamentals*.

[51] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson, "RAID: High-performance, reliable secondary storage," *ACM Computing Surveys*, vol. 26, no. 2, pp. 145–185, 1994.

[52] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal map decoding algorithms operating in the log domain," in *Proceedings IEEE International Conference on Communications*, (Seattle, BC, Canada), pp. 1009–1013, IEEE, 1995.

[53] M. P. C. Fossorier, F. Burkert, S. Lin, and J. Hagenauer, "On the equivalence between SOVA and max-log-MAP decodings," *IEEE Communications Letters*, vol. 2, pp. 137–139, May 1998.

[54] N. Seshadri and C.-E. W. Sundberg, "List Viterbi decoding algorithms with applications," *IEEE Transactions on Communications*, vol. 42, pp. 313–323, February/March/April 1994.

[55] C. Nill and C.-E. W. Sundberg, "List and soft symbol output Viterbi algorithms: Extensions and comparisons," *IEEE Transactions on Communications*, vol. 43, pp. 277–287, February/March/April 1995.

[56] G. W. Stewart, "Gaussian elimination, perturbation theory, and Markov chains," in *Linear Algebra, Markov Chains, and Queuing Models*, (New York, NY, USA), pp. 59–69, Springer-Verlag, 1992.

[57] G. F. Lawler, *Introduction to Stocastic Processes*. Chapman and Hall/CRC, 1995.

[58] A. P. Hekstra, "An alternative to metric rescaling in Viterbi decoders," *IEEE Transactions on Communications*, vol. 37, pp. 1220–1222, November 1989.

[59] P. Siegel, C. B. Shung, T. D. Howell, and H. K. Thapar, "Exact bounds for Viterbi detector path metric differences," in *Proceedings of 1991 IEEE International Conference on Acoustics, Speech, and Signal Processing*, (Toronto, Canada), pp. 109–1096, May 1991.

[60] A. Vityaev and P. H. Siegel, "On Viterbi detector path metric differences," *IEEE Transactions on Communications*, vol. 46, pp. 1549–1554, December 1998.

[61] R. Calderbank, P. Fishburn, and P. Siegel, "State-space characterization of Viterbi detector path metric differences," in *Twenty-Sixth Asilomar Conference on Signals, Systems and Computers*, (Pacific Grove, California), pp. 940–944, October 1992.

[62] P. Siegel. Private communication, 2004.

[63] G. Montorsi and S. Benedetto, "Design of fixed-point iterative decoders for concatenated codes with interleavers," *IEEE Journal on Selected Areas in Communications*, vol. 19, pp. 871–882, May 2001.

[64] E. Boutillon, W. J. Gross, and P. G. Gulak, "VLSI architectures for the MAP algorithm," *IEEE Transactions on Communications*, vol. 51, pp. 175–185, February 2003.

[65] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Norwell, MA, USA: Kluwer Academic Publishers, 1992. ISBN 0-7923-9181-0.

[66] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. Wiley, 1991.

[67] J. M. Wozencraft and I. M. Jacobs, *Principles of Communication Engineering*. Prospect Heights, IL, USA: Waveland Press Inc., 1990. Originally Published 1965 by Wiley.

[68] W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes*. Cambridge, MA, USA: The M.I.T. Press, 2nd ed., 1972.