Maximum a Posteriori Decoding of Turbo Codes

by

Bernard Sklar

Introduction

The process of turbo-code decoding starts with the formation of *a posteriori probabilities* (APPs) for each data bit, which is followed by choosing the data-bit value that corresponds to the *maximum a posteriori* (MAP) probability for that data bit. Upon reception of a corrupted code-bit sequence, the process of decision-making with APPs allows the MAP algorithm to determine the most likely information bit to have been transmitted at each bit time. The metrics needed for the implementation of a MAP decoder are presented here, along with an example to illustrate how these metrics are used.

Viterbi Versus MAP

The MAP algorithm is unlike the *Viterbi algorithm* (VA), where the APP for each data bit is not available. Instead, the VA finds the most likely *sequence* to have been transmitted. However, there are similarities in the implementation of the two algorithms. When the decoded bit-error probability, P_B, is small, there is very little performance difference between the MAP and Viterbi algorithms. However, at low values of bit-energy to noise-power spectral density, E_b/N_0, and high values of P_B, the MAP algorithm can outperform decoding with a soft-output Viterbi algorithm called SOVA [1] by 0.5 dB or more [2]. For turbo codes, this can be very important, since the first decoding iterations can yield poor error performance. The implementation of the MAP algorithm proceeds somewhat like performing a Viterbi algorithm in two directions over a block of code bits. Once this bi-directional computation yields state and branch metrics for the block, the APPs and the MAP can be obtained for each data bit represented within the block. We describe here a derivation of the MAP decoding algorithm for systematic convolutional codes assuming an AWGN channel model, as presented by Pietrobon [2]. We start with the ratio of the APPs, known as the likelihood ratio $\Lambda(\hat{d}_k)$, or its logarithm, called the LLR, as shown below.

$$\Lambda(\hat{d}_k) = \frac{\sum_m \lambda_k^{1,m}}{\sum_m \lambda_k^{0,m}} \tag{1}$$

$$L(\hat{d}_k) = \log \left[\frac{\sum_m \lambda_k^{1,m}}{\sum_m \lambda_k^{0,m}} \right] \tag{2}$$

where $\lambda_k^{i,m}$, the joint probability that data $d_k = i$ and state $S_k = m$ conditioned on the received binary sequence R_1^N, observed from time $k = 1$ through some time N, is described as

$$\lambda_k^{i,m} = P(d_k = i, S_k = m \mid R_1^N) \tag{3}$$

R_1^N represents a corrupted code-bit sequence after it has been transmitted through the channel, demodulated, and presented to the decoder in soft-decision form. In effect, the MAP algorithm requires that the output sequence from the demodulator be presented to the decoder as a block of N bits at a time. Let R_1^N be written as follows:

$$R_1^N = \{R_1^{k-1}, R_k, R_{k+1}^N\} \tag{4}$$

To facilitate the use of Bayes' theorem, Equation (3) is partitioned using the letters A, B, C, D and Equation (4). Thus, Equation (3) can be written in this form:

$$\lambda_k^{i,m} = P\left(\underbrace{d_k = i,\ S_k = m}_{\text{A}} \;\middle|\; \underbrace{R_1^{k-1}}_{\text{B}},\ \underbrace{R_k}_{\text{C}},\ \underbrace{R_{k+1}^N}_{\text{D}}\right) \tag{5}$$

Recall from Bayes' theorem that

$$\begin{aligned} P(A \mid B,C,D) &= \frac{P(A,B,C,D)}{P(B,C,D)} = \frac{P(B \mid A,C,D)\ P(A,C,D)}{P(B,C,D)} \\ &= \frac{P(B \mid A,C,D)\ P(D \mid A,C)\ P(A,C)}{P(B,C,D)} \end{aligned} \tag{6}$$

Hence, application of this rule to Equation (5) yields

$$\lambda_k^{i,\,m} = P\left(R_1^{k-1} \middle| d_k = i,\ S_k = m,\ R_k^N\right)\ P\left(R_{k+1}^N \middle| d_k = i,\ S_k = m,\ R_k\right)$$
$$\times\ P\left(d_k = i,\ S_k = m,\ R_k\right) \Big/ P\left(R_1^N\right) \tag{7}$$

where $R_k^N = \left\{ R_k,\ R_{k+1}^N \right\}$. Equation (7) can be expressed in a way that gives greater meaning to the probability terms contributing to $\lambda_k^{i,\,m}$. In the sections that follow, the three numerator factors on the right side of Equation (7) will be defined and developed as the forward state metric, the reverse state metric, and the branch metric.

The State Metrics and the Branch Metric

We define the first numerator factor on the right side of Equation (7) as the forward state metric at time k and state m, and denote it as α_k^m. Thus, for $i = 1, 0$:

$$P\left(R_1^{k-1} \middle| \overbrace{d_k = i}^{\text{IRRELEVANT}},\ S_k = m,\ \overbrace{R_k^N}^{\text{IRRELEVANT}}\right) = P\left(R_1^{k-1} \middle| S_k = m\right) \ \triangleq\ \alpha_k^m \tag{8}$$

Notice that $d_k = i$ and R_k^N are designated as irrelevant, since the assumption that $S_k = m$ implies that events before time k are not influenced by observations after time k. In other words, the past is not affected by the future, hence $P\left(R_1^{k-1}\right)$ is independent of the fact that $d_k = i$ and the sequence R_k^N. However, since the encoder has memory, the encoder state $S_k = m$ is based on the past, so this term is relevant and must be left in the expression. The form of Equation (8) is intuitively satisfying, since it presents the forward state metric α_k^m at time k as being a probability of the past sequence; that is, dependent only on the current state induced by this sequence, and nothing more. This should be familiar from the convolutional encoder and its state representation as a Markov process [3].

Similarly, the second numerator factor on the right side of Equation (7) represents a reverse state metric, β_k^m, at time k and state m, described below

$$P\left(R_{k+1}^N \middle| d_k = i,\ S_k = m,\ R_k\right) = P\left(R_{k+1}^N \middle| S_{k+1} = f(i,\ m)\right) \ \triangleq\ \beta_{k+1}^{f(i,\,m)} \tag{9}$$

where $f(i, m)$ is the next state, given an input i and state m, and $\beta_{k+1}^{f(i,\,m)}$ is the reverse state metric at time $k + 1$ and state $f(i, m)$. The form of Equation (9) is intuitively satisfying since it presents the reverse state metric, β_{k+1}^m, at future time $k + 1$, as being a probability of the future sequence, which depends on the state (at future

time $k + 1$), which in turn is a function of the input bit and the state (at current time k). This should be familiar because it engenders the basic definition of a finite-state machine [3].

We define the third numerator factor on the right side of Equation (7) as the branch metric at time k and state m, denoted $\delta_k^{i,m}$. Thus, we write

$$P\left(d_k = i,\ S_k = m,\ R_k\right) \triangleq \delta_k^{i,m} \tag{10}$$

Substituting Equations (8) through (10) into Equation (7) yields the following more compact expression for the joint probability:

$$\lambda_k^{i,m} = \frac{\alpha_k^m \ \delta_k^{i,m} \ \beta_{k+1}^{f(i,m)}}{P\ (R_1^N)} \tag{11}$$

Equation (11) can be used to express Equations (1) and (2) as follows:

$$\Lambda(\hat{d}_k) = \frac{\sum_m \alpha_k^m \ \delta_k^{1,m} \ \beta_{k+1}^{f(1,m)}}{\sum_m \alpha_k^m \ \delta_k^{0,m} \ \beta_{k+1}^{f(0,m)}} \tag{12}$$

$$L(\hat{d}_k) = \log\left[\frac{\sum_m \alpha_k^m \ \delta_k^{1,m} \ \beta_{k+1}^{f(1,m)}}{\sum_m \alpha_k^m \ \delta_k^{0,m} \ \beta_{k+1}^{f(0,m)}}\right] \tag{13}$$

where $\Lambda(\hat{d}_k)$ is the likelihood ratio of the kth data bit, and $L(\hat{d}_k)$, the logarithm of $\Lambda(\hat{d}_k)$, is the LLR of the kth data bit, where the logarithm is generally taken to the base e.

Calculating the Forward State Metric

Starting from Equation (8), α_k^m can be expressed as the summation of all possible transition probabilities from time k-1, as follows:

$$\alpha_k^m = \sum_{m'} \sum_{j=0}^{1} P\left(d_{k-1} = j,\ S_{k-1} = m',\ R_1^{k-1} \,\middle|\, S_k = m\right) \tag{14}$$

We can rewrite R_1^{k-1} as $\left\{R_1^{k-2},\ R_{k-1}\right\}$, and from Bayes' theorem,

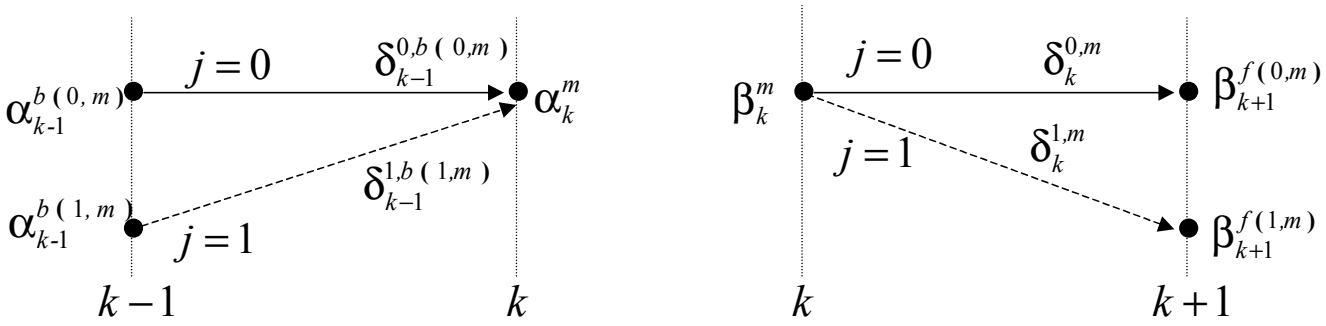
$$\alpha_k^m = \sum_{m'} \sum_{j=0}^{1} P\left(R_1^{k-2} \,\middle|\, S_k = m,\ d_{k-1} = j,\ S_{k-1} = m',\ R_{k-1}\right)$$
$$\times\ P\left(d_{k-1} = j,\ S_{k-1} = m',\ R_{k-1} \,\middle|\, S_k = m\right) \tag{15a}$$

$$= \sum_{j=0}^{1} P\left(R_1^{k-2} \,\middle|\, S_{k-1} = b(j, m)\right)\ P\left(d_{k-1} = j,\ S_{k-1} = b(j, m),\ R_{k-1}\right) \tag{15b}$$

where $b(j, m)$ is the state going backward in time from state m, via the previous branch corresponding to input j. Equation (15b) can replace Equation (15a) since knowledge about the state m' and the input j, at time k-1, completely defines the path resulting in state $S_k = m$. Using Equations (8) and (10) to simplify the notation of Equation (15) yields the following:

$$\alpha_k^m = \sum_{j=0}^{1} \alpha_{k-1}^{b(j, m)} \quad \delta_{k-1}^{j, b(j, m)} \tag{16}$$

Equation (16) indicates that a new forward state metric at time k and state m is obtained by summing two weighted state metrics from time k - 1. The weighting consists of the branch metrics associated with the transitions corresponding to data bits 0 and 1. Figure 1a illustrates the use of two different types of notation for the parameter alpha. We use $\alpha_{k-1}^{b(j, m)}$ for the forward state metric at time k - 1, when there are two possible underlying states (depending upon whether j = 0 or 1). And we use α_k^m for the forward state metric at time k, when the two possible transitions from the previous time terminate on the same state m at time k.



(a) Forward State Metric:

$$\alpha_k^m = \alpha_{k-1}^{b(0,m)} \delta_{k-1}^{0,b(0,m)} + \alpha_{k-1}^{b(1,m)} \delta_{k-1}^{1,b(1,m)}$$

Where $b(j, m)$ is the state going backward in time corresponding to an input j

(b) Reverse State Metric:

$$\beta_k^m = \beta_{k+1}^{f(0,m)} \delta_k^{0,m} + \beta_{k+1}^{f(1,m)} \delta_k^{1,m}$$

Where $f(j, m)$ is the next state given an input j and state m

Branch Metric:

$$\delta_k^{i,m} = \pi_k^i \exp\left(x_k u_k^i + y_k v_k^{i,m}\right)$$

Figure 1

Graphical representation for calculating α_k^m and β_k^m [2].

Calculating the Reverse State Metric

Starting from Equation (9), where $\beta_{k+1}^{f(i,m)} = P\left[R_{k+1}^N \,\middle|\, S_{k+1} = f(i,m)\right]$, we show β_k^m as follows:

$$\beta_k^m = P\left(R_k^N \,\middle|\, S_k = m\right) = P\left(R_k,\ R_{k+1}^N \,\middle|\, S_k = m\right) \tag{17}$$

We can express β_k^m as the summation of all possible transition probabilities to time $k + 1$, as follows:

$$\beta_k^m = \sum_{m'} \sum_{j=0}^{1} P\left(d_k = j,\ S_{k+1} = m',\ R_k,\ R_{k+1}^N \,\middle|\, S_k = m\right) \tag{18}$$

Using Bayes' theorem,

$$\beta_k^m = \sum_{m'} \sum_{j=0}^{1} P\left(R_{k+1}^N \middle| S_k = m,\ d_k = j, S_{k+1} = m',\ R_k\right) \times P\left(d_k = j,\ S_{k+1} = m',\ R_k \middle| S_k = m\right) \tag{19}$$

$S_k = m$ and $d_k = j$ in the first term on the right side of Equation (19) completely defines the path resulting in $S_{k+1} = f(j, m)$, the next state given an input j and state m. Thus, these conditions allow replacing $S_{k+1} = m'$ with $S_k = m$ in the second term of Equation (19), yielding the following:

$$\beta_k^m = \sum_{j=0}^{1} P\left(R_{k+1}^N \middle| S_{k+1} = f(j, m)\right)\ P\left(d_k = j,\ S_k = m,\ R_k\right)$$

$$= \sum_{j=0}^{1} \delta_k^{j,m}\ \beta_{k+1}^{f(j,m)} \tag{20}$$

Equation (20) indicates that a new reverse state metric at time k and state m is obtained by summing two weighted state metrics from time $k + 1$. The weighting consists of the branch metrics associated with the transitions corresponding to data bits 0 and 1. Figure 1b illustrates the use of two different types of notation for the parameter beta. We use $\beta_{k+1}^{f(j,m)}$ for the reverse state metric at time $k + 1$, when there are two possible underlying states (depending on whether $j = 0$ or 1). And, we use β_k^m for the reverse state metric at time k, where the two possible transitions arriving at time $k + 1$ stem from the same state m at time k. Figure 1 presents a graphical illustration for calculating the forward and reverse state metrics.

Implementing the MAP decoding algorithm has some similarities to implementing the Viterbi decoding algorithm [3]. In the Viterbi algorithm, we add branch metrics to state metrics. Then we compare and select the minimum distance (maximum likelihood) in order to form the next state metric. The process is called *add-compare-select* (*ACS*). In the MAP algorithm, we multiply (add, in the logarithmic domain) state metrics by branch metrics. Then, instead of comparing them, we sum them to form the next forward (or reverse) state metric, as seen in Figure 1. The differences should make intuitive sense. With the Viterbi algorithm, the most likely sequence (path) is being sought; hence there is a continual comparison and selection to find the best path. With the MAP algorithm, a soft number (likelihood or log-likelihood) is being sought; hence the process uses all the metrics from all the possible transitions within a time interval, in order to come up with the best overall statistic regarding the data bit associated with that time interval.

Calculating the Branch Metric

We start with Equation (10), which is rewritten below:

$$\delta_k^{i,\,m} = P\left(d_k = i,\ S_k = m,\ R_k\right)$$

$$= P\left(R_k \middle| d_k = i,\ S_k = m\right)\ P\left(S_k = m \middle| d_k = i\right)\ P\left(d_k = i\right) \tag{21}$$

where $R_k = x_k, y_k$, x_k is the noisy received data bit, and y_k is the corresponding noisy received parity bit. Since the noise affecting the data and the parity are independent, the current state is independent of the current input, and can therefore be any one of the 2^{υ} states, where υ is the number of memory elements in the convolutional code system. That is, the constraint length, K, of the code is equal to $\upsilon + 1$. Hence,

$$P\left(S_k = m \ \middle|\ d_k = i\right) = \frac{1}{2^{\upsilon}}$$

and

$$\delta_k^{i,m} = P\left(x_k \middle| d_k = i,\ S_k = m\right) P\left(y_k \middle| d_k = i,\ S_k = m\right)\ \frac{\pi_k^i}{2^{\upsilon}} \tag{22}$$

where π_k^i is defined as $P(d_k = i)$, the a priori probability of d_k.

The probability $P(X_k = x_k)$ of a random variable, X_k taking on the value x_k, is related to the *probability density function* (pdf) $p_{X_k}\left(x_k\right)$ as follows [3]:

$$P\left(X_k = x_k\right) = p_{X_k}\left(x_k\right) dx_k \tag{23}$$

For notational convenience, the random variable X_k, which takes on values x_k, is often termed "the random variable x_k", which represents the meanings of x_k and y_k in Equation (22). Thus, for an AWGN channel where the noise has zero mean and variance σ^2, we use Equation (23) in order to replace the probability terms in Equation (22) with their pdf equivalents, and we write

$$\delta_k^{i,m} = \frac{\pi_k^i}{2^{\upsilon}\sqrt{2\pi}\,\sigma}\ \exp\left[-\frac{1}{2}\left(\frac{x_k - u_k^i}{\sigma}\right)^2\right] dx_k\ \ \frac{1}{\sqrt{2\pi}\,\sigma}\exp\left[-\frac{1}{2}\left(\frac{y_k - v_k^{i,m}}{\sigma}\right)^2\right] dy_k \tag{24}$$

where u_k and v_k represent the transmitted data bits and parity bits, respectively (in bipolar form), and dx_k and dy_k are the differentials of x_k and y_k, and get absorbed into the constant A_k, below. Note that the parameter u_k^i represents data that has no dependence on the state m. However, the parameter $v_k^{i,m}$ represents parity, which does depend on the state m, since the code has memory. Simplifying the notation by eliminating all terms that will appear in both the numerator and denominator of the likelihood ratio resulting in cancellation, we can write

$$\delta_k^{i,m} = A_k \, \pi_k^i \exp\left[\frac{1}{\sigma^2} \left(x_k u_k^i + y_k v_k^{i,m} \right) \right] \tag{25}$$

If we substitute Equation (25) into Equation (1), we obtain

$$\Lambda(\hat{d}_k) = \pi_k \exp\left(\frac{2x_k}{\sigma^2} \right) \frac{\sum_m \alpha_k^m \exp\left(\frac{y_k v_k^{1,m}}{\sigma^2} \right) \beta_{k+1}^{f(1,m)}}{\sum_m \alpha_k^m \exp\left(\frac{y_k v_k^{0,m}}{\sigma^2} \right) \beta_{k+1}^{f(0,m)}} \tag{26a}$$

$$= \pi_k \exp\left(\frac{2x_k}{\sigma^2} \right) \pi_k^e \tag{26b}$$

and

$$L(\hat{d}_k) = L(d_k) + L_c(x_k) + L_e(\hat{d}_k) \tag{26c}$$

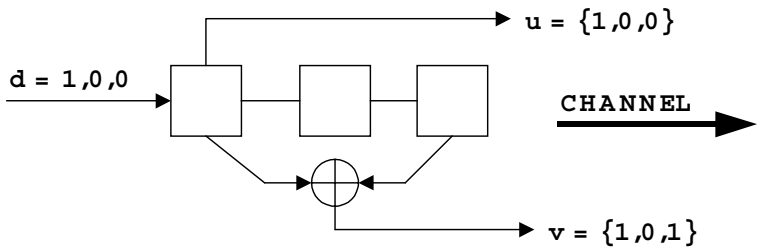
where $\pi_k = \pi^1/\pi^0$ is the input a priori probability ratio (prior likelihood), and π_k^e is the output extrinsic likelihood, each at time k. In Equation (26b), one can think of π_k^e as a correction term (due to the coding) that changes the input prior knowledge about a data bit. In a turbo code, such correction terms are passed from one decoder to the next, in order to improve the likelihood ratio for each data bit, and thus minimize the probability of decoding error. Thus, the decoding process entails the use of Equation (26b) to compute $\Lambda(\hat{d}_k)$ for several iterations. The extrinsic likelihood, π_k^e, resulting from a particular iteration replaces the a priori likelihood ratio, π_{k+1}, for the next iteration. Taking the logarithm of $\Lambda(\hat{d}_k)$ in Equation (26b) yields Equation (26c) which shows that the final soft number $L(\hat{d}_k)$ is made up of

three LLR terms—the a priori LLR, the channel-measurement LLR, and the extrinsic LLR [3].

The MAP algorithm can be implemented in terms of a likelihood ratio $\Lambda(\hat{d}_k)$ as shown in Equations (26a) and (26b). However, implementation using likelihood ratios is very complex because of the multiplication operations that are required. By operating the MAP algorithm in the logarithmic domain [2, 4] as described by the LLR in Equation (26c), the complexity can be greatly reduced by eliminating the multiplication operations.

MAP Decoding Example

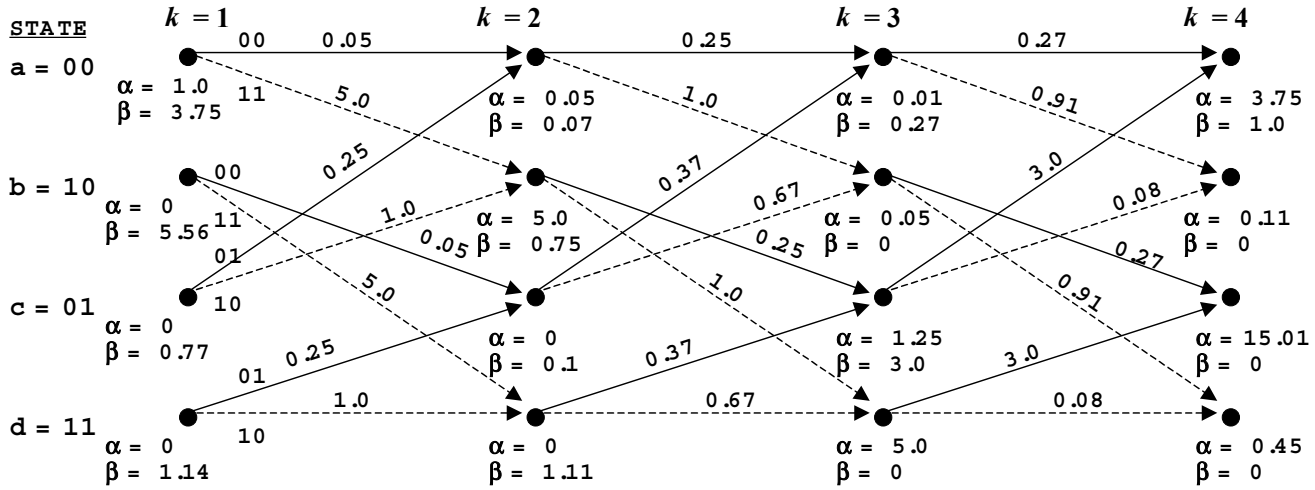
Figure 2 illustrates a MAP decoding example. Figure 2a shows a simple systematic convolutional encoder, with constraint length $K = 3$ and rate ½. The input data consists of the sequence $\mathbf{d} = \{1, 0, 0\}$, corresponding to the times $k = 1, 2, 3$. The output code-bit sequence, being systematic, is formed by consecutively taking one bit from the sequence $\mathbf{u} = \{1, 0, 0\}$ followed by one bit from the parity-bit sequence $\mathbf{v} = \{1, 0, 1\}$. In each case, the leftmost bit is the earliest bit. Thus, the output sequence is 1 1 0 0 0 1, or in bipolar form the sequence is +1 +1 -1 -1 -1 +1. Figure 2b shows the results of some postulated noise vectors, $\mathbf{n_x}$ and $\mathbf{n_y}$, having corrupted sequences **u** and **v**, so they are now designated as $\mathbf{x} = \mathbf{u} + \mathbf{n_x}$ and $\mathbf{y} = \mathbf{v} + \mathbf{n_y}$. As shown in Figure 2b, the demodulator outputs arriving at the decoder corresponding to the times $k = 1, 2, 3$ have values of 1.5, 0.8, 0.5, 0.2, -0.6, 1.2. Also shown are the a priori probabilities of a data bit being 1 or 0, designated as π^1 and π^0, respectively, and assumed to be equally likely for all k times. For this example, all information is now available to calculate the branch metrics and the state metrics, and enter their values onto the decoder trellis diagram of Figure 2c. On the trellis diagram, each transition occurring between times k and $k + 1$ corresponds to a data bit, d_k, that appears at the encoder input at the transition-start time k. At time k, the encoder will be in some state m, and at time $k + 1$ it transitions to a new state (possibly the same state). When such a trellis diagram is used to depict a sequence of code bits (representing N data bits), the sequence is characterized by N transition intervals and $N + 1$ states (from start to finish).



	$k=1$	$k=2$	$k=3$
x =	1.5,	0.5,	-0.6}
y =	{0.8,	0.2,	1.2}
π^1 = π^0 =	{0.5,	0.5,	0.5}

(a) Systematic Convolutional Encoder ($K = 3$, Rate ½)

(b) Received Channel Bits (bipolar) plus Noise



$$L(\hat{d}_1) = \log\left(\frac{3.75}{0.0035}\right) = 3.03 \qquad L(\hat{d}_2) = \log\left(\frac{0}{3.75}\right) = -\infty \qquad L(\hat{d}_3) = \log\left(\frac{0}{3.75}\right) = -\infty$$

(c) Decoder Trellis Diagram

Figure 2

Example of MAP decoding ($K = 3$, rate ½, systematic).

Calculating the Branch Metrics

We start with Equation (25), with $\pi_k^i = 0.5$ (for this exercise, data bits are assumed equally likely for all time), and for simplicity assume that $A_k = 1$ for all time, and that $\sigma^2 = 1$. Thus $\delta_k^{i,m}$ becomes

$$\delta_k^{i,m} = 0.5 \exp\left(x_k u_k^i + y_k v_k^{i,m}\right) \tag{27}$$

What basic receiver function does Equation (27) resemble? The expression "looks" somewhat like a correlation process. At the decoder, a pair of receptions (data-bit related x_k and parity-bit related y_k) arrive at each time k. The branch metric is calculated by taking the product of the received x_k with each of the prototype signals u_k, and similarly the product of the received y_k with each of the prototype signals v_k. For each trellis transition, the magnitude of the branch metric will be a function of how good a match there is between the pair of noisy receptions and the code-bit meaning of that trellis transition. For $k = 1$, Equation (27) is used with the data in Figure 2b for evaluating eight branch metrics (a transition from each state m and for each data value i), as shown below. For notational simplicity, we designate the trellis states as follows: $a = 00$, $b = 10$, $c = 01$, $d = 11$. Note that the code-bit meaning, u_k, v_k, of each trellis transition is written on the transition in Figure 2c (for $k = 1$ only) and was obtained from the encoder structure in the usual way [3]. Also, for the trellis transitions of Figure 2c, the convention is used that dashed lines and solid lines correspond to the underlying data bits 1 and 0, respectively.

$$\delta_{k=1}^{1,\, m=a} = \delta_{k=1}^{1,\, m=b} = 0.5 \ \exp\left[(1.5)\ \ (1) \ + \ (0.8)\ \ (1) \right] = 5.0$$

$$\delta_{k=1}^{0,\, m=a} = \delta_{k=1}^{0,\, m=b} = 0.5 \ \exp\left[(1.5)\ \ (-1) \ + \ (0.8)\ \ (-1) \right] = 0.05$$

$$\delta_{k=1}^{1,\, m=c} = \delta_{k=1}^{1,\, m=d} = 0.5 \ \exp\left[(1.5)\ \ (1) \ + \ (0.8)\ \ (-1) \right] = 1.0$$

$$\delta_{k=1}^{0,\, m=c} = \delta_{k=1}^{0,\, m=d} = 0.5 \ \exp\left[(1.5)\ \ (-1) \ + \ (0.8)\ \ (1) \right] = 0.25$$

Next, we repeat these calculations using Equation (27) for the eight branch metric values at time $k = 2$.

$$\delta_{k=2}^{1,\, m=a} = \delta_{k=2}^{1,\, m=b} = 0.5 \ \exp\left[(0.5)\ \ (1) \ + \ (0.2)\ \ (1) \right] = 1.0$$

$$\delta_{k=2}^{0,\, m=a} = \delta_{k=2}^{0,\, m=b} = 0.5 \ \exp\left[(0.5)\ \ (-1) \ + \ (0.2)\ \ (-1) \right] = 0.25$$

$$\delta_{k=2}^{1,\, m=c} = \delta_{k=2}^{1,\, m=d} = 0.5 \ \exp\left[(0.5)\ \ (1) \ + \ (0.2)\ \ (-1) \right] = 0.67$$

$$\delta_{k=2}^{0,\, m=c} = \delta_{k=2}^{0,\, m=d} = 0.5 \ \exp\left[(0.5)\ \ (-1) \ + \ (0.2)\ \ (1) \right] = 0.37$$

Again, we repeat the calculations for the eight branch metric values at time $k = 3$.

$$\delta_{k=3}^{1,\ m=a} = \delta_{k=3}^{1,\ m=b} = 0.5\ \exp\ [\ (-0.6)\ \ (1)\ +\ (1.2)\ \ (1)\ ] = 0.91$$

$$\delta_{k=3}^{0,\ m=a} = \delta_{k=3}^{0,\ m=b} = 0.5\ \exp\ [\ (-0.6)\ \ (-1)\ +\ (1.2)\ \ (-1)\ ] = 0.27$$

$$\delta_{k=3}^{1,\ m=c} = \delta_{k=3}^{1,\ m=d} = 0.5\ \exp\ [\ (-0.6)\ \ (1)\ +\ (1.2)\ \ (-1)\ ] = 0.08$$

$$\delta_{k=3}^{1,\ m=c} = \delta_{k=3}^{1,\ m=d} = 0.5\ \exp\ [\ (-0.6)\ \ (-1)\ +\ (1.2)\ \ (1)\ ] = 3.0$$

Calculating the State Metrics

Once the eight values of $\delta_k^{i,m}$ are computed for each k, the forward state metrics α_k^m can be calculated with the help of Figures 1 and 2c and Equation (16), rewritten below:

$$\alpha_{k+1}^m = \sum_{j=0}^{1} \delta_k^{j,b(j,m)} \alpha_k^{b(j,m)}$$

Assume that the encoder starting state is $a = 00$. Then,

$$\alpha_{k=1}^{m=a} = 1.0 \quad \text{and} \quad \alpha_{k=1}^{m=b} = \alpha_{k=1}^{m=c} = \alpha_{k=1}^{m=d} = 0$$

$$\alpha_{k=2}^{m=a} = (0.05)(1.0) + (0.25)(0) = 0.05$$

$$\alpha_{k=2}^{m=b} = (5.0)(1.0) + (1.0)(0) = 5.0$$

$$\alpha_{k=2}^{m=c} = \alpha_{k=2}^{m=d} = 0$$

and so forth, as shown on the trellis diagram of Figure 2c. Similarly, the reverse state metric β_k^m can be calculated with the help of Figures 1 and 2c and Equation (20), rewritten below:

$$\beta_k^m = \sum_{j=0}^{1} \delta_k^{j,m} \beta_{k+1}^{f(j,m)}$$

The data sequence and the code in this example was purposely chosen so that the final state of the trellis at time $k = 4$ is the $a = 00$ state. Otherwise, it would be necessary to use tail bits to force the final state into such a known state. Thus, for

the example illustrated in Figure 2, knowing that the final state is $a = 00$, the reverse state metrics can be calculated as follows:

$$\beta_{k=4}^{m=a} = 1.0 \quad \text{and} \quad \beta_{k=4}^{m=b} = \beta_{k=4}^{m=c} = \beta_{k=4}^{m=d} = 0$$

$$\beta_{k=3}^{m=a} = (0.27)(1.0) + (0.91)(0) = 0.27$$

$$\beta_{k=3}^{m=b} = \beta_{k=3}^{m=d} = 0$$

$$\beta_{k=3}^{m=c} = (3.0)(1.0) + (0.08)(0) = 3.0$$

and so forth. All the reverse state metric values are shown on the trellis of Figure 2c.

Calculating the Log-Likelihood Ratio

Now that the metrics δ, α, and β have all been computed for the code-bit sequence in this example, the turbo decoding process can use Equations (12), (13), or (26) for finding a soft decision, $\Lambda(\hat{d}_k)$ or $L(\hat{d}_k)$, for each data bit. When using turbo codes, this process can be iterated several times to improve the reliability of that decision. This is generally accomplished by using the extrinsic likelihood parameter of Equation (26b) to compute and recompute the likelihood ratio, $\Lambda(\hat{d}_k)$, for several iterations. The extrinsic likelihood, π_k^e, of the last iteration replaces the a priori likelihood ratio π_{k+1} for the next iteration.

For this example, let's use the metrics calculated above (with a single pass through the decoder). We choose Equation (13) to compute the LLR for each data bit in the sequence $\{d_k\}$, and then to transform the resulting soft numbers into hard decisions, we use the following decision rules:

$$\hat{d}_k = 1 \text{ if } L(\hat{d}_k) > 0, \text{ and } \hat{d}_k = 0 \text{ if } L(\hat{d}_k) < 0 \tag{28}$$

For $k = 1$, omitting some of the zero factors, we obtain

$$L(\hat{d}_1) = \log\left(\frac{1.0 \text{ x } 5.0 \text{ x } 0.75}{1.0 \text{ x } 0.05 \text{ x } 0.07}\right) = \log\left(\frac{3.75}{0.0035}\right) = 3.03$$

For $k = 2$, again omitting some of the zero factors, we obtain

$$L(\hat{d}_2) = \log \left[\frac{(0.05 \text{ x } 1.0 \text{ x } 0) + (5.0 \text{ x } 1.0 \text{ x } 0)}{(0.05 \text{ x } 0.25 \text{ x } 0.27) + (5.0 \text{ x } 0.25 \text{ x } 3.0)} \right] = \log\left(\frac{0}{3.75}\right) = -\infty$$

For $k = 3$, we obtain

$$L(\hat{d}_3) = \log \left[\frac{(0.01 \times 0.91 \times 0) + (0.05 \times 0.91 \times 0) + (1.25 \times 0.08 \times 0) + (5.0 \times 0.08 \times 0)}{(0.01 \times 0.27 \times 1.0) + (0.05 \times 0.27 \times 0) + (1.25 \times 3.0 \times 1.0) + (5.0 \times 3.0 \times 0)} \right]$$

$$= \log\left(\frac{0}{3.75}\right) = -\infty$$

Using Equation (28) to make the final decisions about the bits at times $k = 1, 2, 3$, the sequence is decoded as {1 0 0}. This is clearly correct, given the specified input to the encoder.

Conclusion

We used basic statistical measures and techniques, such as joint probability, Bayes' theorem, a priori and a posteriori probability, and likelihood, for describing how the MAP algorithm is applied to turbo decoding. We then used a numerical example to show how the trellis diagram is traversed in two directions, resulting in soft-decision outputs.

References

[1] Hagenauer, J. and Hoeher, P., "A Viterbi Algorithm with Soft-Decision Outputs and Its Applications," *Proc. GLOBECOM '89*, Dallas, Texas, November 1989, pp. 1680-1686.

[2] Pietrobon, S. S., "Implementation and Performance of a Turbo/MAP Decoder," *Int'l. J. Satellite Communications*, vol. 15, Jan-Feb 1998, pp. 23-46.

[3] Sklar, B., *Digital Communications: Fundamentals and Applications, Second Edition* (Upper Saddle River, NJ: Prentice-Hall, 2001).

[4] Robertson, P, Villebrun, E., and Hoeher, P., "A Comparison of Optimal and Sub-Optimal MAP Decoding Algorithms Operating in the Log Domain," *Proc. of ICC '95*, Seattle, Washington, June 1995, pp. 1009-1013.

About the Author

Bernard Sklar is the author of *Digital Communications: Fundamentals and Applications, Second Edition* (Prentice-Hall, 2001, ISBN 0-13-084788-7).