Abstract

The goal of this article is to describe the main ideas behind the new class of codes called turbo codes, whose performance in terms of bit error probability has been shown to be very close to the Shannon limit. In this article, the mathematical measures of a posteriori probability and likelihood are reviewed, and the benefits of turbo codes are explained in this context. Since the algorithms needed to implement the decoders have been well documented by others, they are only referenced here, not described in detail. A numerical example, using a simple concatenated coding scheme, provides a vehicle for illustrating how error performance can be improved when soft outputs from the decoders are used in an iterative decoding process.

A Primer on Turbo Code Concepts

Bernard Sklar, Communications Engineering Services

Concatenated coding schemes were first proposed by Forney [1] for achieving large coding gains by combining two or more relatively simple component or building-block codes. The resulting codes had the error correction capability of much longer codes, and were endowed with a structure that permitted relatively easy to moderately complex decoding. A serial concatenation of codes is most often used for power-limited systems such as deep space probes. The most popular of these schemes consists of a Reed-Solomon outer code (applied first, removed last) followed by a convolutional inner code (applied last, removed first) [2]. A turbo code can be thought of as a refinement of the concatenated encoding structure and an iterative algorithm for decoding the associated code sequence.

Turbo codes were introduced by Berrou, Glavieux, and Thitimajshima [3, 4]. For a bit error probability of $10^{-5}$ and code rate $R = 1/2$, the authors report a required $E_b/N_0$ of 0.7 dB. The codes are constructed by applying two or more component codes to different interleaved versions of the same information sequence. For any single traditional code, the final step at the decoder yields hard-decision decoded bits (or, more generally, decoded symbols). In order for a concatenated code such as a turbo code to work properly, the decoding algorithm should not limit itself to passing hard decisions among the decoders. To best exploit the information learned from each decoder, the decoding algorithm must effect an exchange of soft rather than hard decisions. For a system with two component codes, the concept behind turbo decoding is to pass soft decisions from the output of one decoder to the input of the other, and to iterate this process several times to produce better decisions.

A Review of Likelihoods

The mathematical foundations of hypothesis testing rest on Bayes' theorem, which is derived from the relationship between the conditional and joint probability of events $A$ and $B$, as follows:

$$P(A | B) = \frac{P(B | A) P(A)}{P(B)}$$  \hspace{1cm} (1)

A statement of the theorem yields the a posteriori probability (APP), denoted $P(A | B)$.

$$P(A | B) = \frac{P(B | A) P(A)}{P(B)}$$  \hspace{1cm} (2)

which allows us to infer the APP of an event $A$ conditioned on $B$, from the conditional probability, $P(B | A)$, and the a priori probabilities, $P(A)$ and $P(B)$.

For communications engineering applications having additive white Gaussian noise (AWGN) in the channel, the most useful form of Bayes' theorem expresses the APP in terms of a continuous-valued random variable $x$ in the following form:

$$P(d = i | x) = \frac{p(x | d = i) p(d = i)}{p(x)}$$  \hspace{1cm} i = 1, \ldots, M$$  \hspace{1cm} (3)

and

$$p(x) = \sum_{i=1}^{M} p(x | d = i) p(d = i)$$  \hspace{1cm} (4)

where $d = i$ represents data $d$ belonging to the $i$th signal class from a set of $M$ classes, and $p(x | d = i)$ represents the probability density function (pdf) of a received continuous-valued data-plus-noise signal, $x$, conditioned on the signal class $d = i$. $p(x)$ is the pdf of the received signal $x$ over the entire space of signal classes. In Eq. 3, for a particular received signal, $p(x)$ is a scaling factor since it has the same value for each class. Lower case $p$ is used to designate the pdf of a continuous-valued signal, and upper case $P$ is used to designate probability (a priori and APP). Equation 3 can be thought of as the result of an experiment involving a received signal and some statistical knowledge of the signal classes to which the signal may belong. The probability of occurrence of the $i$th signal class $P(d = i)$, before the experiment, is the a priori probability. As a result of examining a particular received signal, we can compute the APP, $P(d = i | x)$, which can be thought of as a "refinement" of our prior knowledge.

The Two-Signal Class Case

Let the binary logical elements 1 and 0 be represented electronically by voltages $+1$ and $-1$, respectively. This pair of transmitted voltages is assigned to the variable $d$, which may now take on the values $d = +1$ and $d = -1$. Let
the binary 0 (or the voltage value -1) be the null element under addition. For an AWGN channel, Fig. 1 shows the conditional pdfs referred to as likelihood functions. The rightmost function \( p(x|d = +1) \) shows the pdf of the random variable \( x \) given the condition that \( d = +1 \) was transmitted. The leftmost function \( p(x|d = -1) \) illustrates a similar pdf given that \( d = -1 \) was transmitted. The abscissa represents the full range of possible values of \( x \) received at the detector. In Fig. 1 is shown one such arbitrary predetection value \( x_k \). A line subtended from \( x_k \) intercepts the two likelihood functions yielding two likelihood values \( \lambda_1 \) and \( \lambda_2 \). A popular hard decision rule, known as maximum likelihood, is to choose the symbol \( d = +1 \) or \( d = -1 \) associated with the larger of the two intercept values, \( \lambda_1 \) or \( \lambda_2 \). This is tantamount to deciding \( d_k = +1 \) if \( x_k \) falls on the right side of the decision line labeled \( y_0 \), otherwise deciding \( d_k = -1 \).

A similar decision rule, known as maximum a posteriori (MAP) or minimum-error rule, takes into account the a priori probabilities, as seen below in Eq. 6. The MAP rule is expressed in terms of APPs as follows:

\[
H_1 \frac{P(d = +1|x)}{H_2} \geq \frac{P(d = -1|x)}{H_2}
\]

Equation 5 states that one should choose the hypothesis \( H_1 \) \((d = +1)\) if the APP \( P(d = +1|x) \) is greater than the APP \( P(d = -1|x) \). Otherwise, choose hypothesis \( H_2(d = -1) \). Using the Bayes' theorem in Eq. 3, the APPs in Eq. 5 can be replaced by their equivalent expressions, yielding:

\[
p(x|d = +1)P(d = +1|x) \geq \frac{H_1}{H_2} p(x|d = -1)P(d = -1|x)
\]

Equation 6 is generally expressed in terms of a ratio, called the likelihood ratio test, as follows:

\[
p(x|d = +1) \frac{H_1}{H_2} p(x|d = -1)P(d = +1|x)
\]

or

\[
p(x|d = +1)P(d = +1|x) \frac{H_1}{H_2} p(x|d = -1)P(d = -1|x)
\]

The log-likelihood ratio (LLR) by taking the logarithm of the likelihood ratio in Eq. 7, we obtain a useful metric called the log-likelihood ratio (LLR). It is the real number representing a soft decision out of a detector, designated \( L(d) \), as follows:

\[
L(d|x) = \log \frac{p(x|d = +1)}{p(x|d = -1)}
\]

where \( p(x|d) \) is the LLR of the channel measurements of \( x \) under the alternate conditions that \( d = +1 \) or \( d = -1 \) may have been transmitted, and \( L(d) \) is the a priori LLR of the data bit \( d \).

To simplify the notation, we represent Eq. 10 as follows:

\[
L'(\hat{d}) = L_e(x) + L(d)
\]

where the notation \( L_e(x) \) emphasizes that this LLR term is the result of a channel measurement made at the detector. Equations 3 through 11 were derived with only a data detector in mind. Next, the introduction of a decoder will typically yield decision-making benefits. For a systematic code, it can be shown [3] that the LLR (soft output) \( L(\hat{d}) \) out of the decoder is equal to

\[
L(\hat{d}) = L'(\hat{d}) + L_e(\hat{d})
\]

where \( L'(\hat{d}) \) is the LLR of a data bit out of the detector (input to the decoder), and \( L_e(\hat{d}) \), called the extrinsic LLR, represents extra knowledge that is gleaned from the decoding process. The output sequence of a systematic decoder is made up of values representing data and parity. Equation 12 partitions the decoder LLR into the data portion represented by the detector measurement, and the extrinsic portion represented by the decoder contribution due to parity. From Eqs. 11 and 12, we write

\[
L(\hat{d}) = L_e(x) + L(d) + L_e(\hat{d})
\]

The soft decision \( L(\hat{d}) \) is a real number that provides a hard decision as well as the reliability of that decision. The sign of \( L(\hat{d}) \) denotes the hard decision; that is, for positive values of \( L(\hat{d}) \) decide +1, for negative values decide -1. The magnitude of \( L(\hat{d}) \) denotes the reliability of that decision.

**Principles of Iterative (Turbo) Decoding**

For the first decoding iteration of the soft input/soft output decoder in Fig. 2, one generally assumes the binary data to be equally likely,
yielding an initial a priori LLR value of $L(d) = 0$ for the third term in Eq. 9. The channel predetection LLR value, $L(x)$, is measured by forming the logarithm of the ratio of $\lambda_1$ and $\lambda_2$, seen in Fig. 1, and appearing as the second term of Eq. 9. The output $L(\hat{d})$ of the Fig. 2 decoder is made up of the LLR from the detector, $L(\hat{d})$, and the extrinsic LLR output, $L(\hat{d})$, representing knowledge gleaned from the decoding process. As illustrated in Fig. 2, for iterative decoding the extrinsic feedback is fed back to the decoder input, to serve as a refinement of the a priori value for the next iteration.

Consider the two-dimensional code (product code) depicted in Fig. 3. The configuration can be described as a data array made up of $k_1$ rows and $k_2$ columns. Each of the $k_1$ rows contains a code vector made up of $k_2$ data bits and $n_2 - k_2$ parity bits. Similarly, each of the $k_2$ columns contains a code vector made up of $k_1$ data bits and $n_1 - k_1$ parity bits. The various portions of the structure are labeled $d$ for data, $p_h$ for horizontal parity (along the rows), and $p_v$ for vertical parity (along the columns). Additionally, there are blocks labeled $L_{eh}$ and $L_{ev}$, which house the extrinsic LLR values learned from the horizontal and vertical decoding steps, respectively. Notice that this product code is a simple example of a concatenated code. Its structure encompasses two separate encoding steps: horizontal and vertical. The iterative decoding algorithm for this product code proceeds as follows:

1. Set the a priori information $L(d) = 0$.
2. Decode horizontally, and using Eq. 13 obtain the horizontal extrinsic information as shown below:
   \begin{align*}
   L_{eh}(\hat{d}) & = L(\hat{d}) - L(\hat{x}) - L(d) \\
   \end{align*}
3. Set $L(d) = L_{eh}(\hat{d})$.
4. Decode vertically, and using Eq. 13 obtain the vertical extrinsic information as shown below:
   \begin{align*}
   L_{ev}(\hat{d}) & = L(\hat{d}) - L(\hat{x}) - L(d) \\
   \end{align*}
5. Set $L(d) = L_{ev}(\hat{d})$.
6. If there have been enough iterations to yield a reliable decision, go to step 7; otherwise, go to step 2.
7. The soft output is:
   \begin{align*}
   L(\hat{d}) & = L(\hat{x}) + L_{eh}(\hat{d}) + L_{ev}(\hat{d}) . \\
   \end{align*}

\section*{Two-Dimensional Single-Parity Code Example}

At the encoder, let the data bits and parity bits take on the values shown in Fig. 4, where the relationships between the data and parity expressed as binary digits (0,1), are as follows:

\begin{align*}
   d_i \oplus d_j & = p_{ij} \\
   d_i & = d_i \oplus p_{ij} \\
   d_j & = d_i \oplus p_{ij} \\
\end{align*}

where $\oplus$ denotes modulo-2 addition. As shown in Fig. 4, the transmitted symbols are represented by the sequence $d_1 \ d_2 \ d_3 \ d_4 \ p_{12} \ p_{34} \ p_{13} \ p_{24}$. At the receiver input, the received symbols are represented by the sequence $x_i, x_j$, where $x_i = d_i + n$ for each received data signal. Thus, the received parity signal, and $n$ represents independent and identically distributed noise. For notational simplicity, we shall denote the received sequence with a single index, as $(x_k)$, where $k$ can be treated as a time index. Using the relationships developed in Eqs. 9 through 11, and assuming Gaussian noise, we can write the LLR for the channel measurement of a received signal $x_k$ as follows:

\begin{align*}
   L(\hat{x}_k) & = \log_e \left( \frac{p(\hat{x}_k|d_k = +1)}{p(\hat{x}_k|d_k = -1)} \right) \\
   & = \log_e \left( \frac{1}{\sigma \sqrt{2\pi}} \exp \left( -\frac{1}{2} \left( \frac{x_k - 1}{\sigma} \right)^2 \right) \right) \\
   & = -\frac{1}{2} \left( \frac{x_k - 1}{\sigma} \right)^2 + \frac{1}{2} \left( \frac{x_k + 1}{\sigma} \right)^2 = \frac{2}{\sigma^2} x_k \\
\end{align*}

where the natural logarithm is used. If we further make a simplifying assumption that the noise variance $\sigma^2$ is unity, then

\begin{align*}
   L(\hat{x}_k) & = 2 x_k \\
\end{align*}

Consider the following example, where the data sequence $d_1 \ d_2 \ d_3 \ d_4$ is made up of the binary digits 1 0 0 1, as shown in Fig. 4. By the use of Eq. 15, it is seen that the parity sequence $p_{12} \ p_{34} \ p_{13} \ p_{24} = 1$ 1 1 1. Thus, the transmitted sequence is

\begin{align*}
   \{d_i, p_{ij}\} & = 1 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \\
\end{align*}

which is shown in Fig. 4 as the encoder output. Expressed in terms of bipolar voltage values, the transmitted sequence is

\begin{align*}
   +1 & -1 +1 +1 +1 -1 +1 \\
\end{align*}

Let us assume that the noise transforms this data-plus-parity sequence into the received sequence

\begin{align*}
   \{x_k\} & = 0.75, 0.05, 0.10, 0.15, 1.25, 1.0, 3.0, 0.5 \\
\end{align*}

From Eq. 21, the assumed channel measurements yield the LLR values

\begin{align*}
   \{L(\hat{x}_k)\} & = 1.5, 0.1, 0.20, 0.3, 2.5, 2.0, 6.0, 1.0 \\
\end{align*}

which is shown in Fig. 4 as the decoder input measurements. It should be noted that, if hard decisions are made on the $\{x_k\}$ or the $\{L(\hat{x}_k)\}$ values shown above, such a detection process would result in two errors, since $d_2$ and $d_3$ would each be incorrectly classified as binary 1.

\section*{Log-Likelihood Algebra}

To best explain the iterative feedback of soft decisions, we introduce the idea of a log-likelihood algebra [5]. For statistically independent $d$, we define the sum of two log likelihood ratios (LLRs) as follows:

\begin{align*}
   d_i \oplus d_j & = p_{ij} \\
   d_i & = d_i \oplus p_{ij} \\
   d_j & = d_i \oplus p_{ij} \\
\end{align*}
where the natural logarithm is used. There are three addition operations in Eq. 22. + is used for ordinary addition, ⊕ is used for modulo-2 addition of the data expressed as binary digits, and is used for log-likelihood addition. The sum of two LLRs is denoted by the operator , where such an addition is defined as the LLR of the modulo-2 sum of the underlying data bits. The development of the Eq. 22 equality is shown in Appendix A. Equation 23 is an approximation of Eq. 22 that will prove useful later in a numerical example. The sum of LLRs as described by Eq. 22 or 23 yield the following interesting results when one of the LLRs is very large or very small:

\[ L(d) \boxplus \infty = -L(d) \]

and

\[ L(d) \boxplus 0 = 0 \]

Note that the log-likelihood algebra described here differs slightly from Hagenauer's reference [5] because of choosing the null element differently. In this article, the null element of the binary set (0,1) has been chosen to be the 0.

**EXTRINSIC LIKELIHOODS**

For the product-code example in Fig. 4, we use Eq. 13 to express the soft output \( L(d_1) \) for the received signal corresponding to data \( d_1 \), as follows:

\[ L(d_1) = L_c(x_1) + L(d_1) \oplus (L_c(x_2) + L(d_2)) \boxplus L_c(x_{12}) \]

(24)

where the terms \((L_c(x_2) + L(d_2)) \boxplus L_c(x_{12})\) represent the extrinsic LLR contributed by the code (i.e., the signal corresponding to data \( d_2 \) and its a priori probability, in conjunction with the signal corresponding to parity \( p_{12} \)). In general, the soft output \( L(d) \) for the received signal corresponding to data \( d \) is:

\[ L(d) = L_c(x) + L(d) \oplus (L_c(x_1) + L(d_1)) \boxplus L_c(x_{12}) \]

(25)

where \( L_c(x), L_c(x_1) \), and \( L_c(x_{12}) \) are the channel LLR measurements of the received signals corresponding to \( d_1, d_2, \) and \( p_{12} \) respectively. \( L(d) \) and \( L(d_1) \) are the LLRs of the a priori probabilities of \( d_1 \) and \( d_2 \), respectively, and \((L_c(x_1) + L(d_1) \boxplus L_c(x_{12}))\) is the extrinsic LLR contribution from the code. Equations 24 and 25 can be best understood in the context of the decoder in Fig. 4. For example, the soft output \( L(d_1) \) is represented by the detector LLR measurement of received signal corresponding to data \( d_1 \), and the extrinsic LLR gleaned from the fact that data \( d_2 \) and parity \( p_{12} \) also provide knowledge about data \( d_1 \), as seen from Eqs. 15 through 17.

**COMPUTING THE EXTRINSIC LIKELIHOODS**

For the example in Fig. 4, the horizontal calculations for \( L_{eh}(\hat{d}) \) and the vertical calculations for \( L_{ev}(\hat{d}) \) are expressed as follows:

\[ L_{eh}(\hat{d}_1) = [L_c(x_2) + L(d_2)] \boxplus L_c(x_{12}) \]

(26a)

\[ L_{ev}(\hat{d}_1) = [L_c(x_3) + L(d_3)] \boxplus L_c(x_{13}) \]

(26b)
extrinsic LLRs yields the improvement in Table 4 (the extrinsic vertical terms are not yet being considered).

\[
\begin{array}{cc}
1.4 & -1.4 \\
-0.1 & 0.1 \\
\end{array}
\]

*Table 4. Improved LLRs due to $L_{oh}(\hat{d})$.*

The original LLR plus both the horizontal and vertical extrinsic LLRs yields the improvement in Table 5.

\[
\begin{array}{cc}
1.5 & -1.5 \\
-1.5 & 1.1 \\
\end{array}
\]

*Table 5. Improved LLRs due to $L_{oh}(\hat{d}) + L_{ov}(\hat{d})$.*

For this example, it is seen that the horizontal parity alone yields the correct hard decisions out of the decoder, but with very low confidence for data bits $d_1$ and $d_4$. After enhancing the decoder LLRs with the vertical extrinsic LLRs, the new LLR solutions have better confidence. Let us pursue one additional horizontal and vertical decoding iteration to see if there are any significant changes in the results. We again use the relationships shown in Eqs. 26 through 29 and proceed with the second horizontal calculations for $L_{oh}(\hat{d})$ using the new $L(\hat{d})$ from the first vertical calculations, shown in Eqs. 34 through 37.

\[
\begin{align*}
L_{oh}(\hat{d}_1) &= (0.1 - 0.1) \oplus 2.5 = 0 = \text{new } L(d_1) \\
L_{oh}(\hat{d}_2) &= (1.5 + 0.1) \oplus 2.5 = -1.6 = \text{new } L(d_2) \\
L_{oh}(\hat{d}_3) &= (0.3 + 1.0) \oplus 2.0 = -1.3 = \text{new } L(d_3) \\
L_{oh}(\hat{d}_4) &= (0.2 - 1.4) \oplus 2.0 = 1.2 = \text{new } L(d_4)
\end{align*}
\]

Next, we proceed with the second vertical calculations for $L_{ov}(\hat{d})$ using the new $L(\hat{d})$ from the second horizontal calculations, shown in Eqs. 38 through 41.

\[
\begin{align*}
L_{ov}(\hat{d}_1) &= (0.2 - 1.3) \oplus 6.0 = 1.1 = \text{new } L(d_1) \\
L_{ov}(\hat{d}_2) &= (0.3 + 1.2) \oplus 1.0 = -1.0 = \text{new } L(d_2) \\
L_{ov}(\hat{d}_3) &= (1.5 + 0.0) \oplus 6.0 = -1.5 = \text{new } L(d_3) \\
L_{ov}(\hat{d}_4) &= (0.1 - 1.6) \oplus 1.0 = 1.0 = \text{new } L(d_4)
\end{align*}
\]

After the second iteration of horizontal and vertical decoding calculations, the soft-output likelihood values are again calculated from Eq. 14, rewritten below:

\[
L(\hat{d}) = L_c(x) + L_{oh}(\hat{d}) + L_{ov}(\hat{d})
\]

The final horizontal and vertical extrinsic LLRs, and the resulting decoder LLRs are displayed in Tables 6–8. For this example, the second full iteration of both codes (yielding a total of 4 iterations) suggests a modest improvement over one full iteration. Now, there seems to be a balancing of the confidence values among each of the four data decisions.

\[
\begin{array}{cc}
1.5 & 0.1 \\
0.2 & 0.3 \\
0 & -1.6 \\
-1.3 & 1.2 \\
1.1 & -1.0 \\
-1.5 & 1.0 \\
\end{array}
\]

*Table 6. Original $L_c(x)$ measurements.*

*Table 7. $L_{oh}(\hat{d})$ after second horizontal decoding.*

*Table 8. $L_{ov}(\hat{d})$ after second vertical decoding.*

The soft output is shown in Table 9.

\[
\begin{array}{cc}
2.6 & -2.5 \\
-2.6 & 2.5 \\
\end{array}
\]

*Table 9. $L(\hat{d}) = L_c(x) + L_{oh}(\hat{d}) + L_{ov}(\hat{d})$.*

**COMPONENT CODES FOR IMPLEMENTING TURBO CODES**

We have described the basic concepts of concatenation, iteration, and soft decisions using a simple product-code model. We next apply these ideas to the implementation of turbo codes. We form a turbo code by the parallel concatenation of component codes (building blocks) [3, 6].

First, a reminder of a simple binary rate-1/2 convolutional encoder with constraint length $K$ and memory $K-1$. The input to the encoder at time $k$ is a bit $d_k$, and the corresponding codeword is the bit pair $(u_k, v_k)$, where

\[
u_k = \sum_{i=0}^{K-1} g_{1i} d_{k-i} \mod 2 \quad g_{1i} = 0,1
\]

\[
v_k = \sum_{i=0}^{K-1} g_{2i} d_{k-i} \mod 2 \quad g_{2i} = 0,1
\]

$G_1 = \{g_{10}, g_{11}\}$ and $G_2 = \{g_{20}, g_{21}\}$ are the code generators and $d_k$ is represented as a binary digit. This encoder has a finite impulse response (FIR), and gives rise to the familiar non-systematic convolutional code (NSC), an example of which is seen in Fig. 5. In this example, the constraint length is $K = 3$, and the two code generators are described by $G_1 = \{111\}$ and $G_2 = \{101\}$. It is well known that, at large $E_b/N_0$, the error performance of an NSC is better than that of a systematic code with the same memory. At small $E_b/N_0$, it is generally the other way around [3]. A class of infinite impulse response (IIR) convolutional codes has been proposed [3] for the building blocks of a turbo code. Such codes are also referred to as recursive systematic convolutional (RSC) codes because previously encoded information bits are continually fed back to the encoder’s input. For high code rates, RSC codes result in better error performance than the best NSC codes at any $E_b/N_0$. A binary rate-1/2 RSC code is obtained from an NSC code by using a feedback loop, and setting one of the two outputs $(u_k$ or $v_k$) equal to $d_k$. Figure 6 illustrates an example of such an RSC code, with $K = 3$, where $a_k$ is recursively calculated as

\[
a_k = d_k + \sum_{i=1}^{K-1} g_{1i} a_{k-i} \mod 2
\]

and where $g_i$ is respectively equal to $g_{1i}$ if $u_k = d_k$, and to $g_{2i}$ if $v_k = d_k$. We assume that the input bit $d_k$ takes on values of 0 or 1 with equal probability. It is stated in [3] that $d_k$ exhibits the same statistical properties as $d_k$. The trellis structure is identical for the RSC code of Fig. 6 and the NSC code of Fig. 5, and these two codes have the same free distance. However, the two output sequences $(u_k)$ and $(v_k)$ do not correspond to the same input sequence $(d_k)$ for RSC and NSC codes. For the same code generators, we can say that RSC codes do not modify the output weight distribution of the output codewords compared to NSC codes. They only change the mapping between input data sequences and output codeword sequences.
CONCATENATION OF RSC CODES

Consider the parallel concatenation of two RSC encoders of the type shown in Figure 5. Good turbo codes have been constructed with their component codes having quite short constraint lengths (K = 3 to 5). An example of such a turbo encoder is shown in Figure 7, where the switch yielding u_k punctures the code, making it rate 1/2. Without the switch, the code would be rate 1/3. Additional concatenations can be continued with multiple component codes. In general, the component encoders need not be identical with regard to constraint length and rate. In designing turbo codes, the goal is to choose the best component codes by maximizing the effective free distance of the code [7]. At large values of E_b/N_0, this is tantamount to maximizing the minimum weight codeword. However, at low values of E_b/N_0 (the region of greatest interest) optimizing the weight distribution of the codewords is more important than maximizing the minimum weight [6].

The turbo encoder in Figure 7 produces codewords from each of two component encoders. The weight distribution for the codewords out of this parallel concatenation depends on how the codewords from one of the component encoders are combined with codewords from the other encoder(s). Intuitively, we would like to avoid pairing low-weight code-

![Figure 5. Nonsystematic convolutional (NSC) code.](image)

![Figure 6. Recursive systematic convolutional (RSC) code.](image)

![Figure 7. Parallel concatenation of two RSC encoders.](image)

words from one encoder with low-weight codewords from the other encoder. Many such pairings can be avoided by proper design of the interleaver. An interleaver that permutes the data in a random fashion provides better performance than the familiar block interleaver [8].

If the encoders are not recursive, a low-weight codeword generated by the input sequence d = (0 0 0 0 0 ... 0 0 1 0 0 ...) with a single binary 1 will always appear again at the input of the second encoder for any choice of interleaver. In other words, the interleaver would not influence the codeword weight distribution if the codes were not recursive. However, if the component codes are recursive, a weight-1 input sequence generates an IIR (infinite weight output). Therefore, for the case of recursive codes, the weight-1 input sequence does not yield the minimum weight codeword out of the encoder. The encoded output weight is kept finite only by trellis termination, a process that forces the coded sequence to terminate in the zero state. In effect, the convolutional code is converted to a block code.

For the Fig. 7 encoder, the minimum weight codeword for each component encoder is generated by the weight-3 input sequence (0 0 0 ... 0 0 1 1 1 1 0 0 ... 0 0) with three consecutive 1s. Another input that produces fairly low-weight codewords is the weight-2 sequence (0 0 0 ... 0 0 1 1 0 0 ... 0 0). However, after the permutations of an interleaver, either of these deleterious input patterns is not likely to appear again at the input to another encoder, so it is unlikely that a minimum weight codeword will be combined with another minimum weight codeword. The important aspect of the building blocks used in turbo codes is that they are recursive (the systematic aspect is merely incidental). It is the RSC code's IIR property that protects against those low-weight encodings which cannot be remedied by an interleaver. One can argue that turbo code performance is determined largely from minimum weight codewords that result from the weight-2 input sequence. The argument is that weight-1 inputs can be ignored since they yield large weights due to the IIR code structure, and for input sequences having weight-3 and larger, a properly designed interleaver makes the number of such input words relatively rare [7–11].

A FEEDBACK DECODER

The Viterbi algorithm (VA) is an optimal decoding method for minimizing the probability of sequence error. Unfortunately, the VA is not able to yield the APP or soft-decision output for each decoded bit. A relevant algorithm for doing this has been proposed by Bahl et al. [12]. The Bahl algorithm was modified by Berrou et al. [3] for use in decoding RSC codes. The APP of a decoded data bit d_k can be derived from the joint probability \( \Lambda_k(m) \) defined by

\[
\Lambda_k(m) = P(d_k = i, S_k = m | R^N) \tag{50}
\]

where \( S_k \) is the state at time \( k \), and \( R^N \) is a received sequence from time \( k = 1 \) through some time \( N \).

Thus, the APP for a decoded data bit \( d_k \), represented as a binary digit, is equal to

\[
P[d_k = i | R^N] = \sum_m \Lambda_k(m), \quad i = 0, 1 \tag{51}
\]

The log-likelihood function is written as the logarithm of the ratio of APPs, as follows:
Fig. 8. Feedback decoder.

\[ L(\hat{d}_k) = \log \frac{\sum_m A_k^1(m)}{\sum_m A_k^0(m)} \]  

(52)

The decoder can make a decision by comparing \( L(\hat{d}_k) \) to a zero threshold.

\[ \hat{d}_k = 1 \text{ if } L(\hat{d}_k) > 0 \]
\[ \hat{d}_k = 0 \text{ if } L(\hat{d}_k) < 0 \]  

(53)

For a systematic code, the LLR \( L(\hat{d}_k) \) associated with each decoded bit \( \hat{d}_k \) can be described as the sum of the LLR of \( d_k \) out of the detector and of other LLRs generated by the decoder (extrinsic information), as was expressed in Eqs. 12 and 13. Consider the detection of a noisy data sequence that stems from the encoder of Fig. 7. The decoder is shown in

Fig. 9. Bit error probability as a function of \( E_b/N_0 \) and multiple iterations.

\[ L(\hat{d}_k) = L_c(x_k) + L_{\text{dec}}(\hat{d}_k) = \log \frac{P(x_k|d_k = 1)}{P(x_k|d_k = 0)} + L_{\text{dec}}(\hat{d}_k) \]  

(56)

where \( L(\hat{d}_k) \) is the soft decision output, and \( L_c(x_k) \) is the LLR channel measurement, stemming from the ratio of likelihood functions \( p(x_k|d_k = 1) \) of the discrete memoryless channel. \( L_{\text{dec}}(\hat{d}_k) = L_{\text{dec}}(\hat{d}_k) \) is a function of the redundant information. It is the extrinsic information supplied by the decoder, and does not depend on the decoder input \( x_k \). Ideally, \( L_c(x_k) \) and \( L_{\text{dec}}(\hat{d}_k) \) are corrupted by uncorrelated noise, and thus \( L_{\text{dec}}(\hat{d}_k) \) may be used as a new observation of \( d_k \) by another decoder for an iterative process. The fundamental principle for feeding back information to another decoder is never feed a decoder with information that stems from itself (because the input and output corruption will be highly correlated).

For the Gaussian channel, we use the natural logarithm in Eq. 56 to describe the channel LLR, \( L_c(x_k) \), as was done in Eqs. 18 through 20. We rewrite the Eq. 20 LLR result below:

\[ L_c(x_k) = -\frac{1}{2} \left( \frac{x_k - 1}{\sigma_0} \right)^2 + \frac{1}{2} \left( \frac{x_k + 1}{\sigma_0} \right)^2 = \frac{2}{\sigma_0^2} x_k \]  

(57)

Both decoders DEC1 and DEC2 use the modified Bahl algorithm [6]. If the inputs \( L_1(\hat{d}_k) \) and \( y_{2k} \) to decoder DEC2 are independent, then the log-likelihood \( L_2(\hat{d}_k) \) at the output of DEC2 can be written as

\[ L_2(\hat{d}_k) = f[L_1(\hat{d}_k)] + L_{22}(\hat{d}_k) \]  

(58)

with
\[ L^*_k(\hat{d}_k) = \frac{2}{\theta_k} x_k + L_{cl}(\hat{d}_k) \]  

where \( j \) indicates a functional relationship. The extrinsic information \( L_{cl}(\hat{d}_k) \) out of DEC2 is a function of the sequence \( \{L_j(d_n)\}_{n \in k} \). Since \( L_j(d_n) \) depends on observation \( R_i \), the extrinsic information \( L_{cl}(\hat{d}_k) \) is correlated with observations \( x_k \) and \( y_k \). Nevertheless, the greater \( |n - k| \) is, the less correlated are \( L_j(d_n) \) and the observations \( x_k \) and \( y_k \). Thus, due to the interleaving between DEC1 and DEC2, the extrinsic information \( L_{cl}(\hat{d}_k) \) and the observations \( x_k, y_k \) are weakly correlated. Therefore, they can be jointly used for the decoding of bit \( d_k \).

\[ z_k = L_{cl}(\hat{d}_k) \]  
acts as a diversity effect in an iterative process. In general, \( L_{cl}(\hat{d}_k) \) will have the same sign as \( d_k \). Therefore, \( L_{cl}(\hat{d}_k) \) may improve the LLR associated with each decoded data bit.

The algorithmic details for computing the LLR, \( L(\hat{d}_k) \), of the APP for each bit has been described by several authors [3–5], and suggestions for decreasing the implementation complexity are still ongoing [13–15]. A reasonable way to think of the process that produces APP values for each bit is to imagine computing an MLSE or a VA in two directions over the block of coded bits. Having metrics associated with states in the forward and backward direction allows us to compute the APP of a bit if there is a transition between two given states. Thus, we can proceed with this bidirectional VA and, in a sliding-window fashion, compute the APP for each code bit in the block. With this view in mind, we can estimate that the complexity of decoding a turbo code is at least two times more complex than decoding one of its component codes using the VA.

### Turbo Code Error Performance Example

Monte Carlo simulations have been presented for a rate-1/2, \( K = 5 \) encoder with generators \( G_1 = \{11111\} \) and \( G_2 = \{10001\} \), and parallel concatenation. The interleaver was a 256 \( \times \) 256 array. The modified Bahi algorithm has been used with a data block length of 65,536 bits [3]. For 18 iterations, the bit error probability \( P_B \) is lower than \( 10^{-6} \) when \( E_b/N_0 = 0.7 \) dB. The error performance improvement as a function of iterations is seen in Fig. 9. Note that, as we approach the Shannon limit of \( -1.6 \) dB, the required bandwidth approaches infinity, and the capacity (code rate) approaches zero. For binary modulation, several authors use \( P_B = 10^{-2} \) and \( E_b/N_0 = 0 \) dB as the Shannon limit reference for a rate-1/2 code. Thus, with parallel concatenation of RSC convolutional codes and feedback decoding, the error performance of a turbo code is at \( 0.7 \) dB from the Shannon limit. Recently, a class of codes that use serial instead of parallel concatenation of the interleaved building blocks have been proposed. It has been suggested that these codes may have superior performance to turbo codes [14].

### Summary

In this article basic statistical measures, such as a posteriori probability and likelihood, are reviewed. We then use these measures for describing the error performance of a soft-input/soft-output decoder. A numerical example helps to illustrate how performance is improved when soft outputs from concatenated decoders are used in an iterative decoding process. We next proceed to apply these concepts to the parallel concatenation of recursive systematic convolutional (RSC) codes, and explain why such codes are the preferred building blocks in turbo codes. A feedback decoder is described in general ways, and its remarkable performance presented.

### References


### Biography

BERNARD SKLAR (LSM) has over 40 years of experience in technical design and management positions at Republic Aviation Corp., Hughes Aircraft, Litton Industries, and The Aerospace Corporation. At Aerospace, he helped develop the MILSTAR satellite system, and was the principal architect for EHF Satellite Data Link Standards. Currently, he is head of advanced systems at Communications Engineering Services, a consulting company he founded in 1984. He has taught engineering courses at several universities, including the University of California, Los Angeles and the University of Southern California, and has presented numerous training programs throughout the world. He has published and presented scores of technical papers. He is the recipient of the 1984 Prize Paper Award from the IEEE Communications Society for his tutorial series on digital communications, and is the author of the book Digital Communications (Prentice-Hall). He is past chair of the Los Angeles Council IEEE Education Committee. His academic credentials include a B.S. degree in math and science from the University of Michigan, an M.S. degree in electrical engineering from the Polytechnic Institute of Brooklyn, New York, and a Ph.D. degree in engineering from the University of California, Los Angeles.
APPENDIX A

Below are presented the algebraic details yielding the results shown in Eq. 22, rewritten as follows:

\[ L(d_1 \oplus d_2) = \log_e \left( \frac{P(d_1 = +1) + P(d_2 = +1)}{1 + \frac{P(d_1 = +1)}{1 - \frac{P(d_2 = +1)}}} \right) \]

(60)

\[ P(d = +1) = \log_e \left( \frac{P(d = +1)}{1 - P(d = +1)} \right) \]

(61)

\[ e^{L(d)} - e^{L(d+)} = P(d = +1) \]

(62)

\[ e^{L(d)} = P(d = +1) \times (1 + e^{L(d)}) \]

(63)

\[ P(d = +1) = \frac{e^{L(d)}}{1 + e^{L(d)}} \]

(64)

\[ P(d = -1) = 1 - P(d = +1) = 1 - \frac{e^{L(d)}}{1 + e^{L(d)}} = \frac{1}{1 + e^{L(d)}} \]

(65)

\[ \frac{1}{e^{L(d)}} = \frac{1}{1 + e^{L(d)}} \]

(66)

\[ L(d_1 \oplus d_2) = \log_e \left( \frac{P(d_1 = +1) \times P(d_2 = +1) + [1 - P(d_1 = +1)] \times [1 - P(d_2 = +1)]}{P(d_1 = +1) \times P(d_2 = +1) + [1 - P(d_1 = +1)] \times [1 - P(d_2 = +1)]} \right) \]

(67)

\[ = \log_e \left[ \frac{e^{L(d_1)} + e^{L(d_2)}}{1 + e^{L(d_1)} + e^{L(d_2)}} \right] \]

(68)

\[ = \log_e \left[ \frac{e^{L(d_1)} + e^{L(d_2)}}{1 + e^{L(d_1)} + e^{L(d_2)}} \right] \]

(69)

\[ \frac{e^{L(d_1)} + e^{L(d_2)}}{1 + e^{L(d_1)} + e^{L(d_2)}} \]

(70)