CHAPTER 12

TURBO CODES

Matthew C. Valenti and Jian Sun

This chapter concerns turbo codes, one of the most powerful types of forward-error-correcting channel codes. Included is not only a discussion of the underlying concepts, but also a description and comparison of the turbo codes used by the Universal Mobile Telecommunications System (UMTS) and cdma2000 third-generation cellular systems.

Channel Coding

Forward-error-correcting (FEC) channel codes are commonly used to improve the energy efficiency of wireless communication systems. On the transmitter side, an FEC encoder adds redundancy to the data in the form of parity information. Then at the receiver, a FEC decoder is able to exploit the redundancy in such a way that a reasonable number of channel errors can be corrected. Because more channel errors can be tolerated with than without an FEC code, coded systems can afford to operate with a lower transmit power, transmit over longer distances, tolerate more interference, use smaller antennas, and transmit at a higher data rate.

A binary FEC encoder takes in k bits at a time and produces an output (or *code word*) of n bits, where $n > k$. While there are 2^n possible sequences of n bits, only a small subset of them, 2^k to be exact, will be *valid* code words. The ratio k/n is called the *code rate* and is denoted by r.

Lower rate codes, characterized by small values of *r,* can generally correct more channel errors than higher rate codes and are thus more *energy* efficient. However, higher rate codes are more *bandwidth* efficient than lower rate codes because the amount of overhead (in the form of parity bits) is lower. Thus the selection of the code rate involves a tradeoff between energy efficiency and bandwidth efficiency.

For every combination of code rate *(r),* code word length *(n),* modulation format, channel type, and received noise power, there is a theoretical lower limit on the amount of energy that must be expended to convey one bit of information. This limit is called the *channel capacity* or *Shannon capacity,* named after Claude Shannon, whose 1948 derivation of channel capacity [1] is considered to have started the applied mathematical field that has come to be known as *information theory.* Since the dawn of information theory, engineers and mathematicians have tried to construct codes that achieve performance close to Shannon capacity. Although each new generation of FEC code would perform incrementally closer to the Shannon capacity than the previous generation, as recently as the early 1990s the gap between theory and practice for binary modulation was still about 3 dB in the most benign channels, those dominated by additive white Gaussian noise (AWGN). In other words, the practical codes found in cell phones, satellite systems, and other applications required about twice as much energy (i.e., 3 dB more) as the theoretical minimum amount predicted by information theory. For fading channels, which are harsher than AWGN, this gap was even larger.

The Dawn of Turbo Codes

A major advancement in coding theory occurred in 1993, when a group of researchers working in France developed (or, in the parlance of coding theorists, “discovered”) *turbo codes* [2]. The initial results showed that turbo codes could achieve energy efficiencies within only a half decibel of the Shannon capacity. This was an extraordinary result that at first was met with skepticism. But once other researchers began to validate the results independently, a massive research effort was soon underway with the goal of explaining and, better yet, enhancing the remarkable performance of turbo codes. Much of this research focused on improving the practicality of turbo codes, which, as will be discussed shortly, have some peculiarities that make implementation less than straightforward.

By the end of the 1990s, the virtues of turbo codes were well known, and they began to be adopted in various systems. Now they are incorporated into standards used by NASA for deep space communications (CCSDS), digital video broadcasting (DVB-T), and both third-generation cellular standards (UMTS and cdma2000).

Parallel Concatenated Encoding with Interleaving

One of the most interesting characteristics of a turbo code is that it is not just a single code. It is, in fact, a combination of two codes[1] that work together to achieve a synergy that would not be possible by merely using one code by itself. In particular, a turbo code is formed from the *parallel concatenation* of two *constituent* codes separated by an *interleaver*. Each constituent code may be any type of FEC code used for conventional data communications. Although the two constituent encoders may be different, in practice they are normally identical. A generic structure for generating turbo codes is shown in Figure 12.1. As can be seen, the turbo code consists of two identical constituent encoders, denoted as *ENC #1* and *ENC #2*. The input data stream and the parity outputs of the two parallel encoders are then serialized into a single turbo code word.

The interleaver is a critical part of the turbo code. It is a simple device that rearranges the order of the data bits in a prescribed, but irregular, manner. Although the same *set* of data bits is present at the output of the interleaver, the *order* of these bits has been changed, much like a

[1]Turbo codes can also be constructed using three or more constituent codes. Such structures are called *multiple* turbo codes. However, they are not used in any standards and are therefore not discussed here.

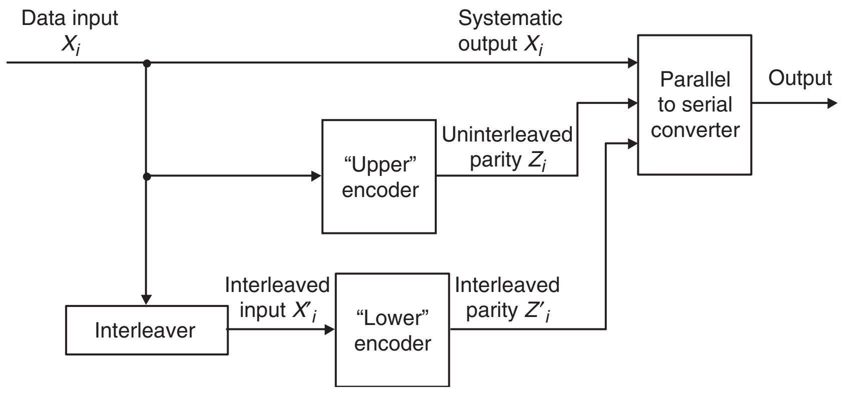

Figure 12.1

A generic turbo encoder.

shuffled deck of cards (although each input word is shuffled in exactly the same way). Without the interleaver, the two constituent encoders would receive the data in the exact same order and thus—assuming identical constituent encoders—their outputs would be the same. This would not make for a very interesting (or powerful) code. However, by using an interleaver, the data $\{X_i\}$ is rearranged so that the second encoder receives it in a different order, denoted $\{X'_i\}$. Thus, the output of the second encoder will almost surely be different than the output of the first encoder—except in the rare case that the data looks exactly the same after it passes through the interleaver. Note that the interleaver used by a turbo code is quite different than the rectangular interleavers that are commonly used in wireless systems to help break up deep fades. While a rectangular channel interleaver tries to space the data out according to a regular pattern, a turbo code interleaver tries to randomize the ordering of the data in an irregular manner.

Why Do Turbo Codes Work So Well?

In order to understand why turbo codes work so well, one must first understand what makes for a good code in general. But first, two terms must be defined. A *linear code* is a code for which the modulo-2 sum of two valid code words (found by XOR-ing each bit position) is also a valid code word. Most codes, turbo codes included, are linear. The *Hamming weight* (also simply known as *weight*) of a code word is the number of ones that it contains. Thus a simple linear code could be composed of two code words, {000} and {111}, where the Hamming weight of the first code word is 0 and the Hamming weight of the second code word is 3. Note that all linear codes must contain the all-zeros code word, since any code word XOR-ed with itself will produce all zeros.

A "good" linear code is one that has mostly high-weight code words (except, of course, the mandatory all-zeros code word). High-weight code words are desirable because it means that they are more distinct, and thus the decoder will have an easier time distinguishing among them. While a few low-weight code words can be tolerated, the relative frequency of their occurrence should be minimized. One way to reduce the number of low-weight code words is by using a turbo encoder. Since the weight of the turbo code word is simply the sum of the weights of the input and the parity outputs of the two constituent code words, we can allow one of these parity outputs to have low weight (as long as the other has high weight). Because the second encoder's input has been scrambled by the interleaver, its parity output is usually quite different from the first encoder's. Thus, although it is possible that *one* of the two encoders will occasionally produce a low-weight output, the probability that *both* encoders simultaneously produce a low-weight output is extremely small. This improvement is called the *interleaver gain* and is one of the main reasons that turbo codes perform so well. Coding theorists say that turbo codes have a "thin distance spectrum," that is the *distance spectrum,* which is a function that describes the number of code words of each possible nonzero weight (from 1 to n), is *thin* in the sense that there are not very many low-weight words present [3].

Convolutional Codes

Although almost any type of encoder could be used for the two constituent encoders shown in Figure 12.1, in practice turbo codes almost

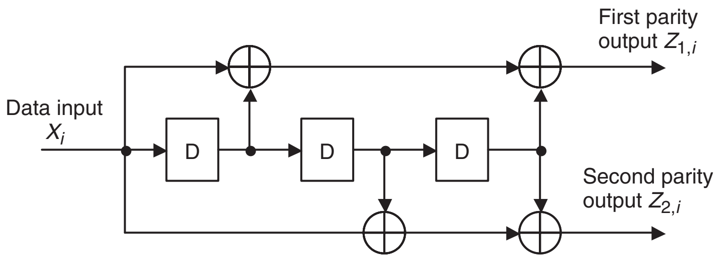

Figure 12.2

A rate 1/2 nonsystematic convolutional (NSC) encoder.

always use *recursive systematic convolutional* (RSC) encoders. RSC codes are very similar in nature to conventional *convolutional codes,* also called *nonsystematic convolutional codes* (NSC), which are commonly used for 2G cellular systems, data modems, satellite communications, and many other applications. An example of an NSC encoder is shown in Figure 12.2. Data $\{X_i\}$ enter from the left and are stored in a linear shift register (D denotes a D flip-flop). Each time a new data bit arrives, the data is shifted to the right into the next flip-flop. Each of the two output bits $\{Z_{1,i}, Z_{2,i}\}$ is computed by XOR-ing a particular subset of the three bits stored in the shift register with the bit at the encoder's input. Because there are two output bits for each input bit, this is a rate $r = 1/2$ encoder. The *constraint length,* denoted by *K,* is the maximum number of input bits (past and current) that either output can depend on. In this case, since each output depends on up to four bits (the three in the shift register plus the one at the input), the constraint length is $K = 4$.

One problem with the encoder shown in Figure 12.2 is that it is *nonsystematic;* that is, the encoder's input bits do not appear at its output. Thus, the code word contains only parity bits and therefore cannot be divided into the separate data and parity fields desired by the turbo encoder. Instead, what we want is a *systematic* encoder, one whose input appears at the output. Unlike its nonsystematic cousin, a systematic code word can be divided into data and parity components. As shown in Figure 12.3, a recursive systematic convolutional code can be created from an NSC by simply feeding one of the two parity outputs back to the input (it is

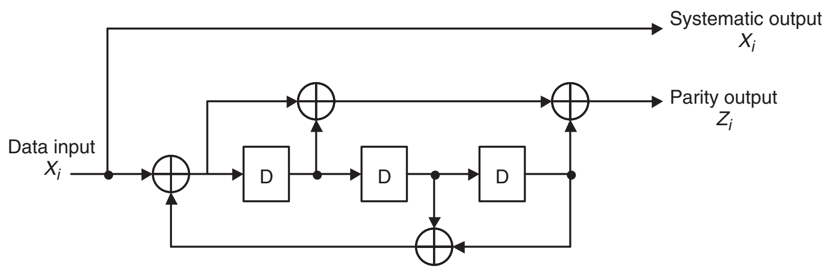

Figure 12.3

The rate 1/2 recursive systematic convolutional (RSC) encoder used by the UMTS turbo code.

this feedback that makes it recursive). Since one of the parity outputs is fed into the input, only the other parity output needs to be transmitted. This allows the data input (also called the systematic output) to be transmitted along with the parity output that was not fed back, while still maintaining a code rate of $r = 1/2$. Also, it turns out that the feedback within the encoder is necessary for the turbo encoder to obtain the maximum interleaver gain [3]. Note that, as indicated by Figure 12.1, only the parity outputs of the two RSC encoders are actually transmitted. The systematic outputs are not needed because they are identical to each other (although ordered differently) and to the turbo code input (which becomes the systematic part of the overall turbo code word).

The UMTS Turbo Code

UMTS, which stands for Universal Mobile Telecommunications System, is one of the two most widely adopted third-generation cellular standards (the other being cdma2000). It is standardized by the Third Generation Partnership Project (3GPP), and the specification is publicly available at the 3GPP website [4]. For FEC, UMTS may use either convolutional or turbo codes (which one depends on the application and available technology). The encoder used by the UMTS turbo code is comprised of a pair of constraint length $K = 4$ RSC encoders, each identical to the one shown in Figure 12.3. As shown in Figure 12.4, the

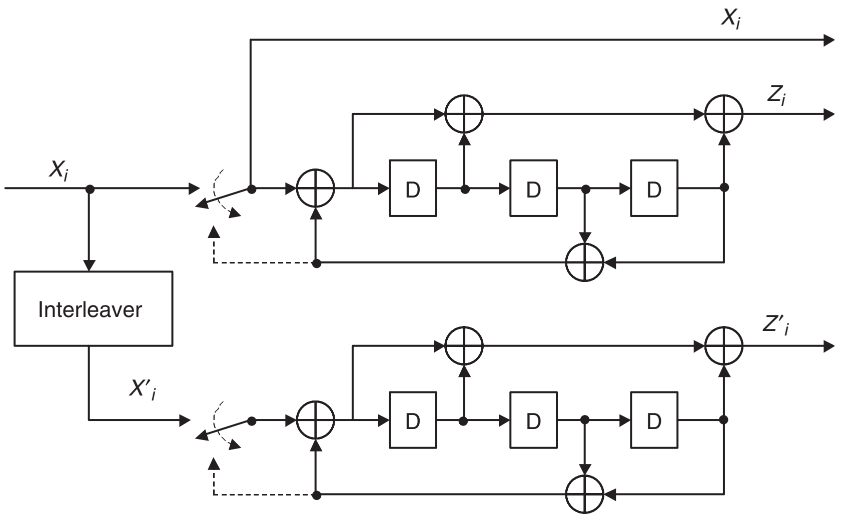

Figure 12.4

The UMTS turbo encoder.

output of the UMTS turbo encoder is a serialized combination of the systematic bits $\{X_i\}$, the parity output of the first encoder $\{Z_i\}$, and the parity output of the second encoder $\{Z'_i\}$. Thus, the overall code rate is approximately $r = 1/3$. In the following, we discuss why it is not exactly 1/3 and explain the action of the switches.

The size of the input data word may range from as few as 40 bits to as many as 5,114 bits. The interleaver, whose size matches that of the input word, scrambles the input according to a prescribed algorithm. The exact interleaving procedure is rather complicated and goes beyond the scope of this chapter (the interested reader is directed to the standard) [4].

Prior to encoding, both constituent encoders start in the *all-zeros state,* that is, each shift register is filled with zeros. After encoding its *k*-bit input, each encoder could be in any of eight distinct states (each state is a unique combination of ones and zeros inside the 3-bit shift register). However, the decoder performs much better if it knows not only the initial state of the encoders, but also their final state. Thus, it is desirable to force the encoders back to a known state after they have encoded the entire input. An obvious choice is to use the all-zeros state, since the encoders need to be in the all-zeros state prior to encoding the next code word.

The NSC encoder shown in Figure 12.2 can be brought back to the all-zeros state by simply inputting a sequence of three zeros, called *tail bits*. RSC codes, however, cannot be brought into the all-zeros state from some other state merely with a sequence of zeros. However, if the feedback bit were to be used as the encoder input, then the XOR of these two bits will be zero, and thus the encoder will return to the all-zeros state after three clock cycles (i.e., the output of the leftmost XOR gate in Figure 12.3 will be zero, since the two inputs are the same). Therefore, the encoder can be brought back to the all-zeros state by inputting the three feedback bits generated immediately after the k-bit code word has been encoded. In the UMTS encoder shown in Figure 12.4 this is achieved by moving the switches from the up position to the down position after the k-bit input has been encoded. Note that because of the interleaver, the state of the two encoders are likely to be different, and thus the tails required for each encoder will also be different. Thus, the transmitted bitstream includes not only the tail bits $\{X_{k+1}, X_{k+2}, X_{k+3}\}$ that correspond to the upper encoder, but also the tail bits $\{X'_{k+1}, X'_{k+2}, X'_{k+3}\}$ that correspond to the lower encoder. In addition to these six tail bits, the corresponding parity bits from the upper encoder $\{Z_{k+1}, Z_{k+2}, Z_{k+3}\}$ and lower encoder $\{Z'_{k+1}, Z'_{k+2}, Z'_{k+3}\}$ are transmitted. Thus, the actual code rate is slightly less than 1/3. In particular, the code rate is $r = k/(3k + 12)$. However, for large k the fractional loss in code rate due to the tail bits is negligible.

The cdma2000 Turbo Code

The other major third-generation cellular standard is cdma2000, which is standardized by the Third Generation Partnership Project 2 (3GPP2).

Like 3GPP, 3GPP2 makes its standards publicly available on its web page [6]. As in UMTS, cdma2000 uses either convolutional or turbo codes for FEC. While the turbo codes used by these two systems are very similar, the differences lie in the interleaving algorithm, the range of allowable input size, and the rate of the constituent RSC encoders. Unlike UMTS, which allows a range of data input sizes (i.e., $40 \leq k \leq 5114$), the size of the data input word (as well as the interleaver) for the cdma2000 turbo code must be one of the following specific values: 378, 570, 762, 1146, 1530, 2398, 3066, 4602, 6138, 9210, 12282, or 20730 bits. Also, the interleaving process used by cdma2000 is different than that used by UMTS. As the details of the interleaver are rather involved, the interested reader is directed to the specification [5].

The constituent RSC encoder used by the cdma2000 turbo code is shown in Figure 12.5. As can be seen, this encoder has three output bits (one systematic plus two parity) for each input bit. Thus, the code rate of this RSC encoder is $r = 1/3$ (neglecting the tail bits). Note that the first parity output $Z_{1,i}$ of the cdma2000 encoder is generated exactly the same way as the parity output Z_i of the UMTS encoder. Thus, these two

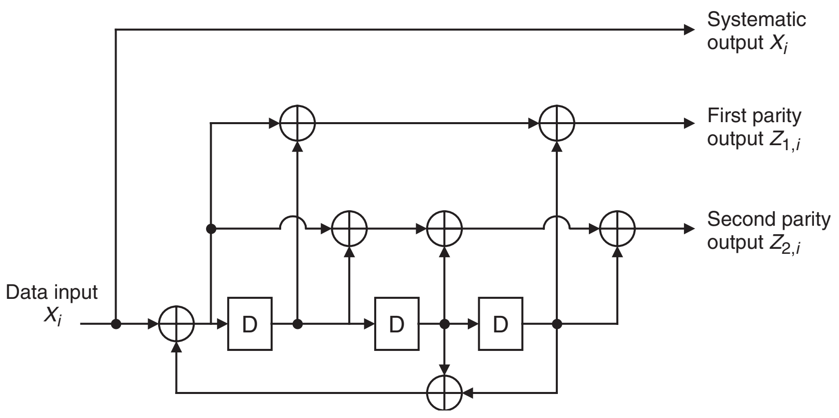

Figure 12.5

The rate 1/3 RSC encoder used by the cdma2000 turbo code.

encoders only differ by the presence of the second parity output $Z_{2,i}$ in the cdma2000 encoder. This encoder is used in the parallel concatenated coding structure shown in Figure 12.1, although now each of the two constituent encoders produces two parity outputs. Thus, the overall code rate for the cdma2000 turbo code is $r = 1/5$. For many applications, such a low rate is undesirable, so cdma2000 also includes a mechanism for transforming the $r = 1/5$ code into a higher rate code. This mechanism, called *puncturing,* involves deleting some of the parity bits prior to transmission. Through puncturing, rates of $r = 1/2$, 1/3, and 1/4 can be achieved. For instance, to achieve rate $r = 1/3$, the encoder deletes the second parity output of each encoder. Since only the first parity outputs are transmitted, the $r = 1/3$ cdma2000 turbo encoder has the same structure as the UMTS turbo encoder (although with different interleaver and range of allowable input word sizes). The puncturing mechanism used to achieve rates $r = 1/2$ and 1/4 are slightly more complicated, but the details can be found in the specification. Finally, cdma2000, like UMTS, uses tail bits to return the encoders back to the all-zeros state.

Turbo Decoding

After encoding, the entire n-bit turbo code word is assembled into a frame, modulated, transmitted over the channel, and decoded. Let U_i represent a modulating code bit (which could be either a systematic or parity bit) and Y_i represent the corresponding received signal (i.e., the output of a correlator or matched filter receiver). Note that while U_i can only be 0 or 1, Y_i can take on any value. In other words, while U_i is a *hard* value, Y_i is a *soft* value. The turbo decoder requires its input to be in the following form:

$$R(U_i) = \ln \frac{P(Y_i \,|\, U_i = 1)}{P(Y_i \,|\, U_i = 0)} \tag{12-1}$$

where $P(Y_i \,|\, U_i = j)$ is the conditional probability of receiving signal Y_i given that the code bit $U_i = j$ was transmitted. Probabilistic expressions such as the one shown in Equation (12-1) are called *log-likelihood ratios* (LLR) and are used throughout the decoding process. Calculation of

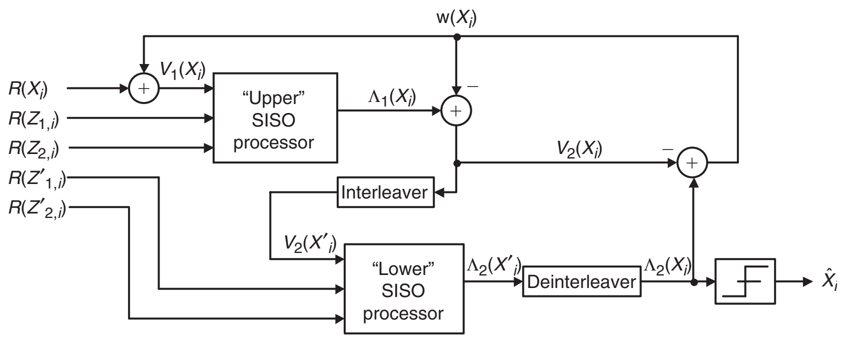

Figure 12.6

An architecture for decoding the UMTS and cdma2000 turbo codes.

Equation (12-1) requires not only the received signal sample Y_i, but also some knowledge of the statistics of the channel. For instance, if binary phase-shift keying (BPSK) modulation is used over an AWGN channel with noise variance σ^2, then the corresponding decoder input in LLR form would be $R(U_i) = 2Y_i/\sigma^2$.

The received values for the systematic and parity bits are put into LLR form and fed into the input of the turbo decoder shown in Figure 12.6. This decoder can be used for both UMTS and cdma2000. In the case of UMTS, the $R(Z_{1,i})$ and $R(Z'_{1,i})$ inputs are the LLR values corresponding to the upper (Z_i) and lower (Z'_i) parity outputs, respectively, of the encoder shown in Figure 12.4. For UMTS, the inputs $R(Z_{2,i})$ and $R(Z'_{2,i})$ are simply set to zero (since there is no corresponding encoder output). For cdma2000, each of the five decoder inputs $R(U_i)$ is the LLR value corresponding to output U_i of the cdma2000 encoder. However, if puncturing was used to increase the code rate to $r > 1/5$, then every bit that was punctured by the encoder must be replaced with a zero at the decoder input.

For each data bit X_i, the turbo decoder must compute the following LLR:

$$\Lambda(X_i) = \ln \frac{P(X_i = 1|\, Y_1 \ldots Y_n)}{P(X_i = 0|\, Y_1 \ldots Y_n)} \qquad (12\text{-}2)$$

This LLR compares the probability that the particular data bit was a one versus the probability that it was a zero, given the *entire* received code word $(Y_1 \ldots Y_n)$. Once this LLR is computed, a hard decision on X_i can be performed by simply comparing the LLR to zero, that is, when $\Lambda(X_i) > 0$ the hard bit estimate is $\hat{X}_i = 1$ and when $\Lambda(X_i) < 0$, $\hat{X}_i = 0$.

The turbo decoder uses the received code word along with knowledge of the code structure to compute $\Lambda(X_i)$. However, because the interleaver greatly complicates the structure of the code, it is not feasible to compute $\Lambda(X_i)$ simply by using a single probabilistic processor. Instead, the turbo decoder breaks the job of achieving a global LLR estimate $\Lambda(X_i)$ into two estimation steps. In the first step, the decoder attempts to compute Equation (12-2) using only the structure of the upper encoder, while during the second step, the decoder computes it using just the structure of the lower encoder. The LLR estimate computed using the structure of the upper encoder is denoted $\Lambda_1(X_i)$ and that computed using the structure of the lower encoder is denoted $\Lambda_2(X_i)$. Each of these two LLR estimates is computed using a *soft-input soft-output* (SISO) processor, the details of which will be discussed shortly.

Because the two SISO processors each produce LLR estimates of same *set* of data bits (although in a different order because of the interleaver), decoder performance can be greatly improved by sharing these LLR estimates between the processors. Thus, the first SISO processor should pass its LLR output to the input of the second SISO processor and vice versa (after appropriate interleaving and deinterleaving). Because of this exchange of information back and forth between processors, the turbo decoding algorithm is iterative. After each iteration, the turbo decoder is better able to estimate the data, although each subsequent iteration improves performance less than the previous one. It is this iterative exchange of information from one processor to the other that gives turbo codes their name. In particular, the feedback operation of a turbo decoder is reminiscent of the feedback between exhaust and intake compressor in a turbo engine.

As with all feedback systems, care must be taken to prevent positive feedback (which would result in an unstable system). Within a turbo decoder, it is important that only the information that is *unique* to a particular SISO processor be passed to the other processor. While each of the two SISO processors receives a unique set of parity inputs (i.e., the middle and lower inputs into each of the SISO processors shown in Figure 12.6), the systematic inputs (i.e., the upper inputs) to the two processors are essentially the same. Thus, the systematic input of each SISO processor must be subtracted from its output prior to feeding the information to the other processor. The difference between an SISO processor's LLR output and its systematic input is called *extrinsic information* and is denoted $w(X_i)$.

The SISO Processor

At the heart of a turbo decoder is the algorithm used to implement the SISO processors. The SISO algorithm uses a *trellis* diagram to represent all possible sequences of encoder states. In particular, the trellis shows the set of states that the RSC encoder may be in at the end of the i^{th} clock cycle, where i ranges from 1 to $k+3$ (assuming three tail bits). Since the RSC encoders used by UMTS and cdma2000 each contain three flip-flops, the number of distinct encoder states at any particular time instance is eight. When the encoder is clocked from time i to time $i+1$, it makes a *state transition* from one state to another. The trellis diagram not only shows the states at each particular time i, but also the set of permissible state transitions leading to states at time $i+1$. An example trellis section for two consecutive time instances, i and $i+1$, is shown in Figure 12.7. The connections between states, called *branches,* show which of the states at time $i+1$ can be reached from states at time i. Each state at time i has two branches leaving it, one corresponding to an input of $X_i = 1$ (solid line) and the other to an input of $X_i = 0$ (dotted line). Every distinct code word is represented by a particular path through the trellis.

The SISO algorithm labels all the branches in the trellis with a *branch metric*. Each branch metric is a function of the processor inputs for the corresponding time instant, with the functional dependency determined by the code structure and which pair of states the particular branch connects. The algorithm is able to obtain LLR estimates of each data bit

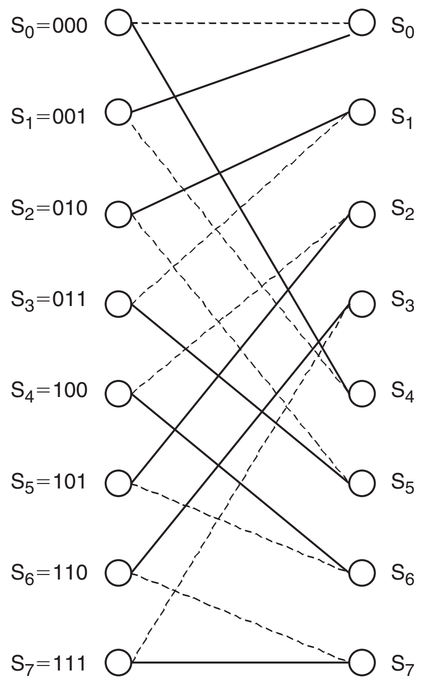

Figure 12.7

A typical section of the code trellis.

$\Lambda(X_i)$ by sweeping through the labeled trellis in a prescribed manner. This sweep can be implemented using one of two algorithms, the *soft output Viterbi algorithm* (SOVA) [6] or the *maximum a posteriori* (MAP) algorithm [7] (also called the BCJR algorithm in deference to its inventors). Both of these algorithms are related to the *Viterbi* algorithm [8], which is commonly used to decode conventional convolutional codes. The key distinction is that while the Viterbi algorithm outputs hard bit decisions, the SOVA and MAP algorithms output soft decisions that can be cast in the form of an LLR.

In general, the SOVA algorithm is less complex than the MAP algorithm, but does not perform as well. However, the complexity of the MAP algorithm can be reduced by implementing it in the log domain. The logarithmic version of the MAP algorithm, called *log-MAP* [9], has reduced complexity because multiplication operations are transformed into additions. However, the logarithmic transformation of addition yields an operation of the form $\ln(e^x + e^y)$, which is nontrivial to compute. Fortunately, this operation is closely approximated by the maximum operator. In [9], an operator called *max-star* is defined as follows:

$$\begin{aligned} \max{}^*(x, y) &= \ln(e^x + e^y) \\ &= \max(x, y) + \ln(1 + e^{-|y-x|}) \\ &= \max(x, y) + f_c(|y - x|) \end{aligned} \tag{12-3}$$

that is, the log-add operation can be implemented by simply taking the maximum of the two arguments and then adding a *correction function* whose argument only depends on the magnitude of the difference between the two arguments of the max-star operator. While computation of the correction function can still be problematic, it can be precomputed and stored in a lookup table. Furthermore, the log-MAP algorithm can be approximated by simply setting the correction term to zero, that is, using $\max^*(x, y) = \max(x, y)$. This variation is called the *max-log-MAP* algorithm. In [9], it is shown that the max-log-MAP algorithm can be implemented using a pair of Viterbi algorithms, one that sweeps through the trellis in the forward direction and a second that sweeps through it in the reverse direction. For this reason, the MAP algorithm and its logarithmic variants are sometimes called the *forward-backward* algorithm. Other flavors of the log-MAP algorithm include the *constant-log-MAP* algorithm, which stores the correction function in a lookup table with just two entries, and the *linear-log-MAP* algorithm, which fits the correction function with a straight line. Details of all four logarithmic versions of the MAP algorithm, and a description of how they can be used to decode the UMTS turbo code, can be found in [10].

Performance of 3G Turbo Codes

This section illustrates the performance of the turbo codes used by the two third-generation cellular standards. Simulations were run to determine the performance of these turbo codes in AWGN with BPSK modulation. In each case, either the log-MAP or linear-log-MAP algorithm was used (the performance of the two algorithms is indistinguishable [10]). For each simulation, a curve showing the bit-error rate (BER) versus the per-bit signal-to-noise ratio (SNR) was computed. The BER is simply the ratio of incorrect data bits divided by the total number of data bits transmitted. The SNR is computed by dividing the energy per received data bit E_b by the single-sided noise spectral density N_o of the channel.

Figure 12.8 shows the performance of the UMTS turbo code with an input frame size of $k = 1530$ bits. This figure shows how performance

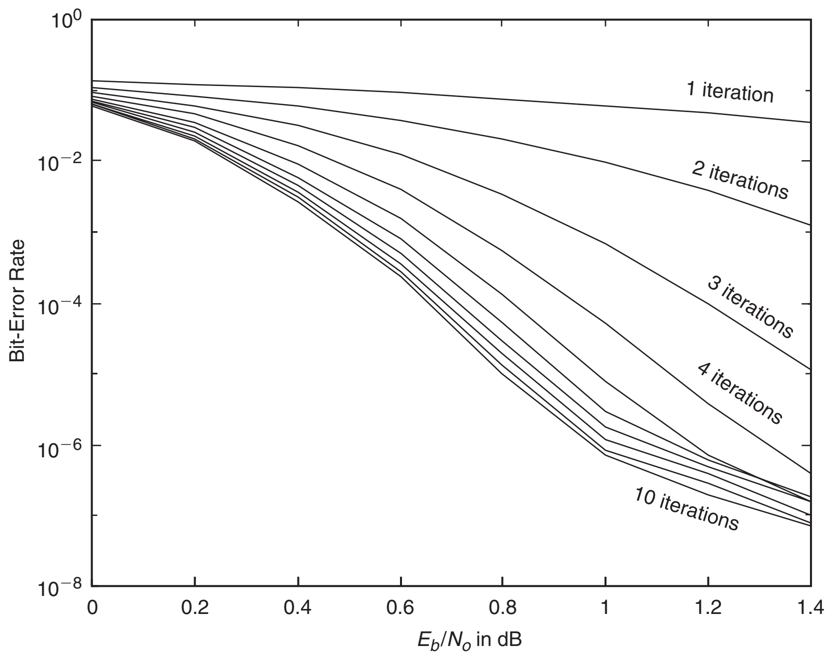

Figure 12.8

Bit-error performance of the UMTS turbo code as the number of decoder iterations varies from one to ten. The encoder input word length is $k = 1530$ bits, modulation is BPSK, and channel is AWGN.

improves as the number of decoder iterations increases. After one iteration, performance is quite poor, and the decoder is unable to achieve a BER lower than 10^{-2} even for an SNR as high as 1.4 dB. However, as the decoder iterates, performance improves until at the tenth iteration it can achieve a BER of 10^{-5} at an SNR of only 0.8 dB. Note how each subsequent iteration improves performance, but that this improvement follows a law of diminishing returns. Thus, although an eleventh (or higher) iteration would provide slightly improved performance, the extra complexity and decoding delay is not justified.

Figure 12.9 shows the performance of the UMTS turbo code as a function of input frame size k. Up to 14 decoder iterations were used, although

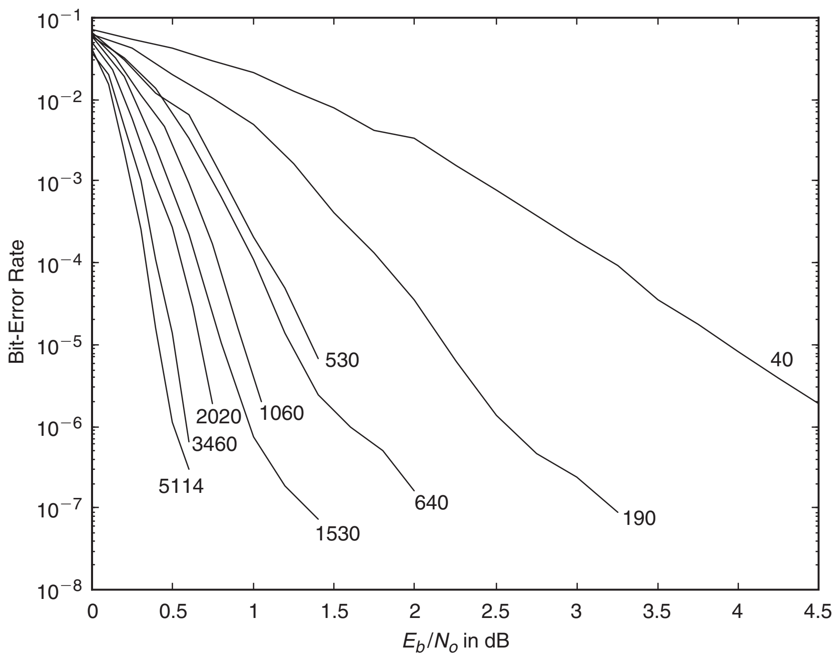

Figure 12.9

Bit-error performance of the UMTS turbo code for various input word lengths. BPSK modulation is used over an AWGN channel.

Table 12.1

Minimum Value of E_b/N_o (in dB) to Achieve a Desired BER Using the UMTS Turbo Code with BPSK Modulation over an AWGN Channel

Frame Size (k)	**Target BER = 10^{-3}**	**Target BER = 10^{-5}**
40	2.41	3.93
190	1.34	2.18
530	0.82	1.36
640	0.75	1.24
1060	0.59	0.94
1530	0.48	0.80
2020	0.38	0.68
3460	0.30	0.51
5114	0.24	0.42

for the smaller frame sizes we found that fewer iterations were required (e.g., for $k = 40$, we only used 8 iterations). As can be seen in this figure, the performance improves with increasing k. This is due to an increase in interleaver gain as the input frame size gets larger. Table 12.1 lists the minimum E_b/N_o required to achieve a BER of 10^{-3} and 10^{-5} for each of the nine frame sizes shown in Figure 12.9.

Figure 12.10 shows the performance of the cdma2000 turbo code for an input frame size of $k = 1530$ bits using BPSK over an AWGN channel. Recall that while the UMTS turbo code has a rate of $r = 1/3$, the cdma2000 turbo code can achieve a wider range of rates. In Figure 12.10, the performance for the four permissible code rates (1/5, 1/4, 1/3, and 1/2) are shown. As with any FEC code, performance improves as the code rate is reduced (since there will be more parity bits to protect the data). Table 12.2 lists the minimum E_b/N_o required to achieve a BER of 10^{-3} and 10^{-5} for each of the four code rates shown in Figure 12.10. Note that the performance of the rate $r = 1/3$ cdma2000 turbo code is virtually identical to that of the UMTS turbo code. This is as expected, since the two encoders are essentially the same when operating at rate 1/3

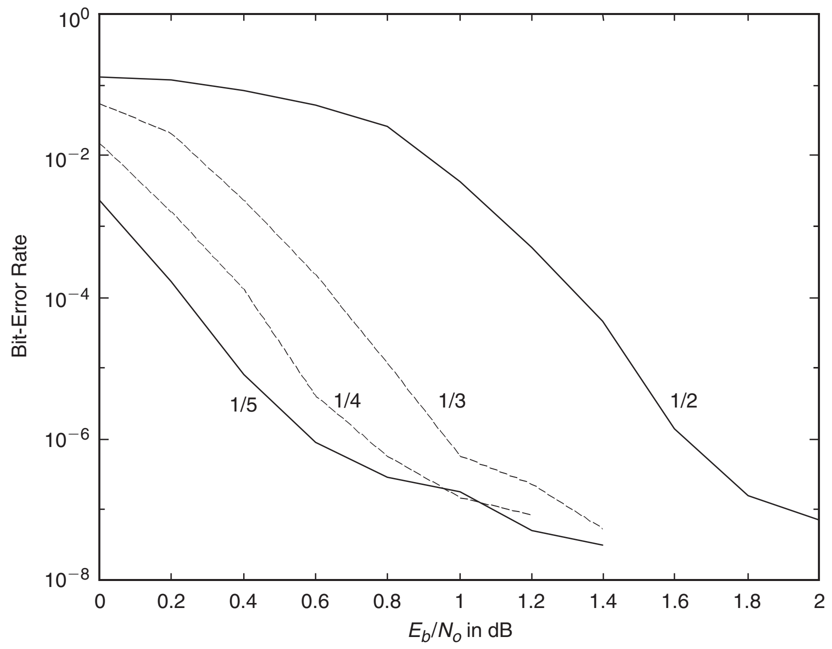

Figure 12.10

Bit-error performance of the cdma2000 turbo code for various code rates. The encoder input word length is $k = 1530$ bits, modulation is BPSK, and channel is AWGN.

Table 12.2

Minimum Value of E_b/N_o (in dB) to Achieve a Desired BER Using the cdma2000 Turbo Code with an Input Frame Size of $k = 1530$ Bits and BPSK Modulation over an AWGN Channel

Code Rate (r)	Target BER = 10^{-3}	Target BER = 10^{-5}
1/2	1.13	1.49
1/3	0.47	0.81
1/4	0.24	0.56
1/5	0.06	0.39

(the only key difference is in the details of interleaving procedure, but this does not significantly affect performance).

Practical Issues

Although turbo codes have the potential to offer unprecedented energy efficiencies, they have some peculiarities that should be taken into consideration. First, while the BER curve falls off sharply with increasing SNR for moderate error rates (e.g., BER > 10^{-5}), the BER curve begins to flatten at higher SNR. This characteristic can be seen in Figure 12.10, for which the BER was simulated down to very small values. The region where the BER curve flattens out is called the *error floor* and hinders the ability of a turbo code to achieve extremely small bit-error rates. The error floor is due to the presence of a few low-weight code words. At low SNR, these code words are insignificant, but as SNR increases, they begin to dominate the performance of the code [3].

The error flooring effect can be combated in several ways. One way is to use a slightly different RSC encoder with a more favorable distance spectrum. However, in order to lower the error floor at high SNR, performance at low SNR will suffer. An interesting approach taken in [11] is to use two *different* RSC encoders. One RSC encoder is optimized to perform well at low SNR, while the other is optimized to reduce the error floor. The resulting *asymmetric turbo code* provides a reasonable combination of performance at both a low and high SNR. Unfortunately, although the error floor has been reduced, it is still present.

Another way to reduce the error floor is to arrange the two constituent encoders in a serial concatenation, rather than in a parallel concatenation [12]. Such serially concatenated convolutional codes (SCCCs) offer excellent performance at high SNR, as the error floor is virtually eliminated (actually, it is pushed way down to BER $\approx 10^{-10}$). However, performance at low SNR is considerably worse than it is for parallel concatenated codes (also called parallel concatenated convolutional codes, or PCCCs). An alternative to choosing between SCCCs and PCCCs is to use *hybrid turbo codes,* which combine features of each type of code [13].

Another problem with turbo codes is that of complexity. As discussed earlier, if the turbo decoder were implemented using the max-log-MAP algorithm, then each half-iteration would require that the Viterbi algorithm be executed twice. If 8 full-iterations are executed, then the Viterbi algorithm will be invoked 32 times. This is in contrast to the decoding of a conventional convolutional code, which only requires the Viterbi algorithm to be executed once. This is why the constraint length of a turbo code's constituent encoder is typically shorter than that of a conventional code. For instance, the conventional convolutional codes used by UMTS and cdma2000 each have a constraint length of $K = 9$.

One easy way to reduce complexity is simply to halt the decoder iterations once the entire frame has been completely corrected. This will prevent over-iteration, which corresponds to wasted hardware clock cycles. However, if the decoder is adaptively halted, then the amount of time required to decode each code word will be highly variable. For a discussion of decoder halting techniques, see Valenti and Sun [10].

Another option for reducing complexity is to implement the entire decoder in analog circuitry, rather than in digital hardware [14]. Analog hardware is particularly appealing because log-add operations such as Equation (12-3) are easily implemented using a Gilbert multiplier. Using analog circuitry, decoding throughputs on the order of hundreds of Mbps are possible for simple turbo codes. However, to date an analog implementation of a decoder suitable for the cdma2000 and UMTS turbo codes has yet to be produced.

Closely related to the issue of complexity are the twin issues of numerical precision and memory management. Because of the forward and backward recursions required by the MAP algorithm and its logarithmic variants, path metrics corresponding to the entire code trellis must be stored in memory. Since a large number of metrics will be stored (e.g., each SISO processor must store $8(5114) = 40{,}912$ metrics when the maximal length UMTS code is used), it is important to represent each metric with as few bits as possible. However, if an insufficient number of bits are used to represent each metric, then the decoder performance will degrade. Wu, Woerner, and Blankenship [15] analyze the numerical precision problem and suggest that each metric be represented by a 12-bit value. Of course, if an analog implementation were to be used, then numerical precision would not be an issue.

A further savings in memory requirements can be achieved by using a *sliding window* algorithm to manage memory [9]. With the sliding window approach, the metrics for only a portion of the code trellis are saved in memory, say from time i to time $i + j$. The entire trellis is then divided into several such windows, with some overlap between windows. Rather than running the MAP algorithm over the entire trellis, it is only run over each window. Since the size of the window is much less than that of the whole trellis, the amount of memory required is greatly reduced. Although this approach may hurt the BER performance, by using sufficient overlap between windows the performance degradation is negligible.

A final practical issue is that of channel estimation and synchronization. In order to transform the received signal into LLR form, some knowledge of the channel statistics is required. For an AWGN channel, the SNR must be known. For a fading channel with random amplitude fluctuations, the per-bit gain of the channel must also be known. If the channel also induces a random phase shift on the signal, then an estimate of the phase would be necessary for coherent detection. As with any digital transmission system, the symbol timing must be estimated using a symbol synchronization algorithm. In addition, it is necessary to synchronize the frame, that is, the decoder needs to know which received bit in a stream of received data corresponds to the first bit of the turbo code word. While such carrier, symbol, and frame synchronization problems are not unique to turbo-coded systems, they are complicated by the fact that turbo codes typically operate at very low SNR. As the performance of synchronization algorithms degrades with reduced SNR, it is particularly challenging to perform these tasks at the low SNRs common for turbo codes. One solution to these synchronization problems is to incorporate the synchronization process into the iterative feedback loop of the turbo decoder itself. In particular, the soft outputs of the SISO processors could be used to help adjust the synchronization after each decoder iteration. For an example of how to use iterative feedback to improve the process of channel estimation (and, consequently, phase synchronization), see the work of Valenti and Woener [16].

References

1. C. E. Shannon, "A Mathematical Theory of Communication," *Bell Systems Technical Journal* 27 (July, Oct. 1948): 379–423, 623–56.
2. C. Berrou, A. Glavieux, and P. Thitimasjshima, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes," *Proceedings of the IEEE International Conference on Communications* (May 1993, Geneva, Switzerland): 1064–70.
3. L. Perez, J. Seghers, and D. J. Costello, "A Distance Spectrum Interpretation of Turbo Codes," *IEEE Transactions on Information Theory* 42 (November 1996): 1698–1708.
4. European Telecommunications Standards Institute (ETSI), "Universal Mobile Telecommunications System (UMTS): Multiplexing and Channel Coding (FDD)," 3 *GPP TS 125.212 Version 3.4.0* (23 September, 2000): 14–20; available online at http://www.3gpp.org.
5. Third Generation Partnership Project 2 (3GPP2), "Physical Layer Standard for cdma2000 Spread Spectrum Systems, Release C," *3GPP2 C.S0002-C, Version 1.0* (May 28, 2002): pp. 115–22; available online at http://www.3gpp2.org.
6. J. Hagenauer and P. Hoeher, "A Viterbi Algorithm with Soft-Decision Outputs and Its Applications," *Proceedings of IEEE GLOBECOM* (November 1989, Dallas, TX): 1680–6.
7. L. R. Bahl, J. Cocke, F. Jelink, and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate,"*IEEE Transactions on Information Theory* 20 (March 1974): 284–7.
8. G. D. Forney, Jr., "The Viterbi Algorithm," *Proceedings of the IEEE* 61 (March 1973): 268–78.
9. A. J. Viterbi, "An Intuitive Justification and Simplified Implementation of the MAP Decoder for Convolutional Codes," *IEEE Journal on Selected Areas of Communication* 1 (February 1998): 260–4.

10. M. C. Valenti and J. Sun, "The UMTS Turbo Code and an Efficient Decoder Implementation Suitable for Software Defined Radios," *International Journal on Wireless Information Networks* 8 (October 2001): 203–16.

11. O. Y. Takeshita, O. M. Collins, P. C. Massey, and D. J. Costello, "A Note on Asymmetric Turbo-Codes," *IEEE Communication Letters* 3 (March 1999): 69–71.

12. S. Benedetto, D. Divsalar, D. Montorsi, and F. Pollara, "Serial Concatenation of Interleaved Codes," *IEEE Transactions on Information Theory* 44 (May 1998): 909–26.

13. D. Divsalar and F. Pollara, "Hybrid Concatenated Codes and Iterative Decoding," *JPL TDA Report PR* 42-130 (April–June 1997): 1–23.

14. H. Loeliger, F. Tarkoy, F. Lustenberger, and M. Helfenstein, "Decoding in Analog VLSI," *IEEE Communications Magazine* (April 1999): 99–101.

15. Y. Wu, B. D. Woerner, and K. Blankenship, "Data Width Requirements in SISO Decoding with Modulo Normalization," *IEEE Transactions on Communications* 49 (November 2001): 1861–8.

16. M. C. Valenti and B. D. Woerner, "Iterative Channel Estimation and Decoding of Pilot Symbol Assisted Turbo Codes over Flat-Fading Channels," *IEEE Journal on Selected Areas of Communications* 19 (September 2001): 1697–1705.