Contents

Chapter 11

Code Concatenation and Advanced Codes

A transmission system's use of the convolutional (or block) and trellis codes of Chapter 10 allows significant improvement in performance with respect to uncoded transmission. The improvement occurs because careful selection of sequences of transmitted symbols can increase codeword separation or minimum distance for a given energy (or volume). Additional improvement through cascade of hard-coding outside of soft-coding further reduces probability of error (perhaps at a small reduction in data rate determined by the rate of the hard external code). This simple improvement is a special case of what is known as **Code Concatenation** and was first seriously studied by Forney in his 1963 MIT dissertation. Often $p = 10^{-6}$ for a BSC model, and very high-rate $\bar{b} \approx 1$ block codes (for instance Reed-Solomon codes) then can reduce probability of error to essentially zero and assume the position of the outer-most hard-decision codes.

There are many successful code-concatenation methods that are more sophisticated than simple cascade of hard and soft coding systems. While coding gains may not add directly, improved probability of bit error can occur from the concatenation, allowing essentially reliable transmission of data rates very close to capacity. Section 11.1 introduces more formally serial and parallel concatenation of codes, while Section 11.2 discusses the related concept of **interleaving**. Interleaving attempts to redistribute a burst of errors or noise that may overwhelm one of the codes, but not all the codes in a concatenation. Section 11.2 describes three popular classes of interleaving methods: **block, convolutional, and random interleaving**. The combination of two or more codes can be considered as one giant code of very long block length with an interleaver, or essentially as "randomly generated" codewords. Surprisingly, the combined code often does **not** have significantly larger free or minimum distance – however, the average number of bit errors that correspond to error events with small distance is very low. This lower number of bit errors is essentially reduced in proportion to the interleaver depth. Thus a maximum-likelihood decoder for the giant combined code could have very low probability of bit error even when the distance-to-noise ratio is poor, thus allowing reliable transmission (at some low but nonzero $\bar{P}_b$) at rates near capacity. However, such a maximum-likelihood decoder would be hopelessly complex.

Iterative Decoding of Section 9.6 is instead a decidedly less complex decoding strategy for concatenated codes. Rather than make a "hard" decision on any of the individual codes, "soft" information about the likelihood of different sequences and/or bits is instead computed for one code and then passed to a decoder for another code, which then also computes its own soft information. The soft information is iteratively passed between all decoders in a recursive cascade of decoders with the hope of converging close to an ability to correctly detect the transmitted message, often lowering $\bar{P}_b$ for data rates very close to but less than capacity. As the AEP in Section 8.3 notes implicitly, any sufficiently long-block-length code with codewords chosen at random has high probability of being a capacity-achieving code so there are many interleaved code concatenations that can approach capacity levels.

Turbo Codes are addressed in Section 11.3 and are examples of interleaved concatenated codes that presume use of iterative decoding and can approach rates of capacity with low $\bar{P}_b$. Section 11.4 directly addresses the use of binary turbo codes in multi-level constellations. Section 11.5 investigates LDPC

codes that essentially can achieve capacity as block length is increased. In LDPC codes, the generator matrix of a block code $G(D) = G(0) = G$, or correspondingly the parity matrix $H(D) = H(0) = H$ have entries essentially chosen at random.

Section **??** discusses shaping methods. At $\bar{b} \geq .5$, the individual symbols in a sequence need to increasingly appear as if the code were created by drawing codewords from a Gaussian distribution. This effect is separate from coding gain achieved by sequence separation or nearest-neighbor reduction and is called **shaping gain** in Chapter 1. A final concatenation of shaping codes is necessary when $\bar{c} \geq .5$ to gain up to an additional 1.53 dB of coding gain that cannot be achieved any other way. Shaping codes are unusual in that most of the complexity is in the transmitter, while the receiver is trivial, and amount to selecting sequences from a large set of extended codewords so that the shaping gain is maximized.

Figure 11.1: Serial Concatenation

11.1 Code Concatenation

Concatenated coding attempts to combine the performance-improvement of two or more codes. Concatenation of codes can follow one of 3 architectures, serial, parallel, or multilevel, each of which are introduced in the 3 following subsections.

11.1.1 Serial Concatenation

Figure 11.1 illustrates **serial concatenation**, which uses two codes called the **inner** and **outer** codes. The inner code may be any of the types of codes studied in earlier chapters. The outer code attempts to provide additional improvement. The interleaver rearranges the order of transmitted symbols, while the corresponding de-interleaver restores the original order. De-interleaving thus can disperse a burst of errors associated with an inner-decoder error event so that these individual errors may be more easily corrected by the outer code's decoder. De-interleaving thus allows the inner channel-code output to appear independent from symbol to symbol.

Serial concatenation with interleaving traditionally uses an inner code that makes hard decisions on its inner-code decoder output. Such hard decisions allow the inner code-and-channel to be modeled by a BSC (or DMC more generally). The outer code then is designed to reduce P_e to negligible levels: The probability of error for the binary outer code that acts on the BSC created by the inner code has

$$\bar{P}_e \approx N_e (4p)^{\lceil d_{free}/2 \rceil} \quad . \tag{11.1}$$

Block or convolutional error-correction outer codes with high rates $\bar{b} \approx 1$ can have $d_{free} \geq 8$. Thus, the inner code's probability of error of perhaps 10^{-6} can be reduced through the use of the outer code to below 10^{-20}. Such low error probabilities essentially mean that the transmission layer is effectively error free, a highly desirable result in many applications. This simple form of serial concatenation allows one of the basic results of Shannon to be achieved, namely arbitrarily reliable transmission (albeit at some small loss in data rate not required in basic capacity theorems of Chapter 8).

Serial concatenation can also use two codes that both are designed for BSC's, meaning that the channel itself is a BSC, as well as the model of the inner-coded-and-interleaved channel. Soft decoding can be used by both decoders exploiting the equality and parity constraint viewpoints of Sections 9.5 and 9.6. Serial concatenation can be extended to cascade more than 2 codes by recursively considering the system in Figure 11.1 to be representative of a new inner code, to which a new outer code and interleaver can be applied.

11.1.2 Parallel Concatenation

Figure 11.2 illustrates parallel concatenation for two codes. The same information sequence is encoded by two different encoders. However, one of the encoders acts on an interleaved version of the input sequence. The two encoder outputs are multiplexed (and interleaved for one of the codes) for channel transport. The naming of an "inner code" and an "outer code" is somewhat vague for parallel concatenation, so the codes are often instead referred to as first and second codes. All the individual-code outputs may be transmitted, or some subset of these outputs may instead be transmitted by regular deletion or "puncturing" of the encoder outputs.

A simple form of parallel concatenation is known as a **product code**. Product codes re-encode the same bits/symbols by adding parity bits with two systematic encoders to the same set of input bits. Figure 11.3 illustrates product coding. One encoder determines horizontal parity bits while a second

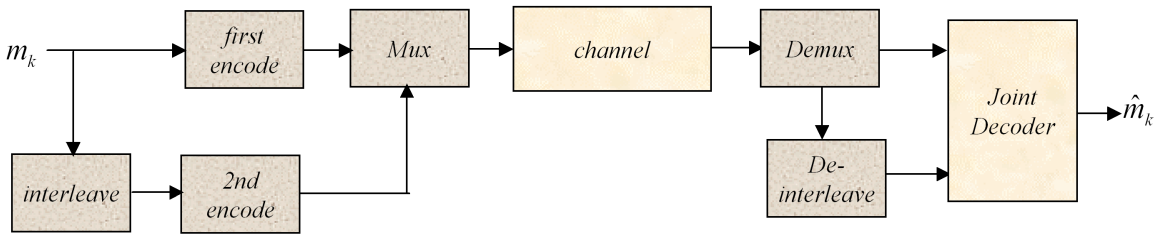

Figure 11.2: Parallel Concatenation.

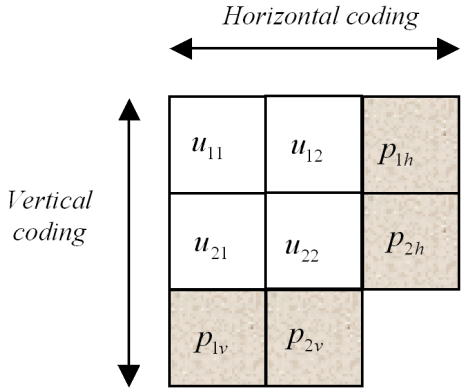

Figure 11.3: Simple product code illustration.

encoder generates vertical parity bits.[1] The ordering of the bits for the two encoders is not the same, thus tacitly illustrating the interleaving. An incorrectly decoded codeword for the horizontal code would leave up to a single bit error in each column, which could typically still be corrected by the vertical code.

In general for parallel concatenation again, de-interleaving again distributes bursts of errors caused by the first decoder's selection of an incorrect codeword, so that the second code can more reliably decode (as long as error bursts don't occur too often). Parallel concatenation can be extended to more than two codes by adding additional interleavers and encoders in the obvious parallel way in Figure 11.2. One could envision a 3-dimensional version of the product code as an example with additional parity bits "coming out of the paper at the reader" in Figure 11.3 (20 of 27 positions in a $3 \times 3 \times 3$ cube).

11.1.3 Multilevel Coding

Multilevel coding uses partition chains to encode different bits of the input stream with different codes. Trivial forms of multilevel coding are the 4-dimensional Wei trellis codes of Chapter 10. Those codes had a first (often rate 2/3) encoder that selected a sequence of 4-dimensional cosets to be transmitted. These first two input bits were protected by the convolutional or trellis part of the code. The remaining input bits were then used to select parallel transitions in a D_4 lattice code for each branch of the trellis. One of those latter bits is protected by the code that corresponds to selecting the upper or lower path in the trivial trellis of a D_4 lattice. The remaining bits could also be coded.

Multilevel coding will not be further pursued in this Chapter. Multilevel coding is typically applicable only to systems with very large b.

[1] The 9^{th} position at the lower right in Figure 11.3 would be present in the serial concatenation of Subsection 11.1.1.

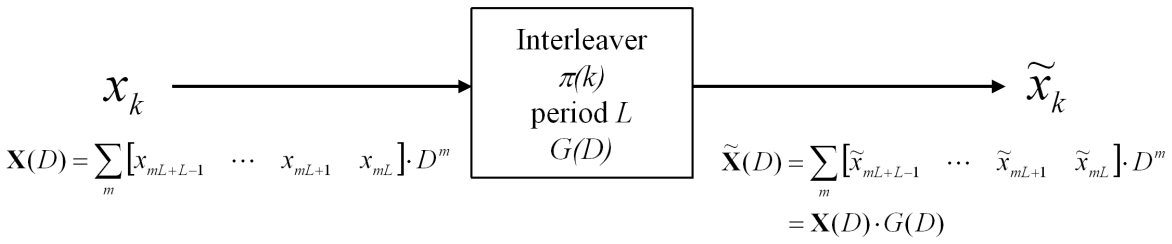

Figure 11.4: Basic Interleaver.

11.2 Interleaving

Interleaving is a periodic and reversible reordering of blocks of L transmitted symbols. Symbols (or bits) are correspondingly reordered by de-interleaving in the receiver. Interleaving is used to disperse error bursts that may occur because of nonstationary channel noise that may be localized to a few dimensions such as an impulse noise burst in time or a loss of a narrow frequency band in OFDM. Error bursts can also occur because of the incorrect decision of an inner/first decoder in Figures 11.1 or 11.2. Errored symbols caused by inner-code detection errors will typically span the entire incorrect codeword, leading to a burst of errored symbols. If these bursts are separated by an interval long with respect to the interleaver period, then they can be distributed more evenly over time (or more generally dimensions) by the de-interleaver in the receiver. The distribution of the errors effectively enables realistic modeling for the inner-code-and-channel as memoryless, i.e., modeled by a BSC, DMC, or other channel for which successive symbol outputs are independent.

Figure 11.4 generically depicts the interleaver as accepting symbols indexed in time by k or in block/packet in m, where L symbols occur within one packet and L is the period of the interleaver. Often L corresponds to a codeword size.

> **Definition 11.2.1 (Depth of an Interleaver)** *The* **depth** J *of an interleaver is defined as the minimum separation in symbol periods at the output of the interleaver between any two symbols that were adjacent at the input of the interleaver.*

The depth of an interleaver has significant implication for a burst of errors entering a de-interleaver at a receiver. If a burst of errors has duration less than the depth, then two symbols affected by the burst cannot be adjacent after de-interleaving.

> **Definition 11.2.2 (Period of an Interleaver)** *The* **period** L *of an interleaver is the shortest time interval for which the re-ordering algorithm used by the interleaver repeats.*

Essentially, the period is established by the detailed description of the interleaver and measures the length of a block of input symbols to which interleaving is applied. The interleaver repeatedly acts with the same algorithm upon successive blocks of L symbols. Often the period of the interleaver is chosen to be equal to the block length of an outer code when block codes are used in serial concatenation.

> **Theorem 11.2.1 (Minimum Distance Magnification in Interleaving (for serial concatenation))** *For inner channels with an outer hard-decision code of block length equal to the period of interleaving, and only one error burst occurs within* $(J-1)\cdot(L-1)$ *symbols, then the outer code's free distance in symbols is multiplied by the depth* J *for inner-channel error bursts.*
>
> **proof:** *Since all adjacent de-interleaver output symbols within a burst on the input to the de-interleaver are separated by at least* $J-1$ *symbols, the number of errors has been effectively reduced by a factor of* J *as long as a subsequent or preceding burst does not introduce errored symbols into the same codewords that have errors from the burst under investigation. The minimum delay of any symbol through the process of interleaving and de-interleaving is* $(J-$

$1) \cdot (L-1)$ *symbols (and would occur when all adjacent symbols are always spaced by exactly* $J-1$ *symbols after interleaving) if every one of* L *symbols is spaced by* $J-1$ *symbols from its previous neighbors. Thus, the burst length must be in general no longer than this minimum delay to prevent different error bursts from placing erorred symbols in the same de-interleaver-output codewords.*

Interleaving with a single code does not improve performance for stationary additive white Gaussian noise, so it is used for channels that exhibit bursts of errors or nonstationary noise. A burst of errors can occur when an "inner" decoder incorrectly decodes and thus interleaving can be of value in systems with two codes as in section 11.3.

Generally, the interleaver follows a relationship from its input $\boldsymbol{x}_k$ to its output $\tilde{\boldsymbol{x}}_k$ of

$$\tilde{\boldsymbol{x}}_k = \boldsymbol{x}_{\pi(k)} \quad , \tag{11.2}$$

where $\pi(k)$ is a function that describes the mapping of interleaver output time indices to interleaver input time indices. Necessarily $\pi(k)$ is one-to-one over the integers modulo its period of L samples. Because of the periodicity,

$$\pi(k) - L = \pi(k-L) \quad . \tag{11.3}$$

The depth can be more precisely defined mathematically using the function π as

$$J = \min_{k=0,\ldots,L-1} | \, \pi^{-1}(k) - \pi^{-1}(k+1) \, | \quad . \tag{11.4}$$

This section augments the traditional coding use of a delay variable D as corresponding to one interleaver period by considering also a symbol delay variable D_{sym} such that

$$D = D_{sym}^{L} \quad . \tag{11.5}$$

As far as the author knows, no other text uses the symbol delay notation, but we find it very useful in simplifying the description of interleaving.

In traditional study of interleaving, a sequence of interleaver-input symbols can be represented by the L-dimensional (row) vector sequence $\boldsymbol{X}(D)$ in Figure 11.4 where each element in the sequence spans one period of the interleaver, with the time index $k = m \cdot L + i \; i = 0, ..., L-1$, yielding to an interleaver block index m,

$$\boldsymbol{X}_m = \left[\boldsymbol{x}_{m \cdot L + (L-1)} \; \boldsymbol{x}_{m \cdot L + (L-2)} \; ... \; \boldsymbol{x}_{m \cdot L} \right] \quad , \tag{11.6}$$

where m thus denotes a time index corresponding to a specific block of L successive interleaver-input symbols, and

$$\boldsymbol{X}(D) = \sum_m \boldsymbol{X}_m D^m \quad . \tag{11.7}$$

Similarly the interleaver output $\tilde{\boldsymbol{X}}(D)$ can be considered an L-symbol-element sequence of interleaver outputs. Then, interleaving can be modeled as a "rate 1" convolutional/block code over the symbol alphabet with generator and input/output relation

$$\tilde{\boldsymbol{X}}(D) = \boldsymbol{X}(D) \cdot G(D) \quad , \tag{11.8}$$

where $G(D)$ is an $L \times L$ nonsingular generator matrix with the following restrictions:

1. one and only 1 entry in each row/column can be nonzero, and
2. nonzero entries are of the form D^l where l is an integer.

The de-interleaver has generator $G^{-1}(D)$, so that $\boldsymbol{X}(D) = \tilde{\boldsymbol{X}}(D) \cdot G^{-1}(D)$. Further, a **causal interleaver** has the property that all nonzero elements have D^l with $l \geq 0$, and elements above the diagonal must have $l \geq 1$ – the de-interleaver for a causal interleaver is necessarily noncausal, and thus must be instead realized with delay because interleaving necessarily introduces delay from interleaver input to de-interleaver output.

The equivalent relationships in terms of symbol delay are expressed similarly

$$\tilde{\boldsymbol{X}}(D_{sym}) = \boldsymbol{X}(D_{sym}) \cdot G(D_{sym}) \quad , \tag{11.9}$$

but the input and output vectors have each entry defined in terms of the symbol-period D-transform as

$$x_i(D_{sym}) = \sum_{k=0}^{\infty} x_{i,k} \cdot D_{sym}^k \quad . \tag{11.10}$$

The entries in the vector $\boldsymbol{X}(D_{sym})$ are **not** simply found by substituting $D = D_{sym}^L$ into the entries in the vector $\boldsymbol{X}(D)$. In fact,

$$\boldsymbol{X}(D_{sym}) = \left[D_{sym}^{L-1} \cdot x_{L-1}(D) \Big|_{D=D_{sym}^L} \; D_{sym}^{L-2} \cdot x_{L-2}(D) \Big|_{D=D_{sym}^L} \; \ldots \; x_0(D) \Big|_{D=D_{sym}^L} \right] \quad . \tag{11.11}$$

A similar relation to (11.11) holds for the output vector of the interleaver. The symbol-spaced generator follows easily by replacing any nonzero power of D, say D^δ in $G(D)$ by substituting $D_{sym}^{L\cdot\delta+(\text{column number}-\text{row number})_{mo}}$ and circularly shifting the entry within the row to the left by (row number $-$ column number) positions.

> **Rule 11.2.1 (Generator Conversion for Interleavers)** Let any non-zero entry in $G(D_{sym})$ be written as D_{sym}^r. A new column index i' is formed by $i' = (r+i)_{\text{mod } L}$ where i is the column index of the original non-zero entry (counting from right to left, starting with 0). Further let $r' = \lfloor \frac{r+i}{L} \rfloor$, then place $D^{r'}$ in the same row in the position of column i'. One can think of this as circularly shifting by $r+i$ positions and increasing the power of D for every factor of L in the exponent of D_{sym}^{r+i}.

(An example use of Rule 11.2.1 will occur shortly.)

It is often easier to avoid the rule and simply directly write $G(D)$ based on a description of the interleaving rule (and to do the same for $G(D_{sym})$. A few examples will help illustrate the concepts.

EXAMPLE 11.2.1 (Simple Block Permutation) A simple period-3 HW example is

$$\pi(k) = \begin{cases} k+1 & \text{if } k = 0 \bmod 3 \\ k-1 & \text{if } k = 1 \bmod 3 \\ k & \text{if } k = 2 \bmod 3 \end{cases} \tag{11.12}$$

or in tabular form:

$\pi(k)$:	-1	1	0	2	4	3	5
k:	-1	0	1	2	3	4	5

which has inverse de-interleaver easily expressed as

$k' = \pi(k)$:	-1	0	1	2	3	4	5
$\pi^{-1}(k') = k$:	-1	1	0	2	4	3	5

The depth is $J = 1$ and thus clearly this interleaver is not very useful.

The corresponding generator is:

$$G(D) = G(D_{sym}) = G(0) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{11.13}$$

with de-interleaving inverse

$$G^{-1}(D) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad . \tag{11.14}$$

Note that the two interleaver descriptions are equivalent if there are only binary entries in the generator matrix. Such interleavers are called "block interleavers" and studied in Section 11.2.1.

EXAMPLE 11.2.2 (Simple Triangular Interleaving) A second period-3 HW-interleaver example is

$$\pi(k) = \begin{cases} k & \text{if } k = 0 \bmod 3 \\ k-3 & \text{if } k = 1 \bmod 3 \\ k-6 & \text{if } k = 2 \bmod 3 \end{cases} \tag{11.15}$$

or in tabular form:

$\pi(k)$:	-7	0	-2	-4	3	1	-1
k:	-1	0	1	2	3	4	5

with inverse

$\pi(k)$:	-1	0	1	2	3	4	5
k:	5	0	4	8	3	7	11

The depth of this interleaver is $J = 4$ symbol periods.

The generator for the second HW example is (recall a delay, D, corresponds to a delay of one period of $L = 3$ time samples)

$$G(D) = \begin{bmatrix} D^2 & 0 & 0 \\ 0 & D^1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ and } G(D_{sym}) = \begin{bmatrix} D_{sym}^6 & 0 & 0 \\ 0 & D_{sym}^3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{11.16}$$

This interleaver is clearly a one-to-one (nonsingular) generator and has inverse

$$G^{-1}(D) = \begin{bmatrix} D^{-2} & 0 & 0 \\ 0 & D^{-1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad . \tag{11.17}$$

This block-level inverse requires a delay of 2 block periods or 6 symbol periods for causal inversion.

EXAMPLE 11.2.3 (A depth-3 Interleaver) An interleaver with period $L = 5$ and depth $J = 3$ has generator

$$G(D_{sym}) = \begin{bmatrix} D_{sym}^8 & 0 & 0 & 0 & 0 \\ 0 & D_{sym}^6 & 0 & 0 & 0 \\ 0 & 0 & D_{sym}^4 & 0 & 0 \\ 0 & 0 & 0 & D_{sym}^2 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{11.18}$$

and could be described as delaying each symbol within a period by its index times $(J-1)$ symbol periods. The inverse is also easily described. Using the traditional notation with D corresponding to a block period, then rule 11.2.1 provides

$$G(D) = \begin{bmatrix} 0 & 0 & D^2 & 0 & 0 \\ D & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & D & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad . \tag{11.19}$$

The second description often complicates easy insight into the implementation and the inverse. The interpretation that each symbol is delayed by its index times $J-1$ symbol periods is not nearly as evident (although true if one examines much harder). This simplified insight is the value of using the symbol-spaced interpretation, and the nice $D_{sym}^{(J-1)i}$ diagonal in (11.18)

11.2.1 Block Interleaving

Block interleaving or **permutation interleaving** is the simplest type of interleaving and has $G(D) = G(D_{sym}) = G(0) = G$. The de-interleaver is trivially $G^{-1} = G^*$ for the block interleaver. The permutation of inputs to outputs is contained within one period in a block interleaver. The first HW example in (11.13) is a block interleaver. Block interleavers are never causal unless $G = I$ (and $G = I$ trivially means no interleaving).

Figure 11.5 illustrates one of the most often encountered block interleavers, which is most often associated with serial code concatenation and a block inner code. Each successive inner-code block of symbols (often a codeword for a block code, but not necessarily the case) is written into a corresponding register/row in the interleave memory. the block interleaver outputs instead successive columns. The number of symbol blocks stored is the depth J of the interleaver. If a symbol group has K symbols and the interleaver depth is J, then $K \cdot J$ symbols must be stored in each of two transmit memories of a block interleaver as shown for $J = 3$ and $K = 4$. The period is $L = K \cdot J$. As $K = 4$ symbols at a time are written into each row of one of the transmit memory buffers, $J = 3$ symbols in each column are read from the other. Symbols occur every T seconds, making the symbol rate $1/T$. The interleaver input clock is thus $1/K = 1/4$ of the symbol clock rate. Thus, one symbol block of $K = 4$ symbols is written into a each row of the write-memory buffer. After $L = K \cdot J = 12$ symbol periods, the entire write buffer will be full. The transmit-memory output clock is $1/J = 1/3$ of that same symbol clock. The read-memory buffer is read $J = 3$ symbols from each column for each period of the interleaver output clock. The interleaver starts/completes writing of the write buffer at exactly the same two points in time as it starts/completes reading the read buffer. Every $L = KJ = 12$ symbol-clock periods, the read and write memories are interchanged.

Ignoring, the 12-symbol-period delay necessary for causal implementation, the generator for this block interleaver (assuming entire rows/columns are read in sequence, i.e., there are no shift-registers)

$$G(D) = G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} . \tag{11.20}$$

The de-interleaver is G' (where prime means transpose) in the receiver and accepts symbols from a decoder and writes them in terms of K symbols per column successively. After all $L = KJ$ symbols have been stored, the symbol blocks are read in horizontal or row order. Again, two memories are used with one being written while the other is read. This type of interleaver is sometimes called a classical block interleaver. For this classical form, the end-to-end delay is (no worse than) $2L$ symbol times and correspondingly there is a total of $4L$ RAM locations necessary (at receiver and transmitter), half ($2L$) of which are in the receiver and the other half ($2L$) in the transmitter.

For the classical block interleaver, a burst of B errored symbols is distributed roughly evenly over J symbol blocks by the de-interleaving process in the receiver. If this is the only burst within the total L receiver symbols and the symbol block length is equal to the length of a codeword for a block code, then the outer code with hard decisions can correct approximately J times more errored symbols. Larger depth J means more memory and more delay, but greater power of the outer code as long as a second burst does not occur within the same KJ symbols. The minimum distance of the code is thus essentially multiplied by J as long as errors are initially in bursts that do not occur very often. Thus interleaving easily improves the performance of concatenation.

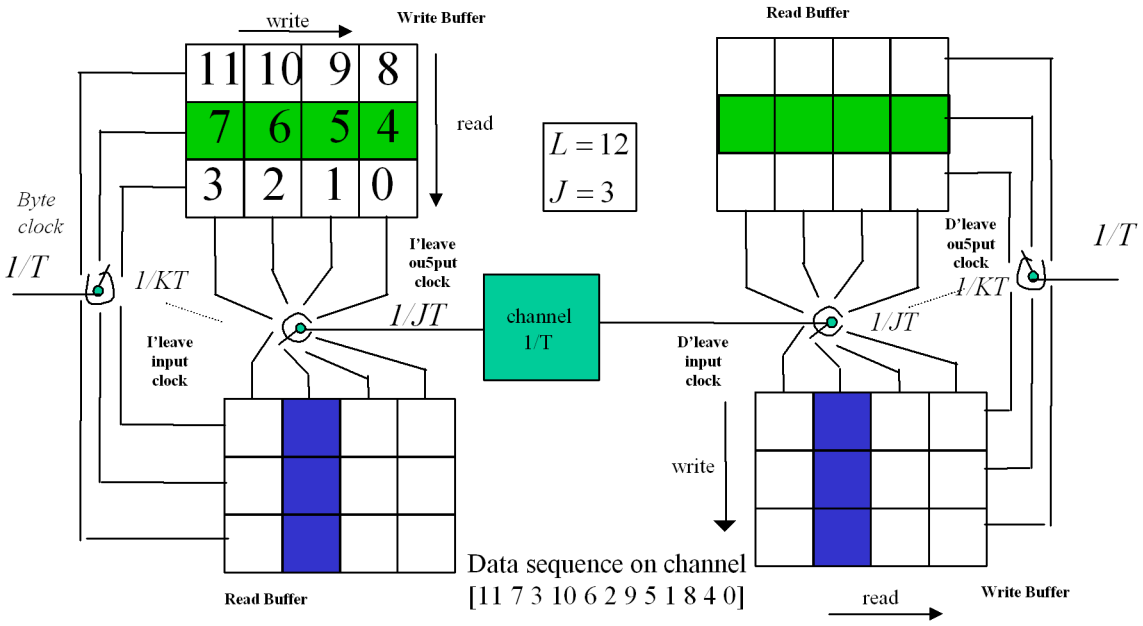

Figure 11.5: Illustration of classical block interleaving and de-interleaving with $L = 12$ and $J = 3$.

11.2.2 Convolutional Interleaving

Convolutional interleavers have $G(D) \neq G(0)$, meaning there is at least one delay element. While more complex in concept, convolutional interleavers can allow a reduction in delay and in memory required for implementation of a given depth J.

Coding theorists have often reserved the name **triangular interleaver** for the special case of a convolutional interleaver where $G(D)$ is a diagonal matrix of increasing or decreasing powers in D proceeding down the diagonal. Example 11.2.2 illustrated such a triangular interleaver. The reason for the names "multiplexed" or "triangular" interleaver follow from Figure 11.6, which illustrates the 3×3 implementation. The delay elements (or the memory) are organized in a triangular structure in both the interleaver and the de-interleaver. Symbols enter the triangular interleaver from the left and with successive symbols cylically allocated to each of the 3 possible paths through the interleaver in periodic succession. The input switch and output switch for the interleaver are synchronized so that when in the upper position, the symbol simply passes immediately to the output, but when in the second (middle) position the symbol is stored for release to the output the next time that the switch is in this same position. Finally, the bottom row symbols undergo two interleaver periods of delay before reaching the interleaver output switch. The deinterleaver operates in analogous fashion, except that the symbol that was not delayed at the transmitter is now delayed by two periods in the receiver, while the middle symbol that was delayed one period in transmitter sees one additional period of delay in receiver, and the symbol that was delayed twice in transmitter is not delayed at all at receiver. Clearly all symbols then undergo two interleaver periods of delay, somehow split between transmitter and receiver. Any symbols in the same block of 3 on the input have at least 3 symbols from other blocks of inputs in between as illustrated in Figure 11.6. The depth is thus $J = 4$.

To generalize what has been known as the triangular interleaver, the depth is restricted to be equal to $J = L + 1$, and $K = L$. Then, $K = L$ symbols within the m^{th} symbol block are numbered from $\boldsymbol{x}_{0,m}$, ..., $\boldsymbol{x}_{i,m}$,..., $\boldsymbol{x}_{L-1,m}$ and the triangular interleaver delays each such symbol by $i \cdot L = i(J-1)$ symbol periods. The generator is

$$G(D) = \begin{bmatrix} D^{L-1} & 0 & 0 & 0 \\ 0 & \ddots & 0 & 0 \\ 0 & 0 & D & 0 \\ 0 & 0 & \ldots & 1 \end{bmatrix} . \tag{11.21}$$

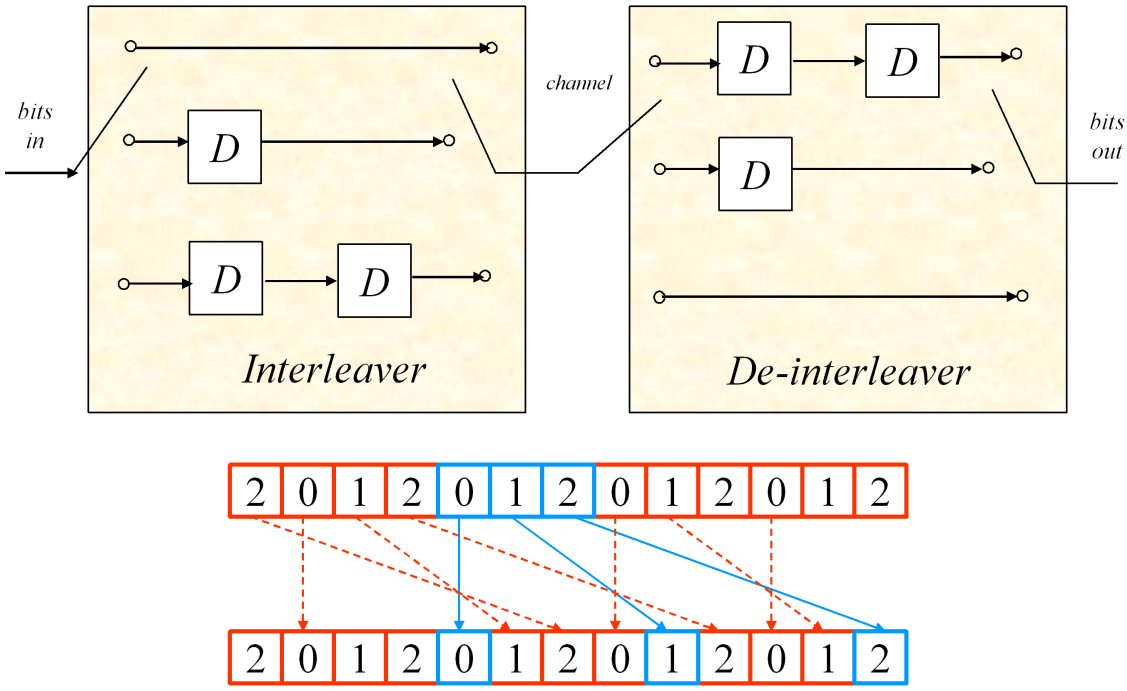

Figure 11.6: Triangular interleaver illustrated for $L = 3$.

The diagonal increasing power of D (where again D corresponds to L symbol periods of delay) in $G(D)$ is that the i^{th} symbol in a symbol block is delayed

$$\Delta_i = i \cdot L = i \cdot (J-1) \text{ symbol periods } i = 0, ..., L-1 \tag{11.22}$$

symbol clocks, assuming symbols arrive sequentially. The single block-of-symbols delay D in Figure 11.6 would be 3 symbol delays, so $D = D_{sym}^3$ and is realized as one storage element. The depth is clearly $J = L + 1$ because the increment in delay between successive symbols is D_{sym}^{J-1}.

The straightforward deinterleaver inverse $G^{-1}(D)$ has negative powers of D in it, so must be realized with delay of $L(L-1) = (J-1)(L-1) = L^2 - L$ symbol periods to be causal. This is equivalent to multiplying $G^{-1}(D)$ by D^{L-1} to obtain

$$G_{causal}^{-1}(D) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \ddots & 0 & 0 \\ 0 & 0 & D^{L-2} & 0 \\ 0 & 0 & ... & D^{L-1} \end{bmatrix} . \tag{11.23}$$

The total delay is $L(L-1) = (J-1)(L-1) = L^2 - L$ symbol periods, which is the theoretical minimum possible. Clearly, the triangular interleaver requires only at most 1/4 the memory and exactly 1/2 the delay of the classical block interleaver, but carries the restriction (so far) of $J = L + 1$ and $K = L$.

In triangular interleaving, the period L can be set equal to the codeword length. The period is no longer the product of K and J as in the block interleaver. Because of the shorter period, synchronization to boundaries of $K \cdot J$ blocks of symbols is no longer necessary (although clearly the de-interleaver in the receiver still needs to know the $L = J - 1$ symbol boundaries of the interleaver). For this reason, the triangular interleaver is often said to not require synchronization – this is somewhat of a misnomer, in that it requires less synchronization than the block interleaver.

The triangular interleaver has an overly restrictive depth of $J = L+1$ only, but has an attractive and simple triangular implementation. **Generalized triangular** interleaving releaves this depth constraint

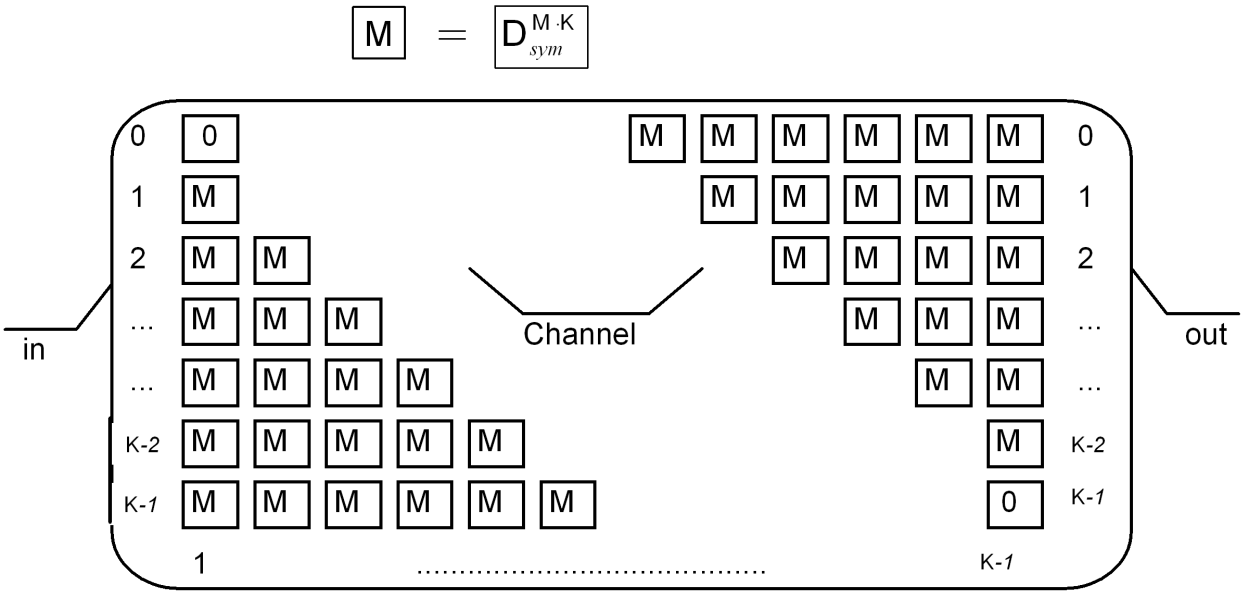

Figure 11.7: The Generalized Triangular Interleaver.

somewhat by essentially replacing each delay element in the triangular implementation by a FIFO (first-in-first-out) queue of M symbol periods. The period remains L symbol periods, but the depth increases to $J = M \cdot L + 1$ symbol periods.

The generalized triangular interleaver appears in Figure 11.7 where the box with an M in it refers to a FIFO containing M symbols and a delay of $M \cdot L$ symbol periods (or M interleaver periods of L symbol periods each). The generalized triangular interleaver is designed for use with block codes of length $N = q \cdot L$ where q is a positive integer. Thus a codeword is divided into q interleaver-period-size groups of size L symbols that are processed by the generalized triangular interleaver. If $q = 1$, one codeword is processed per period, but otherwise $(1/q)^{th}$ of a codeword in general. Often, a symbol is a byte (or "octet" of 8 bits as stated in International Telecommunications Union (ITU) standards).

Thus, the delays for this particular interleaver can be witten as $D_{sym}^{M \cdot L}$. The i^{th} symbol entering the interleaver in Figure 11.7 is delayed by $i \cdot L \cdot M$ block periods. The total delay of any byte is then $\Delta = L \cdot (L-1) \cdot M = (J-1) \cdot (L-1)M$ symbol periods. As with all triangular interleavers, the delay element has power $J - 1$ so $J = M \cdot K + 1$ is again the depth.

The generalized triangular interleaver is typically used with a block code that can correct t symbols in error (typically Reed Solomon block code where a symbol is a byte, then t is the number of parity bytes if erasures are used and 1/2 the number of parity bytes if no erasures are used). The block-code codewords may thus be subdivided into q blocks of symbols so that $N = q \cdot K$. Then Table 11.1 lists the various parameter relations for the generalized triangular interleaver.

> **EXAMPLE 11.2.4 (VDSL Generalized Triangular Interleaver)** The ITU G.993.1 (G.993.2) VDSL (VDSL2) standards use DMT with a Generalized Triangular Interleaver. Table 11.2 works some specific data rates, interleave depths, and consequent delays and memory sizes for the VDSL generalized triangular interleaver. More delay can create application problems, but also improves the correcting capability of the code by the depth parameter.

11.2.3 Enlarging the interpretation of triangular interleavers

Equation (11.22) suggests a more general form of triangular interleaving that follows when $J \leq L + 1$, which is common in use, as shown in Figure 11.8 for example for $J = 2$ and $L = 5$. The delay of each symbol after it enters the interleaver is again $\Delta_i = i \cdot (J-1)\ \forall\ i = 0, ..., L-1$ **symbol** periods. In

Parameter	Value
Interleaver block length (K)	$K=L$ symbols (equal to or divisor of N)
Interleaving Depth (J)	$J=M \times K+1$
(De)interleaver memory size	$M \times K \times (K\text{-}1)/2$ symbols
Correction capability (block code that corrects t symbol errors) With $q=N/K$)	$\lfloor t/q \rfloor \times (M \times K+1)$ symbols $\lfloor t/q \rfloor \times (J)$
End to end delay	$M \times K \times (K\text{-}1)$ symbols

Table 11.1: Table of parameters for Generalized Triangular Interleaver.

Rate (kbps)	Interleaver parameters	Interleaver depth (J)	(De)interleaver memory size	Erasure correction	End-to-end delay
50x1024	$K = 72$ $M = 13$	937 blocks of 72 bytes	33228 bytes	3748 bytes 520 μs	9.23 ms
24x1024	$K = 36$ $M = 24$	865 blocks of 36 bytes	15120 bytes	1730 bytes 500 μs	8.75 ms
12x1024	$K = 36$ $M = 12$	433 blocks of 36 bytes	7560 bytes	866 bytes 501 μs	8.75 ms
6x1024	$K = 18$ $M = 24$	433 blocks of 18 bytes	3672 bytes	433 bytes 501 μs	8.5 ms
4x1024	$K = 18$ $M = 16$	289 blocks of 18 bytes	2448 bytes	289 bytes 501 μs	8.5 ms
2x1024	$K = 18$ $M = 8$	145 blocks of 18 bytes	1224 bytes	145 bytes 503 μs	8.5 ms

Table 11.2: Some calculated delays and data rates for a symbol equal to a byte in the VDSL Interleaver.

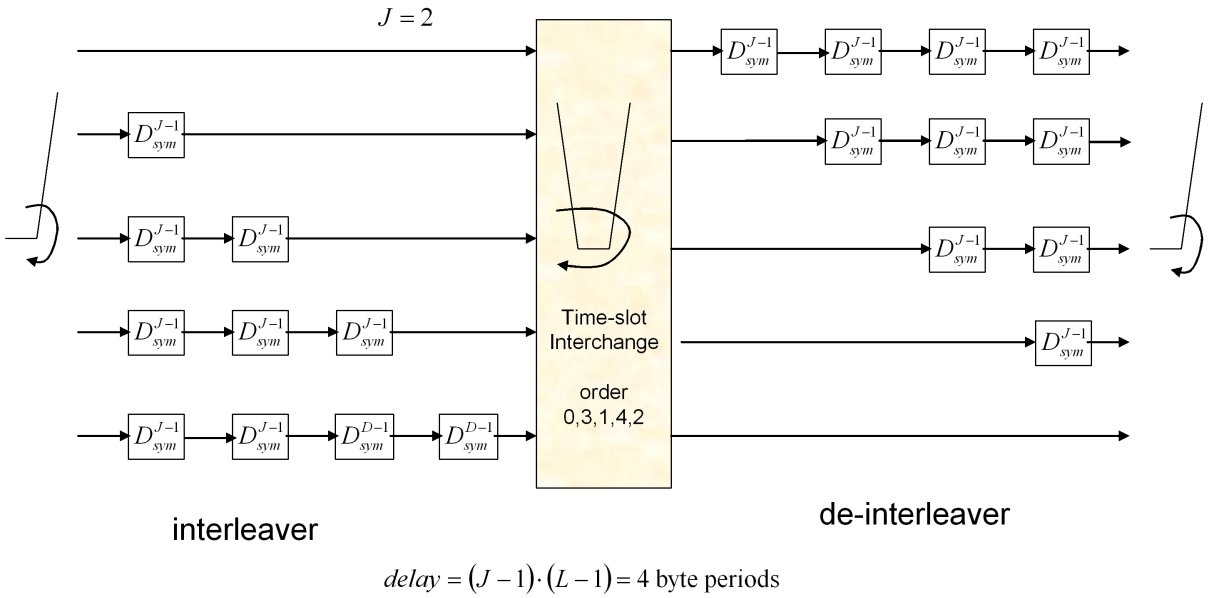

Figure 11.8: Triangular interleaver illustrated for $K = 5$ and $J = 2$.

Figure 11.8, this delay simply increments with each symbol because $J - 1 = 1$. The triangular structure is maintained, but the interleaver-output clocking is irregular as also shown in Figure 11.8 and also in Figure 11.9 for two different depths and a period of 5 symbols. Many authors call the interleavers with $J < L + 1$ a "convolutional interleaver," using the more general term because the triangular implementation escaped notice. The triangular implementation follows by noting the generator in form $G(D_{sym})$ remains diagonal with decreasing (increasing) powers of D_{sym}, while the generator $G(D)$ has a more complicated non-diagonal form. Thus, the first symbol ($i = 0$) in a symbol block is not delayed at all, while the second symbol is delayed by $J - 1$ **symbol** periods, and the last symbol is delayed by $(L - 1)(J - 1)$ symbol periods. The generator matrix for the $J = 3$ example in Figure 11.9 appeared in the third example earlier. It had a complicated $G(D)$, but a simple $G(D_{sym})$. For the example with $J = 2$ of Figure 11.8, the generator is

$$G(D) = \begin{bmatrix} 0 & D & 0 & 0 & 0 \\ 0 & 0 & 0 & D & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{11.24}$$

or

$$G(D_{sym}) = \begin{bmatrix} D_{sym}^4 & 0 & 0 & 0 & 0 \\ 0 & D_{sym}^3 & 0 & 0 & 0 \\ 0 & 0 & D_{sym}^2 & 0 & 0 \\ 0 & 0 & 0 & D_{sym} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{11.25}$$

The diagonal form again is simpler and corresponds directly to the implementation. The interior order is different from the case when $J = L + 1$ and is shown as a "time-slot interchange" in Figures 11.8 and 11.9. The order is easily derived: The input switch position to the interleaver (and output of the de-interleaver) cycles through the period in the normal manner with index $k = 0, ..., L - 1$. The total delay of the k^{th} symbol with respect to the beginning of the period on any line i is $i + i(J - 1) = iJ$ symbol periods. After this delay of iJ symbol periods, the symbol must leave the interleaver and thus the interleaver output switch position (and also the de-interleaver input position) must be then be such

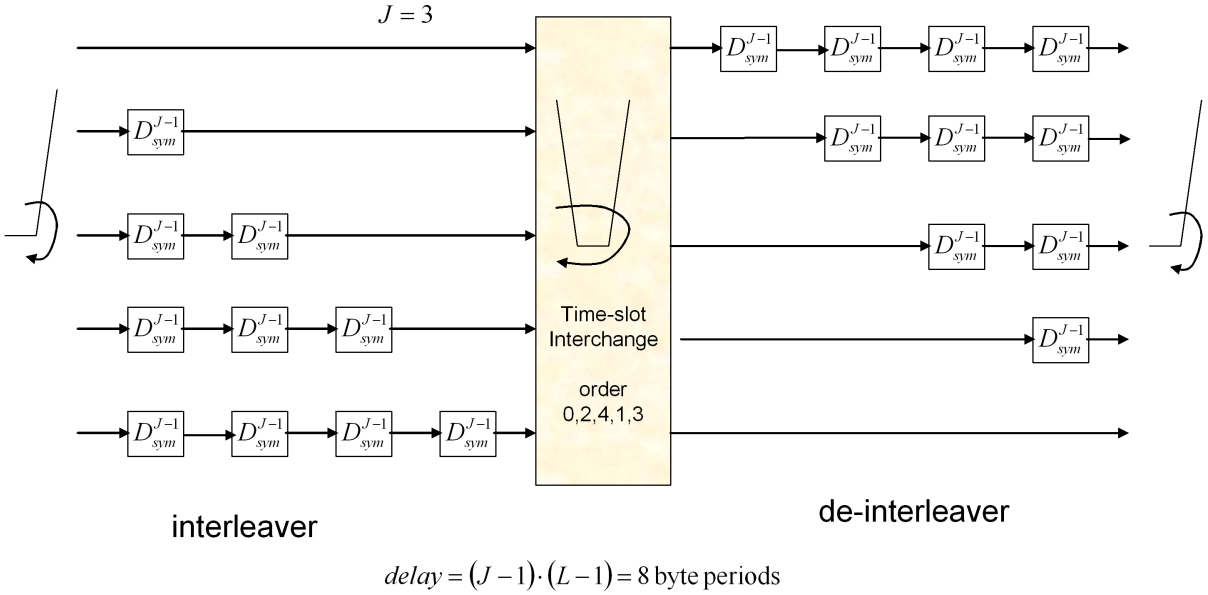

Figure 11.9: Triangular interleaver illustrated for $K = 5$ and $J = 3$.

at time k that

$$(iJ)_L = k \quad , \tag{11.26}$$

since k is also measured from the beginning of an interleaver period. That is, at time k, the output switch position is a function of k, $i(k)$, such that this equation is satisfied for some index i. When the equation is solved for time $k = 1$, let us call that particular time-one solution $i(1) = \Delta$, and $(\Delta J)_L = 1$. For all other times k, then the output position is

$$i = (k \cdot \Delta)_L \quad , \tag{11.27}$$

which is easily proved by substituting (11.26) into (11.27) or $((J \cdot i)_L \cdot \Delta)_L = ((J \cdot \Delta)_L \cdot i)_L = (1 \cdot i)_L = i$. The switch orders in Figure 11.8 both satisfy this equation for the particular depth. Any depth $J \leq L+1$ is possible unless J and L have common factors. If J and L have common factors, then (11.26) does not have a unique solution for each value of i, and the interleaver is no longer a 1-to-1 transformation. The delay of this triangular interleaver and de-interleaver (with time-slot interchange) is then always $(J-1)(L-1)$ symbol periods. The astute reader may note, however, that the memory requirement in the straightforward implementation shown is excessive. The memory can be easily reduced to the theoretical minimum of $(J-1)(L-1)/2$ in each of the interleaver and de-interleaver by a memory-reuse algorithm described momentarily.

The generalized triangular interleaver will follow exactly the same procedure for any depth $J \leq M \cdot L + 1$. Since M can be any positive integer, then any interleave depth is thus possible (as long as L and J are co-prime. The time-slot algorithm still follows (11.27). Again, memory in the straightforward conceptual implementation is not minimum, but delay is again at the minimum of $(J-1)(L-1)$ symbol periods.

Memory reduction to the theoretical minimum

A couple of examples in Tables 11.3 and 11.4 illustrate, for the situations of Figure 11.8 and Figure 11.9, the situation in the triangular interleaver now generalized where RAM cells can be reused. For the situation of Figure 11.8 with $J = 2$, the theoretical minimum number of memory cells for interleaving is 2, while the triangular illustration in Figure 11.8 uses 10 memory cells. Table 11.3 illustrates the storage of symbols (which are bytes in this example). Time is indexed over 3 successive periods of interleaving,

Table 1 for J=2 and L=5												
Line/time	0	1	2	3	4	0’	1’	2’	3’	4’	0’’	1’’
0	-	-	-	-	-	-	-	-	-	-	-	-
1	-	B1	-	-	-	-	B1’	-	-	-	-	B1’’
2	-	-	B2	B2	-	-	-	B2’	B2’	-	-	-
3	-	-	-	B3	B3	B3	-	-	B3’	B3’	B3’	-
4	-	-	-	-	B4	B4	B4	B4	-	B4’	B4’	B4’
CELL1	-	B1	B2	B2	B4	B4	B4	B4	B3’	B3’	B3’	B3’
CELL2	-	-	-	B3	B3	B3	B1’	B2’	B2’	B4’	B4’	B4’

Table 11.3: Minimum-memory scheduling for the triangular interleaver of Figure 11.8 with $L = 5$ and $J = 2$.

with no prime for first period, single prime for second period, and double prime for the two byte intervals shown in the third period. A byte is indexed by the period in which it occurred as B0, B1, B2, B3, or B4 with primes also used. Line 0's bytes (B0) is always immediately passed at time 0 and therefore never uses any memory, and thus does not appear in the table. Hyphens indicate "idle" memory. After byte time 3 of the first interleaver period, the interleaver is in steady state and there are never more than 2 bytes stored at any time (presuming the interleaver reads each memory location before writing any memory location on each byte time slot). Thus in fact two memory cells could be used for this triangular/convolutional interleaver. One half of the bytes in time 1 of the interleave period (called B1 with various primes) are in CELL1, while the other of those bytes are in CELL2. This is true for all bytes, and in general, $1/J$ of the bytes in any symbol position within a period are in any particular cell. Once steady state is reached, all cells are always full. The de-interleaver also only needs 2 CELLS of memory and would be described by letting Bi=B(L-1-i) everywhere (so B4 passes immediately and then bytes B3, B2, B1, and B0 undergo linearly increasing delay).

Table 11.4 shows a similar situation for $J = 3$. After time 1 of the third period, the interleaver is in steady state and uses all the minimum of $4 = \frac{(J-1)(L-1)}{2}$ memory cells. Each memory cell progressively stores the symbols from line 1, then line 4, then line 3, and line 2 before rotating back to line 1 again. The process is regular and repeats on the different memory CELLs offset in time by one period with respect to one another.

An easy way to determine a schedule for the use of the minimum number of memory cells is to realize that the same cell that is read on any byte period of any period must also be written with the next available byte of input with minimum RAM. At design time for a particular specified depth and period, a set of

$$\text{minimum number of cells} = \frac{(J-1)(L-1)}{2} \tag{11.28}$$

"fake" RAM cells can be created in computer software, each with a time that is set to "alarm" exactly $k(J-1)$ symbol periods later where k is the interleaver-input byte-clock index. At each subsequent time period in steady state, one and only one cell's timer will alarm, and that cell should be read and then written and the timer reset to the value of $k(J-1)$. Schedules of "which byte when" will then occur for each storage cell that can be stored and used in later operation. This schedule will repeat over an interval for each cell no longer than

$$S = \text{cell schedule length} \leq \sum_{i=1}^{L-1} i \cdot (J-1) = \frac{1}{2}(J-1) \cdot L \cdot (L-1) = \frac{\Delta}{2} \cdot L \tag{11.29}$$

Table 2 for J=3 and L=5															
L/t	0	1	2	3	4	0’	1’	2’	3’	4’	0’’	1’’	2’’	3’’	4’’
0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
1	-	B1	B1	-	-	-	B1’	B1’	-	-	-	B1’’	B1’’	-	-
2	-	-	B2	B2	B2	B2	-	B2’	B2’	B2’	B2’	-	B2’’	B2’’	B2’’
3	-	-	-	B3	B3	B3	B3	B3	B3 B3’	B3’	B3’	B3’	B3’	B3’’ B3’	B3’’
4	-	-	-	-	B4	B4	B4	B4	B4	B4 B4’	B4 B4’	B4 B4’	B4’	B4’	B4’’ B4’
CELL1	-	B1	B1	B3	B3	B3	B3	B3	B3	B4’	B4’	B4’	B4’	B4’	B4’
CELL2	-	-	B2	B2	B2	B2	B1’	B1’	B3’	B3’	B3’	B3’	B3’	B3’	B4’’
CELL3	-	-	-	-	B4	B4	B4	B4	B4	B4	B4	B4	B2’’	B2’’	B2’’
CELL4								B2’	B2’	B2’	B2’	B1’’	B1’’	B3’’	B3’’

Table 11.4: Minimum-memory scheduling for the triangular interleaver of Figure 11.8 with $L = 5$ and $J = 3$.

symbol periods for each cell. Equality occurs in (11.29) when

$$\frac{m \cdot L}{J-1} \notin Z \text{ for any } m < \frac{\Delta}{2} - 1 \quad . \tag{11.30}$$

When equality occurs then any integer number of periods less than S cannot be divided by the interleaver-output spacing between formerly adjacent interleaver input symbols $J - 1$. In such a case, all cells go through the same length S schedule, just delayed with respect to one another. When the condition produces an integer, then different cells can have different schedulesThe number of schedules for different mutually exclusive groups of cells with the same schedules within each group, but different from group to group, is the number of values of m for which (11.30) is satisified that are not integer multiples of previous values of m that solve the condition in (11.30). but if there are s such schedules of distinct lenghts S_i, then

$$\sum_{i=1}^{s} S_i = \frac{\Delta}{2} \cdot L \quad . \tag{11.31}$$

See problem 11.6 for an interesting development of the above equations.

In the minimum-memory implementation described above, the relationship to the triangular structure in the memory connection is still inherent, but it evolves in time to prevent essentially idle memory cells.

Time variation of depth accompanying a data rate change

This triangular-interleaver interpretation in Figures 11.6, 11.8, and 11.9 (with or without minimum memory requirement) allows graceful change in operation of the interleaver depth between values that maintain L and J co-prime and are both lower and or equal to the upper bound[2]of $J \leq M \cdot L + 1$. The overall delay must be held constant in time to do so. This description calls the new depth J' and the old depth J'. There will be a point in time, call it time 0 at the beginning of a period where all new bytes entering the interleaver will observe depth J' while all bytes already in the interleaver (or de-interleaver) will observe the old depth J'. The corresponding symbol periods will similarly be denoted by T and T'.

[2]The constraint can always be met with the generalized triangular interleaver by using a sufficiently large M and dummy symbols to maintain the co-prime dpeth and period.

(At a constant delay through interleaving and de-interleaver, the symbol rate necessarily must change, which implies a data-rate change.) To maintain the same delay in absolute time, and noting that impulse protection for such a delay remains the same, if the codeword length $N = L$ and correction capability remain the same (just the depth changes to accommodate the same length burst of noise/errors), then

$$T' = \frac{J-1}{J'-1}T \quad , \tag{11.32}$$

another advantage of using the symbol-clock (rather than interleaver period) notation and interpretation. Since bytes through the interleaver/de-interleaver combination must undergo exact the same delay whether before or after the depth change, a byte exiting the interleaver or de-interleaver will be "clocked out" at exactly the same point in absolute time. The time-slot-interchange positions also are at the same time. However, there are two symbol clocks in which to interpret absolute time. A master implementation clock is determined by the greatest common multiple, GCM, of $J-1$ and $J'-1$ as

$$\frac{1}{\delta T} = \frac{GCM}{J-1}\frac{1}{T} = \frac{GCM}{J'-1}\frac{1}{T'} \quad . \tag{11.33}$$

Essentially the higher speed clock will always touch in absolute time all write/read instants of the interleaver for either the old or the new depth. New bytes (symbols) enter the interleaver as shown (the interleaver has the same isosceles triangle structure in concept at any depth) in terms of L clock cycles of the new symbol clock with symbol period $T = \frac{GCM}{J'-1}\frac{1}{\delta T}$ – and any memory element with a new byte after time zero in it will pass symbols $k \cdot (J'-1)T' = k \cdot GCM \cdot \delta T$ time slots later, where k is the index within the constant period of the interleaver. Old bytes within the interleaver exit on the $i(k) = k \cdot \Delta$ line at time instants $kT = k \cdot \frac{GCM}{J-1}\delta T$, while new bytes within the interleaver exit on the $i(k) = k \cdot \Delta'$ line at time instants $kT' = k \cdot \frac{GCM}{J'-1}\delta T$. All these time instants correspond to integer multiples of the high-rate clock. If at any time, both clocks are active then a read operation must be executed before the write operation on that clock cycle occurs (along with all the shifts in the proper order to not drop data within the delay elements shown if indeed the implementation was directly in the isosceles-triangular structure).

When no old bytes exist within the structure any longer (which is exactly the delay of the interleaver/de-interleaver combination), then the higher rate clock can be dropped and the new clock and new depth then used until any other depth change occurs in subsequent operation.

The time-variable structure uses the triangular structure to facilitate the explanation. This structure uses the constant $L(L-1)/2$ locations of memory independent of depth. However, the lower-memory requirement of the cell structure image discussed earlier can again be used even in the depth-variation case. The interleaver situation with larger depth will require a larger number of cells, so the number of cells increases with an increase in depth and decreases with a decrease in depth. For an increase in depth, $(J'-J)\frac{L-1}{2}$ more cells are needed in the interleaver (and the same number less for a decrease in depth). These cells can be allocated (or removed) as they are first necessary (or unnecessary).

Just as in the constant-depth case, an off-line algorithm (software) can be executed to determine the order for each new byte position of cell use by simply watching timers set according to the clock $\frac{1}{\delta T}$ as they sound. Any timer sounding at a time that is only a read can be reserved for potential future use. Any new byte to be written can use the cells reserved for use (either because it was vacated earlier by an isolated read at the old clock instants or because it is an additional cell needed). After $\frac{1}{2}(J-1)L(L-1)$ seconds, the interleaver algorithm (and de-interleaver algorithms) will have completely switched to the new depth. With such minimum-memory-cell use thus established, the depth change can then begin and follow the consequent pattern.

11.2.4 Random Binary Interleaving

In random interleaving, the idea is to eliminate the regular patterns in $G(D)$ or the associated interleaving rule $\pi(k)$ over a very long period. Random interleaving is often used in turbo coding, as in Section 11.3. The **uniform random interleaver** is an abstraction that is really an average over many statistical possibilities for an interleaver with large period L. The idea is that for any pattern of l positions (the

positions may be viewed as the location of 1's), it is equally likely to be interleaved into any of the

$$L_l = \binom{L}{l} \tag{11.34}$$

possibile pattern of l positions. Clearly any interleaver will be deterministic and so could only approximate such an effect. However, over an ensemble of interleavers (essentially making the interleaver cyclostationary), the uniform random interleaver can be hypothesized. Such an interleaver does not necessarily guarantee a fixed depth and $J-1$ spaces between previously adjacent symbols. Those earlier "with depth" interleavers address burst noises. The uniform random interleaver instead is a concept used in trellis coding for AWGNs.

The main objective of random interleaving is to create a very long block length for the concatenated code when viewed as a single code. Codes selected randomly, as long as the block length is very long, often can achieve capacity (Section 8.3). Random interleaving tries to install this element of randomness in the code design without increasing complexity, presuming the use of iterative decoding of the two interleaved codes. The number of ways in which to make a specific type of error is essentially reduced by the large codeword/interleaver length and the unlikely possibility that a pattern of errors for a low-distance error event will just happen to touch the right places for both codes. Section 11.3 investigates turbo codes where the uniform random interleaver is presumed in analysis.

A random uniform interleaver then would translate any pattern of l ones into one of those patterns with probability, L_l^{-1}. This subsection lists 2 types of random interleavers that attempt to approximate uniform random interleaving:

Berrou- Glavieux Block Interleavers

The period $L = K \cdot J = 2^i \cdot 2^j$ is a power of 2 and eight prime numbers are used:

m	1	2	3	4	5	6	7	8
p_m	17	37	19	29	41	23	13	7

The time index within a block is $k = 0, ..., L-1$. Recall that $(\cdot)_M$ means the quantity in brackets modulo M, i.e., the part left over after subtracting the largest contained integer multiple of M. Defining $r_0 = (k)_J$, $c_0 = (k - r_0)/J$, and $m = (r_0 + c_0)_8$ the interleaver order rule is

$$\pi(k) = c(k) + J \cdot r(k) \tag{11.35}$$

where

$$r(k) = (p_{m+1} \cdot (c_0 + 1) - 1)_K \tag{11.36}$$

and

$$c(k) = \left((\frac{K}{2} + 1) \cdot (r_0 + c_0) \right)_J \quad . \tag{11.37}$$

This interleaver has a long period for reasonable values of K and J and causes a robust randomness of the positions of error events in one code with respect to another. An event that results from exceeding free/minimum distance in one constituent code is very unlikely to also be in just the right places after interleaving to exceed the free/minimum distance of another code. Thus the number of free/minimum distance events is greatly reduced, and at lower SNR when many error events in a first decoder are like to occur, this redistribution of error events dominates $\bar{P}_b$. As SNR increases, the unlikely nature of noise exceeding free/minimum distance begins to dominate. One could expect good performance/gain at lower SNR, and then the usual constituent code performance at higher SNR. Such was the insight of Berrou in developing turbo codes (see Section 11.3).

JPL (Jet Propulsion Laboratory) Block Interleaver

For any K and even J, define:

ν	$P(D)$
2	$1+D+D^2$
3	$1+D^2+D^3$
4	$1+D^3+D^4$
5	$1+D^3+D^5$
6	$1+D^5+D^6$
7	$1+D^6+D^7$
8	$1+D^4+D^5+D^6+D^8$
9	$1+D^5+D^9$
10	$1+D^7+D^{10}$
11	$1+D^9+D^{11}$
12	$1+D^6+D^8+D^{11}+D^{12}$
13	$1+D^9+D^{10}+D^{12}+D^{13}$
14	$1+D^9+D^{11}+D^{13}+D^{14}$
15	$1+D+D^{15}$
16	$1+D+D^3+D^{12}+D^{16}$
17	$1+D^3+D^{17}$
18	$1+D^7+D^{18}$
23	$1+D^5+D^{23}$
24	$1+D^{17}+D^{22}+D^{23}+D^{24}$
31	$1+D^{28}+D^{31}$

Table 11.5: A list of maximum-length (primitive) polynomials.

m	1	2	3	4	5	6	7	8
p_m	31	37	43	47	53	59	61	67

Again the time index is $k = 0, ..., L-1$. Defining $r_0 = \left(\frac{i-m}{2} - c_0\right)_J$, $c_0 = \left(\frac{i-m}{2}\right)_J$, and $m = (r_0)_8$ the interleaver order rule is

$$\pi(k) = 2 \cdot r(k) \cdot K \cdot c(k) - (k)_2 + 1 \tag{11.38}$$

where

$$r(k) = (19 \cdot r_0)_{\frac{K}{2}} \tag{11.39}$$

and

$$c(k) = (p_{m+1} \cdot c_0 + 21 \cdot (k)_2)_J \quad . \tag{11.40}$$

Pseudorandom Interleavers

Pseudorandom interleavers make use of pseudorandom binary sequences (PRBS). Such sequences are based on a theory of maximum-length polynomials. A PRBS circuit is a rate one convolutional code with constant 0 input and with feedback based on a polynomial $p(D)$ with implementation $G(D) = \frac{1}{p(D)}$. The degree-ν polynomial is chosen so that with binary arithmetic, it has no nontrivial factors (i.e., it is "prime") and has other properties not discussed here.

Such circuits (if initialized with a nonzero initial condition) will generate a periodic sequence of period $2^\nu - 1$ that thus necessarily must include every nonzero binary pattern of length ν bits. The period of the interleaver can be no greater than the period of the PRBS. Table 11.2.4 lists such maximum-length polynomials[3]:

The pseudorandom interleaver uses such a PRBS to specify the output position of each input symbol. Thus, each successive output bit of the PRBS in combination with the last $\nu - 1$ such bits specifies an address $\pi(k)$ for the interleaver. If an address exceeds the interleaver period (when $L < 2^{\nu-1}$), then it is

[3]These were taken from the book Error Control Systems by Wickert (used in EE387 at Stanford), which has a far more complete listing of all the possible polynomials for each ν (there are many). See Chapter 12 of our text for implementations of scramblers. Note that the polynomials in x in that reference (as well as most coding books) corresponds to an advance, not a delay, which is why the reversal of the polynomials in Table 11.2.4

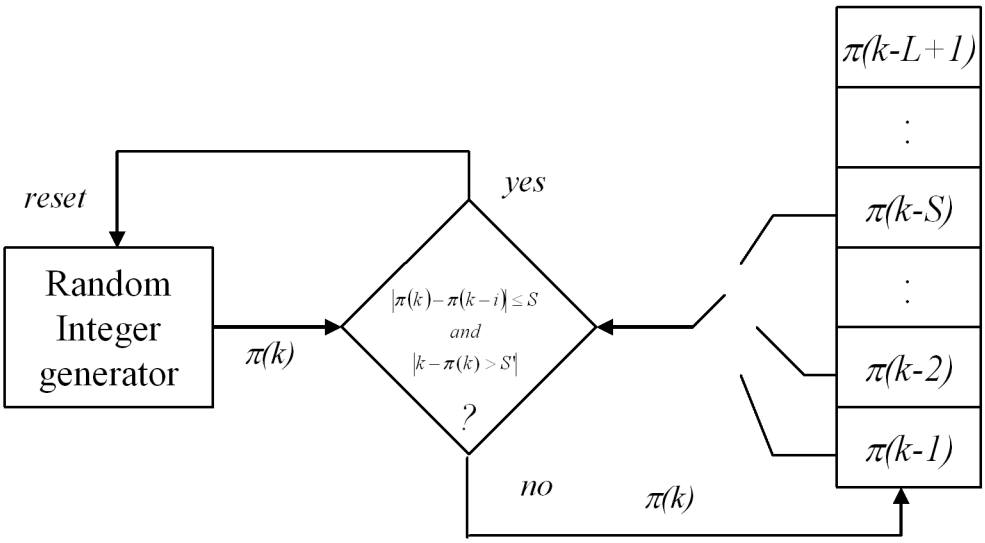

Figure 11.10: S-random interleaver illustrated.

discarded before use, and the PRBS circuit is cycled again. The de-interleaver can regenerate the same sequence and then successively extract the $(\pi(k))^{th}$ symbol and restore it to position k. Clearly, such pseudorandom interleaving corresponds to block interleaving unless the period is exactly $L = 2^{\nu} - 1$, in which case, the structure may have alternative implementations with less memory and delay.

S-Random Interleavers

An S-random interleaver tries to ensure that adjacent symbols are spaced further than S (and integer) after interleaving and to approximate also the random re-distribution of the uniform random interleaver. The S-random interleaver is designed by first chosing an $S \leq \sqrt{\frac{L}{2}}$ and running the algorithm in Figure 11.10. This algorithm will converger if the conditiion on S is met and the second integer S' is ignored. Usually designers run the algorithm several times increasing S' until they find the largest such value for which the algorithm converges. The correlation between intrinsic and extrinsic information for the S-random interleaver decays exponentiall with the difference between interleaver indices (and has the exception value zero for time difference of 0).

11.3 Turbo Codes

Turbo Codes are parallel or serial concatenations of simple good convolutional codes with significant interleaving first discovered by Claude Berrou in 1993. Interleaving is used with turbo codes and essentially the additional gain accrues to a reduction in the nearest neighbor count by the depth of the interleaver. Turbo Code design targets a data rate that is less than capacity, but perhaps just slightly less. Subsection 11.3.1 investigates the rate of simple concatenations, while Subsection 11.3.2 introduces the concept of code puncturing, which is very useful in implementing the desired $\bar{b}$ for a turbo code. Probability-of-error approximations appear in Subsections 11.3.3 and 11.3.4 for parallel and serial concatenations respectively. Subsections 11.3.5 and 11.3.6 enumerate various convolutional codes that have been found to also be good for turbo-code use when $\bar{b} < 1$.

11.3.1 Turbo-Code rate definition

Definition 11.3.1 (Turbo Code) *A* **Turbo Code** *is a parallel or serial concatenation of two convolutional codes with uniform random interleaving (or an approximation to it) to distribute the error events of the two codes with respect to one another. Turbo codes are designed with the expectation of iterative (i.e., "turbo") decoding's use at the receiver.*

parallel concatenations

The rate of a parallel concatenation of two systematic convolutional codes (where the information bits are sent only once, along with the parity bits of both codes) is again

$$\frac{1}{\bar{b}} = \frac{1}{\bar{b}_1} + \frac{1}{\bar{b}_2} - 1 \quad . \tag{11.41}$$

The -1 term is eliminated if the codes are not systematic.[4]

A few examples illustrate the possibilities:

EXAMPLE 11.3.1 (basic rate 1/3) If two convolutional codes both have rate $\bar{b}_1 = \bar{b}_2 = .5$, then $\bar{b} = 1/3$.

EXAMPLE 11.3.2 (rate 1/6) If two different convolutional codes both have rates $\bar{b}_1 = 1/3$ and $\bar{b}_2 = 1/4$, then $\bar{b} = 1/6$.

EXAMPLE 11.3.3 (higher rates) Higher rate turbo codes can be constructed from higher-rate convolutional codes. If two convolutional codes both have rate $\bar{b}_1 = \bar{b}_2 = 3/4$, then $\bar{b} = 3/5$. If two convolutional codes both have rate $\bar{b}_1 = \bar{b}_2 = .8$, then $\bar{b} = 2/3$.

It may be difficult to find a convolutional code with high $\bar{b}$ and good distance properties without a large number of states, so the puncturing of Subsection 11.3.2 becomes the preferred alternative to implementation of the high-rate turbo codes enumerated in Subsection 11.3.5.

serial concatenations

Serial turbo codes have rate equal to the product of the constituent code rates

$$\bar{b} = \bar{b}_1 \cdot \bar{b}_2 \quad . \tag{11.42}$$

A serial turbo code constructed from two rate 1/2 codes would have rate 1/4. Similarly a serial turbo code from two rate 2/3 codes has rate 4/9 (or less than 1/2). Clearly serial concatenation requires very high rates for the two used codes if the concatenation is to be high rate. The puncturing of the Subsection 11.3.2 is helpful for reducing the rate loss in serial concatenation of binary codes.

[4] This formula only applies if all the codes have less than one bit per dimension.

Figure 11.11: Puncturing of rate 1/3 turbo (or convolutional) code to rate 1/2.

Figure 11.12: Puncturing of rate 1/3 turbo (or convolutional) code to rate 2/3.

11.3.2 Puncturing

Before proceeding to Turbo Codes, this subsection first introduces puncturing of a convolutional code; a concept often used in either or both of the convolutional codes in a concatenated Turbo Coding system. Puncturing is the regular/periodic deletion of parity bits from a convolutional encoder output to increase the rate of the code. Such deletion can reduce the free distance of the code. Puncturing allows a single code to be used at different $\bar{b}$, which may be useful in systems that transmit at different data rates depending on conditions. Design or use of a different code for each data rate might complicate implementation. With turbo codes, it is the interleaver that provides the additional gain and it may be that the overall code sees little difference with puncturing from what might have been achieved using a better constituent high-rate code. Thus, puncturing is often used with turbo codes to simplify the implementation at several rates with the same encoder.

For instance, Figure 11.11 illustrates how the rate 1/3 turbo code that results from parallel concatenation of two rate-1/2 convolutional codes can be restored to rate-1/2 by alternately deleting one of the two parity bits. Some performance loss with respect to rate-1/3 turbo coding might be expected, but of course at an increase in data rate. Often, the resultant higher-rate code is still a very good code for the new higher data rate.

A yet higher data rate can be achieved as in Figure 11.12: A frame of 12 bits, 4 information and 8 parity, retains only 2 of the parity bits to increase the data rate to 2/3. Similarly, a rate 2/3 turbo, based on two rate 2/3 convolutional codes, could alternately delete 1 of the 2 parity bits generated at each symbol period.

Generally, a systematic code with k information bits per symbol and $n-k$ parity bits per symbol can be punctured to a higher rate q/p by accepting kq input bits and deleting $nq-kp$ of the parity bits,

$$\frac{kq}{nq-(nq-kp)} = \frac{q}{p} \quad . \tag{11.43}$$

The deleted bits in turbo coding may not be the same for each successive symbol to "distribute" the loss of parity, so j successive symbols may be used. In this most general case, the puncturing can be described by the $nqj \times kpj$ singular permutation/generator matrix G_{punc}. For the puncturing in Figure 11.11 with $j=2$, $k=1$, $n=3$, $q=1$ and $p=2$

$$G_{punc} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad , \tag{11.44}$$

accepting 6 bits but outputing 4. The punctured code words are generated by multiplying the output code sequence from the original code by the generator, or $\boldsymbol{v}_{punc}(D) = \boldsymbol{v}(D) \cdot G_{punc}$ where $\boldsymbol{v}(D)$ may be

written in terms of "stacking" j successive symbols. Puncturing can be very useful to avoid excessive constellation expansion when applying turbo codes to PAM/QAM transmission systems with $\bar{b} \geq .5$.

11.3.3 Analysis of probability of bit error for parallel concatenation

The analysis of Turbo Code probability of error makes use of the presumption of the uniform random interleaver in Section 11.2. This subsection begins with the example of the well-known 4-state rate-1/2 code.

A systematic encoder realization is

$$G_1(D) = \left[1 \quad \frac{1+D+D^2}{1+D^2}\right] \quad . \tag{11.45}$$

The two columns of the parity matrix (or effectively the two output bits of the encoder) have been reversed in order with respect to nominal use of this well-known 4-state convolutional code. Clearly such an output-bit-order reversal changes no property of the code itself (although the number of input bit errors and encoder mappings are different). Any input bit stream with weight $w = 1$ (or only one 1 in the stream so $u(D) = D^m$ where m is some integer) will lead to an infinite string of output 1's corresponding to infinite weight[5], which means that for this encoder the likelihood of a single input bit error occuring is essentially zero. In a parallel concatenated Turbo Code constructed from two uses of this code, inputs with 2 bit errors are thus of more interest. A length-2 input sequence, however, of $1 + D^2$ produces an outputsequence of $v(D) = \left[1 + D^2 \;\; 1 + D + D^2\right]$ and thus produces the minimum-distance (5) error-event code sequence. The parallel concatenation then would make an error if this two-bit input error event were to occur on one of the codes and also after de-interleaving to correspond to $D^m \cdot (1 + D^2)$ for any of $m = 0, ..., L - 1$ within the period L of the interleaver. For the uniform random interleaver, this simultaneous event has probability of occuring

$$\binom{L}{2}^{-1} = \frac{2}{L(L-1)} \quad , \tag{11.46}$$

for a particular value of m and thus probability $L \cdot \frac{2}{L(L-1)} = \frac{2}{L-1}$ that it could occur for any of the L values $m = 0, ..., L - 1$. Furthermore, the rate-1/3 concatenated code has a $d_{free} = 8 = 2 + 3 + 3$ (and corresponds to essentially $v_{turbo}(D) = \left[1 + D^2 \;\; 1 + D + D^2 \;\; \pi(1 + D + D^2)\right]$ where π has been used loosely to denote that the ordering of the 3 parity bits from the second interleaved use of the code occurs in 3 positions that are not the same times as the first 3 parity bits. The coding gain of the overall concatenated code is $8/3 = 2.67 = 4.26$ dB. The original "mother" code had gain $5/2 = 2/5 = 3.97$ dB and thus differs only by .3 dB [6]. However, the nearest neighbor coefficient for this minimum-distance event in the Turbo-Code concatenation is smaller. Since two input bit errors occur when half the minimum distance of the code has been exceeded by noise projection on the error event, then the probability that the $d_{free} = 8$ error event occurs for this turbo code is then

$$\bar{P}_b(d_{free}) \approx \frac{2}{L-1} b \cdot a(d_{free}, b) \cdot Q(\sqrt{d_{free} \cdot \text{SNR}}) \tag{11.47}$$

$$\approx \frac{4}{L-1} \cdot 1 \cdot Q(\sqrt{8 \cdot \text{SNR}}) \quad , \tag{11.48}$$

and corresponds to 2 input bit errors. There are no error events for the convolutional code that correspond to 3 (or any odd number greater than 3) of input bit errors (but there are error events corresponding to 4 and higher even numbers of input bit errors). Table 11.6 lists the error event combinations and shows the coefficent for interleaver depth $L = 101$. Table 11.7 repeats some measured results from a popular author. the error coefficients are also listed for periods of 1,001 and 10,001. The simulation results are very close, except that when the length of the error event becomes large relative to the period, then some accurracy is lost (because Table 11.6 essentially ignored the finite period in enumerating error-event pairs).

[5] An infinite number of channel errors must occur for only 1 input bit error.

[6] Assuming as is usual in convolutional codes that extra dimensions can be added as in method 1 of Section 10.1.

distance	in-ev code 1	in-ev code 2	$2 \cdot a(d,2)$	d_{free} construct	$\frac{4 \cdot a(d,2)}{100}$	$\frac{4 \cdot a(d,2)}{1000}$	$\frac{4 \cdot a(d,2)}{10000}$
8	$1+D^2$	$D^m \cdot (1+D^2)$	2	2+3+3	.04	.004	.004
9	$1+D^2$	$D^m \cdot (1+D^4)$	4	2+3+4	.08	.08	.008
	$1+D^4$	$D^m \cdot (1+D^2)$		2+4+3			
10	$1+D^2$	$D^m \cdot (1+D^6)$	6	2+3+5	.12	.012	.0012
	$1+D^4$	$D^m \cdot (1+D^4)$		2+4+4			
	$1+D^6$	$D^m \cdot (1+D^2)$		2+5+3			
11	$1+D^2$	$D^m \cdot (1+D^8)$	8	2+3+6	.16	.016	.0016
	$1+D^4$	$D^m \cdot (1+D^6)$		2+4+5			
	$1+D^6$	$D^m \cdot (1+D^4)$		2+5+4			
	$1+D^8$	$D^m \cdot (1+D^2)$		2+6+3			
12	$1+D^2$	$D^m \cdot (1+D^{10})$	10	2+3+7	.20	.02	.002
	$1+D^4$	$D^m \cdot (1+D^8)$		2+4+6			
	$1+D^6$	$D^m \cdot (1+D^6)$		2+5+5			
	$1+D^8$	$D^m \cdot (1+D^4)$		2+6+4			
	$1+D^{10}$	$D^m \cdot (1+D^2)$		2+7+3			
t	$1+D^2$	$D^m \cdot (1+D^{2(t-8)})$	2(t-7)		.04(t-7)	.004(t-7)	.0004(t-7)
	$\vdots$	$\vdots$		$\vdots$			
	$1+D^{2(t-8)}$	$D^m \cdot (1+D^2)$					

Table 11.6: Enumeration of $b = 2$ input bit error events and corresponding codeword error event construction for $G_1(D) = (1 + D + D^2)/(1 + D^2)$.

distance	N_b for $L = 10^2$	N_b for $L = 10^3$	N_b for $L = 10^4$
8	3.89×10^{-2}	3.99×10^{-3}	3.99×10^{-4}
9	7.66×10^{-2}	7.96×10^{-3}	7.99×10^{-4}
10	.1136	1.1918×10^{-2}	1.1991×10^{-3}
11	.1508	1.5861×10^{-2}	1.5985×10^{-3}
12	.1986	1.9887×10^{-2}	1.9987×10^{-3}
13	.2756	2.4188×10^{-2}	2.4017×10^{-3}
14	.4079	2.9048×10^{-2}	2.8102×10^{-3}
15	.6292	3.4846×10^{-2}	3.2281×10^{-3}
16	1.197	6.5768×10^{-2}	6.0575×10^{-3}

Table 11.7: 4-state convolutional code as turbo constituent

distance	in-ev code 1	in-ev code 2	$2 \cdot a(d,2)$	d_{free} construct	$\frac{4 \cdot a(d,2)}{100}$
10	$1+D^3$	$D^m \cdot (1+D^3)$	2	2+4+4	.04
12	$1+D^3$	$D^m \cdot (1+D^6)$	4	2+4+6	.08
	$1+D^6$	$D^m \cdot (1+D^3)$		2+6+4	
14	$1+D^3$	$D^m \cdot (1+D^9)$	6	2+4+8	.12
	$1+D^6$	$D^m \cdot (1+D^6)$		2+6+6	
	$1+D^9$	$D^m \cdot (1+D^3)$		2+8+4	
16	$1+D^3$	$D^m \cdot (1+D^{12})$	8	2+4+10	.16
	$1+D^6$	$D^m \cdot (1+D^9)$		2+6+8	
	$1+D^9$	$D^m \cdot (1+D^6)$		2+8+6	
	$1+D^{12}$	$D^m \cdot (1+D^3)$		2+10+4	
t	$1+D^3$	$D^m \cdot (1+D^{1.5(t-8)})$	t-8		.02(t-8)
	$\vdots$	$\vdots$		$\vdots$	
	$1+D^{1.5(t-8)}$	$D^m \cdot (1+D^3)$			

Table 11.8: Enumeration of $b = 2$ input bit error events and corresponding codeword error event construction for $G_2(D) = (1+D^2)/(1+D+D^2)$.

It is interesting to view the same example with the output bit order reversed to the usual instance of this code so that

$$G_2(D) = \begin{bmatrix} 1 & \dfrac{1+D^2}{1+D+D^2} \end{bmatrix} \quad . \tag{11.49}$$

This encoder maps a weight 5 codeword error event of $\begin{bmatrix} 1+D+D^2 & 1+D^2 \end{bmatrix}$ to 3 input bit errors. With concatenation, then this error event corresponds to $d_{free} = 7 = 3+2+2$ indicating that the concatenated code does not have the same codewords and indeed has a lower minimum distance. Nonetheless, this minimum distance requires 3 input-bit positions of errors to coincide for the two codes, which has probability (with uniform random interleaving) of

$$L \cdot \binom{L}{3}^{-1} = \frac{6}{(L-1)(L-2)} \quad . \tag{11.50}$$

Thus, 3-bit error events have very small probability, even if they correspond to minimum-distance codeword events (unless the SNR is very high so that $d_{\min}$ dominates all nearest-neighbor-like coefficients). In fact, the weight two input error event with smallest codeword distance is $1+D^3$ and corresponds to an output distance of $d = 10$. Repeating Table 11.6 now in Table 11.8 yields (there are no odd-number free distances for 2-input-bit-error events).

The second instance of the concatenated code with $G_2(D)$ is clearly better because the distances are larger, and the error coefficients are the same. (The improvement is roughly 1 dB). However, at high SNRs, the dominant free-distance-7 error event will eventually make asymptotic performance worse. So, up to some SNR, the second code is better and then it is worse after that SNR. The eventual dominance of minimum distance at high SNR causes the contribution of different error events to dominate, leading to an unusual flattening of the $\bar{P}_b$ curve known as the "error floor" in turbo coding. In both concatenated codes, the larger free distance from the rate 1/3 code is only a slight improvement. The main improvement is the reduction in bit-error probability caused by division by the interleaver period. Such an improvement is often called **interleaver gain.** Over the range of normal use of codes, a factor of 10 usually corresponds to 1 dB improvement, so the interleaver gain for this code would then be

$$\gamma_{interleaver} = \log_{10}((L-1)/2) \tag{11.51}$$

at least up to a few dB (at which point the approximation of 1 dB per factor of 10 is no longer valid).

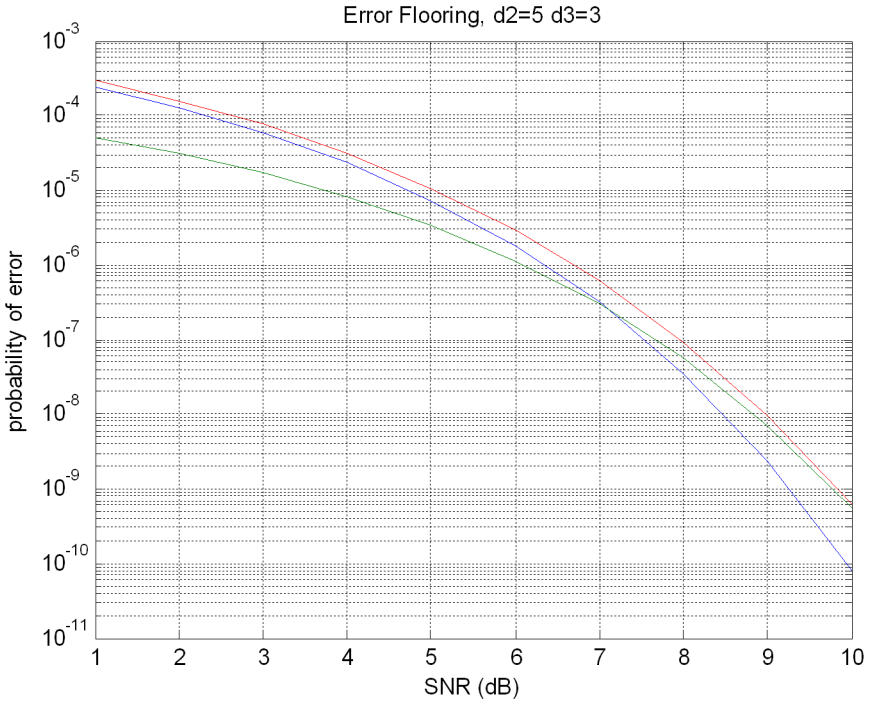

Figure 11.13: Generic depiction of basic error flooring in Turbo Codes.

In a more general situation of a rate $1/n$ code

$$G_{sys}(D) = \left(1 \;\; \frac{g_1(D)}{g_0(D)} \; ... \; \frac{g_{n-1}(D)}{g_0(D)}\right) \quad , \tag{11.52}$$

and any weight-one input information sequence $u(D) = D^m$ produces an infinite-weight output codeword(or large finite weight if the code is terminated in a finite packet length). Thus, again only weight 2 (or higher) input errors are of interest since any $g_0(D)$ must divide $1 + D^j$ for some sufficiently large j. Since input-bit weight 3 errors have a coefficient factor of $\frac{6}{(L-1)(L-2)}$, then they typically have much lower contribution to $\bar{P}_b$ unless the SNR is sufficiently high and the distance for these errors is smaller.

Error Flooring Error flooring occurs when the SNR is sufficiently high that the smaller nearest-neighbor coefficient for error events with 2 errors (or more) is overwhelmed by the exponential decrease with SNR of the Q-function at high SNR. At such high SNR, $d_{\min}$ error events will again dominate. Figure 11.13 depicts this effect generically for two types of error events for which one has a $d_2 = 5$ (output distance corresponding to 2-input bit errors), but has a larger coefficient of .01, while the second has a smaller $d_3 = 3$ (output distance corresponding to 3 input bit errors), but a coefficient of .001. At smaller SNR, the low coefficient of the Q-function term with smaller distance causes this term to be negiligible, but eventually the exponential decay of the Q-function term with larger distance causes it instead to be negligible. In the middle of Figure 11.13 where the overall error probability (the upper curve, which is sum of the two curves) deviates temporarily from a pure exponential decay and "curves less or flattens" temporarily until resuming the exponential decay again for the lower distance portion. Thus, the smaller error-coefficient term dominates at low SNR and provides lower total probability of error. Thus, operation with a Turbo Code attempts to choose parameters so that this portion of the curve corresponds to the range of interest. Over this $\bar{P}_b$ range, if 2-input-bit error events dominate, then

$$\bar{P}_{b,turbo} \approx \frac{2\bar{N}_b(d_2)}{L} Q\left(\frac{d_{\min}}{2\sigma}\right) = \frac{4}{L} Q\left(\frac{d_{2,cat}}{2\sigma}\right) \quad , \tag{11.53}$$

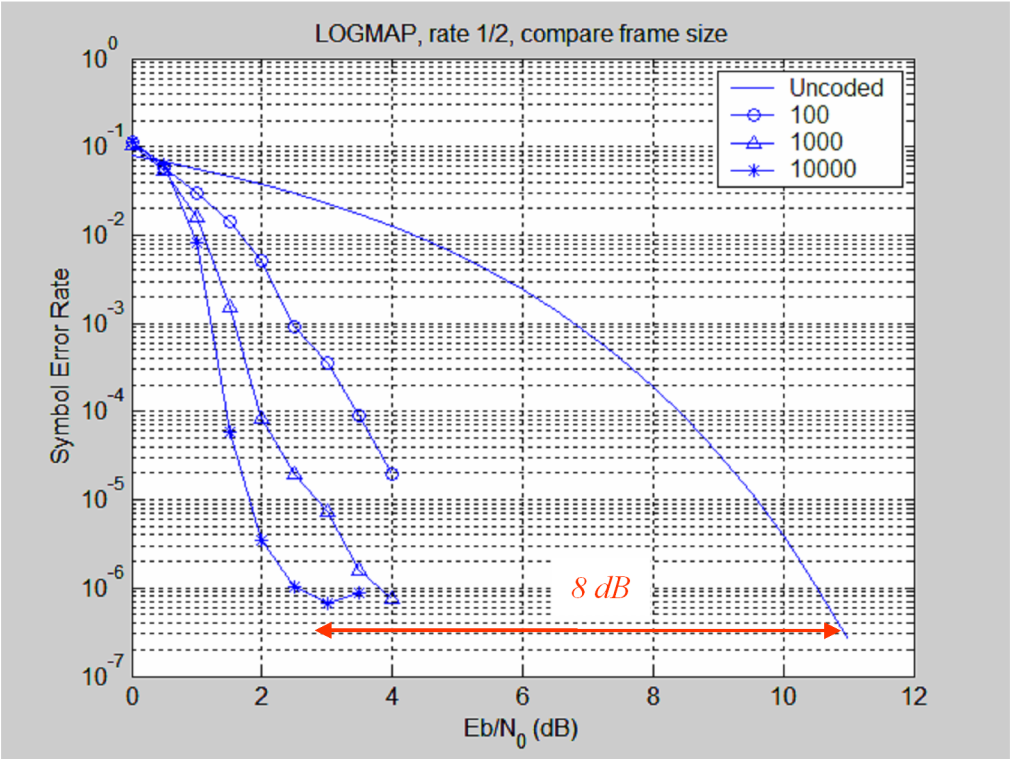

Figure 11.14: Probability of error for the rate-1/2 4-state turbo code, with puncturing to rate 1/2.

where $d_{2,cat}$ is the 2-input-bit-error codeword distance for the concatenated code system. The probability of bit error thus reduces by the factor $L/2$. More generally with rate k/n codes, the situation is more complicated because a simple 2-bit error pattern "cancelling all the denominators" is more complicated and the interleaver possibilities multiply. Aversion of this conceptual complexity uses puncturing of rate 1/n codes rather than direct design for k/n codes.

Figure 11.14 illustrates the probability of error for interleaver depths of $L = 100$, $L = 1000$, and $L = 10000$ for the 4-state turbo rate 1/2 Turbo Code (with encoder G_1) of Table 11.9 with puncturing. From the code itself, one expects a gain of 4.7 dB plus another 3.7 dB for $L = 10000$, so then roughly 8.4 dB also. The actual plotted gain is slightly less at about 8 dB because of the effects and contributions of higher-distance terms.

Figure 11.15 illustrates the convergence of the same code as in Figure 11.14 for the interleaver size of 1000. The probabilty of symbol error (and thus the probability also of bit error for this code) converges within just a few iterations. Figure 11.16 compares the probability of error for SOVA and APP, while the difference is larger at very low SNR, the difference becomes small in the range of operation of $\bar{P}_b = 10^{-6}$. Figure 11.17 illustrates the additional gain of not puncturing and using rate 1/3 instead, which is about .7 dB at $\bar{P}_b = 10^{-6}$ for this example. Actually, slightly larger because this rate 1/3 code also sees a larger d distribution generally than the punctured rate 1/2 code. This gain is small for this code, as is typical with puncturing in turbo codes that little is lost in terms of coding gain.

11.3.4 Analysis of probability of bit error for serial concatenation

Serial concatenation of codes has a somewhat different analysis than parallel concatenation. Analysis of serial concatenation retains the basic concept that an error event with several bit errors is less likely to fall with random interleaving into the exact positions that cause errors in two codes simultaneously. However, the re-encoding into new additional parity bits of the parity (or all) bits of the inner code causes some changes in the probability of error formulas. Often in serial turbo-code concatenation, the

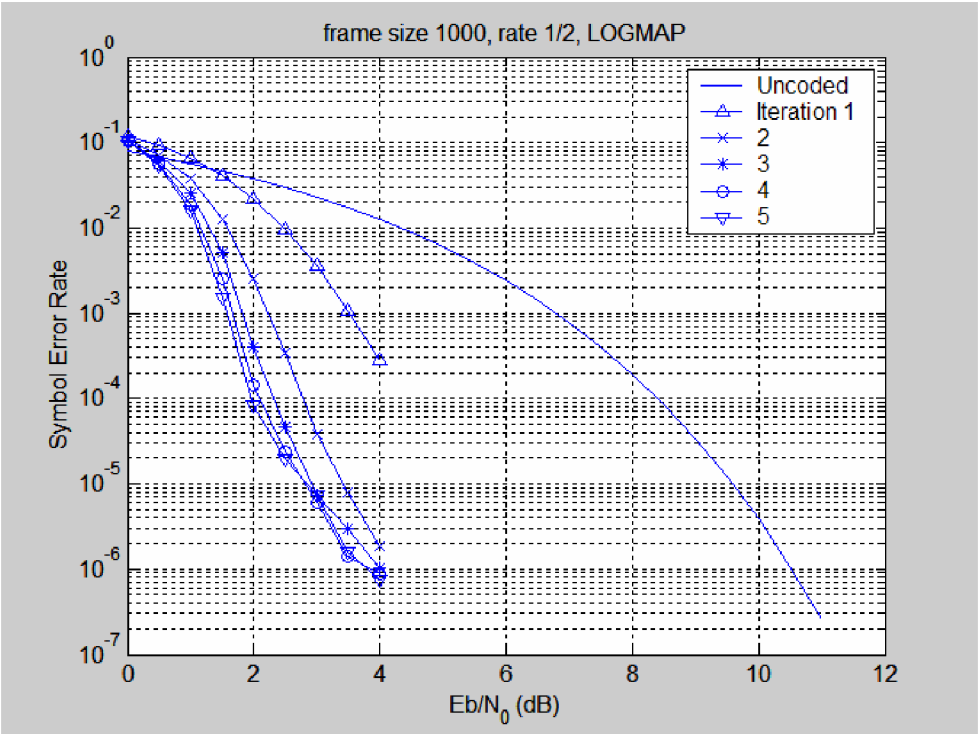

Figure 11.15: APP Convergence of Probability of error for the rate-1/2 4-state turbo code, with puncturing to rate 1/2.

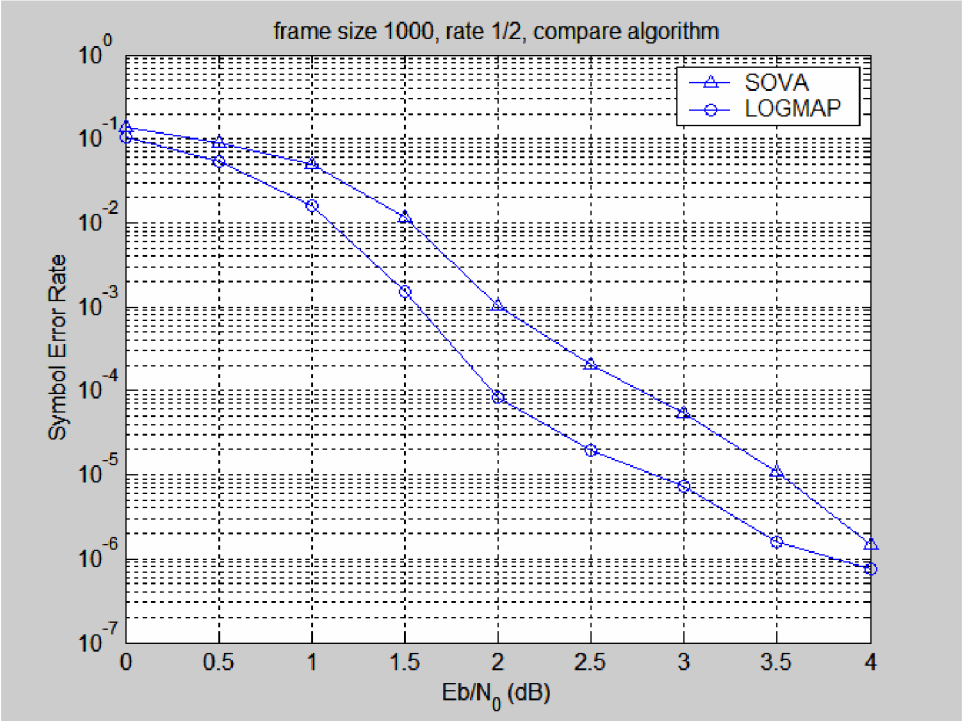

Figure 11.16: Comparison of probability of error for SOVA and APP on the rate 1/2 (punctured) turbo code.

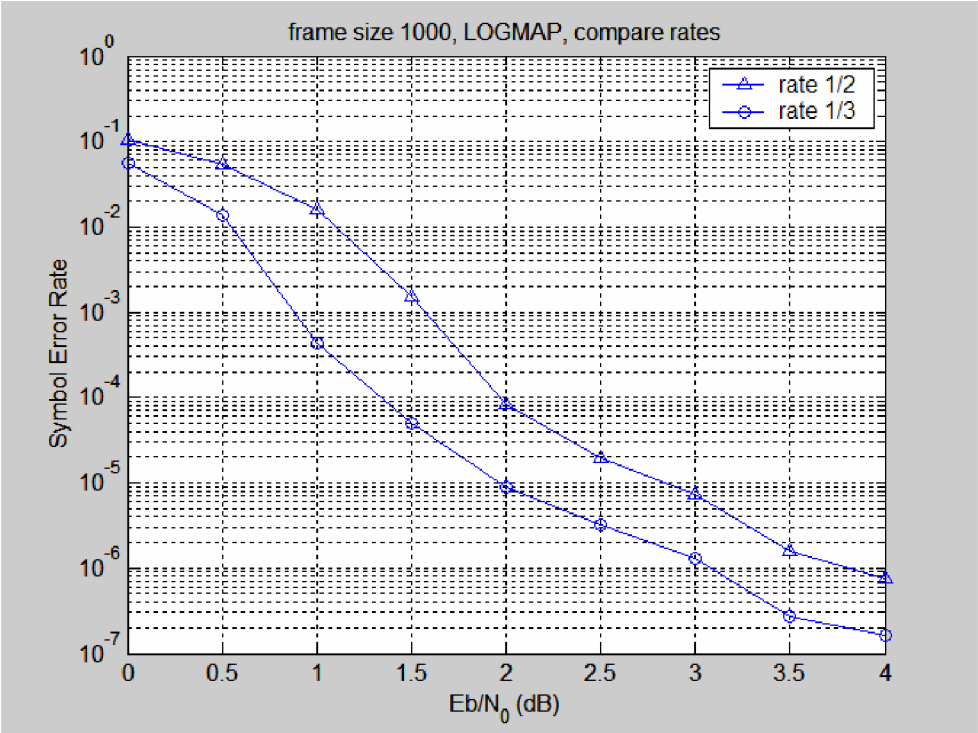

Figure 11.17: Comparison of rate 1/3 and rate 1/2 with puncturing for 4-state Turbo code.

two codes are different and the ensuing analysis accommodates such difference.

First, the error coefficient simply reflects the fact that in serial concatenation the probability that $\left\lceil \frac{d_{free}^{out}}{2} \right\rceil$ bits after uniform random interleaving of depth L again fall in "all the wrong placesin the outer code (out)" is

$$\left(\begin{array}{c} L \\ \left\lceil \frac{d_{free}^{out}}{2} \right\rceil \end{array} \right) \quad . \tag{11.54}$$

This expression is often approximated in serial turbo-code analysis by

$$C \cdot L^{-\left(\left\lceil \frac{d_{free}^{out}}{2} \right\rceil \right)} \quad , \tag{11.55}$$

where the exponential dependence on the codeword (or encoder output) free distance rather than the number of input bit errors, distinguishes serial concatenation from parallel concatenation in Equations 11.46 and 11.50. C is a constant. This coefficient mutiplies the error coefficent for the outer Turbo Code, whatever that coefficient is, for both symbol-error and bit-error probability expressions. While this factor follows parallel concatenation, the adjustment to the Q-function argument requires additional scrutiny for serial concatenation. This text assumes that at least the inner encoder is always systematic so that at least two-input-bit errors are necessary to cause it to have finite output distance and thus to have non-zero probability of the error event. This inner distance for 2-bit errors is called d_2^{in} and similarly the 3-bit error distance is called d_3^{in}. If the inner code's decoder has had the error event occur, then that inner decoder's error probability has Q-function argument $\sqrt{d_2^{in} \cdot \mathrm{SNR}}$. These 2 input bits (which are also part of the output distance in a systematic realization) are "already guaranteed" to be contributing to the output error event for the outer code so the outer decoder's tolerance for errors is reduced to[7]

$$\left\lceil \frac{d_{free}^{out} - 3}{2} \right\rceil \tag{11.56}$$

The overall effect on the Q-function argument for an overall decoder is

$$\left[\left\lceil \frac{d_{free}^{out} - 3}{2} \right\rceil \cdot d_2^{in} + d_w^{in} \right] \cdot \mathrm{SNR} \tag{11.57}$$

where $d_w^{in} = d_2^{in}$ for situations in which the outer code has even d_{free}^{out} and $d_w^{in} = d_3^{in}$ for odd d_{free}^{out}. The reader should recall that the SNR in (11.57) scales down as the product of the two code rates, which is $\bar{b} = \bar{b}_{in} \cdot \bar{b}_{out}$. Thus, the additional error-correcting power (or distance) of the outer code applies to all the extra bits that must additionally be in error in the outer code and thus multiplies d_2^{in} but does mutiply the common bits. However those common bits do have to be in error and whence the last additive term of d_w^{in} in (11.57). The total-bit-error-counting quanitity

$$N_b(d) = \sum_{b=1}^{\infty} b \cdot a(d,b) \tag{11.58}$$

has special use in serial concatenation. With it, the overall probability of bit error expression is then

$$\bar{P}_b \approx L \left(\begin{array}{c} L \\ \left\lceil \frac{d_{free}^{out}}{2} \right\rceil \end{array} \right)^{-1} \cdot \frac{N_b(d_{free}^{in}) \cdot N_b(d_{free}^{out})}{b} \cdot Q\left(\sqrt{\left\{ \left\lceil \frac{d_{free}^{out} - 3}{2} \right\rceil \cdot d_2^{in} + d_w^{in} \right\} \cdot \mathrm{SNR}} \right) \tag{11.59}$$

$$= L \left(\begin{array}{c} L \\ \left\lceil \frac{d_{free}^{out}}{2} \right\rceil \end{array} \right)^{-1} \cdot \frac{N_b(d_{free}^{in}) \cdot N_b(d_{free}^{out})}{b} \cdot Q\left(\sqrt{2 \cdot \left\{ \left\lceil \frac{d_{free}^{out} - 3}{2} \right\rceil \cdot d_2^{in} + d_w^{in} \right\} \cdot \bar{b} \cdot \frac{\mathcal{E}_b}{\mathcal{N}_0}} \right) \tag{11.60}$$

[7] The use of the greatest integer function in (11.56) allows the argument to be reduced by 3 when outer free distance is odd and by only 2 when outer free distance is 2 in agreement with the reality that codes with odd distances essentially get one more position of possible error before experiencing an error event.

where the second expression uses the "energy per bit" form. It is clear that an outer code with odd free distance is preferable it terms of Q-function argument per complexity. However, the outer encoder need not be systematic nor even use feedback. The interleaving gain thus is slightly altered with respect to parallel concatenation to be

$$\gamma_{serial} = \log_{10}\left(\frac{L!}{\left\lceil \frac{d_{free}^{out}}{2} \right\rceil !}\right) \text{ dB} \tag{11.61}$$

over the range of operation of 10^{-4} to 10^{-7}. This factor can be large for $d_{free}^{out} > 2$, but the product of $N_b(d)$ terms reduces the effect of the gain. Error flooring in serial concatenation follows the same basic principle as in parallel concatenation, namely that eventually as SNR increase, minimum distance dominates probability of error and thus an event with larger free distance than d_2^{in} or d_w^{in} but also smaller error coefficient could eventually be expected be the major contributer to probability of error.

11.3.5 Coding Tables for Parallel Concatenation with rate $1/n$, or for rate $n-1/n$ with no puncturing

The following tables are extracted from some papers by Divsalar at the course website. Some of the data in the tables has been augmented by this text's author to include more information pertinent to code implementation and analysis. All the codes appear to be the best or among the best known for the various complexities and parameters listed.

The 4-state rate-1/2 convolutional code that has been previously studied has a systematic realization $[1(1+D^2)/(1+D+D^2)]$. The resultant rate-1/3 turbo code can punctured alternately on the parity bits of the two encoders in each successive symbol with uniform random interleaving on the information bits. The left-most values of d_2, d_3, and d_{free} are for the base or mother code itself. The right-most values are for the turbo code or concatenated code and thus can be used directly in probability of error analysis expressions like (11.53). The rule for adding the 3 right-most additional columns is to find the difference between the d_{free} and d_i listed for the base code, subtract 2 or 3 for d_2 or d_3 respectively, and then add that amount to the distance. $d_{free,cat}$ is the new minimum of the two quantities $d_{2,cat}$ and $d_{3,cat}$. Thus, $d_{2,cat} = 2 \cdot d_2 - 2$ and $d_{3,cat} = 2 \cdot d_3 - 3$ and $d_{free,cat} = \min\{d_{2,cat}, d_{3,cat}\}$. With puncturing, the puncturing patter of 101010 was used for the rate 1/2 code in the worst position in terms of weight for the base code.

Table 11.9 lists the distances for 2 and 3 input bit errors as well as d_{free} for the overall-rate 1/2 turbo code by Divsilar (the overall code values were determined by this author and omitted by Divsilar)" Another more complete set of rate-1/2 punctured codes will appear shortly in Subsection 11.3.6.

2^ν	$g_0(D)$	$g_1(D)$	d_2	d_3	d_{free}	$d_{2,cat}$	$d_{3,cat}$	$d_{free,cat}$
4	7	5	4	3	3	6	5	5
8	13	15	5	4	4	8	6	6
16	23	37	7	4	4	10	8	8

Table 11.9: Rate 1/2 Constituent PUNCTURED convolutional codes for turbo codes

The best rate 1/3 codes found by Divsilar (again with overall turbo code values found by this author) appear in Table 11.10:

The best rate 1/4 codes found by Divsilar (again with overall turbo code values found by this author) appear in Table 11.11:

The best rate 2/3 codes found by Divsilar (again with overall turbo code values found by this author) appear in Table 11.12:

The best rate 3/4 codes found by Divsilar (again with overall turbo code values found by this author) appear in Table 11.13:

2^ν	$g_0(D)$	$g_1(D)$	$g_2(D)$	d_2	d_3	d_{free}	$d_{2,cat}$	$d_{3,cat}$	$d_{free,cat}$
2	3	2	1	4	∞	4	6	∞	6
4	7	5	3	8	7	7	14	11	11
8	13	17	15	14	10	10	26	17	17
16	23	33	37	22	12	12	42	21	21

Table 11.10: Rate 1/3 Constituent convolutional codes for turbo codes

2^ν	$g_0(D)$	$g_1(D)$	$g_2(D)$	$g_3(D)$	d_2	d_3	d_{free}	$d_{2,cat}$	$d_{3,cat}$	$d_{free,cat}$
8	13	17	15	11	20	12	12	38	21	21
16	23	35	27	37	32	16	14	62	31	31

Table 11.11: Rate 1/4 Constituent convolutional for turbo codes

11.3.6 Parallel and Serial Turbo Code Tables with puncturing for base rate 1/2

To enumerate best codes with puncturing, Deaneshgaran, Laddomada and Mondin (DLM) have searched over all rate 1/2 codes and puncturing patterns for different rates to find best parallel and serial concatenations for up to 32 states. The mother codes used in the puncturing search are to create high rate $(n-1)/n$ codes after puncturing. These codes are listed for 4 and 8 states in Table 11.14 and for 16 and 32 states in Table 11.15. An indication of SNR means among all encoders with the same d_2, the one with minimum SNR to get $\bar{P}_b = 10^{-6}$ was selected. The entry d_3 means instead that the code with maximum d_3 was selected. The entry d_2 is a code with largest d_2.

Parallel concatenations with the same code Table 11.16 lists the best parallel-concatenation codes (pesuming the same code is used twice). The codes are rate $(n-1)/n$ after puncturing. The puncturing pattern is octal and corresponds to the mother code rate-1/2 output bit pairs (info, parity) are enumerated from left to right in increasing time and the pucturing pattern is pressed on top with 0 meaning puncture that parity bit. For instance 5352 means 101 011 101 010 so and corresponds to (letting i_k be an information bit and p_k be the corresponding parity bit)

$$(i_1, p_1, i_2, p_2, i_3, p_3, i_4, p_4, i_5, p_5, i_6, p_6) \rightarrow (i_1, i_2, i_3, p_3, i_4, i_5, i_6) \quad . \tag{11.62}$$

Puncturing patterns need not always maintain a systematic code, although it is rare in parallel concatenation to see puncturing of information bits. The distances shown are for the mother code itself. As earlier, the overall $d_{free,cat}$ for performance analysis of the concatenated system is found as $d_{free,cat} = \min_{i=2,3} 2 \cdot d_i - i$.

Serial concatenations - inner codes Table 11.17 lists best known inner codes with puncturing patterns generated from rate 1/2 mother codes. Often the puncturing can delete information bits (meaning the parity bit carries better information under puncturing than the information bit itself). Puncturing is applied to the inner code output and the rate is for the resultant punctured inner code and is $(n-1)/n$.

Serial concatenations - outer codes Table 11.18 lists best known outer codes with puncturing patterns generated from rate 1/2 mother codes. Often the puncturing can delete information bits (meaning the parity bit carries better information under puncturing than the information bit itself). Puncturing is applied to the outer code output and the rate is for the resultant punctured outer code and is $(n-1)/n$.

2^ν	$h_0(D)$	$h_1(D)$	$h_2(D)$	d_2	d_3	d_{free}	$d_{2,cat}$	$d_{3,cat}$	$d_{free,cat}$
4	7	3	5	4	3	3	6	3	3
8	13	15	17	5	4	4	8	5	5
16	23	35	27	8	5	5	14	7	7
16	45	43	61	12	6	6	22	9	9

Table 11.12: Rate 2/3 Constituent convolutional codes for turbo codes

2^ν	$h_0(D)$	$h_1(D)$	$h_2(D)$	$h_3(D)$	d_2	d_3	d_{free} d_{free}	$d_{2,cat}$	$d_{3,cat}$	$d_{free,cat}$
4	7	5	3	1	3	3	3	4	3	3
8	13	15	17	11	4	4	4	6	5	5
16	23	35	33	25	5	4	4	8	5	5

Table 11.13: Rate 3/4 Constituent convolutional codes for turbo codes

A serial turbo-code design would choose one code from Table 11.17 and one from 11.18, using (11.60) to evaluate the performance and the overall code rate is $\bar{b} = \bar{b}_{in} \cdot \bar{b}_{out}$.

SubsectionCDMA 2000 Turbo Code

To be added at a later date. See document at web page for more information.

2^ν	$\left[1 \ \frac{g_1}{g_0}\right]$	d	N_e	$N_b(d)$	d_2	d_3
4	$[\ 1 \ \frac{5}{7}\ ]$	5	1	3	6	5
(d_2)		6	2	6		
		7	4	14		
		8	8	32		
		9	16	72		
4	$[\ 1 \ \frac{7}{5}\ ]$	5	1	2	5	∞
(SNR)		6	2	6		
		7	4	14		
		8	8	32		
		9	16	72		
8	$[\ 1 \ \frac{15}{13}\ ]$	6	2	6	8	6
(d_2)		8	10	40		
		10	49	245		
		12	241	1446		
		14	1185	8295		
8	$[\ 1 \ \frac{17}{13}\ ]$	6	1	4	8	7
(d_2)		7	3	9		
		8	5	20		
		9	11	51		
		10	25	124		
8	$[\ 1 \ \frac{15}{17}\ ]$	6	1	2	6	∞
(SNR)		7	3	12		
		8	5	20		
		9	11	48		
		10	25	126		

Table 11.14: Best 4- and 8-sate Rate 1/2 Constituent (mother) convolutional codes for for use with puncturing in turbo codes

2^ν	$\left[1 \ \frac{g_1}{g_0}\right]$	d	N_e	$N_b(d)$	d_2	d_3
16 (d_2)	[1 $\frac{33}{31}$]	7	2	8	12	7
		8	4	16		
		9	6	26		
		10	15	76		
		11	37	201		
16	[1 $\frac{21}{37}$]	6	1	2	6	∞
		7	1	5		
		8	3	10		
		9	5	25		
		10	12	56		
16 (d_2)	[1 $\frac{27}{31}$]	7	2	8	12	7
		8	3	12		
		9	4	16		
		10	16	84		
		11	37	213		
16 (d_2)	[1 $\frac{37}{23}$]	6	1	4	12	8
		8	6	23		
		10	34	171		
		12	174	1055		
		14	930	6570		
16 (d_2)	[1 $\frac{33}{23}$]	7	2	8	12	7
		8	4	16		
		9	6	26		
		10	15	76		
		11	37	201		
16 (d_2)	[1 $\frac{35}{23}$]	7	2	8	12	7
		8	3	12		
		9	4	16		
		10	16	84		
		11	37	213		
16 (SNR)	[1 $\frac{23}{35}$]	7	2	6	7	∞
		8	3	12		
		9	4	20		
		10	16	76		
		11	137	194		
32 (d_3)	[1 $\frac{71}{53}$]	8	3	12	12	∞
		10	16	84		
		12	68	406		
		14	860	6516		
		16	3812	30620		
32 (SNR d_2)	[1 $\frac{67}{51}$]	8	2	7	20	8
		10	20	110		
		12	68	406		
		14	469	3364		
		16	2560	20864		

Table 11.15: Best 16- and 32-state Rate 1/2 Constituent (mother) convolutional codes for for use with puncturing in turbo codes

$n-1$	4 states	8 states	16 states	32 states
2	$[\ 1\ \frac{5}{7}\ ]$ 13 (3,1,3) $d_2 = 4$, $d_3 = 3$	$[\ 1\ \frac{15}{13}\ ]$ 13 (4,3,10) $d_2 = 5$, $d_3 = 4$	$[\ 1\ \frac{37}{23}\ ]$ 13 (4,2,6) $d_2 = 7$, $d_3 = 4$	$[\ 1\ \frac{67}{51}\ ]$ 13 (5,2,7) $d_2 = 9$, $d_3 = 5$
3	$[\ 1\ \frac{5}{7}\ ]$ 56 (3,4,10) $d_2 = 3$, $d_3 = 3$	$[\ 1\ \frac{15}{13}\ ]$ 53 (3,2,5) $d_2 = 3$, $d_3 = 3$	$[\ 1\ \frac{37}{23}\ ]$ 53 (3,1,3) $d_2 = 4$, $d_3 = 3$	$[\ 1\ \frac{67}{51}\ ]$ 53 (4,2,7) $d_2 = 7$, $d_3 = 4$
4	$[\ 1\ \frac{5}{7}\ ]$ 253 (2,1,2) $d_2 = 2$, $d_3 = 3$	$[\ 1\ \frac{15}{13}\ ]$ 253 (3,9,24) $d_2 = 3$, $d_3 = 3$	$[\ 1\ \frac{37}{23}\ ]$ 253 (3,3,9) $d_2 = 4$, $d_3 = 3$	$[\ 1\ \frac{67}{51}\ ]$ 253 (3,1,3) $d_2 = 5$, $d_3 = 3$
5	$[\ 1\ \frac{5}{7}\ ]$ 1253 (2,2,4) $d_2 = 2$, $d_3 = 3$	$[\ 1\ \frac{17}{13}\ ]$ 1253 (3,15,40) $d_2 = 3$, $d_3 = 3$	$[\ 1\ \frac{27}{31}\ ]$ 1272 (3,2,6) $d_2 = 4$, $d_3 = 3$	$[\ 1\ \frac{71}{53}\ ]$ 1272 (4,108,406) $d_2 = 4$, $d_3 = \infty$
6	$[\ 1\ \frac{5}{7}\ ]$ 5352 (2,22,44) $d_2 = 2$, $d_3 = 3$	$[\ 1\ \frac{17}{13}\ ]$ 5253 (2,1,2) $d_2 = 2$, $d_3 = 3$	$[\ 1\ \frac{27}{31}\ ]$ 5253 (3,12,33) $d_2 = 3$, $d_3 = 3$	$[\ 1\ \frac{71}{53}\ ]$ 5253 (3,3,6) $d_2 = 3$, $d_3 = \infty$
7	$[\ 1\ \frac{5}{7}\ ]$ 25253 (2,7,14) $d_2 = 2$, $d_3 = 3$	$[\ 1\ \frac{15}{17}\ ]$ 25253 (2,7,14) $d_2 = 2$, $d_3 = \infty$	$[\ 1\ \frac{33}{23}\ ]$ 25253 (2,1,2) $d_2 = 2$, $d_3 = 3$	$[\ 1\ \frac{67}{51}\ ]$ 25253 (2,1,2) $d_2 = 3$, $d_3 = 3$
8	$[\ 1\ \frac{5}{7}\ ]$ 125253 (2,9,18) $d_2 = 2$, $d_3 = 3$	$[\ 1\ \frac{15}{13}\ ]$ 125253 (2,4,8) $d_2 = 2$, $d_3 = 3$	$[\ 1\ \frac{37}{23}\ ]$ 125253 (2,1,2) $d_2 = 2$, $d_3 = 3$	$[\ 1\ \frac{67}{51}\ ]$ 125253 (3,17,49) $d_2 = 3$, $d_3 = 3$

Table 11.16: Best puncturing patterns for given high-rate parallel turbo codes

$n-1$	4 states	8 states	16 states	32 states
2	$[\ 1\ \frac{5}{7}\ ]$ 7 (3,1,3) $d_2 = 5$, $d_3 = 3$	$[\ 1\ \frac{15}{13}\ ]$ 7 (4,3,10) $d_2 = 7$, $d_3 = 4$	$[\ 1\ \frac{27}{31}\ ]$ 7 (4,1,5) $d_2 = 11$, $d_3 = 5$	$[\ 1\ \frac{67}{51}\ ]$ 7 (5,2,7) $d_2 = 19$, $d_3 = 6$
3	$[\ 1\ \frac{5}{7}\ ]$ 27 (2,1,4) $d_2 = 4$, $d_3 = 3$	$[\ 1\ \frac{15}{13}\ ]$ 27 (3,2,9) $d_2 = 6$, $d_3 = 3$	$[\ 1\ \frac{27}{31}\ ]$ 27 (3,1,5) $d_2 = 10$, $d_3 = 4$	$[\ 1\ \frac{67}{51}\ ]$ 65 (4,7,51) $d_2 = 18$, $d_3 = 5$
4	$[\ 1\ \frac{5}{7}\ ]$ 67 (2,4,13) $d_2 = 4$, $d_3 = 3$	$[\ 1\ \frac{15}{13}\ ]$ 127 (3,9,52) $d_2 = 6$, $d_3 = 3$	$[\ 1\ \frac{27}{31}\ ]$ 351 (4,16,176) $d_2 = 10$, $d_3 = 4$	$[\ 1\ \frac{67}{51}\ ]$ 325 (4,22,191) $d_2 = 18$, $d_3 = 5$
5	$[\ 1\ \frac{5}{7}\ ]$ 527 (2,6,26) $d_2 = 4$, $d_3 = 2$	$[\ 1\ \frac{15}{13}\ ]$ 527 (2,1,6) $d_2 = 6$, $d_3 = 3$	$[\ 1\ \frac{27}{31}\ ]$ 635 (3,8,43) $d_2 = 10$, $d_3 = 4$	$[\ 1\ \frac{67}{51}\ ]$ 1525 (3,3,21) $d_2 = 18$, $d_3 = 5$
6	$[\ 1\ \frac{5}{7}\ ]$ 3525 (2,84,2693) $d_2 = 4$, $d_3 = 2$	$[\ 1\ \frac{17}{13}\ ]$ 2527 (2,4,20) $d_2 = 6$, $d_3 = 4$	$[\ 1\ \frac{37}{23}\ ]$ 6525 (2,4,22) $d_2 = 10$, $d_3 = 5$	$[\ 1\ \frac{67}{51}\ ]$ 6525 (3,4,34) $d_2 = 18$, $d_3 = 5$
7	$[\ 1\ \frac{5}{7}\ ]$ 12527 (2,15,74) $d_2 = 4$, $d_3 = 2$	$[\ 1\ \frac{17}{13}\ ]$ 12527 (2,7,42) $d_2 = 6$, $d_3 = 4$	$[\ 1\ \frac{33}{23}\ ]$ 32525 (2,6,51) $d_2 = 10$, $d_3 = 4$	$[\ 1\ \frac{67}{51}\ ]$ 32525 (3,14,135) $d_2 = 18$, $d_3 = 5$
8	$[\ 1\ \frac{5}{7}\ ]$ 72525 (2,153,5216) $d_2 = 4$, $d_3 = 2$	$[\ 1\ \frac{15}{13}\ ]$ 52527 (2,4,32) $d_2 = 6$, $d_3 = 3$	$[\ 1\ \frac{33}{23}\ ]$ 72525 (2,6,42) $d_2 = 10$, $d_3 = 4$	$[\ 1\ \frac{67}{51}\ ]$ 152525 (3,23,299) $d_2 = 18$, $d_3 = 5$

Table 11.17: Best puncturing patterns for given high-rate serial (inner code) turbo codes

$n-1$	4 states	8 states	16 states	32 states
2	$[\ 1\ \frac{7}{5}\ ]$ 15 (3,1,2) $d_2 = 3$, $d_3 = \infty$	$[\ 1\ \frac{15}{13}\ ]$ 13 (4,3,10) $d_2 = 5$, $d_3 = 4$	$[\ 1\ \frac{33}{31}\ ]$ 13 (5,7,25) $d_2 = 6$, $d_3 = 5$	$[\ 1\ \frac{71}{53}\ ]$ 13 (6,15,60) $d_2 = 6$, $d_3 = \infty$
3	$[\ 1\ \frac{5}{7}\ ]$ 56 (3,4,10) $d_2 = 2$, $d_3 = 3$	$[\ 1\ \frac{15}{17}\ ]$ 33 (4,29,126) $d_2 = 4$, $d_3 = \infty$	$[\ 1\ \frac{23}{35}\ ]$ 17 (4,29,150) $d_2 = 4$, $d_3 = \infty$	$[\ 1\ \frac{71}{53}\ ]$ 36 (4,1,4) $d_2 = 8$, $d_3 = \infty$
4	$[\ 1\ \frac{5}{7}\ ]$ 253 (2,1,2) $d_2 = 2$, $d_3 = 3$	$[\ 1\ \frac{15}{17}\ ]$ 136 (3,5,16) $d_2 = 3$, $d_3 = \infty$	$[\ 1\ \frac{27}{31}\ ]$ 351 (4,16,176) $d_2 = 10$, $d_3 = 4$	$[\ 1\ \frac{71}{53}\ ]$ 133 (4,28,192) $d_2 = 8$, $d_3 = \infty$
5	$[\ 1\ \frac{5}{7}\ ]$ 1253 (2,2,4) $d_2 = 2$, $d_3 = 3$	$[\ 1\ \frac{17}{13}\ ]$ 1253 (3,15,40) $d_2 = 3$, $d_3 = 3$	$[\ 1\ \frac{33}{31}\ ]$ 653 (4,98,436) $d_2 = 4$, $d_3 = 4$	$[\ 1\ \frac{71}{53}\ ]$ 1272 (4,108,406) $d_2 = 4$, $d_3 = \infty$
6	$[\ 1\ \frac{7}{5}\ ]$ 3247 (2,4,8) $d_2 = 2$, $d_3 = \infty$	$[\ 1\ \frac{17}{13}\ ]$ 5253 (2,1,2) $d_2 = 2$, $d_3 = 3$	$[\ 1\ \frac{27}{31}\ ]$ 3352 (3,7,24) $d_2 = 4$, $d_3 = 3$	$[\ 1\ \frac{71}{53}\ ]$ 5253 (3,3,6) $d_2 = 3$, $d_3 = \infty$
7	$[\ 1\ \frac{7}{5}\ ]$ 15247 (2,6,12) $d_2 = 2$, $d_3 = \infty$	$[\ 1\ \frac{15}{17}\ ]$ 15652 (2,2,4) $d_2 = 2$, $d_3 = \infty$	$[\ 1\ \frac{23}{35}\ ]$ 13632 (3,13,52) $d_2 = 3$, $d_3 = \infty$	$[\ 1\ \frac{71}{53}\ ]$ 13172 (3,4,18) $d_2 = 5$, $d_3 = \infty$
8	$[\ 1\ \frac{7}{5}\ ]$ 65247 (2,9,18) $d_2 = 2$, $d_3 = \infty$	$[\ 1\ \frac{15}{17}\ ]$ 65256 (2,3,6) $d_2 = 2$, $d_3 = \infty$	$[\ 1\ \frac{33}{31}\ ]$ 123255 (3,21,75) $d_2 = 4$, $d_3 = 3$	$[\ 1\ \frac{67}{51}\ ]$ 124672 (3,11,36) $d_2 = 5$, $d_3 = 3$

Table 11.18: Best puncturing patterns for given high-rate serial (outer code) turbo codes

code	m_1	m_2	m_3	m_4
parity1	1	-	1	-
parity2	-	1	-	1

Table 11.19: $b = 1$ with 4QAM, or $\bar{b} = .5$

11.4 Turbo Codes for Higher-Level Constellations

The turbo codes of Section 11.3 can be used effectively with any constellation by simply using log likelihood ratios to compute the initial information from the constellation and otherwise executing iterative decoding between 2 (or more) instances of the code. However, while the codes are very good and have high gain, the gain for multi-level constellations may not be consistent as the number of points in the constellation M grows. Thus, researchers have investigated some codes to be used with S-random interleavers that will provide a relatively constant gap for $\bar{b} \geq 0.5$. This section presents two of those codes, an 8-state code found by Sadjadpour, Sonalkar, and Calderbank (SSC) when working at ATT on DMT DSL systems, where the constant gain is useful for loading algorithm implementation and a 16-state code found by Pons, Sorbara, and Duvaut (PSD), who were also working on DSL. The SSC code has a coding gain (with interleaver of a about 2000 bits) of 6.5 dB and thus a gap of 3 dB at 10^{-7} probability of error. The 16-state PSD has about 6.8 dB coding gain alone with a similar length interleaver and was observed when used with an outer hard-decoding that was not involved in interative decoding to have a gain of about 7.1 dB, and thus a gap of only 2.4 dB. Given that shaping gain would contribute another 1.53 dB, these codes are essentially within a 1-2 dB of capacity or absolute best performance for multi-level transmission systems.

11.4.1 The 8-state SSC Code

The code has the already identified rate-1/2 encoder $G(D) = \left[1 \ \frac{17}{13}\right]$, which thus confirms that this binary convolutional code is a good one to use. This code is used twice. The real innovations of SSC are in the puncturing patterns and actually a few tenths of dB in a clever augmentation to the S-random interleaver search. There are actually two interleavers specified on each of the input branches to the use of this code. One of the interleavers acts on ONLY the systematic bits before they are transmitted, while the other rearranges those bits on the input to the second parity-computing instance of the same code.

Those extra steps over the algorithm in Figure 11.10 are compute the autocorrelation matrix between extrinsic LLR's and information bits, trying to drive these on average to as low of values as possible. The procedure is in the 3rd ATT paper on the web site. The second information-bit-transmitted-only interleaver is a deterministic design that is also described in the paper at the web site.

The length of the interleaverchosen for examples by SSC is $L = 2176$ bits, but the procedure for interleaver design can be repeated for many situations and data rates, thus consequently different interleaver lengths. Such an interleaver length is relatively small for the large gains evident, thus keep interleaver delay in implementations to reasonable levels of a few thousand bits (typically a few ms or less for modern data rates of a few Mbps or more).

The code uses different puncturing patterns for different $\bar{b}$. Tables 11.19-11.26 list the value of $\bar{b}$; the constellation used with coding, with each dimension mapped with gray code (no two adjacent points differ by more than 1 bit in label in a single dimension, see Figure 11.18 for an example); and up to 15 bits per 2-dimensional symbol, along with parity bits if used. A 1 indicates the parity bit is transmitted, while a - indicates the parity bit is punctured for the rate 1/2 code. Vertical lines in the table indicate symbol boundaries with respect to the information bits. Thus, the code in Table 11.19 could be viewed as 8-dimensional, while the codes in Tables 11.20, 11.21, 11.23, and 11.25 could be viewed as 4-dimensional.

code	m_1	m_2	m_3	m_4
parity1	1	-	1	-
parity2	-	1	-	1

Table 11.20: $b = 2$ with 16QAM, or $\bar{b} = 1$

code	m_1	m_2	m_3	m_4	m_5	m_6
parity1	1	-	-	-	-	-
parity2	-	-	-	1	-	-

Table 11.21: $b = 3$ with 16QAM, or $\bar{b} = 1.5$

code	m_1	m_2	m_3	m_4
parity1	1	-	-	-
parity2	-	-	1	-

Table 11.22: $b = 4$ with 64QAM, or $\bar{b} = 2$

code	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}
parity1	1	-	-	-	1	-	-	1	-	-
parity2	-	-	1	-	-	1	-	-	-	1

Table 11.23: $b = 5$ with 256QAM, or $\bar{b} = 2.5$

code	m_1	m_2	m_3	m_4	m_5	m_6
parity1	1	-	-	-	-	-
parity2	-	-	-	1	-	-

Table 11.24: $b = 6$ with 256QAM, or $\bar{b} = 3$

code	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}	m_{11}	m_{12}	m_{13}	m_{14}
parity1	1	-	-	-	-	-	1	-	-	-	1	-	-	-
parity2	-	-	-	1	-	-	-	1	-	-	-	-	-	1

Table 11.25: $b = 7$ with 1024QAM, or $\bar{b} = 3.5$

code	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8
parity1	1	-	-	-	-	-	-	-
parity2	-	-	-	-	1	-	-	-

Table 11.26: $b = 8$ with 1024QAM, or $\bar{b} = 4$

code	m_1	m_2 through m_4	m_5	m_6 through m_8	m_9
parity1	1	-	-	-	1
parity2	-	-	1	-	-

Table 11.27: $b = 9$ with 4096QAM, or $\bar{b} = 4.5$ - flip parity 1 and 2 on alternate symbols

code	m_1	m_2 through m_9	m_{10}
parity1	1	-	-
parity2	-	-	1

Table 11.28: $b = 10$ with 4096QAM, or $\bar{b} = 5$

code	m_1	m_2 through m_5	m_6	m_7 through m_{10}	m_{11}
parity1	1	-	-	-	1
parity2	-	-	1	-	-

Table 11.29: $b = 11$ with 16384QAM, or $\bar{b} = 5.5$- flip parity 1 and 2 on alternate symbols

code	m_1	m_2 through m_{11}	m_{12}
parity1	1	-	-
parity2	-	-	1

Table 11.30: $b = 12$ with 16384QAM, or $\bar{b} = 6$

code	m_1	m_2 through m_6	m_7	m_8 through m_{12}	m_{13}
parity1	1	-	-	-	1
parity2	-	-	1	-	-

Table 11.31: $b = 13$ with 65536QAM, or $\bar{b} = 6.5$ - flip parity 1 and 2 on alternate symbols

code	m_1	m_2 through m_{11}	m_{12}
parity1	1	-	-
parity2	-	-	1

Table 11.32: $b = 14$ with 655536QAM, or $\bar{b} = 7$

code	m_1	m_2 through m_7	m_8	m_9 through m_{14}	m_{15}
parity1	1	-	-	-	1
parity2	-	-	1	-	-

Table 11.33: $b = 15$ with 262144QAM, or $\bar{b} = 7.5$ - flip parity 1 and 2 on alternate symbols

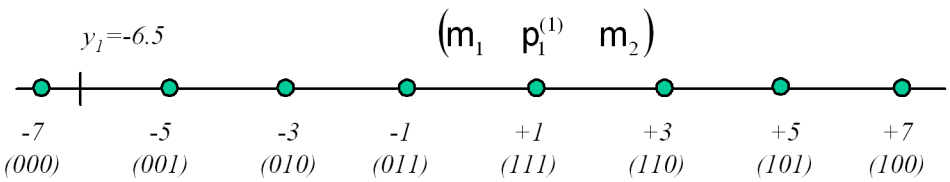

Figure 11.18: Example for feeding decoder.

Feeding the Decoder

EXAMPLE 11.4.1 (Decoder Feeding from one dimension of constellation) An example for one dimension of a 64 QAM constellation and rate 2/3 code illustrates decoding with the AT&T code. Figure 11.18 shows one dimension of the constellation, labelings, and a hypothesized received value of $y = -6.5$. The received value of -6.5 in the first dimension corresponds to the 3 bits $(m_1, p_1^{(1)}, m_2)$. The second parity bit is removed via puncturing. A trellis branch of interest in computation/input of γ_k in the APP of Section 9.3 would have the label $(m_1, p_1^{(1)}) = (1, 1)$ in the encoder. There are two constellation points that correspond to this combination, at +1 and at +3. Thus, gamma for this first code on this branch is

$$\gamma_k(1,1) = \frac{1}{\sqrt{2\pi\sigma^2}} \left(e^{-\frac{1}{2\sigma^2}(7.5)^2} + e^{-\frac{1}{2\sigma^2}(9.5)^2} \right) \tag{11.63}$$

The second γ_{k+1} affected by the same received symbol has a branch in the code trellis with label (0,1), but the second bit was punctured and not transmitted. Thus any point that has $m_2 = 0$ is a possible point for the sum and there are four such points for the first code

$$\gamma_{k+1}(0,1) = \frac{1}{\sqrt{2\pi\sigma^2}} \left(e^{-\frac{1}{2\sigma^2}(.5)^2} + e^{-\frac{1}{2\sigma^2}(3.5)^2} + e^{-\frac{1}{2\sigma^2}(9.5)^2} + e^{-\frac{1}{2\sigma^2}(13.5)^2} \right) \quad . \tag{11.64}$$

For the second code, both parity bits are punctured. Thus, $\gamma_{k+1}(0,1)$ remains the same as for the first code on the same branch, but now

$$\gamma_k(1,1)(code2) = \frac{1}{\sqrt{2\pi\sigma^2}} \left(e^{-\frac{1}{2\sigma^2}(7.5)^2} + e^{-\frac{1}{2\sigma^2}(9.5)^2} + e^{-\frac{1}{2\sigma^2}(11.5)^2} + e^{-\frac{1}{2\sigma^2}(13.5)^2} \right) \quad . \tag{11.65}$$

11.4.2 The 16-state PSD Code

This code uses an S-random interleaver, which can be augmented by the same search as for the SSC code. The generator is $G(D) = \left[1 \ \frac{35}{23}\right]$, which is again among those found in the previous section as good for turbo codes with puncturing. This code adds a few tenths of a dB in coding gain for the doubling of decoder complexity and can otherwise use the same constellations and punturing patterns as the SSC code. Most productive use would require searches of interleaving rules that corresponding to specific data rates to reduce correlation between extrinsic information and intrinsic information between decoders. In this case, the code appears to have coding gain of about 6.8 dB. An outer hard-decoder was found by PSD to add an additional .3 dB (with no additional interleaving), meaning a coding gain of 7.1 dB.

11.4.3 The Third Generation Wireless Turbo Code

Third Generation Wireless, specifically CDMA-2000 3GPP2, use a turbo code that is typically mapped into various forms of BPSK or QPSK signaling on the output bits of the encoder. Figure 11.20 illustrates the base encoder, which typically has punctured outputs. The encoder for both encoders is $G(D) = \left[1 \ \frac{1+D+D^3}{1+D^2+D^3} \ \frac{1+D+D^2+D^3}{1+D^2+D^3}\right]$, which is one of the codes in the earlier tables. The encoder outputs are

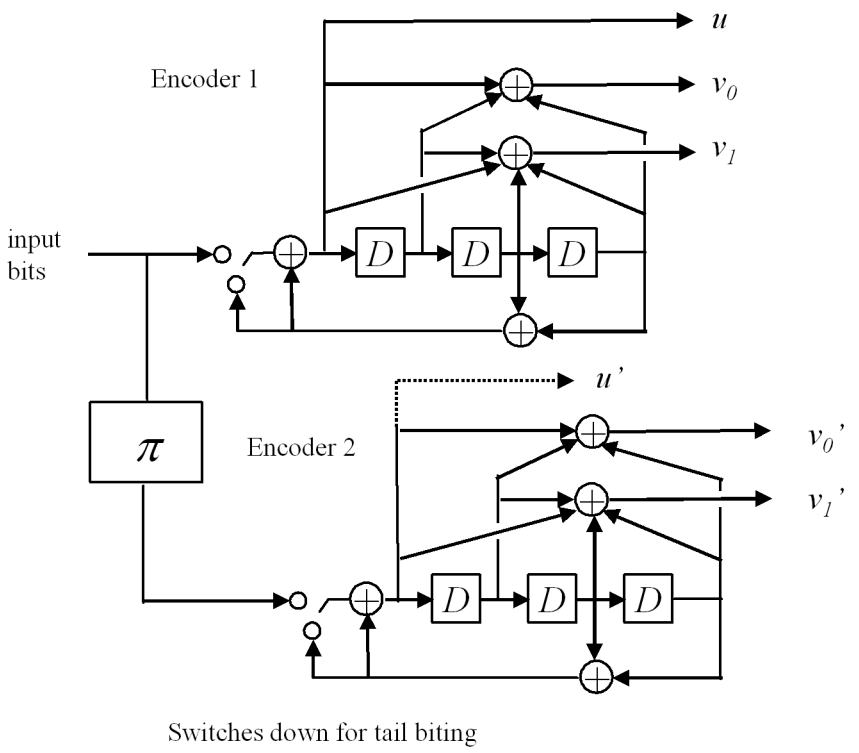

Figure 11.19: 3GPP Turbo encoder base (often punctured).

punc	1/2	1/3	1/4	1/5
u	11	11	11	11
v_0	10	11	11	11
v_1	00	00	10	11
v_0'	01	11	01	11
v_1'	00	00	11	11

Table 11.34: Puncturing patterns for various rates of the 3GPP Turbo Encoders

punctured to create a rate $\bar{b}$ encoder according to Table 11.34 over two successive input bits (or symbol periods). This code is not intended for multi-level constellations and instead is an example of a good one used at low $\bar{b}$.

Various packet lengths are used for the code with tail biting that causes the switch to go to the lower position (the delay is all zeros at the beginning of each packet) with encoder 1's output bits first transmitted for each symbol followed by encoder 2's output bits. When the switch is in the lower position the feedback bit is transmitted. Since this bit is different for the 1st encoder than for the 2nd encoder, both are transmitted and thus the puncturing patterns are changed for the tail biting. Basically $\frac{6}{b}$ bits from the tail are transmitted after puncturing of the tail, or $\frac{3}{b}$ from each of the encoder outputs. The puncturing patterns for the tail biting appear in Table 11.35. The entries 2 and 3 mean to repeat the bit twice or three times respectively.

The interleaver for the 3GPP code approximates a random interleaver, but with a simplified implementation shown in Figure **??**. This interleaver is characterized by an integer $n = 4, 5, ...10$ and the packet sizes (without including tail-biting bits) in Table **??**. A counter is initialized to zero at the beginning of the packet and its output is the input to the circuit in Figure **??**. When a successful address (the packet length is listed as N_{turbo} in Figure **??**), then the counter is incremented. This process is continued until all N_{turbo} input addresses have an interleave output address. The de-interleave process basically

punc	1/2		1/3		1/4		1/5	
u	111	000	222	000	222	000	333	000
v_0	111	000	111	000	111	000	111	000
v_1	000	000	000	000	111	000	111	000
u'	000	111	000	222	000	222	000	333
v'_0	000	111	000	111	000	1111	000	111
v'_1	000	000	000	000	000	111	000	111

Table 11.35: Puncturing patterns for tail biting of the 3GPP Turbo Encoders

n	packet length
378	4
570	5
762	5
1146	6
1530	6
2298	7
3066	7
4602	8
6138	8
9210	9
12282	9
20730	10

Table 11.36: Packet lengths and n for 3GPP turbo code.

reverses the diagram (and requires the solution of the least-significant-bit division problem, which is essentially a set of linear binary equations). The interleaver can also be stored in a large look-up table, but this 3GPP implementation is much simpler. The small look-up table is summarized in Table 11.21.

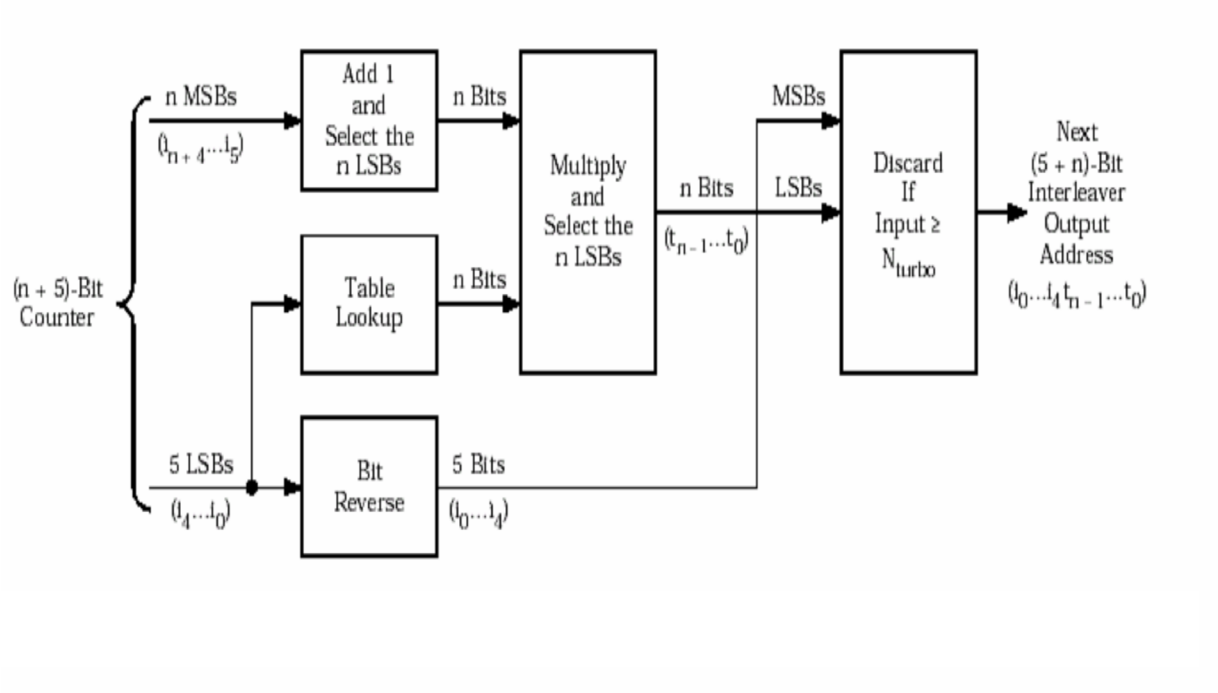

Figure 11.20: 3GPP Turbo encoder interleaver circuit.

Table Index	n = 4 Entries	n = 5 Entries	n = 6 Entries	n = 7 Entries	n = 8 Entries	n = 9 Entries	n = 10 Entries
0	5	27	3	15	3	13	1
1	15	3	27	127	1	335	349
2	5	1	15	89	5	87	303
3	15	15	13	1	83	15	721
4	1	13	29	31	19	15	973
5	9	17	5	15	179	1	703
6	9	23	1	61	19	333	761
7	15	13	31	47	99	11	327
8	13	9	3	127	23	13	453
9	15	3	9	17	1	1	95
10	7	15	15	119	3	121	241
11	11	3	31	15	13	155	187
12	15	13	17	57	13	1	497
13	3	1	5	123	3	175	909
14	15	13	39	95	17	421	769
15	5	29	1	5	1	5	349
16	13	21	19	85	63	509	71
17	15	19	27	17	131	215	557
18	9	1	15	55	17	47	197
19	3	3	13	57	131	425	499
20	1	29	45	15	211	295	409
21	3	17	5	41	173	229	259
22	15	25	33	93	231	427	335
23	1	29	15	87	171	83	253
24	13	9	13	63	23	409	677
25	1	13	9	15	147	387	717
26	9	23	15	13	243	193	313
27	15	13	31	15	213	57	757
28	11	13	17	81	189	501	189
29	3	1	5	57	51	313	15
30	15	13	15	31	15	489	75
31	5	13	33	69	67	391	163

Figure 11.21: 3GPP Turbo interleaver look-up table.

(t_c, t_r)	b	deviation from capacity
(3,6)	.5	1.1 dB
(4,8)	.5	1.6 dB
(5,10)	.5	2.0 dB
(3,5)	.4	1.3 dB
(4,6)	1/3	1.4 dB

Table 11.37: Regular Low-Density Parity Code Average Performance.

11.5 Low-Density Parity Check Codes

Low-Density Parity Check (LDPC) codes were first studied by Gallager of MIT in the early 1960's. They essentially are an implementation of the concept of random coding, which is known from Chapter 8 to lead to code designs that can approach capacity. These block codes have very long block length N and a parity matrix $H(D) = H(0) = H$ that is essentially chosen randomly to have a number of 1's that grows linearly with block length (rather than as the square of the block length as might be expected if 0's and 1's were selected with equal probability for the H matrix entries). Also, from the expertise gained in Section 9.6, "cycles of 4" are avoided so that iterative decoding using the constraint-based structure of Section 9.6 can be used effectively.

Definition 11.5.1 (cycle of 4) *A* **cycle of 4** *occurs in a code when for any 1's in the (i, j) and (i, k) positions in any row i of H, there is at least one other row $i' \neq i$ that also has 1's in the j^{th} and k^{th} positions.*

Definition 11.5.2 (Regular Parity Matrix) *A* **regular code** *with a regular parity matrix has exactly t_r 1's in every row and exactly t_c 1's in every column.*

The $(n - k) \times n$ parity matrix H is chosen in regular LDPC codes to have exactly t_c 1's in each column and exactly t_r 1's in each row, from which one notes

$$(n - k) \cdot t_r = n \cdot t_c \tag{11.66}$$

or equivalently

$$\bar{b} = 1 - \frac{t_c}{t_r} \quad . \tag{11.67}$$

Essentially, if uniformly distributed integers from 1 to n were selected, t_r at a time, they would represent the positions of the 1's in a row of H. Successive rows could be generated by selecting successively groups of t_r integers. If a row is obtained that is either linearly dependent on previous rows, or forms a 4-cycle (which one can determine is simply a rectangle of 4 $1's$ in the corners somewhere in the stacked rows of H generated), then the row is discarded and the process continued. While large n is desired, to ensure a low-density of 1's as described, the parameters t_c, n, and k must be chosen to satisfy

$$\frac{(n - k)(n - k - 1)}{t_c(t_c - 1)} \geq n \quad . \tag{11.68}$$

This relation basically forces long n to be used for high-rate codes. On average, codes having such a parity matrix can have very high coding gain. For instance, Richardson and Urbanke have recently noted the average LDPC code parameters (we made adjustments to their deviation from capacity numbers to reflect capacity for an ensemble average of large-n LDPC codes rather than the restricted capacity they studied) in Table 11.37. This table assumes that the codes are used on the AWGN with $\bar{b} < 1$ – that is, transmitted signals are simple ± 1 and there is no issue of shaping gain at such low $\bar{b}$.

In a paper in IEEE Communication Letters, Chung, Forney, Richardson and Urbanke show that for the binary-input AWGN channel, a rate 1/2 code can be constructed that is within 0.0045 dB of the Shannon capacity. This code did not have a constant number of 1's per row or column (i.e., was not regular), and it appears such non-uniform distribution of 1's is necessary to get extremely close to capacity.

Clearly any specific random construction for an H matrix needs to be explicitly tested to determine its gain and parameters. However, with the results in Table 11.37 to guide design, one could generate codes with confidence that a code with approximately the gains illustrated would be found. Additional coding gain to capacity can be achieved if the inner LDPC system operates at a $\bar{P}_b$ of approximately 10^{-6} or 10^{-7} and an external hard-decoded block code (like a Reed Solomon code of rate .9 or greater) is applied, and will often gain the remaining 1-2 dB of coding gain as well as reduce the probability of error to essentially error-free levels. LDPC codes are based on good distance properties for very complicated codes and do not exhibit the error-flooring effect of turbo codes.

There are various approaches to generation of the H matrix. Gallager originally used a recursive procedure in which smaller LDPC matrices are inserted into larger H matrices. However, this textbook considers codes that appear to be well suited for multilevel transmission, as well as for low-rate transmission, for which the construction follows the random procedure.

Since the error-floor problem associated with interleaving gain in Turbo codes is not evident in good LDPC codes, LDPC codes more uniformly address the issue of increasing minimum distance and simultaneously controlling the growth in nearest neighbors at all small distances. Decoding of LDPC codes follows according to the constraint-decoding methods discussed in Chapter 9.

11.5.1 IBM Array LDPC Codes

IBM researchers E. Eleftheriou, S. Olcer, G. Cherubini, B. Marcus, and M. Blaum recently reported a construction of some LDPC codes that is easy to reproduce and also essentially performs within 1-2 dB of capacity at all $\bar{b}$. Again, the remaining 1-2 dB can be obtained by serial outer concatenation of a long-length Reed Solomon block code with appropriate longer-length interleaving than the block length of the LDPC code. Shaping gain of 1.53 dB at large $\bar{b}$ (and less at smaller $\bar{b} > 1$) is independent, see Section **??**.

The IBM LDPC codes use a Vandermonde matrix structure for H that depends on a $p \times p$ (p is a prime integer) circular shift matrix

$$\alpha = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \quad . \tag{11.69}$$

The consequent parity matrix is ALMOST

$$H = \begin{bmatrix} I & I & ... & I & I \\ I & \alpha & \alpha^2 & ... & \alpha^{t_r-1} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ I & \alpha^{\tilde{t}_c-1} & \alpha^{2(\tilde{t}_c-1)} & ... & \alpha^{(t_r-1)(\tilde{t}_c-1)} \end{bmatrix} \quad . \tag{11.70}$$

The word ALMOST us capitalized because linearly dependent rows of H (going from top to bottom) are eliminated as they occur, so while the codeword length remains $t_r \cdot p$, the number of rows $n - k < t_c \cdot p$. Table 11.38 lists how many (m) rows of the parity matrix in (11.70) are actually linearly dependent and thus removed. This number is always $m = t_c - 1$. In removing some rows, the number of ones per column reduces in some columns, so that t_c value now represents the maximum number of ones in any column. Some columns have instead $\tilde{t}_c - 1$ ones in them.

The H construction above is known to guarantee aversion of any 4-cycles. The notation $\tilde{t}_c$ is used because the deletion of linearly dependent rows actually reduces t_c in m of the columns and so the codes are not uniform (that is, t_c is not quite constant).

Longer block length typically means higher gain with this construction. Table 11.38 illustrates the codes and parameters when used for $\bar{b} \geq .5$. Both the gap to capacity when the code is used at $\bar{b} < 1$ on the AWGN and when used (as to be shown later in this section) for $\bar{b} > 1$ are enumerated, with the latter being 1.5 dB higher because LDPC codes do not address shaping gain. The gap to capacity varies only slightly with $\bar{b}$ for these codes as illustrated in Figure 11.22.

(n,k)	m	p	t_c	t_r	$\bar{b}$	Γ at 10^{-7}	high-b Γ	γ_f
(276,207)	2	23	3	12	.7572	3.8 dB	5.3 dB	4.2 dB
(529,462)	2	23	3	23	.8733	3.0 dB	4.5 dB	5.0 dB
(1369,1260)	2	37	3	37	.9204	2.3 dB	3.8 dB	5.7 dB
(2209,2024)	3	47	4	47	.9163	1.8 dB	3.3 dB	6.2 dB
(4489,4158)	4	67	5	67	.9263	1.5 dB	3.0 dB	6.5 dB
(7921,7392)	5	89	6	89	.9332	1.3 dB	2.8 dB	6.9 dB

Table 11.38: IBM Array-Code LDPC parameters.

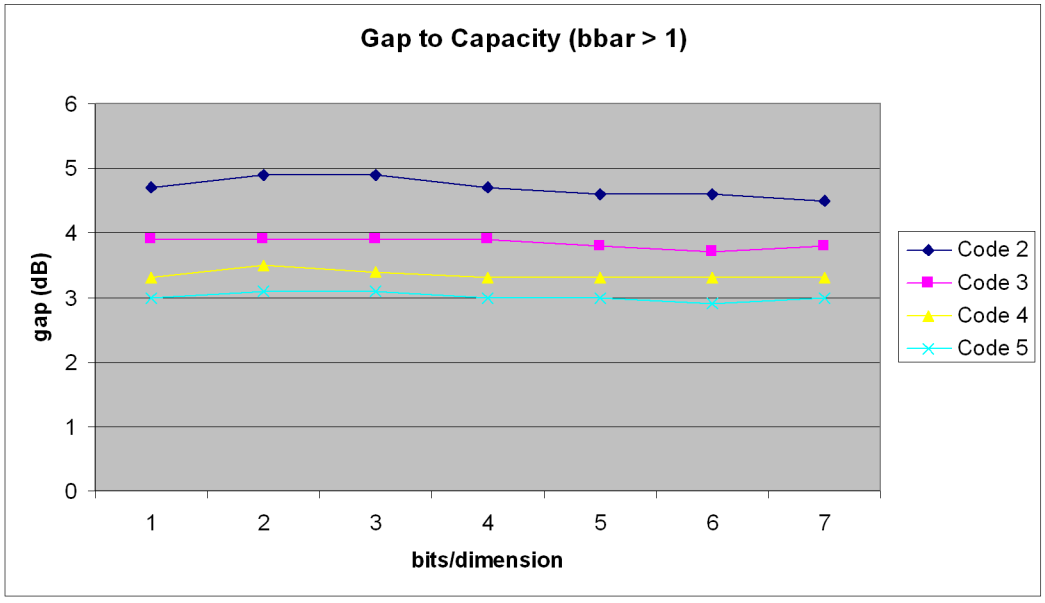

Figure 11.22: Illustration of nearly constant gap to capacity for integer $\bar{b}$.

bit label	value	subset number
(0,0,0)	-7	0
(0,0,1)	-5	1
(0,1,1)	-3	3
(0,1,0)	-1	2
(1,0,0)	+1	0
(1,0,1)	+3	1
(1,1,1)	+5	3
(1,1,0)	+7	2

LDPC codes are iteratively decoded using the parity and equality constraint processing of Section 9.6. The IBM researchers noticed that there can be a very large number of processors and this large number is not necessary for constellations with large $\bar{b} > 2$. For such large-constellation applications, the IBM LDPC code is mapped into SQ QAM constellations by directly mapping the encoder output into 1-dimensional constituent components. The modulator takes up to 4 bits at a time from the LDPC encoder output. A useful quantity $\bar{b}_o$ is defined as the number of bits per dimension at the encoder output (so $\bar{b}_o = (n/k) \cdot \bar{b}$. When $\bar{b}_o = .5$, then BPSK is transmitted using each successive LDPC encoder output bit to modulate the polarity, positive or negative (for 1 or 0 respectively). When $\bar{b}_o = 1$, two successive LDPC encoder output bits are mapped into QPSK. When $\bar{b}_o = 1.5$, 3 successive output bits map into 8 SQ QAM, and when $\bar{b}_o = 2$, 4 successive encoder output bits map into 16 QAM. For $\bar{b}_o > 2$ and an integer, only 4 output bits from the LDPC encoder are used: 2 of these bits are used to encode one of 4 subsets in a one-dimensional partitioning, and the other 2 bits are used identically for the other dimension in SQ QAM. The remaining "uncoded" bits are used to select points with the sets - points within a set have an intra-coset distance of nearly 12 dB better than uncoded and thus exceed the gain of the code. An example appears in Table 11.5.1 for the encoding of one of the dimensions of 64 QAM. The first bit or msb is thus not coded by the LDPC encoder. This procedure simplifies decoding (because the additional "uncoded" bits are determined by simple slicing in the receiver), and also reduces the redundancy in the use of the code because there are fewer redundant bits per symbol. Furthermore, the gain is not reduced.

Such a procedure does need to match data rates and redundancies. This development will assume that SQ QAM constellations (or two-dimensional constellations) are used. these constellations can always be viewed as 2 successive PAM constellations as above. The number of bits per 2D symbol is defined as

$$b_2 = \frac{R}{\text{2D symbol rate}} = \frac{q}{p} \quad ;. \tag{11.71}$$

The integers q and p may be chosen to approximately arbitrarily closely any data rate and number of bits per QAM symbol. Two situations are of interest: (1) the number of bits is relatively low so that no parallel transitions are used and (2) parallel transitions will be used because the number of bits per QAM symbol is sufficiently large. Figure 11.23 illustrates the first case: In this case, q successive uses of the code produce $n \cdot q$ output bits in response to $k \cdot q$ input bits. This also corresponds to $k \cdot p$ two-dimensional symbols since $kq/b_2 = kp$. A larger number of bits per two-dimensional symbol

$$\tilde{b}_2 = \lceil \frac{n}{k} \cdot b_2 \rceil \tag{11.72}$$

represents the redundancy introduced by the LDPC code. The quantity $\tilde{b}_2 \leq 4$ for the first case, and if $\tilde{b}_2 > 4$, then the second case is considered (as later in this subsection). For the first case, some $\tilde{l}$ 2D symbols will need to carry $\tilde{b}_2$ bits while l other 2D symbols will carry only $\tilde{b}_2 - 1$ bits. The following equations are solved with $\tilde{l}$, $\tilde{b}_2$, l all integers:

$$\tilde{l} \cdot \tilde{b}_2 + l \cdot (\tilde{b}_2 - 1) = nq \tag{11.73}$$

$$\tilde{l} + l = kp \quad . \tag{11.74}$$

An integer solution is

$$l = kp \cdot \tilde{b}_2 - nq \tag{11.75}$$

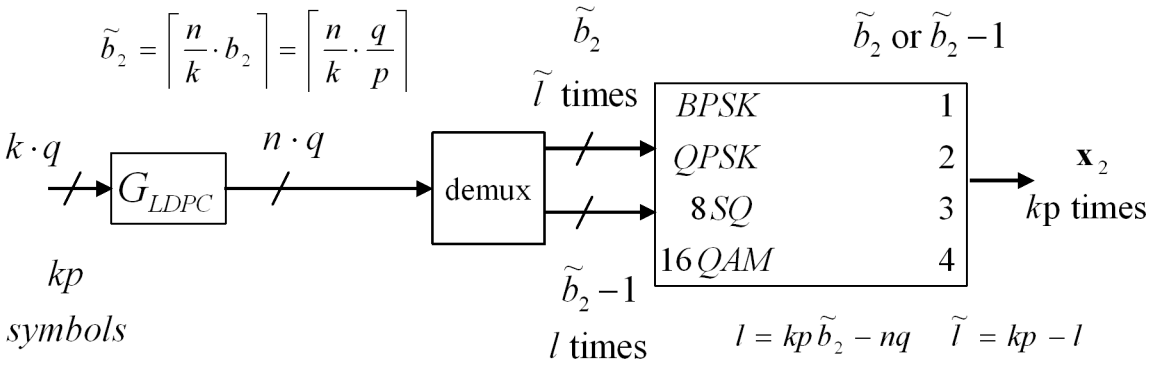

Figure 11.23: Multi-level use of LDPC code with $\tilde{b}_2 \leq 4$.

$$\tilde{l} \;=\; kp - l \quad . \tag{11.76}$$

It is possible that among the q blocks of bits or code uses that up to $q-1$ individual two-dimensional symbols have bits from two adjacent codewords. The LLR's for the bits of interest are computed as if the other bits are either (1) unknown or (2) the previous decoder's decisions for the earlier block are correct (in which latter case there is a small chance for error propagation).

The second case appears in Figure 11.24. In this case $n \cdot b_2$ input bits are grouped for 4 successive LDPC encoder uses. This number of bits is also equal to $4k + b_{parallel}$ so that, if (n,k) are known/chosen for a particular LDPC (or more generally binary block) code, then

$$b_{parallel} = n \cdot b_2 - 4 \cdot k \quad . \tag{11.77}$$

The symbols are then split between the l of n symbols that carry $b_2 + i$ bits each and the $\tilde{l}$ that carry $b_2 + i + 1$ bits each, where

$$\begin{aligned} n &= l + \tilde{l} && (11.78) \\ b_{parallel} &= l(b_2 - 4 + i) + \tilde{l}(b_2 - 3 + i) \quad . && (11.79) \end{aligned}$$

Then i is chosen as the smallest integer $i = 4, 5, 6, ...$ that produces $l \geq 0$ in satisfying

$$l = n \cdot (b_2 - 3 + i) - b_{parallel} \quad . \tag{11.80}$$

Then,

$$\tilde{l} = 4n - l \quad . \tag{11.81}$$

An example occurs in Figure 11.25 for the use of the (2209,2024) IBM code and a data rate of 20 Mbps and a two-dimensional symbol rate of 4 MHz.

Multi-carrier transmission is somewhat easier because it naturally spreads bits over a long symbol interval and many two-dimensional symbols. If B bits per N carriers are used with an (n,k) block code, then

$$\tilde{B} = \lceil B \cdot \frac{n}{k} \rceil \tag{11.82}$$

are loaded. If all carriers have less than 4 bits, then Figure 11.26 illustrates the straightforward implementation. If some carriers have a number of bits exceeding 4, then the loading algorithm is re-run with a FM (or MA) loading algorithm until

$$B' + \sum_{i:b_i > 4} (b_i - 4) = B \tag{11.83}$$

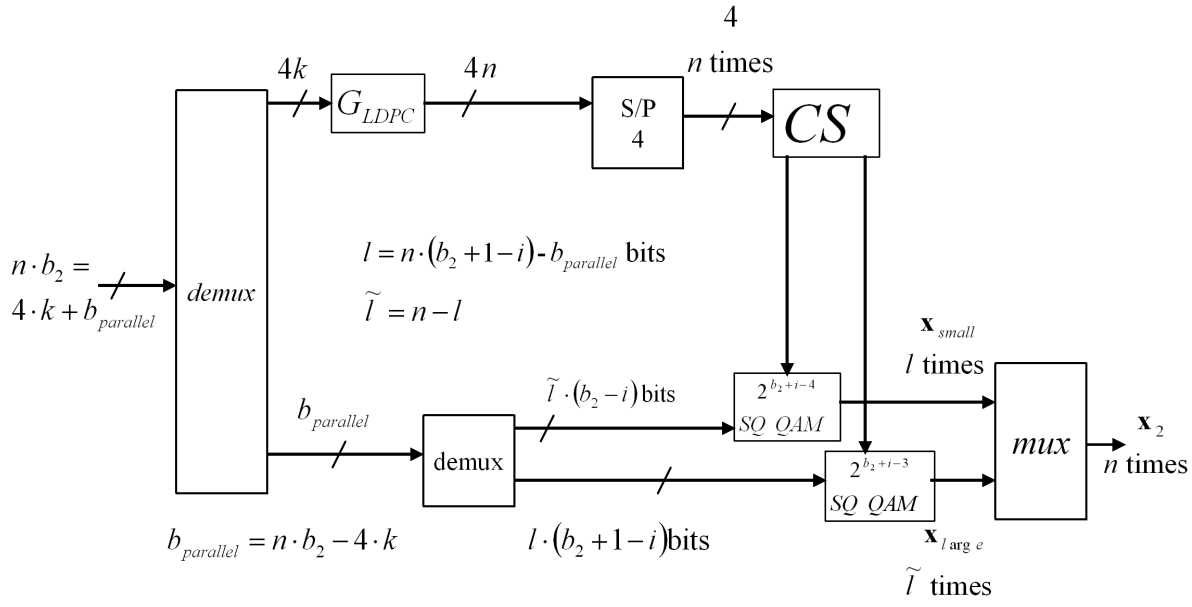

Figure 11.24: Multi-level use of LDPC code with $\tilde{b}_2 > 4$.

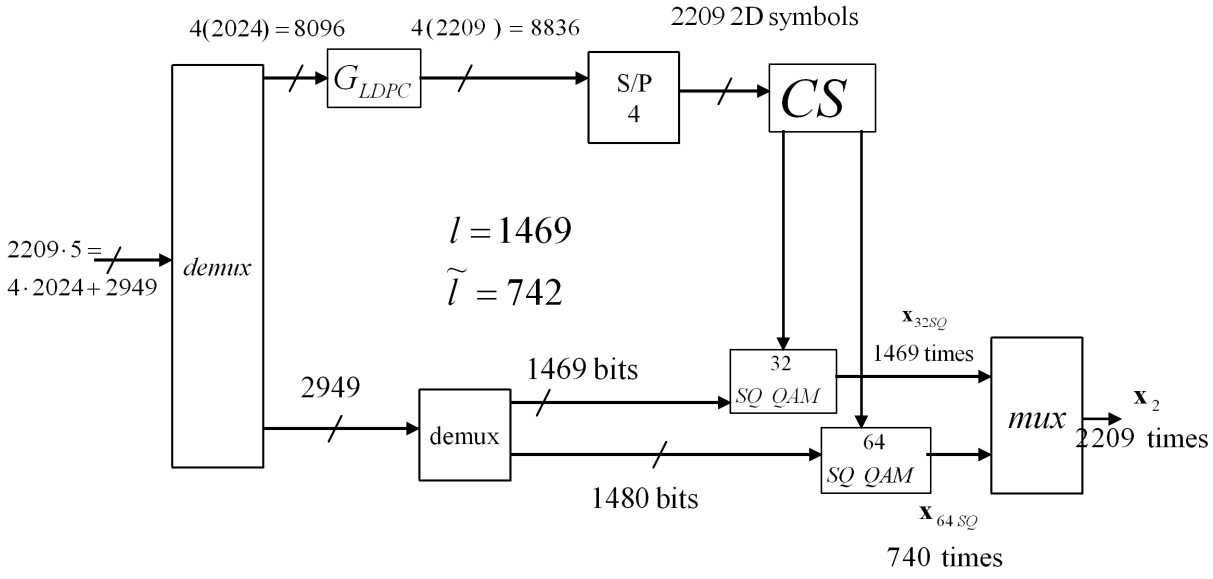

Figure 11.25: Specific 20 Mbps example of LDPC code with $b_2 = 5$.

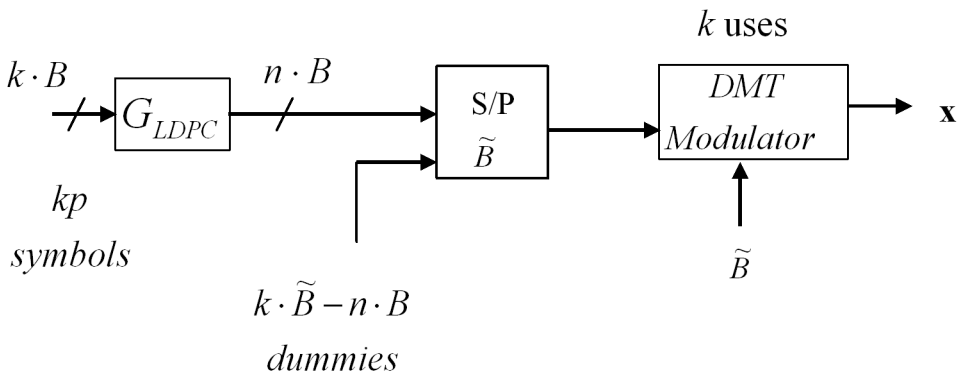

Figure 11.26: Use of LDPC with multi-carrier loading.

where B' solves

$$\lceil \frac{n}{k} \cdot B \rceil = 4 \cdot N_{>4} \sum_{i:b_i \leq 4} b_i \quad . \tag{11.84}$$

The number of parallel bits is

$$b_{parallel} = \sum_{i:b_i > 4} (b_i - 4) \quad . \tag{11.85}$$

11.5.2 Feeding the Decoder

To obtain the initial intrinsic probability from an AWGN received channel output value, the mapping allows each bit's value of the encoder output to be identified with one or more points in the constellation. This set of points is called either $X_{v_m=1}$ or $X_{v_m=0}$ The probability is obtained by the following sum

$$p(v_k = 1) = \sum_{x \in X_{v_k=1}} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(y-x)^2} \cdot p_x \quad . \tag{11.86}$$

A similar sum is computed for $v_k = 0$. Then, LDPC decoding is initialized with the prior given by

$$\text{LLR}_k = \ln \frac{p(v_k = 1)}{p(v_k = 0)} \quad . \tag{11.87}$$

11.5.3 Lee's LDPC Routines

2006 EE379B student Chien-Hsin Lee has graciously provide the software routines to construct various LDPC codes, both parity and systematic generator matrices as well as an encoder and decoder function. The matlab software is at the web site:

To generate the LDPC H matrix for any IBM code, the function get_h_matrix is useful. This function called by H = get_h_matrix(prime,tr,tc,m) has 4 inputs

1. prime = the prime number
2. tr = the number of ones per row
3. tc = the maximum number of ones per column
4. m = the number of dependent rows removed (from Tables earlier $m = t_c - 1$)

The sole output is the parity matrix.

A second routine provides the systematic encoder and is ¿¿ [H G] = systematic(H). The inputs and outputs are obvious.

A third routine provides the encoder output and also the AWGN channel outputs when noise of a specified SNR is added: [rx_bit, coded_bit, message_bit] = encoder(SNR_dB,G,random,message_bit). The inputs and outputs are again obvious.

And, finally a decoder routine specified by [decoded_bits,llr_bits, iter] = ldpc_decoder_spa(H,bits,max_iter,var,fast). var is the variance of the noise, and fast=1 stops the decoder if the parity check equations are all satisfied, while fast=0 means execute max_iter iterations of decoding.

The program listings appear to end this subsection:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% function [H_no_dep H] = get_h_matrix(p,rw,cw,first_1_start);
%% Generate LDPC H Matrix Uses IBM's Method As Per Cioffi's Class Note
%% Example: to Generate (529,462) code, p=23, rw=23, cw=3, first_1=2
%%          H = get_h_matrix(23,23,3,2),
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Definition of input variables
%% p      :  Prime number of the size of base matrix of size p-by-p
%% rw     :  Row weight equals to number of base matrix per row, eq to K
%% cw     :  Column weight equals to number of base matrix per column,eq to J
%% first_1:  Set to 2 per IBM's Method right shift 1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Definition of output variables
%% H_no_dep  : the parity check matrix with no dependent rows
%% H         : without removing the dependent rows
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% To Do  :
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% EE379B, Chien-Hsin Lee, 06/2006
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [H_no_dep] = get_h_matrix(p,rw,cw,first_1_start);
% generate base matrix
a = eye(p,p);
a = [a(first_1_start:p,:);a(1:first_1_start-1,:)];

% generate H matrix
H = [];
for row = 1:cw
    current_row = [];
    for cl = 1:rw
        current_row = [current_row,a^((row-1)*(cl-1))];
    end
    H = [H;current_row];
end

% remove dependent rows
% this need to be update with mod2 operation
[Q R] = qr(H);
H_no_dep = [];
for i = 1:p*cw
    if( abs(R(i,i)) > 1E-9)
        H_no_dep = [H_no_dep;H(i,:)];
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% function [H_sys, G_sys] = get_g_matrix(H);
```

```
%% This routine remove the dependent row from original H matrix.
%% Find the systematic G matrix and the H matrix work with this G matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Definition of input variables
%% H      : parity check matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Definition of output variables
%% H      : H to work with systematic G matrix
%% G      : Systematic G matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% To Do  :
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% EE379B, 06/2006, Chien-Hsin Lee
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [H_final, G_sys] = systematic(H);
[row col] = size(H);
H_in      = H;
offset    = 0;
r = 1;  c = 1;
num_dep_row   = 0;
%% Gaussion Elimination PART1
%% This find the leading 1 in each column
%% and eliminate the 1s in the row below
for i = 1:row
    row_order(i) = i;
end
while( r <= row && c <= col)
    row_lst = find(H(r:row,c));
    if( isempty(row_lst))
        c = c + 1;
    else
        if( length(row_lst) > 1)
            for i = 2:length(row_lst)
                H(row_lst(i)+r-1,:) = mod(H(row_lst(1)+r-1,:) + H(row_lst(i)+r-1,:),2);
            end
        end
        H_temp                   = H;
        H_temp(r,:)              = H(row_lst(1)+r-1,:);
        H_temp(row_lst(1)+r-1,:) = H(r,:);
        H                        = H_temp;
        row_temp                 = row_order;
        row_temp(r)              = row_order(row_lst(1)+r-1);
        row_temp(row_lst(1)+r-1) = row_order(r);
        row_order                = row_temp;
        r = r + 1;
        c = c + 1;
    end
end
r = row;
while( H(r,:) == 0 & r ~= 0)
    H(r,:) = [];
    r = r-1;
end
num_dep_row = row-r;
```

```
%flg = sprintf('Report: %d dependent rows are removed;\n  Old matrix is %d by %d;\n  New matrix is %
%disp(flg);
row = r;

%% Column Permutation To Put Diaginal In All 1's
for c = 1:col
    column(c) = c;
end
r = 1; c = 1;
while( r <= row && c <= col)
    if( H(r,c) ~= 1)
        H(:,c:col) = [H(:,c+1:col),H(:,c)];
        column(c:col) = [column(c+1:col),column(c)];
        r = r-1; c= c-1;
    end
    r = r+1; c=c+1;
end

%% Gaussian Elimination Part II To Get The Systematic H
for i = 0:row-1
    %% find backward leading cols and eliminate the upper trangle 1s
    col_lst = find(H(:,row-i));
    if( length(col_lst) > 1)
        for j = 1:length(col_lst)-1
            H(col_lst(j),:)=mod(H(col_lst(j),:)+H(row-i,:),2);
        end
    end
end

%% Get G And H Matrix
G_sys = [H(:,row+1:col)',eye(col-row)];
H_no_dep = [];
if(num_dep_row > 0)
    x = row_order(row+1:row+num_dep_row);
    x = sort(x);
    j = 1;
for i = 1:row+num_dep_row
        if(i~=x(j))
            H_no_dep = [H_no_dep;H_in(i,:)];
        else
            j=j+1;
        end
    end
else
    H_no_dep = H_in;
end

for i = 1:col
    H_final(:,i)=H_no_dep(:,column(i));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% function [encoded_bit message_bit] = encoder(SNR,G,random,message_bit)
%% LDPC Encoder
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Definition of input variables
%% SNR_dB      : Receiver SNR in dB
%% G           : Generator matrix
%% random      : encoder generate random bit streams
%% message_bit : message_bit to be encoded.
%%             : if random is 1, message_bit is ignored
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Definition of output variables
%% rx_bit      : coded_bit + AWGN
%% coded_bit   : encoded bit
%% message_bit : message bit to be encoded
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% To Do  :
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% EE379B, 06/2006, Chien-Hsin Lee
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [rx_bit, coded_bit, message_bit] = encoder(SNR_dB,G,random,message_bit)

[k,n]      = size(G);
if(random == 1)
    message_bit = round(rand(1,k));
end

noise        = randn(1,n)/sqrt(10^(SNR_dB/10));
coded_bit    = mod(message_bit*G,2);
modulated_bit= coded_bit*2-1;
rx_bit       = modulated_bit + noise; %% sending -1 only

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% function [decoded_bits llr_bits iter]=ldpc_decoder_spa(H,bits,max_iter,var,fast)
%% SPA LDPC Decoder
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Definition of input variables
%% H        : parity check matrix
%% bits     : Received bits
%% max_iter : maximun number of iteration
%% var      : variace of the noise
%% fast     : 1: stop at good parity, 0: run till maximum iteration.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Definition of output variables
%% decoded_bit : decoded bits
%% llr_bit     : llr of each input bits
%% iter        : number of iteration used
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% To Do  :
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% EE379B, 06/2006, Chien-Hsin Lee
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [decoded_bits,llr_bits, iter ] = ldpc_decoder_spa(H,bits,max_iter,var,fast)
[row col] = size(H);
iter      = 0;
pass      = 0;
```

```
llr_int   = zeros(1,col);    % intrinsic
llr_b2c   = zeros(row,col);  % bit node to check node
llr_c2b   = zeros(row,col);  % check node to bit node
tr        = (-1).^mod(sum(H')',2);

%% intrinsic bit probobility
for i = 1:col
    llr_int(i) = ((bits(i)+1)^2 - (bits(i)-1)^2)/2/var;
end
llr_bits        = llr_int;
decoded_bits(i) = (sign(llr_bits(i))+1)/2;

while(iter < max_iter && (pass ~= 1 || fast == 0) )
iter = iter + 1;
%% update bit note to check node LLR
for j = 1:col
        row_ptr  = find(H(:,j));
        for i = 1:length(row_ptr)
            llr_temp =llr_int(j);
            for k = 1:length(row_ptr)
                if( k ~= i)
                    llr_temp = llr_temp + llr_c2b(row_ptr(k),j);
                end
            end
            llr_b2c(row_ptr(i),j) = llr_temp;
        end
end

%% update probibilty of check equation
for i = 1:row
        col_ptr = find(H(i,:));
        for j = 1:length(col_ptr);
            X = 1;
            for k = 1:length(col_ptr);
                if( k ~= j)
                    X = X*tanh(llr_b2c(i,col_ptr(k))/2);
                end
            end
            llr_c2b(i,col_ptr(j)) = tr(i)*2*atanh(X);
        end
end

    %% check result
    for i=1:col
        llr_bits(i)     = llr_int(i)+sum(llr_c2b(:,i));
        decoded_bits(i) = (sign(llr_bits(i))+1)/2;
    end
    pass    = 1-sum(mod(decoded_bits*H',2));
end %% while()
```

Exercises - Chapter 11

11.1 *Interleaver*

Assume a generalized triangular interleaver throughout this problem:

a. (3 pts) Find $G(D)$ and $G^{-1}(D)$ for a generalized triangular interleaver with $J = 4$ and $K = 7$.

b. (1 pt) compute delay (end to end)

c. (1 pt) How much does the gain (for bursty error) of hard convolutional code appear multiplied if error bursts are infrequent.

11.2 *Serial Code Concatenation*

Assume two convolutional codes are concatenated serially with rates $\bar{b}_1$ for the inner code and $\bar{b}_2$ for the outer code.

a. (1 pts) Find the overall $\bar{b}$ for the concatenated system.

b. (1 pt) Suppose the inner code is a trellis code and repeat part a.

c. (2 pts) How does interleaving affect the rates of the two systems in parts a and b?

d. (1 pt) Suppose the parity bits added by a systematic encoder in part a are not re-encoded by the outer code - then what is the rate of the concatenated system?

11.3 *Convolutional Interleaving - Final 2003 (24 pts)*

A generalized triangular interleaver designed for bytes (symbols) has period $L = 7$ and depth $J = 5$.

a. Find the Generator Matrix for the interleaver. (2 pts)

b. Find the inverse generator matrix for the de-interleaver. (2 pts)

c. Draw the interleaver and de-interleaver and show use of byte-wide registers as D and provide the clock rate at which these registers pass information. A cascade of m D registers can be abbreviated D^m. How many bytes of memory are used? (2 pts)

d. What is the delay through the interleaver and de-interleaver. (1 pt)

e. Assuming the byte-level code associated with this interleaver corrects dfree bytes in error - what is the number of bytes that can be corrected from burst disturbances with the interleaver (if the burst is 35 samples or less). (1 pt)

11.4 *Interleaving and Wireless – 10 pts*

A wireless channel uses a 16 QAM modulator to transmit data. The symbol clock is 4 MHz. In addition to the channel being an AWGN, there is a gain on each channel that can vary with symbol period. When in a nominal state, the channel has SNR=18 dB. In a fading state, the SNR is either 21 dB with probability 0.999 or -9 dB with probability .001. Each of the gains is independent of the others.

a. What is the data rate? (1 pt)

b. What is the probability of symbol error in the nominal state? (1 pt)

c. What is the probability of a symbol error in the fading state? (2 pts)

d. Suppose that a 50and that the channel is always in the fading state. Use the best 4-state $d = 5$ convolutional code as an outer code with hard decoding and design an interleaving scheme of minimal end-to-end delay for this code that ensures that the probability of symbol error that is less than 10^{-6}. (6 pts)

11.5 *Zero Stuffing and block iterative decoding*

Two convolutional codes are concatenated in parallel with rates $\bar{b}_1$ for the inner code and $\bar{b}_2$ for the outer code. The first code has constraint length ν_1 and the second code has ν_2. A few bits are placed at the end of a length-L packet to force the first encoder's state to zero. For parts c through f, assume $\bar{b}_1 = 1/n_1$ and $\bar{b}_2 = 1/n_2$, and that the two codes are systematic.

a. (2 pts) How many bits are wasted to force the first encoder's state to zero? When is this number insignificant?

b. (2 pts) Do these bits also force the second code to state zero in general? What might we do to also force this code to state zero?

c. (2 pts) Suppose instead of separately computing the likelihood or probability densities for input zero and one bits independently, could we instead propagate the log of the likelihood ratio? (the likelihood ratio is the ratio of the probability density of a "1" to a "0"). What would be the resultant eventual detection rule using some set of Likelihood ratios at some stage in decoding?

d. (3 pts) Find an iterative expression for the alternative of propagating the log likelihood ratio of part c in iterative decoding, instead of propagating the Likelihood function itself. Explicitly enumerate the values of the intrinsic log likelihood ratio for the assumption of a uniform input and transmission over a BSC with parameter p.

e. (5 pts) Find an iterative expression for the alternative of propagating the log likelihood ratio of part c in iterative decoding, instead of propagating the Likelihood function itself. Explicitly enumerate the values of the intrinsic log likelihood ratio for the assumption of a uniform input and transmission over an AWGN with noise σ^2 and bipolar transmission levels ± 1 on the code outputs.

f. (1 pt) Comment on propagation of the likelihood ratio as an alternative to propagation the distributions themselves.

11.6 *Schedules – 11 pts*

A given triangular interleaver has depth $J = 3$ and period $L = 4$.

a. What is the minimum number of memory cells? (1 pt)

b. Create a scheduling diagram similar to those in Tables 11.3 and 11.4 for this triangular interleaver? (5 pts)

c. How many different length schedules are there in your answer for part a? (1 pt)

d. Compute S and compare to the sum of the lengths of the different-length schedules. (1 pt)

e. Explain the condition in Section 11.2 that $\frac{mL}{J-1}$ should not be an integer if the number of periods is smaller than half the delay minus one. (3 pts)

11.7 *Puncturing and Generators – 10 pts*

The same rate 2/3 convolutional code is used twice in a parallel concatenation system.

a. What is code rate of the concatenated system with no puncturing? (1 pts)

b. Find a puncturing scheme to make each of the convolutional codes rate 3/4 and what is the new rate of the concatenated system? (3 pts)

c. What is the size of the generator matrix that describes the puncturing? (1 pt)

d. Show the G matrix for your scheme. (3 pts)

e. Repeat part b for a new convolutional code rate of 4/5? (2 pts)

11.8 *Puncturing and Generators – 10 pts*
The same rate 2/3 code is used twice in a serial concatenation system.

a. What is code rate of the concatenated system with no puncturing? (1 pts)

b. Find a puncturing scheme to make each of the convolutional codes rate 3/4 and what is the new rate of the concatenated system? (3 pts)

c. What is the size of the generator matrix that describes the puncturing? (1 pt)

d. Show the G matrix for your scheme. (3 pts)

e. Repeat part b for a new convolutional code rate of 4/5? (2 pts)

11.9 *Turbo-Coded Trits (Final 2001): (12 pts)*
Prof. Bo Zhoe of the University of California at Bear has again escaped trans-bay security and is loose on Sand Hill Road in Menlo Park. He has invented 3-level logic for integration on circuits, thus allowing the use of trits, which measure the data content of a constellation according to

$$t = \log_3(M) \quad . \tag{11.88}$$

The levels that can be processed are 0,1, and 2, which translated in logic into voltages -1, 0, and +1. An earlier success with 3-way trellis code partitioning has encouraged him to investigate turbo codes and what he calls LDTC (low density trinity codes). He starts with the design of a linear convolutional code based on 1 trit in and two trits out.

a. Compute the code rate $\bar{t}$ in trits per dimension. (1 pt)

b. Using up to 9 states, design a trellis that provides maximum distance between codeword sequences? (3 pts)

c. Find the coding gain in a manner analogous to binary convolutional codes. (1 pt)

d. Find a systematic $G(D)$ and corresponding "trinity" matrix for your code in part b). (2 pts)

e. The same code with systematic realization is used in two parallel-concatenated codes with a uniform random interleaver between them. Find an approximate expression for the probability of bit error assuming iterative decoding with the APP algorithm on each decoder. (Hint, you may have to investigate the error event corresponding to dmin and use some approximation to determine how likely it is to affect a second decoder) (2 pts)

f. (Zhoe also claims it is possible to use 3-level logic for the trinity matrix of any ternary linear code. Find relations for the extrinsic probability of an equality constraint and for a "trinity" constraint. Can you say something about the use of LLR here and whether it makes sense? (2 pts)

g. Do you see any advantage for the 3-level codes with respect to binary codes when used in the turbo or LDTC context? What would you do to Zhoe if you were a venture capitalist working on Sand Hill Road? (1 pt)

11.10 *High-performance Turbo Coding - 9 pts - Final 2003*
An AWGN channel uses QAM with symbol rate fixed at 1 MHz and 21 dB of SNR.

a. What is the highest data rate that can be achieved with the 16-state 4D Wei code (ok to use gap approximation here) at $\bar{P}_e = 10^{-6}$? (1 pt)

b. Using the AT&T Turbo code, repeat part a. (1 pt)

c. For the AT&T Turbo code, show the encoder with an PRBS-based approximation to a uniform random interleaver, constellation with bit labels.

d. For the receiver using the APP-based iterative-decoding (you may assume a very long block length so that essentially one block is decoded and complexity per symbol time is constant):(4 pts)

(i) How many states are in the trellises for each of the APP decoders used? (1 pt)

(ii) Assuming one multiply-and-add is an operation, how many operations are used in each of the APP decodings per symbol? (1 pt)

(iii) If iterative decoding coverges after 5 cycles (of both decoders), what is the total number of operations per second for the iterative decoding? (1 pt)

(iv) How many operations are necessary to compute input prior probabilities from the constellation (you can assume table look of a probability density evaluation or of a log function is one operation)? (1 pt)

(v) Using your answer in part d, compare the complexity in operations per second to the complexity in operations per second for the 4D 16-state Wei code. What might be done to reduce the complexity without compromising performance too much? (2 pts)

11.11 *SBS detection with and without LDPC coding on EPR4 channel: (20 pts)*

In this problem you are asked to use matlab to compare uncoded SBS detection and SBS detection with LDPC coding at the output of an EPR4 channel with AWGN. The matlab files that you need for this problem are available on the class web page.

Assume the input is binary and is equally likely. A single frame of the input sequence of length 1088 bits that you need to use for this exercise is given in the file m.mat. We would like to pass this frame through the channel EPR4channel.m 16 times and find the average BER. The function EPR4channel.m takes d, the distance between constellation points, and the frame number i ($i = 1, 2, ..., 16$) as arguments, in addition to input frame $\text{xFrame}_i(D)$, and returns output frame $\text{yFrame}_i(D)$. Assume $\mathcal{E}_b/\mathcal{N}_0$=5 dB, unless mentioned otherwise. Adjust $\mathcal{E}_x$ appropriately to get the required value of $\mathcal{E}_b/\mathcal{N}_0$.

a. (2 pts) What is the number of training symbols required to estimate the variance of the AWG noise of the channel to within 0.05 dB and with 90% confidence?

b. (2 pts) Use a training sequence of the length found in part a to estimate noise variance of the channel EPR4channel.m.

c. (1 pt) Use precoding and 2-level PAM modulation. After going through the channel, use uncoded SBS detection to detect the information bits. Find the number of bit errors in the detected sequence over the 16 transmitted frames.

d. (1 pt) Now let us add LDPC coding. Use the LDPC code given in the file ldpc_code.mat, which has a 136×1224 parity check matrix that has $t_c = 3$ ones in each column. Pass the input data frame through encoder.m to get the LDPC codeword of length $N = 1224$. This codeword will be transmitted through the channel 16 times for BER calculations. (1 pt)

e. (1 pt) Pass the codeword through a binary precoder and 2-level PAM modulator, and then go through the EPR4channel.m.

f. (1 pt) For each received symbol y_k, $k = 1, ..., N$, where N is the codeword length, calculate the channel probabilitie densities $p_{y_k/\tilde{y}_k}$ for each possible value of $\tilde{y}_k$, where yt_k is the noiseless channel output. Normalize these conditional probabilities so that they add up to 1.

g. (2 pts) Calculate the a priori probabilities $p_k(0)$ and $p_k(1)$, $k = 1, ..., N$, by adding the corresponding channel probabilities found in part (f) based on the mapping of SBS detection.

h. (1 pt) Calculate the log likelihood ratio LLR_k, $k = 1, ..., N$. To avoid dividing by zero use: $\text{LLR}_k = \log\left(\frac{\max(p_k(1),\epsilon)}{\max(p_k(0),\epsilon)}\right)$, where $\epsilon = 10^{-5}$.

i. (4 pts) Use the function ldpc_iteration, which performs a single LDPC decoding iteration, to decode the received codeword. Initialize the messages from checks to bits to all zeros, and use the a priori LLR sequence found in part h. The vector message_bit_locations in ldpc_code.mat indicates the locations of message bits in the LDPC codeword. Find the number of bit errors in the detected information sequence over the 16 transmitted frames, for the cases of 1, 3, and 10 LDPC decoding iterations.

j. (3 pts) Plot the BER versus $\mathcal{E}_b/\mathcal{N}_0$ for uncoded SBS detection and LDPC coded SBS detection with 1, 3, and 10 decoding iterations over the $\mathcal{E}_b/\mathcal{N}_0$ range 3 to 8 dB. Use a step of 0.2 dB in your final plots.

k. (2 pts) What is the LDPC coding gain with 1, 3, and 10 decoding iterations at BER=10^{-3} relative to uncoded SBS detection.

11.12 *Multilevel LDPC decoding on AWGN channel (22 pts)*

This problem uses matlab to compare the performance of uncoded transmission and symbol-by-symbol detection with the performance of an LDPC code and iterative decoding at the output of the same AWGN channel. The information rate is with 8-level PAM transmission. The matlab files that you need for this problem are available on the class web page. The uncoded 8-level PAM constellation uses the following "Gray-code" bit mapping:

m_k , m_{k+1} , m_{k+2}	000	001	011	010	110	111	101	100
constellation point:	$-7d/2$	$-5d/2$	$-3d/2$	$-d/2$	$d/2$	$3d/2$	$5d/2$	$7d/2$

Each binary input message is equally likely. A single frame of the binary input sequence of length 1089 bits for this exercise is given in the file m.mat. This exercise passes this frame through the channel AWGNchannel.m 16 times and finds the corresponding average BER. The function AWGNchannel.m takes the frame number i (i=1, 2, 16) as an argument, in addition to input frame xFrame_i(D), and returns output frame yFrame_i(D). Assume $\frac{\mathcal{E}_b}{\mathcal{N}_0} = \frac{\text{SNR}}{2b} = 16$dB, unless mentioned otherwise and adjust $\mathcal{E}_b$ appropriately to get the required value of $\frac{\mathcal{E}_b}{\mathcal{N}_0}$.

a. What is the number of training symbols required to estimate the variance of the AWG noise of the channel to within 0.075dB of its true value with 98% confidence? (2 pts)

b. Use a training sequence of the length found in part a to estimate noise variance of the channel AWGNchannel.m. (2 pts)

c. Use uncoded 8-level PAM modulation given above. Use SBS detection to detect the information bits from the output of the AWGN. Find the number of bit errors in the detected sequence over the 16 transmitted frames. (2 pts)

d. Use uncoded 8-level PAM modulation given above. Use SBS detection to detect the information bits from the output of the AWGN.

e. Find the number of bit errors in the detected sequence over the 16 transmitted frames. (2 pts)

f. Pass the codeword through the 8-level PAM modulator, and then go through the AWGNchannel.m. Calculate the loss in dB caused by the any bandwidth expansion that would be necessary to accommodate the rate loss in part d. (1 pt)

g. For each received symbol y_k, $k = 1, ..., N/3$. where N is the codeword length, calculate the channel probabilities $p_{y_k/\tilde{y}_k}$ for each possible value of $\tilde{y}_k$, where $\tilde{y}_k$ is the noiseless channel output. Normalize these conditional probabilities so that they add up to 1. (3 pts)

h. Calculate the intrinsic probabilities $p_k(0)$, $p_k(1)$, $p_{k+1}(0)$, $p_{k+1}(1)$, $p_{k+2}(0)$, $p_{k+2}(1)$, $k = 1, ..., N/3$, by adding the corresponding channel probabilities found in part f based on the symbol to bit demapping shown above. (3 pts)

i. Calculate the log likelihood ratio LLR_k, $k = 1, ..., N$. To avoid dividing by zero use:

$$\text{LLR}(k) = \log\left(\frac{\max(p_k(1), \epsilon)}{\max(p_k(0), \epsilon)}\right) \quad , \tag{11.89}$$

where $\epsilon = 10^{-5}$. (1 pt)

j. Use the function ldpc_iteration, which performs a single LDPC decoding iteration, to decode the received codeword. Initialize the LLR messages/extrinsic-probabilities from parity check nodes to equality nodes to all be zero, and use the á priori LLR sequence found in part h to initialize the decoding process. The vector message_bit_locations in ldpc_code.mat indicate the locations of message bits in the LDPC codeword. Find the number of bit errors in the detected information sequence over the 16 transmitted frames, for the cases of 1, 3, and 10 LDPC decoding iterations. (2 pts)

k. Plot the BER versus $\frac{\mathcal{E}_b}{\mathcal{N}_0}$ for uncoded SBS detection and LDPC coded SBS detection with 1, 3, and 10 decoding iterations over the $\frac{\mathcal{E}_b}{\mathcal{N}_0}$ range 14 to 21 dB. Use a step of 0.2dB in your final plots. Comment on the slight loss in part e and how it affects the result here. (3 pts)

11.13 *Combined APP and LDPC decoding on EPR4 channel - Al-Rawi's Triathalon: (24 pts)*

This problem uses matlab to compare uncoded SBS detection and combined APP soft detection with LDPC soft decoding at the output of an EPR4 channel with 2-level PAM transmission. The matlab files that are necessary for this problem are available on the class web page.

In this iterative scheme, extrinsic soft information is passed back and forth between the APP soft-channel detector and the soft LDPC decoder. Number of iterations is expressed as ($N_{channel}$, N_{ldpc}), where $N_{channel}$ is the number of passes through the APP detector and LDPC decoder. Each pass through the LDPC decoder uses N_{ldpc} decoding iterations. Extrinsic output of the APP soft-channel detector is used as à priori input to the LDPC decoder, and if $N_{channel} > 1$, the extrinsic output of the LDPC decoder is used as à priori input to the APP detector. The final output is always taken from the LDPC decoder. Notice that the APP detector uses raw probabilities and the LDPC decoder uses probabilities in LLR domain, so you need to do the necessary conversions when passing information back and forth between the two. When converting from LLR to probability, you need to truncate your LLR values so as to keep them within the range from 100 to +100, to avoid getting infinite values in matlab.

Assume the binary input is equally likely. A single frame of the binary input sequence of length 1088 bits that you need to use for this exercise is given in the file m.mat. We would like to pass this frame through the channel EPR4channel.m 16 times and find the average BER. The function EPR4channel.m takes d, the distance between constellation points, and the frame number i (i=1, 2, 16) as arguments, in addition to input frame xFrame_i(D), and returns output frame yFrame_i(D). Assume $\frac{\mathcal{E}_b}{\mathcal{N}_0} = 0.4$dB, unless mentioned otherwise. Adjust $\mathcal{E}_b$ appropriately to get the required value of $\frac{\mathcal{E}_b}{\mathcal{N}_0}$.

a. What is the number of training symbols required to estimate the variance of the AWG noise of the channel to within 0.15dB and with 99.9% confidence? (2 pts).

b. Use a training sequence of the length found in part a to estimate noise variance of the channel EPR4channel.m. (2 pts)

c. Use precoding and 2-level PAM modulation. After going through the channel, use uncoded SBS detection to detect the information bits. Find the number of bit errors in the detected sequence over the 16 transmitted frames. (1pt)

d. Plot the BER versus $\frac{\mathcal{E}_b}{\mathcal{N}_0}$ for uncoded SBS detection over the $\mathcal{E}_b$ range 0 to 8 dB. Use a step of 0.2dB. (1 pt)

e. Use the LDPC code given in the file ldpc_code.mat, which has a 136×1224 parity check matrix that has $t_c = 3$ ones in each column. Pass the input data frame through encoder.m to get the LDPC codeword of length $N = 1224$. This codeword will be transmitted through the channel 16 times for BER calculations. (1 pt)

f. Pass the codeword through a binary precoder and 2-level PAM modulator, and then go through the EPR4channel.m. What is the rate loss? How many additional bits would need to be augmentted to the end of the channel input sequence to force a return to state zero in the EPR4 trellis? Are these bits necessary here? (1 pt)

g. For each received symbol y_k, $k = 1, ..., N$, where N is the codeword length, calculate the channel probabilities $p_{y_k/\tilde{y}_k}$ for each possible value of $\tilde{y}_k$, where $\tilde{y}_k$ is the noiseless channel output. Normalize these conditional probabilities so that they sum to 1. (1 pt)

h. Use the APP algorithm for soft detection, followed by LDPC soft decoding in the iterative scheme explained above. Assume that each APP detection iteration requires $22N \cdot 2^{\nu}$ computational operations, and each LDPC decoding iteration requires $(8t_c - 1)N$ computational operations, where N is the codeword length, and 2^{ν} is the number of states in the trellis. What is the minimum complexity to get zero bit errors in the 16 received frames. (5 pts)

i. Assuming we are transmitting at 1Mb/s rate, express this complexity of part h in number of operations per second (ops/sec). (2 pts)

j. Find the number of bit errors in the detected sequence over the 16 frames for the following iterative schemes ($N_{channel}$, N_{ldpc}) = (1, 1), (1, 3), and (3, 1). (3 pts)

k. Plot the BER versus $\frac{\mathcal{E}_b}{\mathcal{N}_0}$ for combined soft APP detection and LDPC decoding with the schemes in part j over the $\mathcal{E}_b$ range 0 to 2 dB. Use a step of 0.1dB in your final plots. (3 pts)

l. What is the coding gain of iterative schemes in part j over SBS detection at $\bar{P}_b = 10^{-3}$. (2 pts)

11.14 *Are you a Jedi Knight yet? - Final 2003 – 12 pts*

You are a young Stanford graduate in the year 2134 and have just joined the Alliance for Intergalactic Liberty as a transmission engineer. An evil empire is jamming transmissions that are crucial to the Alliance survival, leaving an AWGN channel with SNR=15 dB for wireless transmission with a fixed symbol rate of $1/T = 10$ MHz. Your Alliance friends need the highest possible data rate you can provide them on this link.

a. What is the maximum data rate with the fixed symbol rate? (1 pts)

b. Design code and transmission system that achieves 90% of this highest rate at low probability of bit error. (10 pts - better designs will earn more points)

   (i) show or describe your constellation

   (ii) describe or illustrate your encoder(s) and any significant parts

   (iii) describe the receiver

c. Suppose a very large look-up table is allowed in both the transmitter and the receiver, and a delay of 100 milliseconds is not of concern and you can somehow then get 1 dB of additional shaping gain. How much higher might you transmit? (1 pt)

11.15 *Puncturing and Generators – 10 pts*

An AWGN has an SNR of 14.2 dB at some given symbol rate. The probability of error goal for QAM transmission with $b = 4$ is $\bar{P}_b \leq 10^{-6}$. (For parts b and c, use Turbo Codes, for parts d,e, and f, use LDPC codes, and for part g use either.)

a. What is the capacity c in bits/symbol of this channel? (1 pt)

b. Design a turbo code at this same symbol rate that meets the data rate and probability of error goal. Show the encoders, interleavers, and constellation. (5 pts)

c. For the code in part b, what is a reasonable size for the interleaver period? W hat is a good value for S if this interleaver is an S-random interleaver? (2 pts)

d. Repeat part b using an IBM LDPC code and show exact numbers of bits, dummy bits, etc in the encoder. (6 pts)

e. How many parity checks are there in your code and how many bits contribute to each? (2 pts)

f. In the decoder, how many equality nodes are used? (2 pts)

g. What could you do to double the data rate achieved in parts b or d at the same probability of error? (2 pts)

Bibliography