Super Codes: A Flexible Multi–Rate Coding System

Steven S. Pietrobon

Small World Communications, 6 First Avenue, Payneham South SA 5070, Australia.
E–mail: steven@sworld.com.au

Abstract: *We define super codes as the serial concatenation of punctured convolutional codes. Puncturing pattern and interleaver design techniques are presented so as to maximise the free distance of the code, and thus the performance at very low bit error rates (BER). The performance of super codes is compared to turbo codes. For BERs around 10^{-5}, turbo codes perform 0.1, 0.4, and 0.6 dB better than super codes for rates 3/4, 1/2, and 1/3, respectively. However, due to the use of serial concatenation, super codes have an interleaver gain of N^{-2}, compared to only N^{-1} for turbo codes and interleaver size N. Thus, super codes are expected to perform significantly better at BERs around 10^{-10}.*

Keywords: turbo coding, super coding, parallel concatenation, serial concatenation

1. INTRODUCTION

In this paper we investigate the use of punctured rate 1/2 systematic encoders in both serial and parallel concatenated encoders. Puncturing allows the same constituent decoder in an iterative decoder to be used for decoding a variety of coding rates. We investigate three coding rates, 1/3, 1/2 and 3/4, although other coding rates are possible.

Parallel concatenated convolutional codes (commonly referred to as turbo codes) and iterative decoding techniques with soft–in–soft–out (SISO) decoders [1,2] are a powerful method of obtaining very high error control decoder performance with relatively moderate decoder complexity. The complex rate 1/2 turbo code presented in [3] was able to come within 0.55 dB of Shannon capacity for transmission of 1 bit/sym (information bits per signalling interval). This gives a coding gain of 9.05 dB compared to uncoded quadrature phase shift keying (QPSK) at a bit error rate (BER) of 10^{-5}. Less complex turbo codes are able to achieve coding gains of around 7 dB.

A key to any turbo code decoder are SISO decoders that individually decode the systematic encoders in a turbo code. The best known SISO decoder is the maximum a posteriori (MAP) algorithm [4].

Serial concatenated convolutional codes (which we call super codes) are an alternative technique for obtaining high performance [5–7]. The main characteristic of super codes is that they have a much lower error "floor" than turbo codes. The floor is the performance region where a steep fall in BER against E_b/N_0 (energy ber bit to single sided noise density ratio) changes to a much more shallower slope. The gradual slope is caused by the small free distance of turbo and super codes. By increasing the free distance of a turbo or super code, this floor can be lowered, usually at the expense of worse performance at higher BERs. Since the floor is not flat, at infinite E_b/N_0 the BER is zero.

The basic turbo and super code encoder architectures will first be described. A method for tail–biting for super codes is presented. Descriptions of various interleaving techniques will also be given. Decoder implementation and performance simulations of turbo and super codes are then presented.

2. ENCODING

Before describing the turbo code encoder we will first describe systematic recursive convolutional (SRC) encoders. This is followed by descriptions of tail–biting and some interleaver designs.

2.1. SRC Encoders

The use of systematic recursive convolutional (SRC) encoders is an important element of any turbo or super code. In fact, the recursive nature of these codes is what gives these codes their power [7–9]. The systematic part is also important as punctured systematic codes perform better than punctured non–systematic codes at higher BERs [10] (important in iterative decoding). For super codes, only the inner decoder needs to be recursive [7].

Figure 1 illustrates a rate 1/2 16 state SRC encoder with code polynomials $g_0 = 23_8$ and $g_1 = 31_8$.

We have that x_k is the input bit at time k and y_k^0 and y_k^1 are the coded bits at time k. The state of the encoder is given by the vector $S_k = (s_k^0, s_k^1, s_k^2, s_k^3)^T$. An important encoder parameter is the number of delay elements ν. For the encoder in Figure 1 $\nu = 4$. With continuous encoding $y_k^0 = x_k$ for all k which makes the encoder systematic.

2.2. Block Coding With Tail

If the data is encoded into blocks of length N then $s_0 = s_N = 0$. That is, the encoder must start in state 0 at the beginning of the block and end in state 0 at the end of

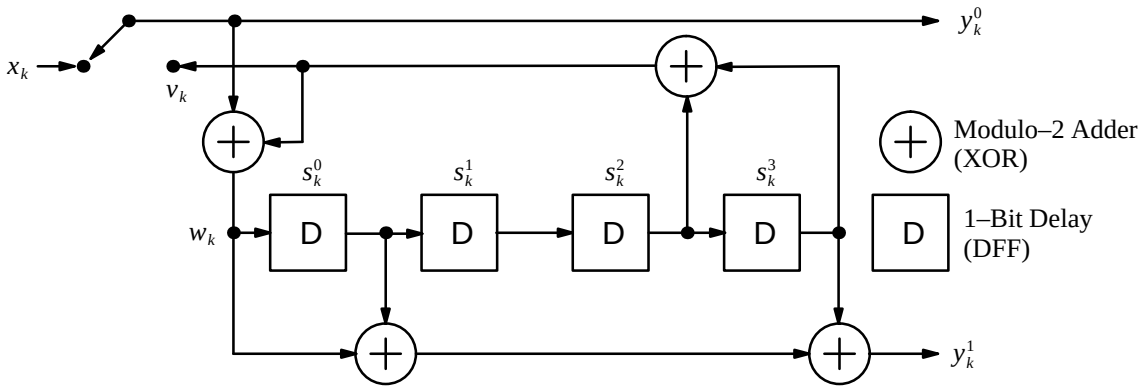

Figure 1: Rate 1/2 16 state systematic recursive convolutional encoder.

the block. Setting the initial state to zero is relatively simple and can be performed by resetting the encoder delay elements to zero at time zero. To force the final state to zero a sequence of ν zero's must be fed into the shift register in Figure 1. This is achieved by letting $y_k^0 = v_k$ for $k = N$–ν to N–1 (v_k is the feedback term connected to the multiplexer in Figure 1). These ν bits are commonly referred to as the "tail".

If the next block of data starts at time N then the encoder will already be in state zero. Thus, there is no need to reset the encoder to state zero. If the block of data does not have a tail (implying the final state is unknown) the encoder must reset the state to zero for the start of the next block.

2.3. Block Coding With Tail–biting

In tail–biting, the starting state is made to equal to the final state and thus no tail is required. We can write that

$$S_{k+1} = GS_k + X_k \tag{1}$$

where G is a $\nu \times \nu$ binary matrix and X_k is a $\nu \times 1$ matrix that is a function of the encoder inputs. For the encoder in Figure 1 we have

$$G = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \quad X_k = \begin{pmatrix} x_k \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{2}$$

It was shown in [11] that if the starting and ending state are identical, i.e., $S_0 = S_N = S^c$, then

$$S^c = (I + G^N)^{-1} S_N^0 \tag{3}$$

where S_N^0 is the final state given that the starting state was zero. In order that S^c is defined, we require that $G^N \neq I$. We can show that if the divisor polynomial $g^0(D)$ is a primitive, then the set $\{0, I, G, G^2, \ldots, G^{\nu-1}\}$ is isomorphic with GF(2^ν). Thus, $G^N \neq I$ iff 2^ν–1 does not divide N (for $\nu = 1$, this implies that two state codes cannot be made tail–biting). For example, the encoder in Figure 1 has a primitive divisor polynomial and $\nu = 4$. Thus, if 15 does not divide N, the encoder can be made tail–biting.

It was shown in [11] that

$$S_k = G^k S_0 + \sum_{i=1}^{k} G^{k-i} X_{i-1} \tag{4}$$

where addition is modulo–2. For $S_0 = 0$,

$$S_k^0 = \sum_{i=1}^{k} G^{k-i} X_{i-1}. \tag{5}$$

That is, assuming we start in state S^c, then

$$S_k = G^k S^c + S_k^0. \tag{6}$$

We also have the input/output equation

$$Y_k = G' S_k + X'_k \tag{7}$$

where G' is an $n \times \nu$ binary matrix and X'_k is an $n \times 1$ matrix that is a function of the encoder inputs. Thus we can write

$$Y_k = G'(G^k S^c + S_k^0) + X'_k\,. \tag{8}$$

If we start from state 0, then

$$Y_k^0 = G' S_k^0 + X'_k \tag{9}$$

If we input the all–zero sequence, but start from state S^c, then

$$Y_k^c = G' G^k S^c. \tag{10}$$

That is, $Y_k = Y_k^0 + Y_k^c$. If an interleaver follows the encoder, we can use this fact to reduce the encoding delay. That is, we can encode starting from state 0, interleave the data, and then add an interleaved version of (10). Since G^k has a period of 2^ν–1, small lookup tables or circuits can be used to generate (10) based on the interleaved address.

2.4. Turbo Code Encoder

Figure 2 illustrates the basic punctured turbo encoder structure. The outputs y_k^0 and c_k^1 form the outputs of a rate 1/2 RSC. ENC1 represents the encoder circuitry in Figure 1 without the systematic and tail generating circuitry. The double vertical lines represent a parallel to serial converter. The parity output c_k^1 is punctured (with the punc-

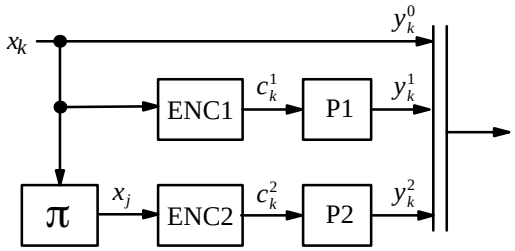

Figure 2: Parallel concatenated punctured convolutional (turbo code) encoder.

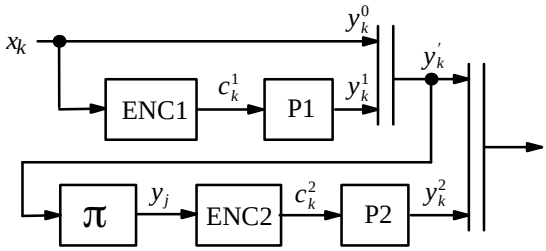

Figure 3: Serial concatenated punctured convolutional (super code) encoder.

turing pattern selecting one out of k_1 bits) to give an overall rate of $R_1 = k_1/(k_1+1)$.

The information sequence x_k is interleaved by π to give the interleaved sequence x_j. The interleaver is assumed to take a block of size N with a mapping from $k \rightarrow j$, $0 \leq k, j \leq N-1$. The interleaved output x_j is fed into a second RSC encoder (ENC2) to produce the parity sequence c_k^2. The interleaved output is not transmitted since it is equivalent to the original information sequence. As for ENC1, c_k^2 is punctured to give a rate $R_2 = k_2/(k_2+1)$. The two RSC encoders are commonly called the "constituent" encoders of a turbo encoder. The overall code rate is given by

$$R = \frac{1}{1/R_1 + 1/R_2 - 1} = \frac{k}{k+2} \qquad (11)$$

if $k = k_1 = k_2$. For example, we can obtain code rates of 1/3, 1/2, and 3/4, and 7/8 with $k_1 = 1$, 2, and 6, respectively.

The design criteria used for the constituent codes is to maximise $d_{\min,2}$ (the smallest distance for an information weight two sequence) [12,13]. This is quite different to normal convolutional codes where the commonly used design criteria is to maximise $d_{\min}$ (the minimum weight of non–zero code words). Since the design criteria is the same for both constituent codes this implies that the constituent encoders should use the same codes.

Table 1 gives the best known constituent codes for $\nu = 4$ and $R_i = 1/2$, 2/3 [12–15], and 3/4 to 15/16 (new results). Also listed are $d_{\min,2}$ and $d_{\min}$. The number in brackets indicates the average number of paths for that weight.

A peculiarity of punctured systematic codes is that the trellis may have some zero weight paths where the starting and final state are identical and not equal to zero. This leads to zero weight cycles. Normally, this would mean the code is catastrophic (infinite input gives finite output). However, since the information weight is also zero for these zero cycles, the code is not catastrophic.

These zero cycles are not desirable since the number of neighbours will increase with block length, reaching infinity for infinite size block length. Thus, we call these codes *quasi–catastrophic* and indicate this in Table 1.

Table 1: Constituent codes for ν = 4.

R_i	g_1	g_0	$d_{\min}$	$d_{\min,2}$	Quasi–catastr.?
1/2	33	23	7(2)	12(1)	no
1/2	31	23	6(1)	12(1)	no
2/3	31	23	4(1/2)	7(1)	no
3/4	31	23	3(1/3)	4(1/3)	no
4/5	31	23	3(1)	4(3/4)	yes
5/6	31	23	2(1)	2(1)	yes
6/7	31	23	3(4/3)	3(1/3)	yes
7/8	31	23	2(1/7)	2(1/7)	yes
8/9	31	23	2(1/8)	2(1/8)	yes
9/10	31	23	2(4/9)	2(4/9)	yes
10/11	31	23	2(2.2)	2(2.2)	yes
11/12	31	23	2(3/11)	2(3/11)	yes
12/13	31	23	2(5/12)	2(5/12)	yes
13/14	31	23	2(8/13)	2(8/13)	yes
14/15	31	23	2(5/7)	2(5/7)	yes
15/16	31	23	2(∞)	2(∞)	yes

Some comment needs to be made about the codes listed in Table 1. The rate 5/6 code has both $d_{\min,2} = d_{\min} = 2$ while the next higher rate has $d_{\min,2} = d_{\min} = 3$. We believe this is because the puncturing period divides $2^{\nu-1}$ and the code is quasi–catastrophic. If the puncturing period equals $2^{\nu-1}$ then there are zero cycles for all non–zero states (since there is a zero weight path that goes through all non–zero states). Thus, all path weights will have an infinite number of paths. Therefore, punctured codes with rates 15/16 and above can be expected to perform very poorly.

A time–varying puncturing period maybe able to reduce this problem.

2.5. Super Code Encoder

A super code encoder is shown in Figure 3. The overall code rate is

$$R = R_1 R_2 = \frac{k}{k+2} \quad (12)$$

if $k_2 = k_1 + 1$, $k_1 = k$. Thus, for overall rates of 1/3, 1/2, and 3/4 the outer code rate is 1/2, 2/3, and 6/7 and the inner code is of rate 2/3, 3/4, and 7/8, respectively.

If tail–biting is used, the encoding technique described previously can be used to decrease the encoding delay of the outer decoder. For turbo codes, the technique does not provide any advantage since the interleaver only interleaves data bits.

Since high rate punctured codes are used in the super code encoder, the codes given in Table 1 are also used for super codes. The interleaver gain is given by $N^{-\lfloor (d_{\min}+1)/2 \rfloor}$ [7]. Thus, the interleaver gain is N^{-3} for $k_1 = 1$ (we use the second code from Table 1), N^{-2} for $k_1 = 2,3,4,6$ and N^{-1} for $k_1 = 5,7,8...$ Since we desire codes with an interleaver gain of N^{-2} or better, we only consider super codes with the outer code having $d_{\min} \geq 3$.

For some super codes, it may be preferable to have $k_2 = k_1 - 1$, $k_2 = k$, e.g., $R_1 = 2/3$ and $R_2 = 1/2$ to give a $R = 1/3$ code. For $k_2 = 1$ the interleaver gain is reduced to N^{-2}.

2.6. Interleaving

An important parameter of a turbo code is the interleaver size N. It can be shown that for turbo codes, the decoder bit error rate (or interleaver gain) is inversely proportional to N [8,9]. Thus, one should try to make N as large as possible within the constraints of complexity and decoding delay.

Another important design parameter is the actual interleaving pattern used. It has been found that with very high probability, a randomly generated interleaver will give very good performance. In fact, to analyse turbo and super codes, performance bounds are averaged over all possible interleaver patterns. For high SNR these bounds are very close to actual measured performance of turbo code decoders [7–9].

Various interleaver construction methods can be found in [15–21]. We will present the most common construction methods.

2.6.1. Pseudo Random

A pseudo–random interleaver pattern can be constructed by generating a sequence of random numbers between 0 and N–1. This can be efficiently done by forming two arrays. The first array contains the interleaving sequence and is initially empty. The second array contains the numbers 0 to N–1 in sequence. A uniform random number between 0 and 1 is generated and multiplied by the size of the second array (in this case N). The random number is quantised to an integer and forms the address for selecting the number from the second array which is placed in the first array. The second array is then reshuffled so that it is of size N–1. Another random number is generated between 0 and 1 and multiplied by the size of the second array (in this case N–1). This forms the address for the second array and the number selected is placed into the first array. The process repeats until the second array is empty and the first array is full. The first array then forms the random interleaver pattern.

A problem with randomly generated arrays is that they can have some very low weight error patterns that are not correctable. Even though these patterns don't occur very often, this leads to the code having a small free distance and thus have a BER curve that declines very slowly with SNR.

2.6.2. *S*–random

An effective way of preventing the problems of a pseudo–random interleaver is to use a spread or *S*–random interleaver [22]. An *S*–random interleaver is generated in the same way as a random interleaver, except that a test is performed on each randomly generated number. If the difference between the random number and the previous S random numbers is between $-S$ and S, the random number is rejected and another random number is selected until one is found that satisfies the criteria. This ensures that any pairs of bits that are less than S apart in time are separated by at least S after interleaving.

Note that this algorithm is not always guaranteed to complete. The larger the value of S the less likely that a pattern can be found. We have found that a reasonable value for S is to choose a value that is equal to $\lfloor N^{0.438} \rfloor$.

2.6.3. Symmetric *S*–random

A significant implementation simplification can be obtained by using a *symmetric* interleaver. A symmetric interleaver has the property that if $I \rightarrow J$, then $J \rightarrow I$ for $I, J \in 0,...,N-1$. This implies that the deinterleaving pattern is exactly the same as the interleaving pattern. The *S*–random property can also be applied to symmetric interleavers.

With a random access memory (RAM), one can usually perform a read followed by a write in a single clock cycle at the same address location. Initially the data would written in sequentially. The data would then be read in interleaved order while new data is written in interleaved order. The new data is then read out in sequential order (while the next block is also written in sequential order). This is equivalent to a deinterleave operation. However, since a deinterleave is equivalent to an interleave for a symmetric interleaver, the new data is correctly interleaved. Thus, a single memory can be used, compared to two memories for a non–symmetric interleaver (as shown in [24], a single memory can also be used for non–symmetric interleavers, but complicated address generating circuitry is required).

2.6.4. Interleaving for Punctured Codes

The encoders shown in Figures 2 and 3 are composed of two punctured systematic encoders. We can see that if we use a random interleaver, then due to the puncturing, an information bit can have either none, one, or two parity bits. This leads to an uneven protection of each of the information bits. That is, some bits will be protected more than others.

To overcome this situation we should use an interleaver that uniformly protects the information bits. The first and simplest example of this is the *odd–even* interleaver for rate 1/2 turbo codes [16]. In an odd–even interleaver all the even positioned bits are mapped to even positions and all the odd positioned bits are mapped to odd positions. A rate 1/2 turbo code has puncturing patterns of 10 and 01 for the first and second encoders. Thus, this leads to each information bit having one parity bit and an overall increase in performance compared to an interleaver without this constraint.

In general, one should design the interleaver according to the puncturing pattern used, trying to uniformly spread the parity bits over the information bits.

For example, for turbo codes and $k = 2$, the puncturing patterns (for the first and second convolutional encoder, respectively) are 10 and 01. For $k = 3$, the patterns are 100 and 010 (100 and 001 are also possible, but they will give the same performance). Assume bit position I maps to bit position J. Thus, to keep an even distribution we have the constraint that $I \bmod 3 = J \bmod 3$ ($I \bmod 3$ is the remainder after dividing I by 3). We shall call this a mod–3 interleaver (similarly, an odd–even interleaver is a mod–2 interleaver).

For $k = 4$, the puncturing patterns are 1000 and 0010, i.e., every second bit will either have none or one parity bit. Again, we have $I \bmod 4 = J \bmod 4$. We can loosen this constraint for $I \bmod 4 = 1$ or 3. For these values of $I \bmod 4$, $J \bmod 4 = 1$ or 3.

For any k and with puncturing patterns 100...00 and 0...010...0 (the 1 is in position k div 2) the mod–k interleaver constraints are defined as

$$\min(i, k - i) = \min(j, k - j) \qquad (13)$$

where $i = I \bmod k$ and $j = J \bmod k$. Note that this constraint is slightly different to the $k = 3$ constraint described above. In this case, $i = 1$ or 2 can go to $j = 1$ or 2. This will a give a more random interleaver, but at the expense of less uniformity in protecting the information bits.

For super codes, the interleavers designed for turbo codes can also be used. In this case, a mod–k_2 interleaver should be used. In general, the parity bit of the outer code can be placed in any position. However, in order to use a mod–k_2 interleaver, the position of the parity bit is also important and plays a part in designing the puncturing pattern. As before, the puncturing patterns are designed so as to provide uniform protection of the data bits.

Table 2 shows the order of the data and parity bits of the outer code mod–k_2 with the associated puncturing pattern of the inner code for $k_1 = k = 1$ to 6.

As can be seen, the pattern used depends on whether k is odd or even. For $k = 1$, each data bit is followed by its parity bit. Thus, the inner code pattern always selects the data bit. For $k = 2$, a mod–3 interleaver fixes D_0, while D_1 and P_1 can alternate in position. Since D_1 already has a parity bit, we let the inner code puncturing pattern select the parity bit for D_0.

Table 2: Super code puncturing patterns ($k_1 = k$)

k	Outer Code Pattern	Inner Code Pattern
1	D_0P_0	10
2	$D_0D_1P_1$	100
3	$D_0D_1P_0D_2$	0100
4	$D_0D_1D_2P_2D_3$	10000
5	$D_0D_1D_2P_0D_3D_4$	001000
6	$D_0D_1D_2D_3P_3D_4D_5$	1000000

For $k = 3$ we let P_0 be in position 2, since that position is fixed in a mod–4 interleaver. The inner code selects position 1 as D_1 and D_2 alternate in a mod–4 interleaver. The other patterns can be constructed in a similar manner.

In general if k is odd, the outer code selects the parity bit for data position 0 mod k and places it in position $[(k+1) \text{ div } 2] \bmod (k+1)$. The inner code selects the parity bit for data position $(k \text{ div } 2) \bmod (k+1)$.

For k even, the outer code selects the parity bit for data position $(k \text{ div } 2) \bmod k$ and places it in position $[(k \text{ div } 2)+1] \bmod (k+1)$. The inner code selects the parity bit for data position 0 mod $(k+1)$.

If the turbo code data is continuously encoded then the interleaver size N should be divisible by k (k_2 for super codes). This ensures that the correct puncturing pattern is applied to each block of data. This restriction is not required for data transmitted in blocks (this is because the encoder is reset at the beginning of each block). If phase locked loops are used to generate the required clocks, then it may also be desirable to have k or k_2 divide N for turbo and super codes, respectively.

The puncturing patterns for $k_2 = k$ super codes are in general more complex than for $k_1 = k$ super codes. The cases for $k = 2$ and 3 are relatively simple and are shown in Table 3. In general, the outer code pattern will have a period equal to the least common multiple of n_1 and k_2, e.g., for $k = 1$ to 3, we have lcm(3,1) = 3, lcm(4,2) = 4, lcm(5,3) = 15.

Software for generating a mod–k symmetric or non–symmetric S–random interleavers can be found at [23].

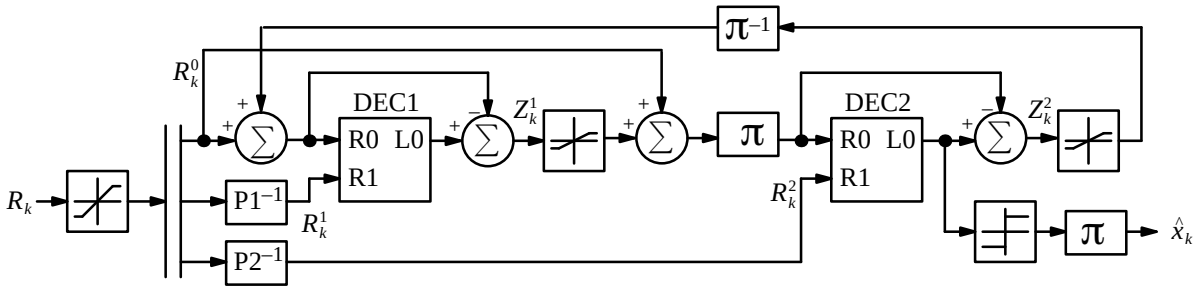

Figure 4: Punctured turbo code decoder.

Table 3: Super code puncturing patterns ($k_2 = k$)

k	Outer Code Pattern	Inner Code Pattern
1	$D_0D_1P_1$	1
2	$D_0D_1P_0D_2$	01

3. DECODER IMPLEMENTATION

A turbo code is far too complex to decode with a single decoder. Instead, each convolutional code in the turbo code is decoded separately with soft information being passed from one decoder to the next in an iterative fashion.

Let the encoder modulate a binary phase shift keyed (BPSK) signal with additive white Gaussian noise (AWGN). The received signal at time k is

$$R_k^i = 1 - 2y_k^i + n_k^i \tag{14}$$

where $y_k^i \in \{0, 1\}$, $i = 0{,}1$ are the coded bits for a rate 1/2 systematic code (we can easily extend this to other rates), and n_k^i is the Gaussian distributed noise component with zero mean and normalised variance σ^2. For a systematic code $y_k^0 = x_k$.

The term $1 - 2y_k^i$ implies that in two's complement notation, the most significant bit is equal to y_k^i. We have

$$\frac{1}{\sigma^2} = 2R\frac{E_b}{N_0} \tag{15}$$

where E_b/N_0 is the energy per bit to single sided noise density ratio and R is the code rate.

The first step in iteratively decoding a turbo code is to decode the data from the first encoder. We shall assume that a maximum a posteriori (MAP) decoder [4] is used to decode the individual codes in the turbo code. A MAP decoder finds the most likely bit to have been transmitted given a received noisy sequence (compared to say the Viterbi algorithm which finds the most likely sequence).

Ignoring decoder delay, the output of a MAP decoder in the log domain can be expressed as

$$L_k^i = W_k^i + R_k^i + Z_k^i \tag{16}$$

where W_k^i is the a priori information of y_k^i and Z_k^i is known as the extrinsic information. The a priori information is defined as

$$W_k^i = -\ln\left(\frac{P_r(y_k^i = 1)}{P_r(y_k^i = 0)}\right). \tag{17}$$

Figure 4 illustrates a punctured turbo code decoder. The decoders are implemented in the log–domain. In our simulations we added a limiter circuit at the receiver to simulate the limiting effect of digital demodulators. The limiting amplitude is given by

$$R_{\text{lim}} = (\sigma\sqrt{2/\pi}\, e^{-1/2\sigma^2} + 1 - 2Q(1/\sigma))/0.65 \tag{18}$$

where $Q(\cdot)$ is the error function. This simulates the effect of an automatic gain control circuit in the receiver [24].

We also have a gain and limiting function for the extrinsic function. We found that with a large number of iterations, having a gain less than one improves decoder performance. For twelve iterations, a gain of 1/4 was used with the output limited to $\pm R_{\text{lim}}$.

Figure 5 shows a punctured super code decoder. In this case the inner code is decoded first, followed by decoding the outer code.

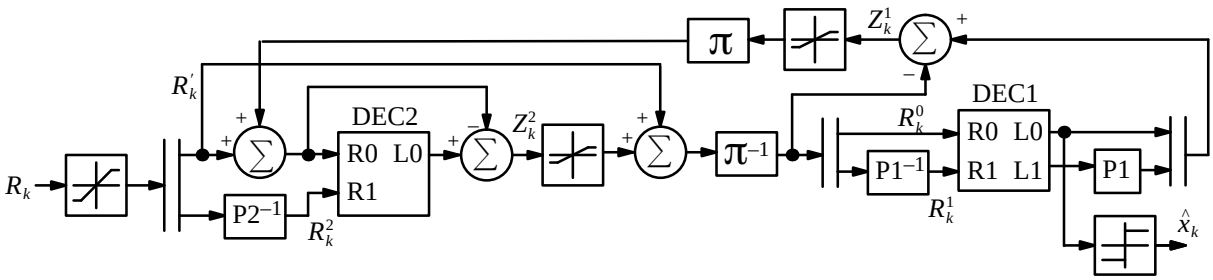

Figure 5: Punctured super code decoder.

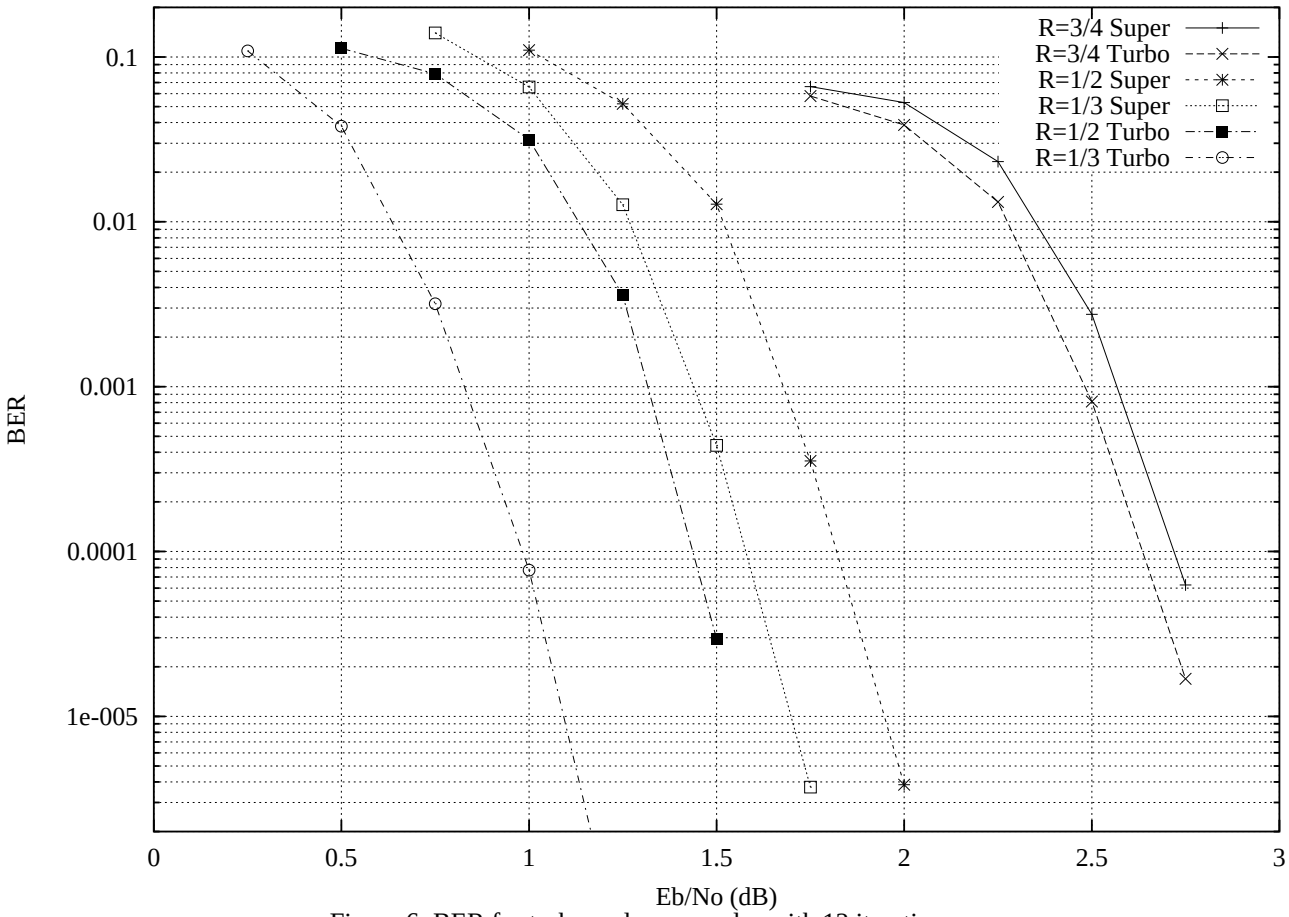

Figure 6: BER for turbo and super codes with 12 iterations.

4. SIMULATION RESULTS

In all our simulations a block log–MAP decoder was used. All the encoders added a tail which slightly reduces the actual code rate. Code rates of 1/3, 1/2, and 3/4 were simulated over an AWGN channel for both turbo and super codes. Except for the rate 1/3 turbo code with g_0 = 23 and g_1 = 33, all codes used g_0 = 23 and g_1 = 31.

A maximum of up to 12 iterations were simulated. To speed the decoding process, if two consecutive full iterations gave zero errors, the algorithm was stopped. All mod–k interleavers were symmetric with $S = \lfloor N^{0.438} \rfloor$.

Interleaver sizes were 4080 for rate 1/2 and 3/4 super codes with $k_2 = k$. It was found that the rate 1/3 super code with $k_2 = k$ performed only slightly better than the rate 1/2 super code. Improved performance was obtained with $k_1 = k$. To keep the same data block size, the interleaver size was reduced to 3060 for the rate 1/3 super code. Similarly, to keep the same data block size, interleaver sizes were 2040, 2720, and 3497 for rate 1/3, 1/2, and 3/4 turbo codes, respectively.

For the rate 1/3 super code, it was found that slightly better performance was achieved by decoding the inner code first, even though it is the more powerful code. To obtain reliable frame and bit error rates, 64 frame errors were counted (this is a better criteria than counting bit errors since decoding errors occur in bursts). Some of the simulation points below 10^{-5} BER have less than 10 frame errors due to excessively long simulation times.

Figure 6 shows the bit error rate (BER) of the turbo and super codes that were obtained. For a BER of 10^{-5}, super codes perform 0.1, 0.4 and 0.6 dB worse than turbo codes for rates 3/4, 1/2 and 1/3, respectively. It is expected that since the interleaver gain is N^{-2} for super codes, compared to N^{-1} for turbo codes, that the super codes will perform significantly better than the turbo codes at a BER of 10^{-10}.

5. CONCLUSIONS

A method has been presented of using punctured rate 1/2 recursive systematic convolutional codes in a serial concatenated scheme. We call these codes super codes. At moderate BERs, super codes perform worse than turbo codes. However, it is expected that super codes will perform better than turbo codes at very low BERs.

A method has also been presented of using tail–biting for serial concatenated codes that reduces encoder delay. In our simulations, tails were added and block MAP decoding was used. Future work will investigate the use of

tail–biting, sliding–block MAP decoders, and data quantisation.

REFERENCES

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error–correcting coding and decoding: Turbo–Codes," *IEEE Int. Conf. Commun.*, Geneva, Switzerland, pp. 1064– 1070, May 1993.

[2] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: Turbo–codes," *IEEE Trans. Commun.*, vol. 44, pp. 1261–1271, Oct. 1996.

[3] C. Berrou, "Some clinical aspects of turbo codes," *Int. Symp. Turbo Codes*, pp. 26–31, Brest, France, Sep. 1997.

[4] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. IT–20, pp. 284–287, Mar. 1974.

[5] L. C. Perez and D. J. Costello, Jr., "Serial concatenation of convolutional codes," *Int. Symp. on Commun. Theory and Applic.*, pp. 338–344, Ambleside, England, July 1995.

[6] J. Y. Couleaud, "High gain coding schemes for space communications," ENSICA Final Year Report, *Uni. of South Australia*, Sep. 1995.

[7] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial concatenation of interleaved codes: Performance, analysis, design, and iterative decoding," *IEEE Trans. Inform. Theory*, vol. 44, pp. 909–926, May 1998.

[8] S. Benedetto and G. Montorsi, "Unveiling turbo codes: Some results on parallel concatenated coding schemes," IEEE Trans. Inform. Theory, vol. 42, pp. 409–429, Mar. 1996.

[9] L. C. Perez, J. Seghers, and D. J. Costello, Jr., "A distance spectrum interpretation of turbo codes," IEEE Trans. Inform. Theory, vol. 42, pp. 1698–1709, Nov. 1996.

[10] M.–G. Kim, "On systematic punctured convolutional codes," *IEEE Trans. Commun.*, vol. 45, pp. 133–139, Feb. 1997.

[11] C. Berrou, C. Douillard, and M. Jezequel, "Multiple parallel concatenation of circular recursive systematic convolutional (CRSC) codes," *Annales des Télécommun.*, vol. 54, pp. 166–172, Mar.–Apr. 1999.

[12] D. Divsalar and F. Pollara, "On the design of turbo codes," *JPL TDA Progress Report*, vol. 42–123, pp. 99–121, 15 Nov. 1995.

[13] S. Benedetto and G. Montorsi, "Design of parallel concatenated convolutional codes," *IEEE Trans. Commun.*, vol. 44, pp. 591–600, May 1996.

[14] M. S. C. Ho, S. S. Pietrobon, and T. Giles, "Improving the constituent codes for turbo encoders," submitted to *GLOBECOM'98*, Feb. 1998.

[15] Ö. F. Açikel and W. E. Ryan, "Punctured turbo–codes for BPSK/QPSK channels," *IEEE Trans. Commun.*, vol. 47, pp. 1315–1323, Sep. 1999.

[16] S. A. Barbulescu and S. S. Pietrobon, "Interleaver design for turbo codes," *Electronic Letters*, vol. 30, pp. 2107–2108, 8 Dec. 1994.

[17] S. A. Barbulescu and S. S. Pietrobon, "Terminating the trellis of turbo–codes in the same state," *Electronic Letters*, vol. 31, pp. 22–23, 5 Jan. 1995.

[18] J. D. Andersen and V. V. Zyablov, "Interleaver design for turbo coding," *Int. Symp. on Turbo Codes*, Brest, France, pp. 154–156, Sep. 1997.

[19] J. Hokfelt and T. Maseng, "Methodical interleaver design for turbo codes," *Int. Symp. on Turbo Codes*, Brest, France, pp. 212–214, Sep. 1997.

[20] F. Daneshgaran and M. Mondin, "Design of interleavers for turbo codes based on a cost function," *Int. Symp. on Turbo Codes*, Brest, France, pp. 255–258, Sep. 1997.

[21] M. S. C. Ho, S. S. Pietrobon, and T. Giles, "Interleavers for punctured turbo codes," *IEEE Asia–Pacific Conf. on Commun. and Singapore Int. Conf. on Commun. Systems*, vol. 2, pp. 520–524, Nov. 1998.

[22] D. Divsalar and F. Pollara, "Multiple turbo codes for deep–space communications," *JPL TDA Progress Report*, vol. 42–121, pp. 66–77, 15 May 1995.

[23] Small World Communications, "Random interleaver generator," Feb. 1998. http://www.sworld.com.au/software/int.zip

[24] S. S. Pietrobon, "Implementation and performance of a turbo/MAP decoder," *Int. J. Satellite Commun.*, vol. 16, pp. 23–46, Jan.– Feb. 1998.