# On Punctured Serially Concatenated Turbo Codes

Steven S. Pietrobon, Email: steven@sworld.com.au
Small World Communications, 6 First Avenue, Payneham South SA 5070, Australia.

## 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 $10^{-10}$.*

## 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 super code encoder architectures will first be described. Decoder implementation and performance simulations of turbo and super codes are then presented.

### 1.1: Turbo code encoder

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.

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

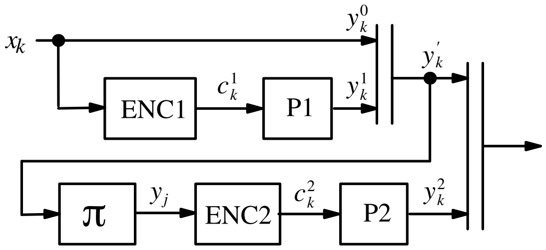


**Figure 1: Serial concatenated punctured convolutional (super code) encoder.**

$2^\nu - 1$ 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.

**Table 1: Constituent codes for $\nu = 4$.**

| $R_i$ | $g_1$ | $g_0$ | $d_{min}$ ($N_{min}$) | $d_{min,2}$ ($N_{min,2}$) | 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 |

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

## 1.2: Super Code Encoder

A super code encoder is shown in Figure 1. The overall code rate is

$$R = R_1 R_2 = \frac{k}{k+2} \tag{1}$$

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.

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 $N^{-2}$.

## 1.3: 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 [15–19]. We will present the most common construction methods.

### 1.4: Interleaving for Punctured Codes

The encoders shown in Figure 1 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

$$\min(i, k - i) = \min(j, k - j) \qquad (2)$$

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.

**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 |

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 $D_0$.

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) div 2] mod ($k$+1). The inner code selects the parity bit for data position ($k$ div 2) mod ($k$+1).

For $k$ even, the outer code selects the parity bit for data position ($k$ div 2) mod $k$ and places it in position [($k$ div 2)+1] mod ($k$+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 [21].

**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 |

## 2: 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.

Figure 2 shows a punctured super code decoder. In this case the inner code is decoded first, followed by decoding the outer code.

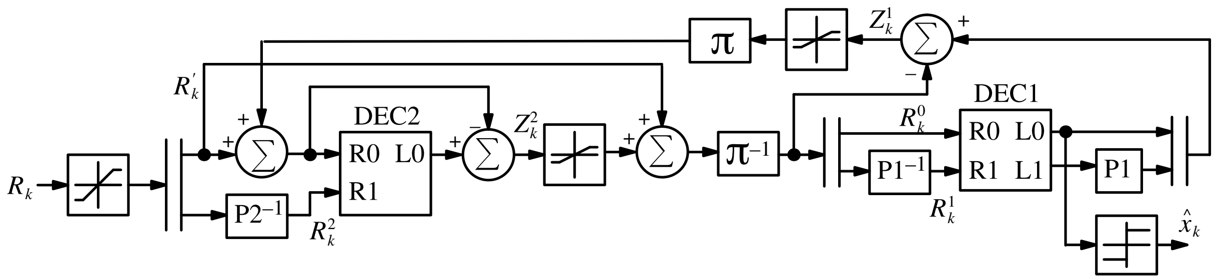


**Figure 2: Punctured super code decoder.**

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

$$L_k^i = W_k^i + R_k^i + Z_k^i \tag{3}$$

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

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

$$R_{\lim} = (\sigma \sqrt{2/\pi}\, e^{-1/2\sigma^2} + 1 - 2Q(1/\sigma))/0.65 \tag{4}$$

where $Q(\cdot)$ is the error function. 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 $\pm R_{\lim}$.

## 3: 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 3 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 $10^{-10}$.

## 4: 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.

Future work will investigate the use of tail–biting, sliding–block MAP decoders, and data quantisation.

## 5: 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.

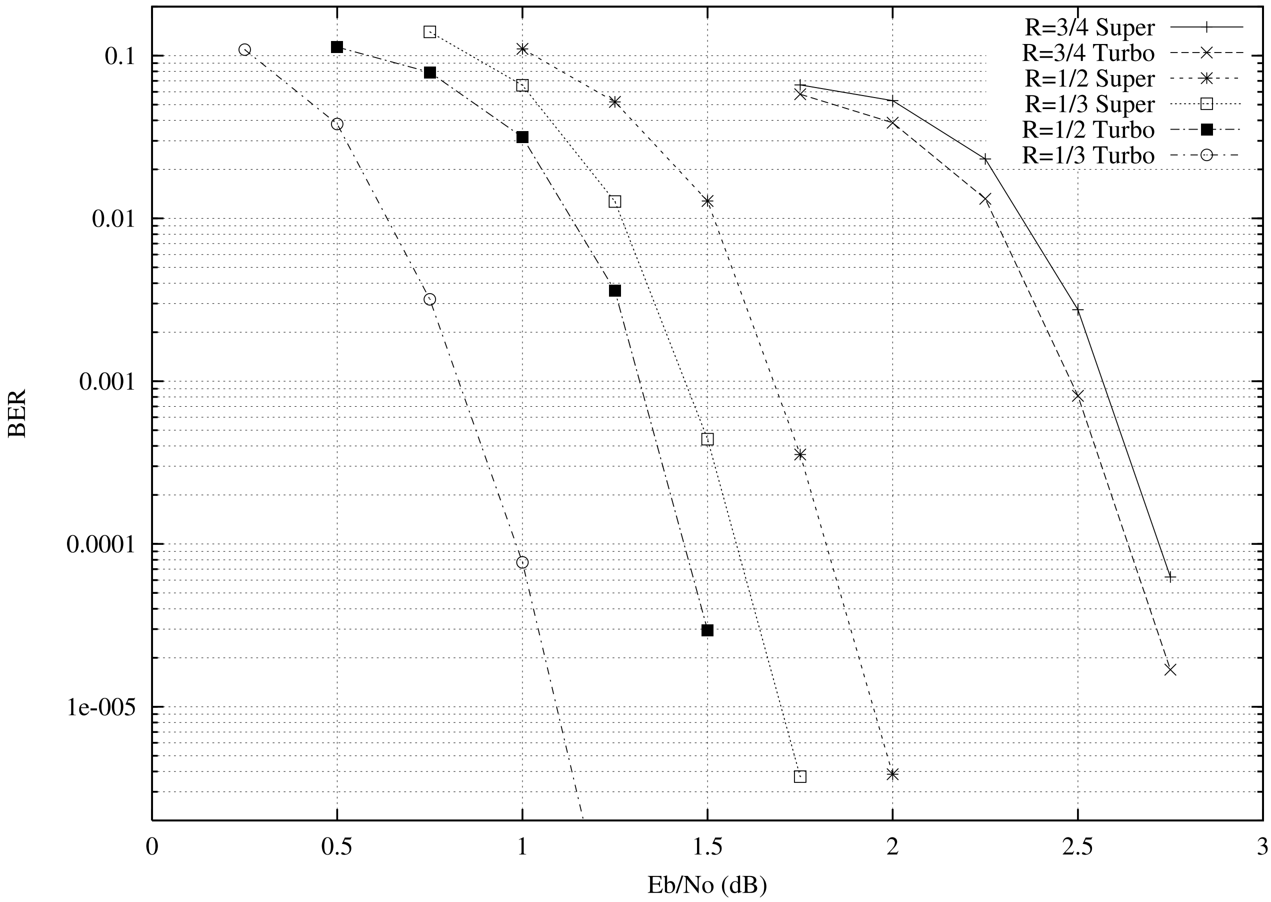


**Figure 3: BER for turbo and super codes with 12 iterations.**


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

[18] 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.

[19] 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.

[20] 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.

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