Single Parity Check Product Codes

David Rankin and T. Aaron Gulliver[1]
Dept. of Electrical and Electronic Engineering
University of Canterbury
Private Bag 4800
Christchurch, New Zealand

February 25, 2000

Submitted to IEEE Transactions on Communications

Abstract

This paper considers the performance of single parity check (SPC) multidimensional product codes in an Additive White Gaussian Noise channel. Asymptotic performance bounds are derived and compared to simulation results. A new code structure based on SPC product codes is introduced. This structure involves interleaving between the encoding of each dimension. Weight distribution analysis is used to explain the extremely good performance results for this class of codes.

1 Background

Product codes, or iterated codes, were introduced by Elias [1] in 1954 and represent the first method capable of achieving error free coding with a non-zero code rate (as the number of dimensions increases to infinity).

A product code can be constructed in the following way [6]. The data to be transmitted is arranged in a hypercube of dimension d with the length in each dimension defined by $\{k_1, k_2, \ldots, k_d\}$. The i^{th} dimension is encoded with a systematic (n_i, k_i, d_{min_i}) code, and this is repeated for all $i = 1, \ldots, d$ dimensions. The order of encoding is not important. The resulting d-dimensional product code has block length

$$N = \prod_{i=1}^{d} n_i, \tag{1}$$

[1]Now with the Dept. of Electrical and Computer Engineering, University of Victoria, P.O. Box 3055, STN CSC, Victoria, B.C., Canada V8W 3P6

and code rate

$$r = \prod_{i=1}^{d} r_i, \tag{2}$$

where the rate of the code in the i^{th} dimension is $r_i = k_i/n_i$.

The two-dimensional code consists of a data block, parity checks on the rows, the checks on columns and the checks on checks. The three-dimensional code can be represented by a cube with three of the six faces being the parity check bits of the SPC codes in each dimension.

This paper considers only product codes formed from binary, systematic, linear component codes, specifically single parity check (SPC) component codes which have the same length in every dimension. The code rate and minimum distance are

$$r = \left(\frac{n-1}{n}\right)^d \quad \text{and} \quad d_{min} = 2^d, \tag{3}$$

respectively. Product codes based on SPC product codes have also been consider explicitly by Battial [8], in which the length of the product code increases in every dimension, and Hunt [9].

If the checks on checks are not included in the product code the block length is smaller and the code rate is higher. The minimum distance and code rate are

$$r = \frac{1}{1 + d/(n-1)} \quad \text{and} \quad d_{min} = d + 1, \tag{4}$$

respectively. Therefore the asymptotic performance of the codes with checks on checks will be better because the product of the code rate and minimum distance exceeds that of the SPC product codes without checks on checks as n and d increase.

1.1 Decoding

Maximum likelihood (ML) decoding of a product code is computationally infeasible for most practical codes. Therefore a simpler sub-optimal decoding algorithm is usually employed.

The key to good performance is soft decision decoding. In particular, soft input, soft output decoding algorithms are extremely useful because *iterative decoding* can be implemented. Iterative decoding works by passing the soft outputs of one decoding stage to the next stage as soft inputs. Two soft-in/soft-out algorithms will be considered in this paper, both of which are based on maximum *a posteriori* (MAP) decoding, which minimises the probability of bit error. One algorithm separates out the extrinsic information, while the other does not.

1.1.1 BCJR Decoding

One of the earliest MAP decoding algorithms was proposed by Bahl, Cocke, Jelinek and Raviv [2], and is known as the BCJR algorithm. We will expand upon the application of the BCJR algorithm to SPC codes. Consider the transmission of a binary codeword $X = \{x_1, x_2, \ldots, x_n\}$

through a memoryless Additive White Gaussian Noise (AWGN) channel. The received word is $Y = \{y_1, y_2, \ldots, y_n\}$. The BCJR algorithm computes the *a posteriori* probabilities (APPs) $Pr\{x_i = a_i|Y\}$ where $a_i = \{0, 1\}$ and Y is the received signal. These probabilities are equal to

$$Pr\{x_i = a_i|Y\} = \sum_{\chi=a_i} \sigma_i(m', m), \tag{5}$$

where χ is the output symbol corresponding to the state transition from m' to m. The state transition probability can be written as

$$\sigma_i(m', m) = \alpha_{i-1}(m')\gamma_i(m', m)\beta(m), \tag{6}$$

where the forward probabilities $\alpha_i(m) = Pr\{S_i = m; Y_1^i\}$ and backward probabilities $\beta_i(m) = Pr\{Y_{i+1}^n|s_i = m\}$ can be recursively calculated using

$$\alpha_i(m) = \sum_{m'} \alpha_{i-1}(m')\gamma_i(m', m), \tag{7}$$

and

$$\beta_i(m) \quad = \quad \sum_{m'} \beta_{i+1}(m')\gamma_{i+1}(m, m'), \tag{8}$$

with the boundary conditions $\alpha_0(0) = 1$, $\alpha_0(m) = 0$ for $m \neq 0$, $\beta_n(0) = 1$, and $\beta_n(m) = 0$ for $m \neq 0$. The channel transition probabilities $\gamma_i(m', m)$ are defined as

$$\gamma_i(m', m) = \sum_{\chi} p_i(m|m')q_i(\chi|m', m)R(Y_i, \chi), \tag{9}$$

for all output symbols χ, where $p_i(m|m')$ is the probability that the trellis can transition from state m' to state m, and $q_i(\chi|m', m)$ is the probability that the transition from m' to m has output symbol χ. Note that both of the above probabilities are deterministic and depend only on the trellis of the code. Finally, $R(Y_i, \chi)$ is the probability that the i^{th} received symbol is equal to χ. This is the only part of the algorithm that is dependent on the received channel values.

To illustrate the simplicity of the BCJR algorithm for SPC codes recall that the trellis of these codes has only two states. The calculation of the APPs is then extremely simple

$$p_i(m|m')q_i(\chi|m', m) = \frac{1}{2} \qquad 1 \leq i \leq n - 1. \tag{10}$$

The only exception is when $i = n$ (or $i = 0$), in which case the trellis cannot transition to the "1" state (or from the "1" state). However, the values of $\alpha_n(m)$ are irrelevant because the APPs depend upon $\alpha_{i-1}(m') \times \beta_i(m)$ where $i \leq n$. A similar argument can be applied to computing $\beta_i(m)$ when $i = 0$. Since $R(Y_i, \chi) = Pr\{Y_i|\chi = a_i\}$, and calculation of this probability is extremely simple for the AWGN channel, the overall complexity is quite low. For transmission over an AWGN channel

$$Pr\{Y_i|\chi = b_i\} = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(y-\chi)^2}{2\sigma^2}\right), \tag{11}$$

where σ^2 is the noise variance and $a_i = \{0, 1\}$ is mapped to $b_i = \{1, -1\}$, respectively (BPSK modulation is assumed throughout this paper).

The BCJR algorithm as described here accepts the conditional channel probabilities $Pr\{Y_i|\chi = a_i\}$ and returns the APPs $Pr\{\chi_i = a_i|Y\}$, taking into account the structure of the code. Now consider the decoding of a multidimensional SPC product code using the BCJR algorithm. First, the received channel values Y_i are converted into the conditional probabilities $Pr\{Y_i|\chi = a_i\}$. Then the SPC codes are decoded in a single dimension (the choice of dimension is arbitrary), so for this dimension the APPs $Pr\{\chi_i = a_i|Y\}$ for each bit are determined. These APPs are then passed to the decoder for the next dimension to be used as channel input probabilities. This process is repeated for every SPC code in every dimension (one dimension at a time). Once all the dimensions have been decoded the result is optimal in the sense that the probabilities which have been passed between the decoders are all independent (due to the structure of the code). However, this decoding procedure is normally iterated because this improves performance, with decreasing returns, up to approximately the number of dimensions in the product code.

1.1.2 Log-Map Decoding

In [3], Hagenauer, Offer and Papke defined a version of the optimal MAP algorithm in the log-likelihood domain. They considered as an example the use of this algorithm to decode a simple two-dimensional SPC product code (without checks on checks). This can easily be extended to more than two dimensions. Throughout this paper the log-MAP algorithm will refer to the optimal MAP algorithm in the log domain, not any sub-optimal version of the algorithm.

The *a posteriori* probabilities for an SPC code, in terms of a log-likelihood, can be written as [3]

$$\begin{aligned} L(\hat{u}_k) &= \log \frac{Pr\{u = +1|y\}}{Pr\{u = -1|y\}} \\ &= L_c y_k + L(u_k) + \sum_{\substack{j=1 \\ j \neq k}}^{N} \boxplus L(x_j; y_j), \end{aligned} \tag{12}$$

where the binary bipolar mapping $\{0, 1\} \to \{1, -1\}$ is used, $L_c = 4/\sigma^2$, and

$$L(x_j; y_j) \triangleq \begin{cases} L_c y_k + L(u_k) & 1 \leq k \leq K \\ L_c y_k & K+1 \leq k \leq N, \end{cases} \tag{13}$$

where K is the number of information bits and N is the length of the code. The last term in (12) can be expanded to

$$\begin{aligned}
\sum_{\substack{j=1\\j\neq k}}^{N} \boxplus L(u_j) &\triangleq L\left(\sum_{\substack{j=1\\j\neq k}}^{J} \oplus u_j\right) \\
&= \log \frac{\prod_{\substack{j=1\\j\neq k}}^{N}\left(e^{L(u_j)}+1\right)+\prod_{\substack{j=1\\j\neq k}}^{N}\left(e^{L(u_j)}-1\right)}{\prod_{\substack{j=1\\j\neq k}}^{N}\left(e^{L(u_j)}+1\right)-\prod_{\substack{j=1\\j\neq k}}^{N}\left(e^{L(u_j)}-1\right)} \\
&= \log \frac{1+\prod_{\substack{j=1\\j\neq k}}^{N}\tanh(L(u_j)/2)}{1-\prod_{\substack{j=1\\j\neq k}}^{N}\tanh(L(u_j)/2)} \\
&= 2\text{arctanh}\left(\prod_{\substack{j=1\\j\neq k}}^{N}\tanh(L(u_j)/2)\right). \qquad (14)
\end{aligned}$$

Although this is an excellent algorithm for decoding product codes without checks on checks [3], the performance, as will be shown in the next section, is quite poor. The reason for this poor performance is the small minimum distance of these codes.

A goal of this paper is an efficient algorithm to decode a full (with checks on checks) product code with a log-MAP algorithm. The key point is that the checks on checks must also be allowed some kind of extrinsic information. The most reasonable approach is to simply consider all bits to be equivalent in the SPC code, in the sense that the extrinsic information will be calculated from the log-likelihood of every other bit in the code, such that *all* bits have both *a priori* and extrinsic information. The concept of separable MAP filtering [5] is similar, as an independent "refinement factor" is passed between the decoders. However, the passing of extrinsic information is handled differently here. The algorithm derived here is an extension of Battial's work on replication decoding [4]. Replication decoding is optimal MAP decoding if all the $n-k$ independent replicas of the code are taken into account. For the case of SPC codes the dual implementation of the replication decoding algorithm is extremely simple. However, some extensions of the definition are required. As described in [4] the *a priori algebraic value* does not separate the channel information from the *a priori* information. For iterative decoding this is undesirable since the extrinsic information from the previous decoding stage is included as *a priori* information in the current decoding stage.

Battail, Decouvelaere and Goldlewski [4] presented the following expression for the log-likelihood ratio (LLR) of the *a posteriori* probabilities for the i^{th} symbol, A_i, based on the dual code

$$A_i = a_i + \log \frac{\sum_{c'\in C'_{i0}} \prod_{j\in J'_c} \tanh(a_j/2) + \sum_{c'\in C'_{i1}} \prod_{j\in J'_c} \tanh(a_j/2)}{\sum_{c'\in C'_{i0}} \prod_{j\ inJ'_c} \tanh(a_j/2) - \sum_{c'\in C'_{i1}} \prod_{j\in J'_c} \tanh(a_j/2)}, \qquad (15)$$

where $a_i = \log(p_{i0}/p_{i1})$ is the *a priori algebraic value* and th e second term of A_i is the extrinsic information. For SPC codes, the dual code contains only two codewords, namely all-zeros and all-ones, so the expression can be simplified to

$$A_i = a_i + \log \frac{1 + \prod_{\substack{j=1 \\ j \neq i}}^{n} \tanh(a_j/2)}{1 - \prod_{\substack{j=1 \\ j \neq i}}^{n} \tanh(a_j/2)}. \tag{16}$$

In the derivation of a_i, Battial assumed that the symbols were equally likely. To include the *a priori* information, the following can be used

$$\begin{aligned} a_i &= \log \frac{Pr\{x_i = 0|y_i\}}{Pr\{x_i = 1|y_i\}}, \\ &= \log \frac{Pr\{y_i|x_i = 0\}}{Pr\{y_i|x_i = 1\}} + \log \frac{Pr\{x_i = 0\}}{Pr\{x_i = 1\}}, \end{aligned} \tag{17}$$

so that

$$A_i = \log \frac{Pr\{y_i|x_i = 0\}}{Pr\{y_i|x_i = 1\}} + \log \frac{Pr\{x_i = 0\}}{Pr\{x_i = 1\}} + \log \frac{1 + \prod_{\substack{j=1 \\ j \neq i}}^{n} \tanh(a_j/2)}{1 - \prod_{\substack{j=1 \\ j \neq i}}^{n} \tanh(a_j/2)}. \tag{18}$$

This expression is similar to that in [3], except that the extrinsic information can also be calculated for the parity bit. Iterative decoding of the SPC product code using the log-MAP algorithm is very similar to that with the BCJR algorithm except, as in [3], only the extrinsic information is passed between the decoders for each dimension. The implementation of the log-MAP decoder for the full product code is simpler than that of the BCJR algorithm, yet the performance, as shown in the next section, is considerably better.

2 Soft-In/Soft-Out Performance

2.1 Performance with the BCJR Algorithm

This section presents the performance of iteratively decoded multidimensional SPC product codes using the BCJR MAP algorithm. The simulations employed product codes with $(8,7)$ SPC component codes in each dimension. The code rate and block length for the five dimensional codes is $r = 0.5129$ and $N = 32768$ while the minimum distance of this code is $d_{min} = 32$. The performance of this family of $(8,7)$ SPC product codes is shown in Fig. 1. The capacity of the binary input AWGN channel is attained at 0.17dB for rate 0.5129, so the five-dimensional SPC product code at a BER of 10^{-5} is only 2.3dB away from capacity. This performance using BCJR decoding is quite good given the very low decoding complexity. The main disadvantage of these codes is the large block size required for the high-dimensional codes.

The results shown in Fig. 1 are after a number of decoding iterations (until performance converges), typically at least up to the number of dimensions of the product code. To illustrate this convergence, Fig. 2 shows the performance of the five-dimensional $(8,7)$ SPC product code after each decoding iteration. As expected, iterative decoding improves the performance with each decoding cycle, with decreasing returns, up to about the number of dimensions of the code. This decrease is due to the increasing correlation of the APPs for each bit. Initially the APPs are independent (after one decoding iteration), but with each additional iteration the correlation increases and the performance improvement decreases.

2.2 Log-MAP Performance

This section presents the performance of multidimensional SPC product codes using the log-MAP decoding algorithm. First, consider the family of (8,7) SPC product codes without checks on checks. The blocklengths for this family of codes are smaller and the code rates higher than full SPC product codes. The five dimensional code has code rate and blocklength, $r = 0.5833$ and $N = 28812$, respectively. Unfortunately the minimum distance of this code is only 6. The performance of this family of codes is shown in Fig. 3, and clearly shows an asymptotic error floor due to the poor minimum distance. However, the performance at low SNR is quite good, and is in fact better than that for the same family of codes with checks on checks. This is due to the lower number of minimum distance codewords and the higher code rate without checks on checks.

The performance of the family of $(8,7)$ SPC product codes *with* checks on checks and using the log-MAP decoding algorithm is given in Fig. 4. In this case, only the extrinsic information is passed between decoders. For comparison, the BCJR algorithm results are also shown in Fig. 4. Clearly passing only the extrinsic information between the decoders for each dimension improves performance. Furthermore, the implementation of the log-MAP decoder is simpler.

In general, the performance improves with the number of dimensions, as seen with the family of $(8,7)$ SPC codes, but increasing the size of the component codes also improves performance. Fig. 5 presents the performance of a four-dimensional $(20,19)$ SPC code (code rate $r = 0.8145$) after 18 decoding iterations. The capacity of the binary input AWGN channel for this code rate is 2.15dB, therefore this code is only 0.87dB away from capacity at a BER of 10^{-5}. This is remarkable for such a simple (although large) code.

2.3 Performance Bounds and Approximations

Consider the union bound on the probability of bit error for a binary linear code over the AWGN channel

$$\begin{aligned} P_b &\leq \sum_{i=1}^{M} \frac{1}{2}\frac{i}{N}\mathrm{erfc}\left(\sqrt{\frac{E_b}{N_o}r d_{i0}}\right) \\ &= \frac{1}{2}\sum_{i=1}^{N}\frac{iA_i}{N}\mathrm{erfc}\left(\sqrt{\frac{E_b}{N_o}ri}\right), \end{aligned} \tag{19}$$

where M is the number of codewords of length N, d_{i0} is the Hamming distance from the i^{th} codeword to the all-zero codeword and A_i is the number of codewords of weight i. Note that it is assumed (without loss of generality) that the all-zero codeword has been transmitted.

Asymptotically, as the signal-to-noise ratio (SNR) tends to infinity, the term corresponding to the minimum distance of the code dominates the summation, leading to the following approximation

$$P_b \approx \frac{1}{2}\frac{d_{min}}{N}A_{d_{min}}\mathrm{erfc}\left(\sqrt{\frac{E_b}{N_o}rd_{min}}\right). \tag{20}$$

For a d-dimensional SPC product code with identical $(n, n-1, 2)$ component codes in each dimension, the code parameters are (3) and

$$A_{d_{min}} = \binom{n}{2}^d \text{ and } N = n^d, \tag{21}$$

so that

$$P_b \approx \frac{1}{2}(n-1)^d\mathrm{erfc}\left(\sqrt{\frac{E_b}{N_o}\left(\frac{2(n-1)}{n}\right)^d}\right). \tag{22}$$

A comparison of this bound with the simulation results for the family of $(8, 7)$ product codes is given in Fig. 6. This confirms that the bound is tight for small codes at high SNR. The reason for the deviation at low SNR is the greater contribution from other low weight codewords. The asymptotic convergence indicates that the error floor which affects turbo codes is not a factor with high-dimensional SPC product codes because of the large minimum distance.

Another point to be noted is that the performance bound at high SNR is based on ML decoding of the complete product code. Since iterative log-MAP decoding of the entire code is sub-optimal, it is interesting that the performance still converges towards the asymptotic bound.

An approximation can also be obtained for the performance of multidimensional SPC product codes without checks on checks. The code parameters are given in (4, and

$$A_{d_{min}} = (n-1)^d \text{ and } N = (n-1)^d\left(1 + \frac{d}{n-1}\right), \tag{23}$$

so the asymptotic bound on performance is

$$P_b \approx \frac{d+1}{1+d(n-1)}\mathrm{erfc}\left(\sqrt{\frac{E_b}{N_o}\frac{d+1}{(1+d/(n-1))}}\right). \tag{24}$$

A comparison between this bound and the simulation results is given in Fig. 7. In this case, there is a distinct difference between the asymptotic bounds and the simulation results. Furthermore, the slope of the simulation results is slightly greater than that of the asymptotic bound. The reason for this difference lies in the assumptions made when calculating the bound. In particular, the sub-optimal decoding algorithm can often decode beyond the minimum distance of the code.

3 Randomly Interleaved SPC Product Codes

In this section, the traditional concept of SPC product codes is extended by interleaving the data block between the encoding of each dimension. The reason for this interleaving is simple, if a random interleaver is used, the code becomes more random-like. This will, with a high probability, reduce the number of codewords at minimum distance, but it may also reduce the minimum distance if a poor interleaver is chosen.

Encoding these interleaved codes is relatively simple. Every data bit is encoded in a single dimension, creating $(n-1)^{d-1}$ parity bits on a single face of the data hypercube. The data is then randomly shuffled and all the previously encoded bits are encoded in the second dimension (also generating checks on checks). This random shuffling of the data and encoding is continued for all remaining dimensions, typically a different random interleaver is used between the encoding of each dimension. It is also possible to include the previously calculated parity bits in the random interleaving, and the next section will analyse the weight distribution for both these cases.

The key problem in decoding these interleaved codes is that the traditional method of decoding all bits in every dimension is no longer applicable. Instead what must be done is an "unravelling", which is essentially the opposite of the encoding process. The last dimension to be encoded is the first to be decoded, all bits are part of a codeword and therefore all bits can be fully decoded. The next step is to deinterleave the data and decode only those bits which were encoded in the second to last dimension. This means that the parity bits added when the final dimension was encoded *cannot be used* because the bits added by encoding the final dimension are not part of codewords in the previous (interleaved) dimensions. The decoding process is repeated down to the first dimension of encoded bits, and then the extrinsic information on the data bits (after appropriate interleaving) is passed back to the outermost code for the next decoding iteration. Note that although this decoding algorithm uses less information about the parity check equations than the full product code algorithm, the random interleaving significantly improves the performance of the code, as will be shown.

3.1 Random Interleaving Analysis

The aim of this section is to find the expected number of low weight codewords of these randomly interleaved (RI) SPC product codes. It should be noted that the performance *cannot* be improved in the two dimensional case, however we will show that in three or more dimensions the RI SPC product codes have fewer low weight codewords (compared to normal SPC product codes).

We begin by investigating the *partial weight distribution*, E_{ij}, which is the number of partial codewords with input weight i and parity weight j. A partial codeword is formed by encoding the product code in only one dimension. Assume, without loss of generality, that the rows will be encoded, so that the problem is to encode of $S = (n-1)^{d-1}$ single parity check equations subject to the input data matrix. Let the number of ones in the j^{th} row be given by a_j. The input weight, i, is given by

$$\sum_{j=1}^{S} a_j = i, \tag{25}$$

with

$$0 \leq a_j \leq n-1 \qquad \forall j. \tag{26}$$

Our aim is to find all sets $A = [a_1, a_2, \ldots, a_S]$ which satisfy these constraints. These are the input patterns to the encoding of a single dimension. This approach allows the output weights to easily be enumerated, the parity weight is simply the number of rows which contain an odd number of ones.

There are $(n-1)^S$ combinations of the a_j's, which represents all possible input classes. Enumeration of these sets is required to find the partial weight distribution. The combinatorial properties of these sets can be used to develop an efficient algorithm to do this. Since the input weight is simply the sum of the a_js, and the output weight is the number of odd weight rows, the sets are *invariant under permutation* with respect to the input and output weights. Also unchanged by permutation is the number of input patterns corresponding to each set. Therefore we only need to find a representative for every permutation group. The following algorithm will generate a set of lexicographically least representatives.

Step 1: *Initialisation.* Let the vector $A = \{0, 0, \ldots, 0\}$ where A has length S.

Step 2: *Increment.* Find the right-most element in A which can be incremented, and increase it by one. Then set all elements to the right of this element to the same value. An element can be incremented if it is less than $n-1$.

Step 3: *Iteration.* Repeat step 2 until the operation is no longer possible. The set of representatives is then complete.

Proof First, by construction, it is obvious that the representatives are unique (i.e. one cannot be a permutation of another). However it must also be shown that there exist no other representatives. The proof is by contradiction. Consider a representative not generated by this algorithm. Because it can be any element from the permutation group, it can permuted such that it is is lexicographic order. However, this element has already been generated by the algorithm because all possible lexicographically least elements have been generated. Hence the algorithm produces a complete set of representatives. QED.

For example, consider a two dimensional code with $n = 4$ and $S = 3$. The algorithm will produce a set of representatives for the vectors $A = (a_1, a_2, a_3)$, where a_j's corresponds to the number of ones in each row, $0 \leq a_j \leq 3$. The resulting representatives are

$$\begin{array}{ccc} (0,0,0) & (0,0,1) & (0,0,2) \\ (0,0,3) & (0,1,1) & (0,1,2) \\ (0,1,3) & (0,2,2) & (0,2,3) \\ (0,3,3) & (1,1,1) & (1,1,2) \\ (1,1,3) & (1,2,2) & (1,2,3) \\ (1,3,3) & (2,2,2) & (2,2,3) \\ (2,3,3) & (3,3,3) & \end{array} \tag{27}$$

Now it remains to determine the number of elements in each permutation group. This is a case of ordered partitions [10] in which rows with the same numbers of ones are indistinguishable. From [10][Thm 1&2,Chp IV.1], there are $C(S; k_1, k_2, \ldots, k_n)$ such permutations where

$$C(S; k_1, k_2, \ldots, k_n) = \frac{S!}{k_1! k_2! \ldots k_n!}, \tag{28}$$

and k_i is the number of elements which are the same and S, the total number of elements, is given by

$$S = \sum_{i=1}^{n} k_i. \tag{29}$$

For example consider the two dimensional code with $n = 4$ and $i = 6$. The representatives are $(0,3,3)$, $(1,2,3)$ and $(2,2,2)$ with output weights, 2,2 and 0, respectively. The group containing $(1,2,3)$ has six elements since $C(3;1,1,1) = 6$, that containing $(2,2,2)$ has one element, and that for $(3,3,0)$ has $C(3;2,1) = 3$ elements.

Once the groups have been enumerated, the number of input patterns, and hence partial codewords, corresponding to each must be calculated. Since each row contains $n-1$ data bits, of which a_j (for $1 \leq j \leq S$) of the positions are filled by ones there must be $\binom{n-1}{a_j}$ ways of placing these ones in the row. The total number of input patterns, W_k, corresponding to the k^{th} unique set is then

$$W_k = \prod_{j=1}^{S} \binom{n-1}{a_j}. \tag{30}$$

This must be multiplied by the number of elements in the group, $C(S; k_1, \ldots, k_n)$, in order to find the number of partial codewords corresponding to the given output weight, which gives

$$E_{ij} = \sum_{k \in \Phi} W_k C(S; k_1, \ldots, k_n), \tag{31}$$

where Φ is the set of representatives with output weight j.

To continue the example, the group $(2,2,2)$ has one element, but involves $\binom{3}{2}.\binom{3}{2}.\binom{3}{2} = 27$ ways to place the ones in the data matrix. The partial codewords corresponding to these input patterns have a zero weight output (parity check weight). The group $(1,2,3)$ has six elements and $\binom{3}{1}.\binom{3}{2}.\binom{3}{3} = 9$ ways to place the ones, in each permutation, therefore $9.6 = 54$ partial codewords have an output weight of two. Finally, the group $(3,3,0)$ has three elements and $\binom{3}{3}.\binom{3}{3}.\binom{3}{0} = 1$ ways of placing the ones, thus there are $1.3 = 3$ more partial codewords with weight two outputs. Therefore for $1 = 6$

$$E_{i0} = 27 \text{ and } E_{i2} = 57. \tag{32}$$

The sum over the E_{ij} for all j gives the total number of codewords of weight i, $\binom{(n-1)^d}{i}$, which is $\binom{9}{6} = 84$ for this example.

The next step in the analysis involves increasing the output weight as more dimensions are encoded. This relies upon the fact that the data bits are randomly shuffled before each dimension is encoded. Because the input patterns can be assumed to be uniformly distributed, for a given input weight i, the probability that the output has weight j is given by

$$P_{ij} = \frac{E_{ij}}{\sum_j Eij}. \tag{33}$$

This will always be the case whenever a new dimension (with random interleaving) is encoded. If the checks on checks are not transmitted, the expected number of codewords for a given output weight can be determined by by a kind of normalised convolution of the weight distribution E_{ij} with itself. Specifically, after encoding another dimension the updated partial distribution is given by

$$E'_{ij} = \sum_{j=a+b} E_{ia}P_{ib}. \tag{34}$$

This process can be iterated for the required number of dimensions by letting $E_{ij} = E'_{ij}$. For the example, if the second dimension of the two dimensional code with $n = 4$ and input weight $i = 6$ is encoded, the weight enumerator (without checks on checks) is

$$\begin{aligned} E'_{i0} &= 27 \times (27/84) = 8.68 \\ E'_{i2} &= 27 \times (57/84) + 57 \times (27/84) = 36.64 \\ E'_{i4} &= 57 \times (57/84) = 38.68 \end{aligned}$$

Now consider the weight distribution when checks on checks are transmitted. There are two simple cases. In the first case, the parity bits which have been generated by encoding the previous dimension are interleaved with the data before encoding the present dimension. In the second case, only the data is interleaved before the next encoding. The first case is much easier to analyse because the random interleaving will uniformly distribute the ones in both the data and previous parity bits. Encoding the subsequent dimension is equivalent to encoding an SPC product code with the input weight equal to the output weight of the previous dimension. However, the product code to be encoded is no longer square, for example if the second dimension of a three dimensional RI SPC is encoded, it is equivalent to encoding a

$(n-1) \times (n-1) \times n$ SPC product code in a single dimension. This has the effect of changing $S = (n-1) \times (n-1)$ to $S = (n-1) \times n$. The weight distribution is updated by

$$E'_{ij} = \sum_a E_{ia} P_{i+a,j-a}, \tag{35}$$

where P_{ij} corresponds to the probability of finding a partial codeword with input weight i and parity weight j for the *larger* SPC product to be encoded.

For example, if the $n = 4$ two dimensional code is encoded in the second dimension (with the parity bits included in the random interleaving), then for $i = 6$

$$\begin{aligned} E'_{i0} &= E_{i,0} \times P_{i,0} = 3.16 \\ E'_{i2} &= E_{i,0} \times P_{i,2} + E_{i2} \times P_{i+2,0} = 30.02 \\ E'_{i4} &= E_{i,0} \times P_{i,4} + E_{i2} \times P_{i+2,2} = 44.61. \\ E'_{i6} &= E_{i,2} \times P_{i+2,4} = 6.21 \end{aligned}$$

Note that the P_{ij} were computed with $S = 4$ rows and $n - 1 = 3$ as the maximum number of ones in any row. Furthermore, the number of data bits, and hence the maximum input weight to this larger code, is $n \times (n-1) = 12$.

The case where only the data bits are interleaved can be analysed in a similar way. However the checks on checks need to be considered separately. For example, the weight enumerator for a three dimensional code with checks on checks and no interleaving of the parity bits can be written as

$$A_{ij} = \sum_{j=a+b+c+d+e+f+g} E_{ia} P_{ib} P_{ic} P'_{ad} P'_{ae} P'_{bf} P''_{dg}. \tag{36}$$

The input-output weight enumerator function (IOWEF) is A_{ij}, and represents the expected number of codewords with input weight i and parity check weight j. E_{ia}, P_{ib} and P_{ic} are the number of parity bits and the appropriate probabilities for each of the three checks on the data bits (with $S = (n-1)^2$). The probabilities P'_{ad}, $P'_{a,e}$ and $P'_{b,f}$ correspond to the checks on checks introduced in the encoding of the second and third dimensions, respectively (with $S = (n-1)$). P''_{df} corresponds to the checks on the checks on checks encoded in the third dimension (with $S = 1$).

This analysis shows that the expected weight distribution for a three-dimensional code with $n = 8$ and random interleaving (where the checks are also interleaved) is

$$B_0 = 1 \quad B_2 = 0.28 \quad B_4 = 21.85 \quad B_6 = 160.38 \quad B_8 = 2668.48 \ldots \tag{37}$$

The expected number of codewords (using the same code), in which only the data bits are interleaved is

$$B_0 = 1 \quad B_2 = 0.31 \quad B_4 = 20.66 \quad B_6 = 217.35 \quad B_8 = 3060.3 \ldots \tag{38}$$

A normal SPC product code of the same size has the following distribution

$$B_0 = 1 \quad B_8 = 21952\ldots \tag{39}$$

This clearly shows that the number of low weight codewords is reduced, but at the cost of perhaps reducing the minimum distance. However when more dimensions are used there are significantly fewer low weight codewords and the minimum distance is not likely to be much lower than without interleaving. For example, the four dimensional code with $n = 8$ (randomly interleaving the parity checks as well as the data), has only 10398 codewords up to and including weight 16. The corresponding normal SPC product code has 614656 codewords of weight 16. The first significant codeword weight of the average randomly interleaved four dimensional code is at weight eight with 20.8 codewords. The results given in the next section show a clear improvement in performance at low SNR due to this reduction in the number of low weight codewords. However, the analysis does predict an 'error floor' at high SNR (results for a four-dimensional $n = 8$ code with only the data bits interleaved has an error floor near $BER = 10^{-6}$).

3.2 Random Interleaving Performance

This section presents the performance of randomly interleaved SPC product codes. Fig. 8 presents a comparison using the $(8, 7)$ family of codes and Log-MAP decoding. One set of curves represents random interleaving between dimensions, while the other does not. note that the random interleaving is only on the data bits.

Note that the two-dimensional performance with random interleaving is slightly worse. This occurs because the random interleaver used creates low weight codewords (in fact for two dimensions, random interleaving cannot improve performance). The three-dimensional performance is slightly better, but the four-dimensional performance shows considerable improvement. The five-dimensional performance with random interleaving is better but not markedly so, yet with $E_b/N_O = 1.5$dB the code achieves a bit error rate of 10^{-5}. This is only 1.3dB away from capacity for a rate 0.5128 code. Given the simplicity of decoding, the performance is quite exceptional.

The improvement in performance, at no extra cost in terms of code rate or blocklength, is due to the reduction in the number of low weight codewords. Note that no attempt was made to optimize the interleavers, only randomly generated interleavers were used. It should be possible to improve the error floor through better interleaver design.

The performance of a four-dimensional $(20, 19)$ SPC product code with random interleaving is shown in Fig. 9. A BER of 10^{-5} is achieved at $E_b/N_O = 2.78$dB, which is only 0.63dB from capacity on the binary input AWGN channel. Compared to $E_b/N_O = 3.02$dB for a BER of 10^{-5} with the full SPC product code, this is a significant improvement just by adding interleaving.

Fig. 10 summarises the previous simulation results on a single graph. The points marked indicate the E_b/N_O required to achieve a BER of 10^{-5} for the given code rate. These results are compared to the binary input AWGN capacity and the classical turbo code result [7]. The performance of both the $(8,7)$ and $(10,9)$ families of randomly interleaved (RI) SPC product codes are shown for dimensions $d = 2$ to 5, inclusive. The performance of the four and five-dimensional codes is quite exceptional. Interestingly the four-dimensional $(10,9)$ RI SPC product code performs almost as well as the five-dimensional $(8,7)$ SPC product code at almost the same code rate (however the minimum distance of the five-dimensional code is much better, hence it will perform better, in comparison, at some lower BER).

4 Conclusions

SPC product codes are very simple, low complexity codes which have remarkably good performance. It was shown here that the performance of SPC product codes can be improved by separating the extrinsic information from the *a posteriori* probabilities. Furthermore, transmitting the checks on checks is essential to obtaining a high minimum distance and hence good asymptotic performance as the number of dimensions increases. Performance can also be improved by using larger component codes. Of course increasing the number of dimensions and/or the length of the component codes increases the blocklength of the product code and also changes the code rate.

Randomly interleaved SPC product codes were introduced, and were shown to perform better than non-interleaved codes for three or more dimensions. The analysis presented indicates that random interleaving reduces the number of low-weight codewords but may also also reduce the minimum distance. Although these codes cannot be fully decoded like a non-interleaved or regular SPC product code, this does not significantly degrade performance, and decoding is still quite simple.

Finally, it should be noted that these codes have a decoding complexity which is much lower than virtually any other code structure.

References

[1] P. Elias, "Error free coding," *IRE Trans. Inform. Theory*, Vol. 4, pp. 29–37, Sept. 1954.

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

[3] J. Hagenauer, E. Offer and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inform. Theory*, Vol. 42, pp. 429–445, March 1996.

[4] G. Battail, M. Decouvelaere and P. Goldlewski, "Replication decoding," *IEEE Trans. Inform. Theory*, Vol. 25, pp. 429–445, May 1979.

[5] J. Lodge, R. Young, P. Hoeher and J. Hagenauer, "Separable MAP Filters for the decoding of product and concatenated codes," Proc. *IEEE ICC*, pp. 1740–1745, Geneva, May 1993.

[6] W.W. Peterson and E.J. Weldon, Jr., *Error Correcting Codes*, 2nd Ed., MIT Press, 1972.

[7] C. Berrou, A. Glavieux and P. Thitimajshima, "Near Shannon limit error correcting coding and decoding: Turbo codes," Proc. *IEEE ICC*, pp. 1740–1745, Geneva, May 1993.

[8] G. Battail "Building long codes by combination of simple ones, thanks to weighted-output decoding", in *Proc. URSI ISSSE* (Erlangen, Germany, Sept. 1989), pp. 634-637.

[9] A. Hunt "Hyper-codes: High-performance low complexity error correcting codes" Masters Thesis, Dept. of Systems and Computer Engineering, Carleton University, Ottawa, ON, May 1998.

[10] A.P. Street and W.D. Wallis, "Combinatorial Theory: An Introduction", Charles Babbage Research Centre, Manitoba, CA, 1977.

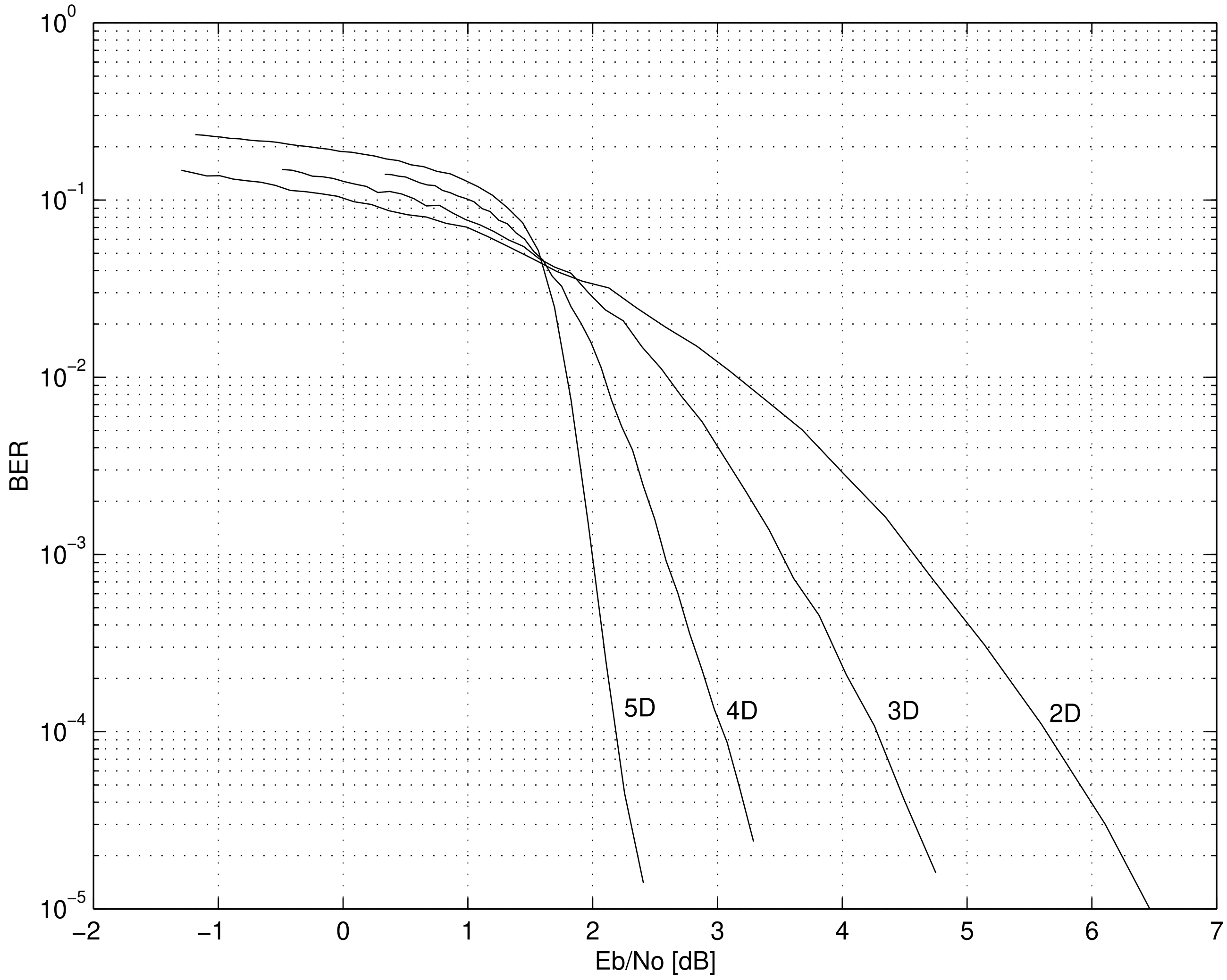

Figure 1: Performance of the (8,7) SPC product codes with two to five dimensions using BCJR decoding.

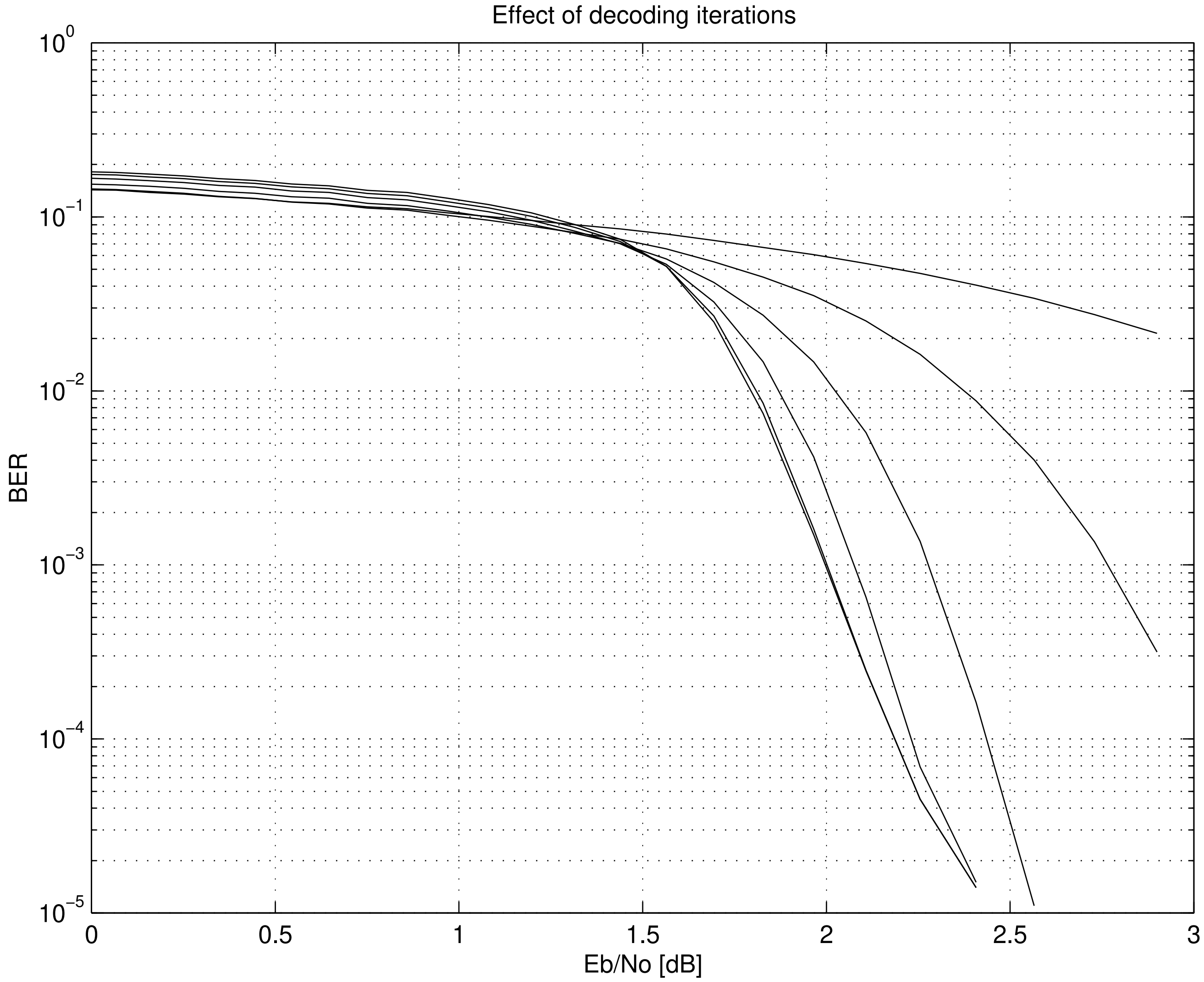

Figure 2: Iterative decoding performance of the 5D (8,7) SPC product code using BCJR decoding and up to six decoding iterations.

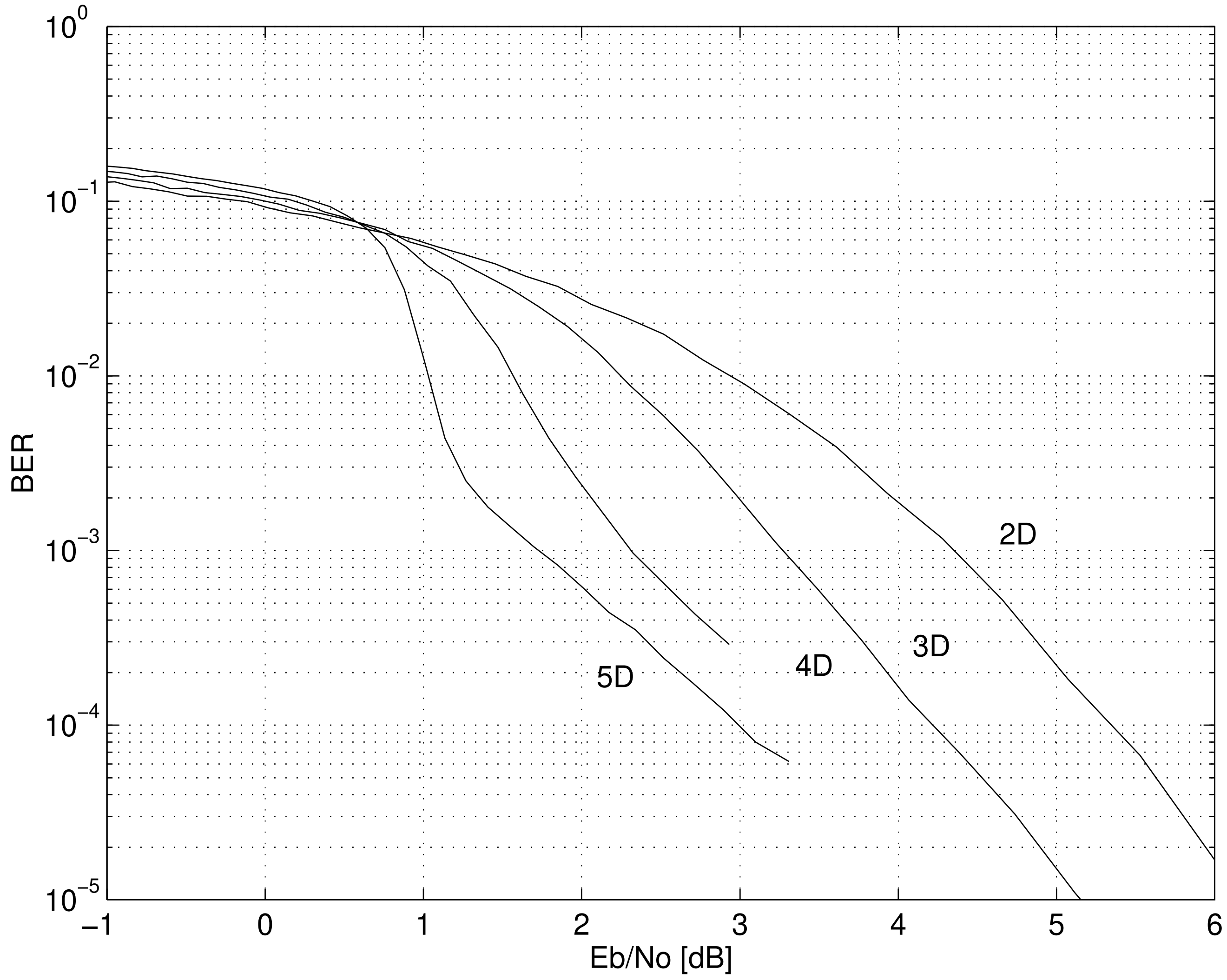

Figure 3: Performance of the (8,7) SPC product codes without checks on checks using iterative log-MAP decoding.

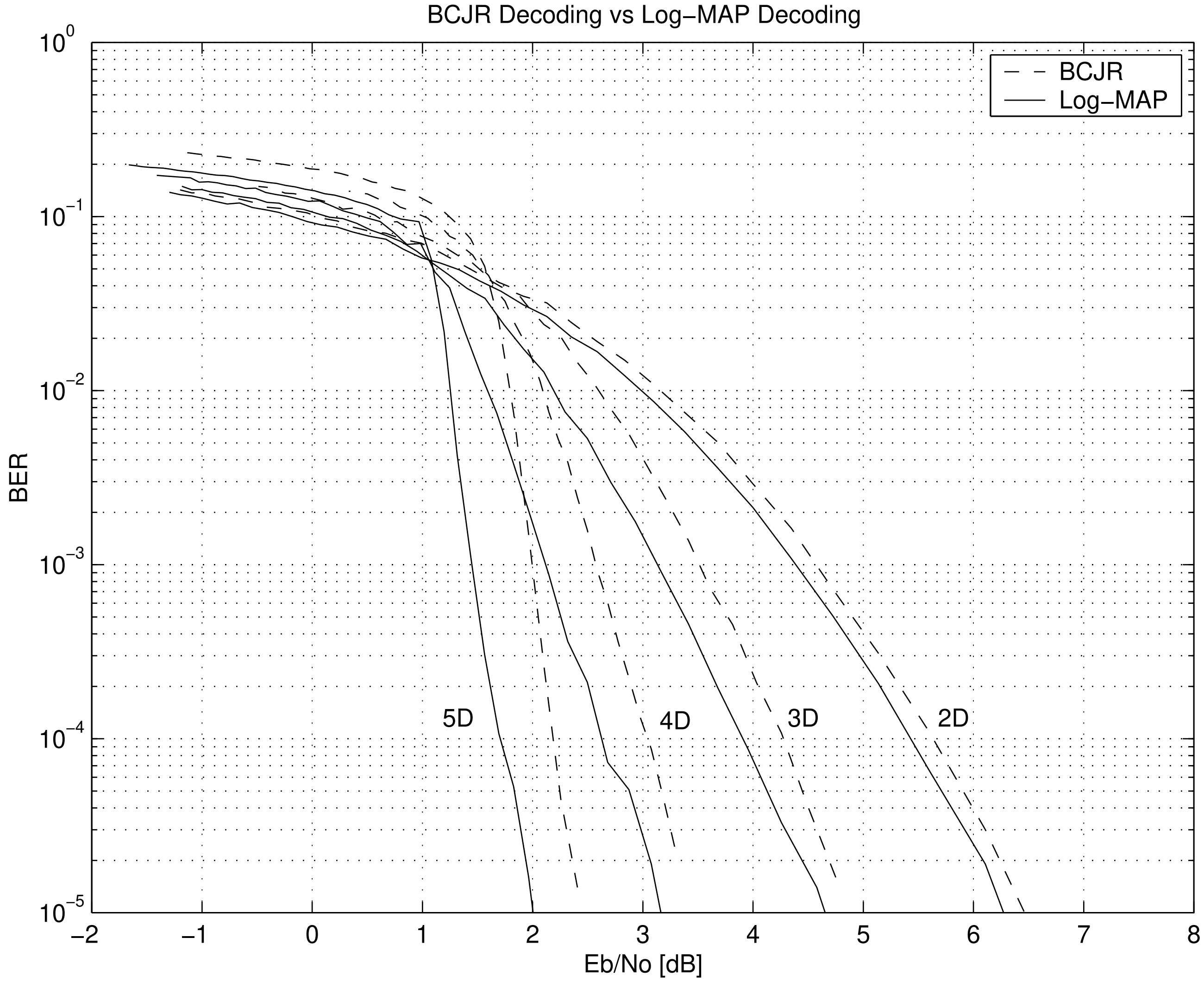

Figure 4: A comparison between BCJR decoding and log-MAP decoding of the full d-dimensional $(8,7)$ SPC product code, $d = 2..5$.

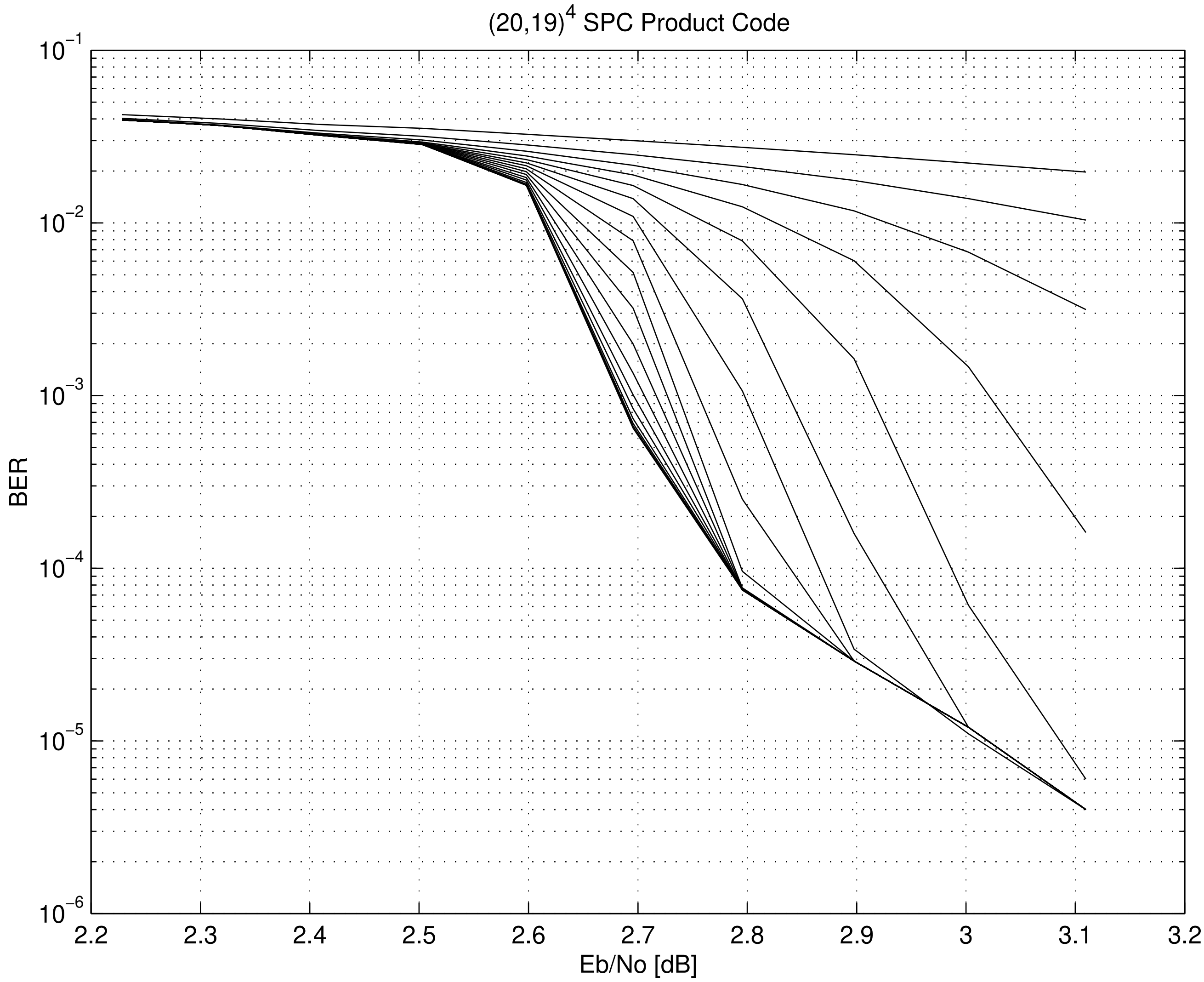

Figure 5: Performance of the four-dimensional (20,19) SPC product code using iterative log-MAP decoding.

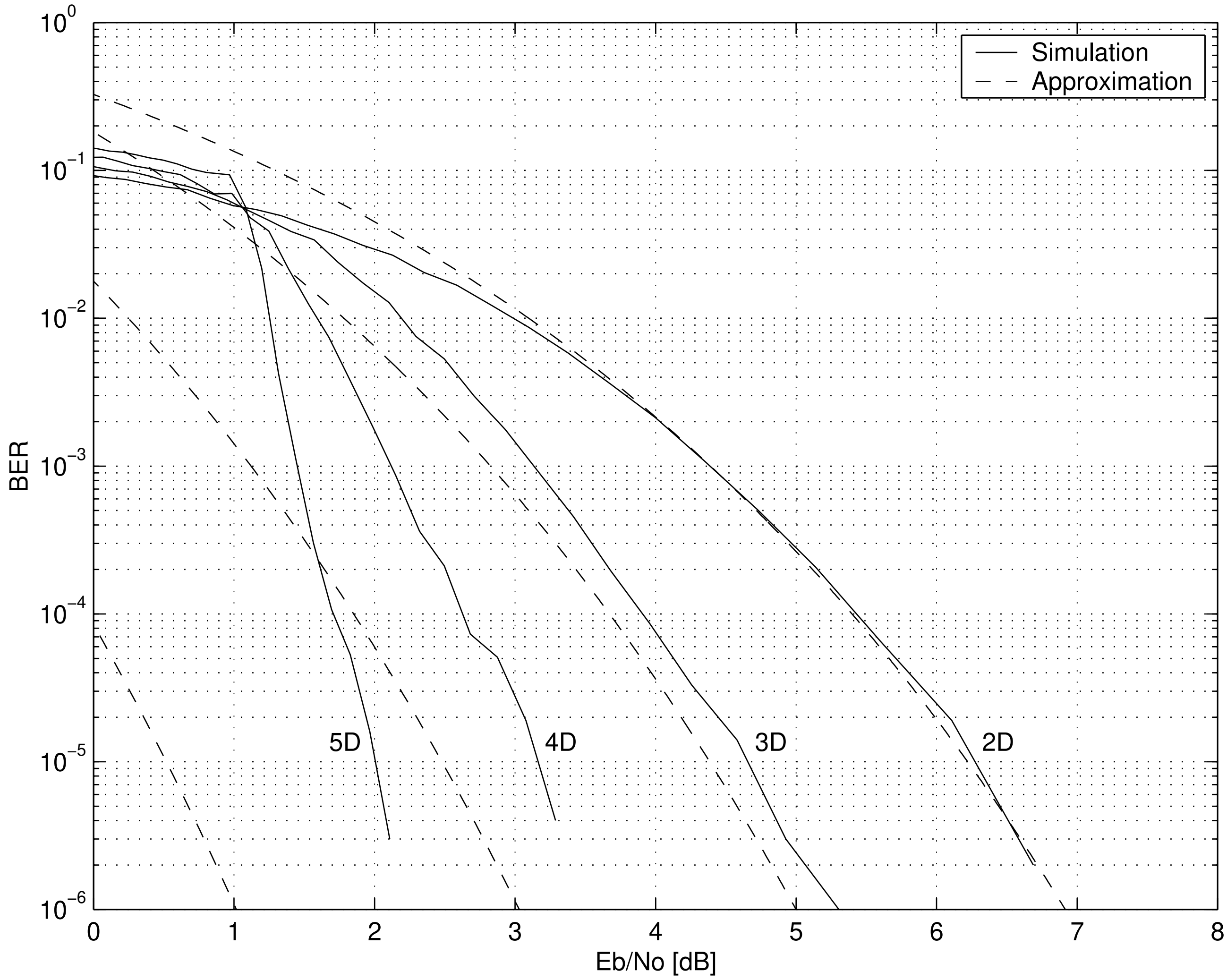

Figure 6: A comparison of the asymptotic bounds with the simulation results for the $(8, 7)$ family of SPC product codes.

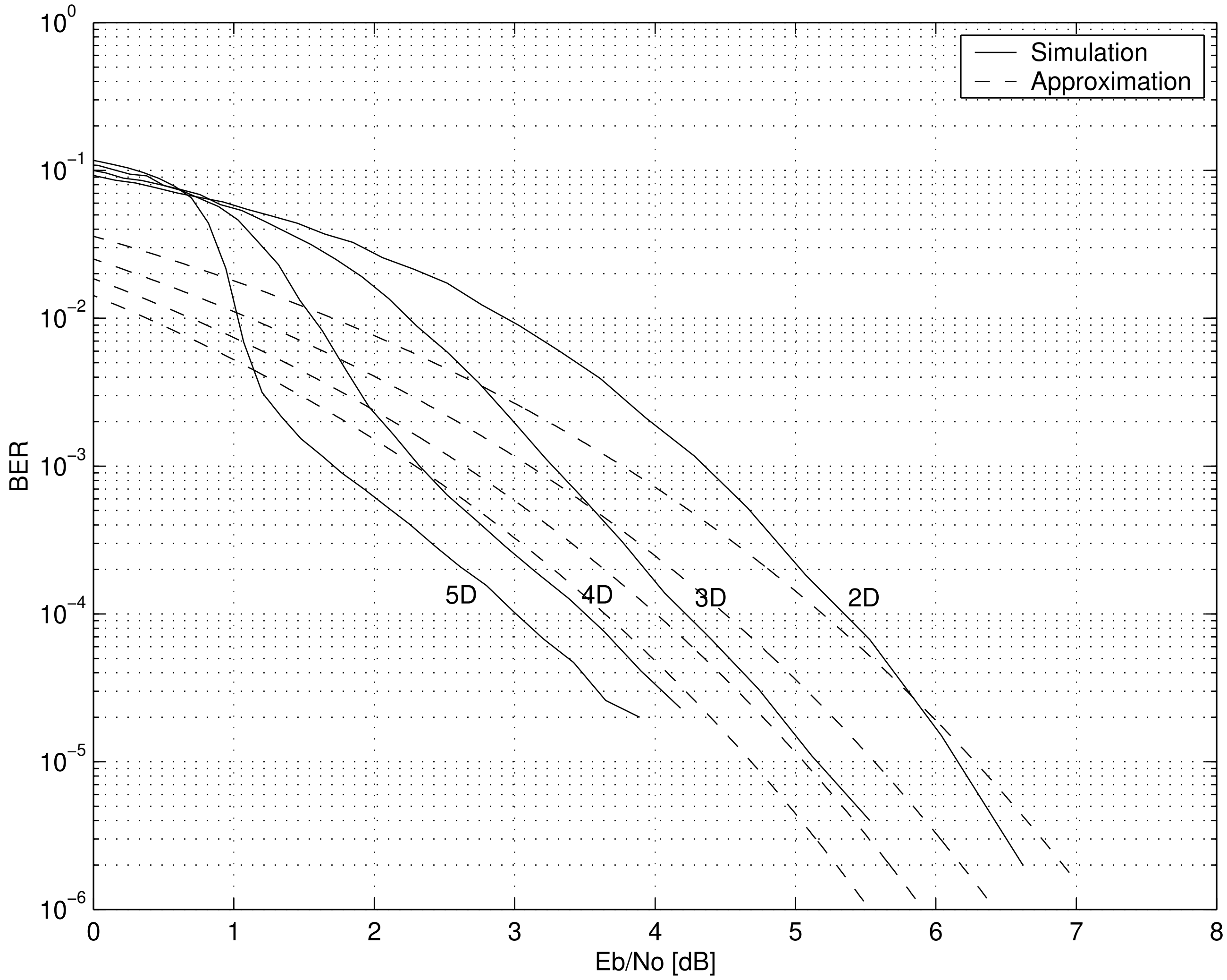

Figure 7: A comparison of the asymptotic bounds and simulation results for the $(8,7)$ family of SPC product codes without checks on checks.

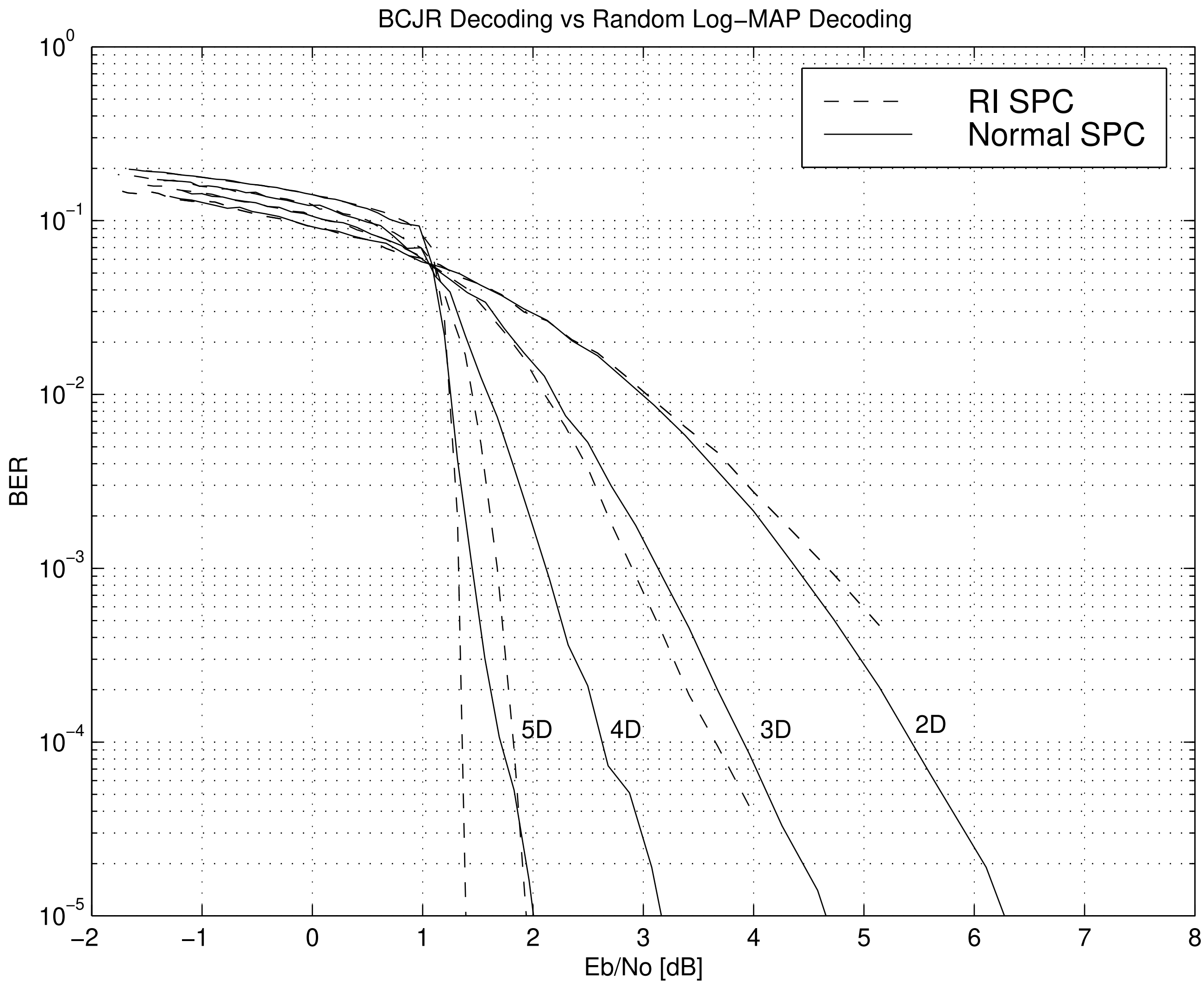

Figure 8: A comparison between the performance of randomly interleaved SPC product codes and normal SPC product codes for the d-dimensional $(8, 7)$ family of codes, $d = 2..5$

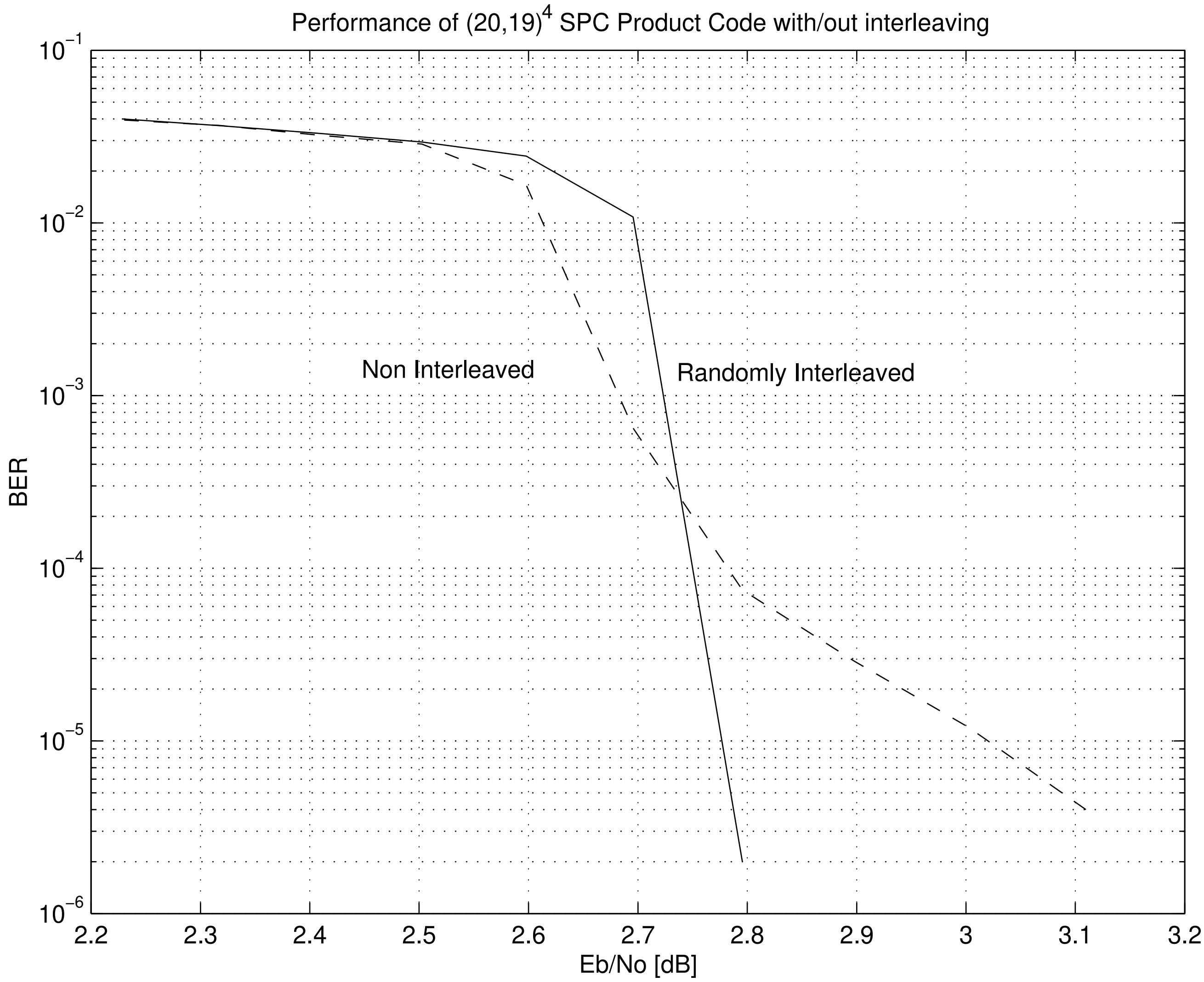

Figure 9: Performance of the four-dimensional $(20, 19)$ SPC Product Code with and without random interleaving.

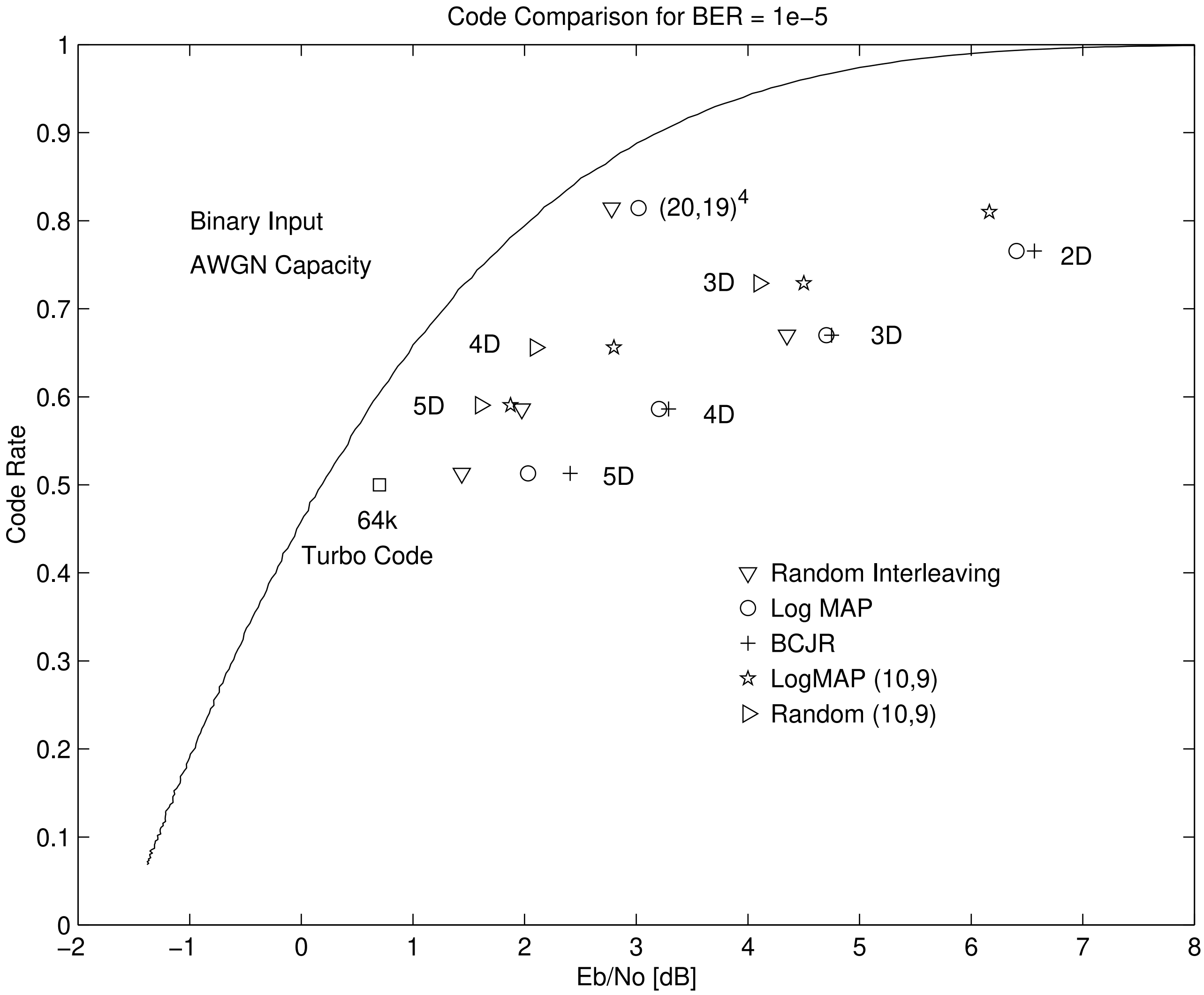

Figure 10: A comparison of various normal and randomly interleaved SPC product codes ffrom the d-dimensional $(10, 9)$ and $(8, 7)$ families of codes , $d = 2..5$.