
© 1993 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.
SEPARABLE MAP "FILTERS" FOR THE DECODING
OF PRODUCT AND CONCATENATED CODES
J. Lodge1, R. Young1, P. Hoehler2, J. Hagenauer2
1Communications Research Centre, Ottawa, Canada
2German Aerospace Research Establishment (DLR), Oberpfaffenhofen, Germany

Abstract
Very efficient signalling in radio channels requires the design
of very powerful codes having special structure suitable for
practical decoding schemes. In this paper, powerful codes are
obtained by using simple block codes to construct
multidimensional product codes. The decoding of
multidimensional product codes, using separable symbol-by-
symbol maximum a posteriori (MAP) "filters", is described.
Simulation results are presented for three-dimensional product
codes constructed with the (16,11) extended Hamming code.
The extension of the concept to concatenated convolutional
codes is given and some simulation results are presented.
Potential applications are briefly discussed.

1. Introduction
In practice, very efficient signalling in radio channels
requires more than the design of very powerful codes. It
requires designing very powerful codes that have special
structure so that practical decoding schemes can be used with
excellent (but not necessarily truly optimal) results. Examples
of two such approaches include the concatenation of
convolutional and Reed-Solomon coding, and the use of very
large constraint-length convolutional codes with reduced-state
decoding. In this paper, an alternate approach is introduced.
The initial simulation results are very encouraging.

The work discussed in this paper was motivated by
concepts introduced in [1] for the decoding of concatenated
convolutional codes. In that paper it is shown that symbol-by-
symbol MAP decoding for the inner code allows soft decisions
to be passed to the outer decoder, resulting in impressive
performance. The inner decoding algorithm can be thought of
as a type of nonlinear filter that accepts as its input a noisy
signal. Then it makes use of the structure inherent in the inner
code to produce a noisy output "decoded" signal (that is
hopefully less corrupted in some sense than the original input
signal). Here we apply and extend the same philosophy for
multidimensional product codes. A logical extension to the
case of concatenated convolutional codes is then proposed.

The organization of this paper is as follows. In Section 2
the background behind the concept is summarized. We discuss
MAP "filtering" for block codes and the decoding of
multidimensional product codes, using separable MAP "filters".
Simulation results for a multidimensional block code are
presented in Section 3. Then, in Section 4 convolutional
coding is considered, and potential applications are discussed in
Section 5.

Figure 1. A block diagram of the system model.

2. Background
The symbol-by-symbol MAP algorithm can be used for
codes that can be represented by a trellis of finite duration. It is
well known that decoding trellises exist for convolutional codes
[2] as well as for linear block codes [3]. For the system model
shown in Figure 1, we provide a brief summary of the symbol-
by-symbol MAP algorithm as given in [4] and the appendix of
[2]. The simple time-invariant 4-state trellis, shown in Figure 2,
is used to illustrate the concepts. This trellis corresponds to a
rate-1/3 convolutional code. In general (e.g., for block codes)
the trellis may be time-varying with the number of states, Mt,
being a function of the time index t. It is assumed that at the
start and the end of the time interval of interest, the coder is in
the zero state. Any given input sequence Dt of binary (e.g., 0
or 1) k-vectors, that satisfies the above end conditions, will
correspond to a particular path through the trellis that is
described by a sequence of states
\[ S_{t-1} = [S_{t-1} = 0, ... , S_t = m, ... , S_{t+1} = 0] \]  
where \[ S_t \epsilon [0, ... , M_t - 1]\].

For each path through the trellis the coder produces a
particular channel input sequence
\[ X_t = [X_t, ... , X_{t+1}] \]  
where \[ X_t \] is a n-vector denoted by
\[ X_t = [x_{t}, ..., x_{t+n}] \]  
of binary (e.g., -1 or +1) elements. In the example trellis of
Figure 2, k=1, n=3 and M=4 for all t. For notational
convenience, the functional dependence of \[ X_t \] on \[ S_{t-1} \] and \[ S_t \] is
only shown when required. The corresponding channel output
sequence is given by
\[ Y_t = [Y_t, ... , Y_{t+n}] \]  
where \[ Y_t \] is an n-vector denoted by
\[ \alpha_i(m) = \sum_{m'=0}^{M-1} \alpha_{i-1}(m') \gamma_i(m',m) \]  
\[ \beta_i(m) = \sum_{m'=0}^{M-1} \beta_{i+1}(m') \gamma_{i+1}(m,m') \]  

Here we refer to \( \gamma_i(m',m) \) as the branch probability and it is given by

\[ \gamma_i(m',m) = \Pr(S_i = mlS_{i-1} = m') \prod_{j=1}^{n} p(y_j \mid x_j(m',m)) \]

where the first term on the right-hand side is usually a straightforward function of the probability distribution of the input data and the coder structure. The second term on the right-hand side is a product of conditional symbol probability densities as given in equation (6). The branch probabilities account for the "present" n-vector of channel symbol probability densities, while the "past" channel outputs are accounted for by the forward recursion defined by equation (11), and the future channel outputs are accounted for by the backward recursion in equation (12).

Consider applying these techniques to obtain the a posteriori probabilities of the coded bits (i.e., the elements of \( X_j \)) rather than on the information bits (i.e., the elements of \( D_j \)). If the coded bits are assumed to be independent, with \( p_0 \) and \( p_1 \) being the probability that any given bit is a 0 or 1, respectively, then

\[ p(x_j = 0; \mathcal{Y}_j) = p(x_j = 0; y_j) = p(y_j \mid x_j) p_0 \]

However, the coded bits are not independent due to the structure imposed by the coder. Consequently, we would like to use the MAP processing to determine the probabilities, \( p(x_j = 0; \mathcal{Y}_j \mid C) \), where the conditioning on \( C \) refers to the knowledge of the coding structure. This can easily be done by defining the set of all transitions for which \( x_j = 0 \);

\[ A = \{(m',m) : x_j (m',m) = 0\} \]

and then summing over the joint transition probabilities to obtain the joint probability

\[ p(x_j = 0; \mathcal{Y}_j \mid C) = \sum_{(m',m) \in A} \sigma_i(m',m). \]

The noisy codeword enters the MAP "filter" as a vector of independent probabilities, and then is output from the filter with the probabilities (which are no longer independent) being refined according to the structure of the code. A similar procedure can be used for determining the probability that the information bit \( d_j \) is zero by replacing the set \( A \) by

\[ A' = \{(m',m) : d_j (m',m) = 0\}. \]

1741
In this paper, we distinguish between the terms "MAP filter" and "MAP decoder", with the former computing the \( a \) \( \text{posteriori} \) probabilities of the coded bits and the latter the \( a \) \( \text{posteriori} \) probabilities of the decoded bits. (Clearly for systematic codes, the \( a \) \( \text{posteriori} \) probabilities of the information bits are a subset of the probabilities for the coded bits.) If hard decisions are performed on the output of the MAP filter, the minimum average probability of coded bit error is achieved. However, the resulting word may not be a valid code word. A good choice for a valid codeword can be obtained by iterating the filtering operation until a valid code word is obtained. Of course, the assumption of independent probabilities by the MAP algorithm is erroneous when the algorithm is used iteratively. Nevertheless, the MAP algorithm demonstrates an ability to "capture" valid codewords when it is iterated. However, valid codewords selected in this way are not truly optimal by any commonly-used criterion.

We first consider symbol-by-symbol MAP “filtering” for block codes. Two linear block codes can be used to construct a more powerful code by constructing a product code with them. In this case the new codeword can be viewed as a rectangular matrix (or equivalently a two-dimensional array) with each row being a codeword of one of the linear block codes, while each column is a codeword of the other code. The rate, Hamming distance, and asymptotic coding gain of the product code are equal to the products of those quantities for the individual codes. While the following condition is not necessary, identical systematic codes will be assumed in both dimensions here. Therefore, the overall code word is a square array, with parity bits along two of its four sides, and the distance and rate of the overall code are the squares of those values for the original code. Clearly this concept can be extended to higher dimensions. For example, in three dimensions the overall codeword is a cubic array with parity bits along three of the six sides, while in four dimensions the overall codeword is a four-dimensional hypercube with parity bits along four of the eight sides.

Note that the product code can be thought of as a special case of concatenated coding with interleaving between the coding stages. For example, a two-dimensional product code can be viewed as concatenated coding with a block interleaver that has a number of columns equal to the word length of the component code, and a number of rows equal to the number of information bits. The special property of this choice of interleaver/coding arrangement is that component codewords for each of the component codes can be obtained at the output of the final stage, by simply subdividing the output symbols into different subsets. This property is important for generalizing the separable MAP filtering concept to concatenated convolutional codes (see Section 4).

In multidimensional signal processing, digital filtering is often performed using “separable” filters. That is, in order to avoid excessive computational requirements, one-dimensional filtering is performed sequentially in each of the \( N \) dimensions, rather than performing a single massive \( N \)-dimensional digital filter. In this paper, we investigate the analogous approach for the decoding of multidimensional product codes. That is, one-dimensional MAP filters will be used sequentially in each dimension. Consider the two-dimensional case first. One-dimensional MAP filtering can be done across the rows giving a new set of refined probabilities, taking into account only the horizontal structure of the overall code. These new probabilities (for which the elements in a column are still independent) are then further refined by one-dimensional MAP filtering down the columns to complete a single filtering cycle. This process can be iterated any number of times. The extension to the cases with more than two dimensions is obvious, with a single filtering cycle consisting of sequentially performing one-dimensional filtering in each of the dimensions.

In the multidimensional signal processing case, iterating the filtering does not make sense because the filters are linear. However, in the product coding case, the filters are highly nonlinear and additional filtering cycles can significantly improve the performance. Note that the independence assumption for each MAP filter operation is valid for the entire first cycle. Thus, up to this point the probabilities computed by each MAP filter operation are the "real probabilities" of symbol error. For subsequent iterations the independence assumption is erroneous.

Straightforward application of the iterative MAP filtering algorithm for the decoding of multidimensional product codes can produce very good results [5]. However, in many cases one of a number of variations of the MAP filtering approach can yield improved performance. Here, we will outline two such variations.

The first variation makes use of the following factorization. From equations (9), (13), (14), and (16), it can be shown that

\[
\begin{align*}
\rho(x_{1j} = 0; y_i | C) &= \\
&= \rho(x_{1j} = 0; y_{ij} | C)
\end{align*}
\]

\[
\sum_{(m, m') \in A} \left[ (p_0)^{-1} \alpha_{i-1} (m') \beta_{i} (m) \Pr \{ S_i = m | S_{i-1} = m' \} \right]
\]

\[
\prod_{q \neq j} \rho(y_{iq} | x_{iq} (m', m))
\]

where the first term is dependent only upon the \( j \)-th element at time \( t \) and the summation is independent of this element and therefore represents the "refinement factor" due to the structure of the coder. In general such a factorization is not possible with the APP because it is necessary to divide equation (18) by \( p(y_i | \cdot) \) which is dependent upon all of the elements in the time interval of interest. However, a similar expression exists for \( p(x_{1j} = 1; y_i | C) \) and the \( p(y_i | \cdot) \) terms cancel when the likelihood ratio, given by,
\[
L = \frac{\Pr(x_j = 0; Y^r_t; C)}{\Pr(x_j = 1; Y^r_t; C)} = \frac{p(x_j = 0; Y^r_t; C)/p(Y^r_t)}{p(x_j = 1; Y^r_t; C)/p(Y^r_t)}
\]

(19)

is computed. Therefore the likelihood ratio factors into the form

\[
L_{t_j}(1) = f_{t_j}(1)L_{t_j}(0),
\]

(20)

for one iteration and

\[
L_{t_j}(K) = \left[ \prod_{q=1}^{K} f_{t_j}(q) \right] L_{t_j}(0)
\]

(21)

for \( K \) iterations, where \( f_{t_j}(q) \) represents the refinement factor for the \( q \)-th iteration. Note that the above factorization is only applicable to MAP filtering and the MAP decoding of systematic codes, and is not generally applicable to MAP decoding. "Partial factor MAP filtering" is a variation for which the input to the MAP filters, for component codes in a given dimension, includes only those refinement factors that correspond to the filtering passes in the other dimensions. This variation reduces the error introduced by the independence assumption in the MAP processing.

A second variation takes advantage of the fact that the probabilities resulting from one complete filtering cycle are the last truly valid probabilities (because the independence assumption for each MAP filter operation is valid up to this point of the processing). The resulting "real probabilities" are saved, and for subsequent cycles equation (9) is replaced by

\[
\sigma_j(m', m) = \alpha_{j-1}(m') \beta_j(m) \gamma_j(m', m)
\]

(22)

where the modified branch probability \( \gamma_j \) is computed using the "real probabilities". The forward and backward recursions are still computed using the filtered probabilities.

3. Block Coding Example

In this section a three-dimensional product code constructed using the (16,11) extended Hamming code is considered. The one-dimensional MAP filter for this code has 32 states. The three-dimensional product code has a rate of 0.325 and an asymptotic coding gain of 13.18 dB. Monte Carlo simulations where performed for which 100 codewords, each with 1331 information bits, were processed for a range of signal-to-noise ratios. Figure 3 shows the number of codewords containing errors for one-cycle, three-cycle, and six-cycle MAP filtering. A couple of points are evident. Firstly, impressive performance can be achieved by constructing multidimensional product codes using comparatively simple component codes. Partial factor MAP filtering was used here. The results are notably better than those presented in [5] for full factor MAP filtering. Secondly, the improvement due to iterative filtering can be very dramatic. Note that the results for one cycle of filtering correspond to the more conventional approach to the decoding of concatenated codes, where soft-in soft-out decoding algorithms are used. The capability of iterative filtering to capture the correct codeword is quite striking.

4. Extension to Convolutional Codes

In this paper, very good performance has been achieved by constructing multidimensional product codes, using fairly simple block codes. The ability to perform iterative MAP filtering becomes more important as the number of dimensions increase. For many radio applications convolutional coding has proven to be more attractive than block coding. Therefore it is natural to seek similar techniques for convolutional codes. Convolutional codes are linear if the encoders are started in the all-zero state. Therefore it is possible to treat them as long block codes and construct massive product codes with them. While these huge powerful block codes may be appropriate for some applications, for other applications an approach that results in sequential processing of the in-coming signal would be more attractive. One potential approach is to search for a method of concatenating the convolutional codes in such a way that the combination of coding and interleaving results in codewords from all of the component codes being present in the final output. That is, the final output can be subdivided into valid codewords for any one of the component codes by appropriately grouping the output bits into symbols. Here, we describe such a technique that results in the construction of very large powerful convolutional codes by appropriately combining smaller component convolutional codes.

![Figure 3. The number of codewords containing errors for 1 cycle, 3 cycles, and 6 cycles of MAP filtering. A total of 100 codewords was processed for each value of \( E_b/N_0 \), with each codeword containing 1331 information bits.](image-url)
The first important observation is that convolutional encoders are linear and shift-invariant [6]. Therefore a sum of valid codewords, each with a different delay, is still a valid codeword. The second is that time-division interleaving can be implemented as is illustrated in Figure 4. Note that this structure does not destroy the time-invariant property, unlike most interleaving schemes. Therefore this type of combined encoder/interleaver can be used as a building block for the type of composite code that is desired. This concept is illustrated in Figure 5 for a two-tier example code. Each tier contains a number of identical coders with inputs interconnected to the coder outputs of the previous tier. The interconnection must be done in a manner such that the codewords arriving from the previous tier are linearly transformed through the current tier.

Here, we develop such an interconnection using a recursive approach. Starting with a rate $k_1/n_1$ convolutional coder at the first tier, we wish to add a second tier consisting of rate $k_2/n_2$ coders. In our interconnection there will be $k_2$ coders at tier 1 and $n_1$ coders at tier 2. The concatenation of tier 1 and tier 2 is treated as a supercoder of rate $k'_2/n'_2 = k_1k_2/n_1n_2$. To connect a third tier of rate $k_3/n_3$ coders we repeat the above process. There will be $k_3$ supercoders at tier 2 and $n'_2$ coders at tier 3 and after the interconnection this will produce a supercoder of rate $k''_3/n''_3 = k_2k_3/n'_2n_3$. In general, interconnecting tier $i$ to tier $i+1$ requires $k_{i+1}$ supercoders at tier $i$ and $n'_i$ coders at tier $i+1$. This concatenation is treated as a supercoder of rate $k''_{i+1}/n''_{i+1}$ for subsequent interconnections. The final supercoder resulting from concatenating $N$ tiers of convolutional coders has a rate

$$
\frac{k'_N}{n'_N} = \frac{\prod_{i=1}^{N} k_i}{\prod_{i=1}^{N} n_i}.
$$

(23)

The actual interconnection of tier $i$ to tier $i+1$ is straightforward. If we denote the $j$th coder at stage $i$ as $c_{ij}$, then our interconnection strategy is to connect the $m$th output of supercoder $c_{ij}$ to the $j$th input of coder $c_{i+1,j,m}$.

The individual codewords from the convolutional coders are dispersed as they propagate through subsequent tiers. In order to facilitate MAP filtering, we must be able to construct valid codewords from each tier. Let us denote the output sequence of $n'_N$ bits as

$$
(b(0), b(1), b(2), \ldots, b(n'_N - 1))
$$

Then, the $m$th code symbol from the $i$th tier is

$$
(b(m), b(m + p), b(m + 2p), \ldots, b(m + (n_i - 1)p))
$$

where

$$
p = \begin{cases} 
\prod_{j=1}^{i-1} n_j, & \text{for } i < N \\
1, & \text{for } i = N
\end{cases}
$$

(24)

and

$$
m = \left\{ 0, 1, 2, \ldots, \frac{n'_N}{n_i} - 1 \right\}.
$$

(25)

Note that each of the component codewords (appropriately interleaved) is present at the output. The purpose of the interleaving is to make the distance of the composite code approximately proportional to the product of the component codes. Long interleaver depths can be achieved if the interleaving factors for the tiers are chosen to be mutually prime.

In processing a continuous stream of received bits, some form of block processing is necessary because receiver memory and delay are not unlimited. However, by nature, convolutional codes are not ideally suited to block processing. Our strategy is to overlay a two segment processing window onto the incoming stream. The first segment of the window
identifies the portion of bits that will be decoded and the second segment acts as a view into the future for the processing. After each decoding process is completed, the window is moved forward to the position just past the last decoded bit. The forward and backward recursions of the MAP processing are performed over the entire window, however, the decoding phase does not output bits from the future segment.

Many good convolutional codes are not systematic. If the chosen component codes are not systematic, MAP decoding should be used in the final cycle in order to recover the information bits.

As an example consider the two-tier concatenated code, as shown in Figure 5, constructed using rate-2/3 16-state systematic convolutional codes. Here, the values of 9 and 10 have been selected for the interleaving depths $m_1$ and $m_2$, respectively. The resulting concatenated code has a rate of $4/9$. The results for 1, 2, 4 and 8 cycles of MAP processing, using the “real probabilities” to compute the branch probabilities, can be seen in Figure 6. Given the modest complexity of this particular code, the results are quite impressive. As with the block coding example, the benefits of the additional cycles of MAP processing are clear.

5. Discussion

The initial simulation results, for the application of MAP filtering to multidimensional product codes, are quite encouraging. These ideas were extended to convolutional codes, for which component convolutional codes are combined to form a large convolutional code. Promising simulation results were presented for this type of code.

As would be expected with such powerful coding techniques, the decoding process is quite computationally intensive. Therefore, the development of efficient implementation techniques is an important area for future work. For some codes, it is possible that simpler algorithms (e.g., [1]) can replace the MAP processing without severely degrading the performance.

A number of potential application areas are evident. Firstly, powerful codes with a high coding rate can be implemented. For example, the four-dimensional code, constructed with (25,24) single parity bit component codes, has a rate of 0.85 and an asymptotic coding gain of 11.3 dB. Secondly powerful codes with relatively short block lengths can be implemented for packet data applications. Perhaps the ultimate example is the two-dimensional code constructed with the (24,12) extended Golay code, which provides an asymptotic coding gain of 12 dB with a wordlength of only 144 information bits. The third application is for extremely power efficient coding (e.g., deep-space applications).

References


