

# MAP DECODING OF LINEAR BLOCK CODES BASED ON A SECTIONALIZED TRELLIS OF THE DUAL CODE

Peter J. Schreier, Daniel J. Costello, Jr.

Dept. of Electrical Engineering, Univ. of Notre Dame, Notre Dame, IN 46556

## Abstract

Block codes for use in turbo coding schemes provide an alternative to punctured convolutional codes when high rate component codes are needed. Since block codes have large, time-varying trellis diagrams, full Maximum A Posteriori (MAP) Soft-In Soft-Out decoders are very complex. It is shown how to modify the MAP algorithm to utilize a sectionalized trellis diagram of the dual code for decoding, which minimizes computational complexity for high rate component codes. This paper also gives simulation results for some high rate block turbo codes.

## 1 Introduction

Convolutional codes have received a great deal of attention as component codes in parallel concatenated (turbo) coding schemes. However, many applications require block lengths in the range of a few hundred to a few thousand bits. For short block lengths, the rate loss caused by the trellis termination of convolutional codes is no longer negligible. Turbo codes that employ block component codes do not suffer a rate loss, but their decoding can be extremely complex because block codes typically have large, time-varying trellis diagrams.

The aim of this paper is to modify the optimal MAP decoder for a block component code such that it minimizes computational complexity. Since parallel concatenation reduces the resulting overall rate of the code, our interest is focused on high rate block codes with  $k > n - k$ . This paper will demonstrate that it is most advantageous to use a sectionalized trellis diagram of the dual code to decode high rate block codes. For an extended BCH(32,21) code, decoding based on the optimum sectionalized trellis of the dual code saves 73% in real operations compared to decoding based on the original bit-level trellis.

The paper is organized as follows:

Section 2 contains a short introduction to sectionalized trellises. It introduces parameters to describe the complexity of a trellis diagram.

Section 3 presents in detail how to use the sectionalized trellis diagram of the dual code in a MAP Soft-In Soft-Out (SISO) decoding scheme that accepts and delivers soft values. A SISO decoder is the heart of a turbo decoding scheme. Section 3.1 shows how to most efficiently implement the MAP algorithm based on the trellis of the dual code, and the number of required floating

point operations and storage locations is derived. In section 3.2 it is demonstrated how to choose a sectionalization that minimizes the computational complexity. As examples, complexity measures for decoding an RM(32,26) code and an extended BCH(32,21) code are given.

Section 4 presents simulation results for RM(32,26) and extended BCH(32,21) turbo codes, and investigates the influence of interleavers.

## 2 Sectionalized Trellises

Consider an  $(n, k)$  linear block code  $C$  that has a minimal bit-level trellis. In a bit-level trellis, each state transition is associated with one code bit. In a sectionalized trellis, code bits are grouped together in sections, using boundary locations  $Q = \{q_0 = 0, q_1, q_2, \dots, q_{V-1}, q_V = n\}$  in the trellis, where  $q_{i-1} < q_i$ ,  $1 \leq i \leq V$ , and  $V$  is the total number of sections. The sectionalized trellis of the code  $C$  using boundary locations  $Q$  is denoted by  $T_C(Q)$ . A section of the trellis consists of the set of all states at time instants  $q_{i-1}$  and  $q_i$ , denoted by  $S_{q_{i-1}}$  and  $S_{q_i}$ , respectively, and all transitions between states  $s_{q_{i-1}} \in S_{q_{i-1}}$  and states  $s_{q_i} \in S_{q_i}$ . There are  $v_i = q_i - q_{i-1}$  code bits in section  $i$ , forming the sequence  $\mathbf{x}_i = (x_{q_{i-1}+1}, \dots, x_{q_i})$ . In a sectionalized trellis there can be parallel branches connecting two states, forming a composite branch. Let  $X(s_{q_{i-1}} \rightarrow s_{q_i})$  be the set of all code sequences  $\mathbf{x}_i$  that correspond to the transition  $s_{q_{i-1}} \rightarrow s_{q_i}$ .

There are  $0 \leq z_i \leq v_i$  information bits associated with section  $i$ . We denote the  $j$ th information bit in the  $i$ th section as  $u_{i,j}$ . Similarly  $x_{i,m}$  and  $y_{i,m}$  stand for the  $m$ th code bit and  $m$ th received bit in the  $i$ th section, respectively. Observe that  $1 \leq i \leq V$ ,  $1 \leq j \leq z_i$ , and  $1 \leq m \leq v_i$ . If there is only one section, we drop the subscript  $i$ .

Let  $C_{a,b}$ ,  $0 \leq a < b \leq n$ , denote the subcode of  $C$  consisting of those codewords in  $C$  whose nonzero components are confined to the positions in the set  $\{a+1, a+2, \dots, b\}$ . Let  $\dim C$  denote the dimension of the linear code  $C$ . Then the number of states  $|S_{q_i}|$  at time instant  $q_i$  in the trellis is given by [6]

$$\log_2 |S_{q_i}| = \dim C - \dim C_{0,q_i} - \dim C_{q_i,n}. \quad (1)$$

The measure  $\log_2 |S_{q_i}|$  is referred to as the state space dimension at time  $q_i$ . Let  $p_{a,b}(C)$ ,  $0 \leq a < b \leq n$ , denote the punctured code of length  $b - a$  obtained from  $C$  by removing the first  $a$  and the last  $n - b$  components of each codeword in  $C$ . The number of distinct composite

branches  $v_{dist,i}$  in the  $i$ th section of a trellis is obtained by

$$\log_2 v_{dist,i} = \dim p_{q_{i-1},q_i}(C) - \dim C_{q_{i-1},q_i}. \quad (2)$$

The total number of composite branches  $v_{comp,i}$  in the  $i$ th section is obtained by

$$\log_2 v_{comp,i} = \dim C - \dim C_{0,q_{i-1}} - \dim C_{q_i,n} - \dim C_{q_{i-1},q_i}. \quad (3)$$

Finally, the number of parallel branches  $v_{par,i} = |X(s_{q_{i-1}} \rightarrow s_{q_i})|$  between two adjacent states  $s_{q_{i-1}}$  and  $s_{q_i}$  forming a composite branch is given by

$$\log_2 v_{par,i} = \dim C_{q_{i-1},q_i}. \quad (4)$$

The trellis of the dual code has the same state space dimension at each time instant as the original code [6]. Still, since  $C$  has  $k$  information bits and the dual code  $C^\perp$  has  $n - k$  information bits, the total number of branches differs. If we consider high rate codes with  $k > n - k$  the trellis of  $C^\perp$  will be very sparse, so that the overall branch complexity of the trellis of  $C^\perp$  will be much smaller than for the trellis of  $C$ .

The trellis complexity parameters depend on the order of code bits, i.e., on the permutation of columns in the generator matrix  $\mathbf{G}$ . We are interested in a column ordering that produces a trellis that can be decoded with minimal complexity. A permutation that yields the smallest state space dimension at every time of the code trellis and the smallest overall branch complexity is called an optimum permutation [6]. In an optimum permutation, information bits do not necessarily correspond to the first  $k$  positions in the code bit. For turbo decoding we need a representation of  $\mathbf{G}$  that still exhibits a separation of information and parity bits. Such a form is called quasi-systematic. For example, the optimum permutation of the RM(8,4) code has the information bits at positions  $I = \{1, 2, 3, 5\}$ . A quasi-systematic form is given by

$$\mathbf{G} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \end{bmatrix}. \quad (5)$$

For a quasi-systematic code, there is a mapping  $m_i(j) : j \rightarrow m$  that gives the position of the code bit that is associated with information bit  $u_{i,j}$  within section  $i$ . For example, if we use a trellis of the RM(8,4) code with boundary locations  $\mathcal{Q} = \{0, 2, 8\}$ , then  $m_1(1) = 1$ ,  $m_1(2) = 2$ ,  $m_2(1) = 1$ , and  $m_2(2) = 3$ .

### 3 MAP Decoding Based on the Sectionalized Trellis of the Dual Code

Using log-likelihood values (“L-values”), the MAP decoder must provide us with a posteriori values

$$L(\hat{u}_j) = L(u_j|\mathbf{y}) = \log \frac{p(u_j = +1|\mathbf{y})}{p(u_j = -1|\mathbf{y})}. \quad (6)$$

For high rate codes it is often advantageous to use the trellis of the dual code to evaluate (6). Hartmann and Rudolph [4] found a way to calculate the probabilities  $p(u_j = \pm 1|\mathbf{y})$  using the codewords of the dual code  $C^\perp$ . The idea is to express  $p(u_j = \pm 1|\mathbf{y})$  as a function over the whole code space and to use the finite Fourier Transform. Hartmann and Rudolph were not interested in soft values, however, so Hagenauer *et al.* [3] adapted their idea to deliver and accept soft values. A more intuitive approach that yields the same result was given by Battail *et al.* in [1]. They use the nonhomogeneous polynomial representation of the code  $C$  to transform the MAP decision rule into the dual domain. Below, we show how to utilize the sectionalized trellis diagram of the dual code,  $T_{C^\perp}(\mathcal{Q})$ , to evaluate the transformed equation. It is essential that the notation employed be very strict. All symbols that refer to the dual code are denoted by a tilde above the symbol. All symbols without a tilde still refer to the original code. It is especially important to note that the mapping  $m_i(j)$  is still defined for the original code. According to [3], the MAP rule (6) can be expressed in the dual domain as

$$L(\hat{u}_j) = L_c y_{m(j)} + L(u_j) + L_e(\hat{u}_j), \quad (7)$$

where  $L(u_j)$  is a priori information, and the extrinsic information  $L_e(\hat{u}_j)$  is given by (8) at the bottom of the next page. In (8), we used

$$L(x_m; y_m) = \begin{cases} L_c \cdot y_m + L(u_j), & \text{for info bits} \\ L_c \cdot y_m, & \text{otherwise.} \end{cases} \quad (10)$$

The term  $L_c$  is called channel reliability factor. For an AWGN channel,  $L_c = 4E_s/N_0$ . Note that in (7) a posteriori values are evaluated for information bits only, which is sufficient for turbo decoding. For serially concatenated product codes, however, a posteriori values are needed for all code bits. This obviously requires more real operations, but actually simplifies notation, since we don't need the mapping  $m_i(j)$  later on.

In (8), we have introduced the notation  $\langle a \rangle_x$  to denote a value equal to  $a$  if  $x = -1$  and  $1$  if  $x = +1$ . In a product this simply means that  $x$  decides if  $a$  is included in the product or not. Note that (7) is the sum of three independent estimates for  $L(\hat{u}_j)$ : the received bit  $L_c y_{m(j)}$ , the a priori information  $L(u_j)$ , and the extrinsic information  $L_e(\hat{u}_j)$ . However, a closed form that does not extract the extrinsic information and thus includes  $m(j)$  in the products does not exist in the dual domain. We will see that the absence of such a closed form diminishes the computational savings achievable by employing sectionalized trellises.

Equation (8) does not make use of a trellis diagram yet. However, it is possible to utilize a sectionalized trellis of the dual code to evaluate (8). Let  $\tilde{\sigma}' = \tilde{s}_{q_{i-1}}$  be the state at the left and  $\tilde{\sigma} = \tilde{s}_{q_i}$  the state at the right boundary location of section  $i$  in the trellis of the dual code. We define the branch transition probability from state  $\tilde{\sigma}'$  to  $\tilde{\sigma}$  in section  $i$  of  $T_{C^\perp}(\mathcal{Q})$  as

$$\tilde{\gamma}_i(\tilde{\sigma}', \tilde{\sigma}) = \sum_{\tilde{x}_i \in \tilde{X}(\tilde{\sigma}' \rightarrow \tilde{\sigma})} \prod_{m=1}^{v_i} \langle \tanh(L(x_{i,m}; y_{i,m})/2) \rangle_{\tilde{x}_{i,m}}. \quad (11)$$

The sum here accounts for possible parallel branches between two states  $\tilde{\sigma}'$  and  $\tilde{\sigma}$ . To allow the extraction of the  $j$ th information bit in section  $i$ , we also define a branch transition operation that does not consider this information bit or its transmitted value as

$$\tilde{\gamma}_{i,j}^X(\tilde{\sigma}', \tilde{\sigma}) = \sum_{\tilde{\mathbf{x}}_i \in \tilde{X}(\tilde{\sigma}' \rightarrow \tilde{\sigma})} \prod_{\substack{m=1 \\ m \neq m_i(j)}^{v_i}} \langle \tanh(L(x_{i,m}; y_{i,m})/2) \rangle_{\tilde{x}_{i,m}} \quad (12)$$

If we include only those transitions in the computation that correspond to  $\tilde{x}_{i,m_i(j)} = +1$  or  $\tilde{x}_{i,m_i(j)} = -1$ , we denote the transition operation by  $\tilde{\gamma}_{i,j}^{X^+}(\tilde{\sigma}', \tilde{\sigma})$  or  $\tilde{\gamma}_{i,j}^{X^-}(\tilde{\sigma}', \tilde{\sigma})$ , respectively. Obviously,  $\tilde{\gamma}_{i,j}^X(\tilde{\sigma}', \tilde{\sigma}) = \tilde{\gamma}_{i,j}^{X^+}(\tilde{\sigma}', \tilde{\sigma}) + \tilde{\gamma}_{i,j}^{X^-}(\tilde{\sigma}', \tilde{\sigma})$ . The extrinsic information is then given by (9) at the bottom of this page.

The state probabilities  $\tilde{\alpha}_i(\tilde{\sigma})$  and  $\tilde{\beta}_i(\tilde{\sigma})$  are found by a forward recursion and a backward recursion:

$$\tilde{\alpha}_i(\tilde{\sigma}) = \sum_{\tilde{\sigma}'} \tilde{\gamma}_i(\tilde{\sigma}', \tilde{\sigma}) \cdot \tilde{\alpha}_{i-1}(\tilde{\sigma}'), \quad (13)$$

$$\tilde{\beta}_{i-1}(\tilde{\sigma}') = \sum_{\tilde{\sigma}} \tilde{\gamma}_i(\tilde{\sigma}', \tilde{\sigma}) \cdot \tilde{\beta}_i(\tilde{\sigma}). \quad (14)$$

### 3.1 Efficient Implementation and Complexity of the Algorithm

We now consider an efficient implementation of the presented algorithm on a computer, where memory requirements are not a first priority, and determine its complexity. When counting floating point operations, divisions are assumed to have the same complexity as multiplications, and subtractions the same as additions. Furthermore, we assume that the log and tanh function values are stored in look-up tables. Complexity measures that refer to  $T_{C^\perp}(Q)$  are denoted by a tilde above the symbol. Keep in mind, however, that  $z_i$  still means the number of information bits in the  $i$ th section of  $T_C(Q)$ .

**Step 1: Computing the transition probabilities**  $\tilde{\gamma}_i(\tilde{\sigma}', \tilde{\sigma})$ . This is done using (11). We define

$$\hat{\gamma}_i(\tilde{\mathbf{x}}_i) = \prod_{m=1}^{v_i} \langle \tanh(L(x_{i,m}; y_{i,m})/2) \rangle_{\tilde{x}_{i,m}}. \quad (15)$$

First,  $k$  additions are required to compute the  $L(x_{i,m}; y_{i,m})/2$  values from (10). The division by two can be achieved by a binary shift.

Then, for all sections  $i$ :

- Evaluate the terms  $\hat{\gamma}_i(\tilde{\mathbf{x}}_i)$  for all branches with distinct labeling  $\tilde{\mathbf{x}}_i$ , and save them in a temporary table. Only the  $\tanh(\cdot)$  terms that correspond to  $\tilde{x}_{i,m} = -1$  are included in the product. Since each code bit  $\tilde{x}_{i,m}$  is equal to  $-1$  in exactly half the codewords, the construction of the temporary table involves a total of  $(v_i/2 - 1) \cdot \tilde{v}_{dist,i} \cdot \tilde{v}_{par,i} + 1$  multiplications if  $v_i > 1$ , and none otherwise. In this equation, the  $+1$  term is necessary, because the all-“ $-1$ ”/all-“ $+1$ ” codeword pair together requires one more multiplication than the average of the remaining codeword pairs.
- If there are parallel branches in section  $i$ , the computation of  $\tilde{\gamma}_i(\tilde{\sigma}', \tilde{\sigma})$  requires  $\tilde{v}_{dist,i} \cdot (\tilde{v}_{par,i} - 1)$  additions of  $\hat{\gamma}_i(\tilde{\mathbf{x}}_i)$  terms.

- The transition operation that does not consider the code bit  $\tilde{x}_{i,m_i(j)}$  was defined in (12). We set

$$\hat{\gamma}_{i,j}^X(\tilde{\mathbf{x}}_i) = \prod_{\substack{m=1 \\ m \neq m_i(j)}^{v_i}} \langle \tanh(L(x_{i,m}; y_{i,m})/2) \rangle_{\tilde{x}_{i,m}}. \quad (16)$$

Construct a temporary table  $\hat{\gamma}_{i,j}^X(\tilde{\mathbf{x}}_i)$  for all branches with distinct labeling  $\tilde{\mathbf{x}}_i$  and all  $1 \leq j \leq z_i$ . The construction of this table does not require floating point operations if  $v_i \leq 2$ . In this case,  $\hat{\gamma}_{i,j}^X(\tilde{\mathbf{x}}_i)$  is either directly given by a single tanh term or is equal to 1. Otherwise, it is best to use the  $\hat{\gamma}_i(\tilde{\mathbf{x}}_i)$  table from above to compute

$$\hat{\gamma}_{i,j}^X(\tilde{\mathbf{x}}_i) = \frac{\hat{\gamma}_i(\tilde{\mathbf{x}}_i)}{\langle \tanh(L(x_{i,m_i(j)}; y_{i,m_i(j)})/2) \rangle_{\tilde{x}_{i,m_i(j)}}} \quad (17)$$

which involves  $z_i/2 \cdot \tilde{v}_{dist,i} \cdot \tilde{v}_{par,i}$  multiplications.

- Finally, if there are parallel branches in section  $i$ , the computation of  $\tilde{\gamma}_{i,j}^{X^+}(\tilde{\sigma}', \tilde{\sigma})$  and  $\tilde{\gamma}_{i,j}^{X^-}(\tilde{\sigma}', \tilde{\sigma})$  requires  $2z_i \cdot \tilde{v}_{dist,i} \cdot (\tilde{v}_{par,i}/2 - 1)$  additions.

$$L_e(\hat{u}_j) = \log \frac{\sum_{\tilde{\mathbf{x}} \in C^\perp} \prod_{\substack{m=1 \\ m \neq m(j)}^n \langle \tanh(L(x_m; y_m)/2) \rangle_{\tilde{x}_m}}}{\sum_{\substack{\tilde{\mathbf{x}} \in C^\perp \\ \tilde{x}_{m(j)} = +1}} \prod_{\substack{m=1 \\ m \neq m(j)}^n \langle \tanh(L(x_m; y_m)/2) \rangle_{\tilde{x}_m}} - \sum_{\substack{\tilde{\mathbf{x}} \in C^\perp \\ \tilde{x}_{m(j)} = -1}} \prod_{\substack{m=1 \\ m \neq m(j)}^n \langle \tanh(L(x_m; y_m)/2) \rangle_{\tilde{x}_m}}}. \quad (8)$$

$$L_e(\hat{u}_{i,j}) = \log \frac{\sum_{\tilde{\sigma}' \rightarrow \tilde{\sigma}} \tilde{\alpha}_{i-1}(\tilde{\sigma}') \cdot \tilde{\gamma}_{i,j}^X(\tilde{\sigma}', \tilde{\sigma}) \cdot \tilde{\beta}_i(\tilde{\sigma})}{\sum_{\substack{\tilde{\sigma}' \rightarrow \tilde{\sigma} \\ \tilde{x}_{i,m_i(j)} = +1}} \tilde{\alpha}_{i-1}(\tilde{\sigma}') \cdot \tilde{\gamma}_{i,j}^{X^+}(\tilde{\sigma}', \tilde{\sigma}) \cdot \tilde{\beta}_i(\tilde{\sigma}) - \sum_{\substack{\tilde{\sigma}' \rightarrow \tilde{\sigma} \\ \tilde{x}_{i,m_i(j)} = -1}} \tilde{\alpha}_{i-1}(\tilde{\sigma}') \cdot \tilde{\gamma}_{i,j}^{X^-}(\tilde{\sigma}', \tilde{\sigma}) \cdot \tilde{\beta}_i(\tilde{\sigma})}. \quad (9)$$

Step 1 requires a total of

$$A_1 = k + \sum_{i=1, \tilde{v}_{par,i} > 1}^V \tilde{v}_{dist,i} \cdot [z_i (\tilde{v}_{par,i} - 2) + \tilde{v}_{par,i} - 1] \quad (18)$$

and

$$M_1 = \sum_{i=1}^V \{ (v_i/2 - 1) \cdot \tilde{v}_{dist,i} \cdot \tilde{v}_{par,i} + 1 \}_{v_i > 1} + \{ z_i/2 \cdot \tilde{v}_{dist,i} \cdot \tilde{v}_{par,i} \}_{v_i > 2} \quad (19)$$

additions and multiplications, respectively. The terms in braces are only included in the sum if the corresponding conditions are fulfilled.

**Step 2: Computing the state probabilities  $\tilde{\alpha}_i(\tilde{\sigma})$  and  $\tilde{\beta}_i(\tilde{\sigma})$ .** This is accomplished using (13) and (14).

- Computing the  $\alpha$ 's requires  $\tilde{v}_{comp,i}$  multiplications and  $|\tilde{S}_{q_i}| \cdot (\tilde{v}_{in,i} - 1) = \tilde{v}_{comp,i} - |\tilde{S}_{q_i}|$  additions, where  $\tilde{v}_{in,i}$  denotes the number of branches merging at each node  $\tilde{S}_{q_i}$ .
- Computing the  $\beta$ 's requires  $\tilde{v}_{comp,i}$  multiplications and  $|\tilde{S}_{q_i}| \cdot (\tilde{v}_{out,i} - 1) = \tilde{v}_{comp,i} - |\tilde{S}_{q_{i-1}}|$  additions, where  $\tilde{v}_{out,i}$  denotes the number of branches leaving each node  $\tilde{S}_{q_i}$ .

Thus, step 2 involves

$$A_2 = \sum_{i=1}^{V-1} (\tilde{v}_{comp,i} - |\tilde{S}_{q_i}|) + \sum_{i=2}^V (\tilde{v}_{comp,i} - |\tilde{S}_{q_{i-1}}|) \quad (20)$$

and

$$M_2 = \sum_{i=1}^{V-1} \tilde{v}_{comp,i} + \sum_{i=2}^V \tilde{v}_{comp,i} \quad (21)$$

additions and multiplications, respectively.

**Step 3: Computing the log-likelihood values  $L_e(\hat{u}_{i,j})$ .**

This is done using (9). We define the symbols

$$\Sigma_{i,j}^+ = \sum_{\tilde{\sigma}' \rightarrow \tilde{\sigma}, \tilde{x}_{i,m_i(j)} = +1} \tilde{\alpha}_{i-1}(\tilde{\sigma}') \cdot \tilde{\gamma}_{i,j}^{X^+}(\tilde{\sigma}', \tilde{\sigma}) \cdot \tilde{\beta}_i(\tilde{\sigma}) \quad (22)$$

$$\Sigma_{i,j}^- = \sum_{\tilde{\sigma}' \rightarrow \tilde{\sigma}, \tilde{x}_{i,m_i(j)} = -1} \tilde{\alpha}_{i-1}(\tilde{\sigma}') \cdot \tilde{\gamma}_{i,j}^{X^-}(\tilde{\sigma}', \tilde{\sigma}) \cdot \tilde{\beta}_i(\tilde{\sigma}) \quad (23)$$

$$\begin{aligned} \Sigma_{i,j} &= \sum_{\tilde{\sigma}' \rightarrow \tilde{\sigma}} \tilde{\alpha}_{i-1}(\tilde{\sigma}') \cdot \tilde{\gamma}_{i,j}^X(\tilde{\sigma}', \tilde{\sigma}) \cdot \tilde{\beta}_i(\tilde{\sigma}) \quad (24) \\ &= \Sigma_{i,j}^+ + \Sigma_{i,j}^- \end{aligned}$$

Then do the following for all sections  $i$  that contain information bits, i.e.,  $z_i > 0$ :

- Compute  $\Sigma_{i,j}^+$  and  $\Sigma_{i,j}^-$  for all  $1 \leq j \leq z_i$ . Evaluation of the products  $\tilde{\alpha} \cdot \tilde{\beta}$  requires  $v_{comp,i}$  multiplications. If there are no parallel branches in this section, each  $\tilde{\gamma}_{i,j}^{X^\pm}(\tilde{\sigma}', \tilde{\sigma})$  is 0 for half the branches. In this case, the computation of all  $\Sigma_{i,j}^\pm$ 's requires only  $z_i \cdot v_{comp,i}$  multiplications and  $2 \cdot z_i \cdot (v_{comp,i}/2 - 1)$

additions. Otherwise, it requires  $2 \cdot z_i \cdot v_{comp,i}$  multiplications and  $2 \cdot z_i \cdot (v_{comp,i} - 1)$  additions. There is also the possibility of checking if  $v_i = 1$ . In this case,  $\tilde{\gamma}_{i,j}^{X^\pm}(\tilde{\sigma}', \tilde{\sigma})$  is equal to 1, and does not need to be included in the product. This check makes sense if there are a lot of sections that contain only one code bit.

- Compute  $\Sigma_{i,j} = \Sigma_{i,j}^+ + \Sigma_{i,j}^-$ , which involves  $z_i$  additions.
- Compute  $\log(\Sigma_{i,j} / [\Sigma_{i,j}^+ - \Sigma_{i,j}^-])$  for all information bits in this section, which involves  $z_i$  multiplications and  $z_i$  additions.

Step 3 requires a total of

$$A_3 = \sum_{i=1, z_i > 0}^V \left\{ \begin{array}{ll} 2z_i \cdot \tilde{v}_{comp,i}, & \tilde{v}_{par,i} > 1 \\ z_i \cdot \tilde{v}_{comp,i}, & \tilde{v}_{par,i} = 1 \end{array} \right\} \quad (25)$$

and

$$M_3 = k + \sum_{i=1, z_i > 0}^V \tilde{v}_{comp,i} + \left\{ \begin{array}{ll} 2z_i \cdot \tilde{v}_{comp,i}, & \tilde{v}_{par,i} > 1 \\ z_i \cdot \tilde{v}_{comp,i}, & \tilde{v}_{par,i} = 1 \end{array} \right\} \quad (26)$$

additions and multiplications, respectively.

The total computational complexity for computing extrinsic values  $L_e(\hat{u}_{i,j})$  based on  $T_{C^\perp}(Q)$  is now given by  $A = A_1 + A_2 + A_3$  additions and  $M = M_1 + M_2 + M_3$  multiplications.

**Memory Requirements.** The MAP algorithm based on  $T_{C^\perp}(Q)$  requires roughly a total of

$$\underbrace{2 \sum_{i=1}^V |\tilde{S}_{q_i}|}_{\tilde{\alpha}'\text{'s and } \tilde{\beta}'\text{'s}} + \underbrace{\sum_{i=1}^V \tilde{v}_{dist,i}}_{\tilde{\gamma}'\text{'s}} + 2 \underbrace{\sum_{i=1}^V z_i \cdot \tilde{v}_{dist,i}}_{\tilde{\gamma}^{X^\pm}\text{'s}} \quad (27)$$

floating point storage locations.

Note that this figure is intended to give the reader only a rough idea of the storage required to implement the MAP algorithm. No efforts were made to minimize this figure, but it also does not consider the memory required for temporary tables, integer variables, and — most importantly — the trellis description. The fewer sections a trellis has, the easier is its description. This can be another factor in favor of using a sectionalized trellis diagram.

## 3.2 Optimum Sectionalization of Trellises

For implementation on a computer where the memory requirement is not a high priority, we are interested in a sectionalization that minimizes the computational complexity. In order to be able to decide which sectionalization is optimum, we need to know the relative complexity of additions and multiplications. Modern microprocessors perform multiplications almost as fast as additions. On a computer with Intel-Pentium II processor, we determined that a multiplication takes 1.08 times the time of an addition.

DECODING USING $T_C(Q)$	add's	mult's	total	memory
<b>RM(32,6) code</b>				
no trellis ( $V = 1$ )	2,257	14	2,271	9
bit-level trellis ( $V = 32$ )	245	1,558	1,803	1,338
opt. sectionalization ( $V = 6$ ): $Q = \{0, 4, 12, 16, 24, 28, 32\}$	547	390	937	226
<b>RM(32,16) code</b>				
no trellis ( $V = 1$ )	2,621,487	34	2,621,521	19
bit-level trellis ( $V = 32$ )	8,025	19,200	27,225	9,658
opt. sectionalization ( $V = 10$ ): $Q = \{0, 2, 4, 8, 12, 16, 20, 24, 28, 30, 32\}$	8,184	9,344	17,528	1,602
<b>RM(32,26) code</b>				
no trellis ( $V = 1$ )	$3.0 \cdot 10^9$	54	$3.0 \cdot 10^9$	29
bit-level trellis ( $V = 32$ )	2,765	4,554	7,319	1,338
opt. sectionalization ( $V = 21$ ): $Q = \{0, 2, 4, 6, 7, 8, 10, 11, 12, 13, 14, 16, 18, 19, 20, 21, 22, 24, 25, 26, 28, 32\}$	2,781	3,974	6,755	978

Table 1: Complexity measures for MAP SISO decoding based on  $T_C(Q)$

DECODING USING $T_{C^\perp}(Q)$	add's	mult's	total	memory
<b>RM(32,6) code</b>				
no trellis ( $V = 1$ )	$4.6 \cdot 10^8$	$1.2 \cdot 10^9$	$1.6 \cdot 10^9$	15
bit-level trellis ( $V = 32$ )	1,184	2,550	3,734	1,362
opt. sectionalization ( $V = 19$ ): $Q = \{0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 16, 17, 18, 19, 20, 24, 28, 32\}$	1,194	2,083	3,277	858
<b>RM(32,16) code</b>				
no trellis ( $V = 1$ )	1,141,127	1,507,378	2,621,505	35
bit-level trellis ( $V = 32$ )	6,410	19,200	25,610	9,722
opt. sectionalization ( $V = 11$ ): $Q = \{0, 2, 3, 5, 8, 9, 12, 16, 17, 20, 24, 32\}$	6,876	11,808	18,684	2,476
<b>RM(32,26) code</b>				
no trellis ( $V = 1$ )	1,753	1,872	3,625	55
bit-level trellis ( $V = 32$ )	708	2,570	3,278	1,442
opt. sectionalization ( $V = 7$ ): $Q = \{0, 2, 4, 8, 16, 24, 28, 32\}$	702	1,201	1,903	810

Table 2: Complexity measures for MAP SISO decoding based on  $T_{C^\perp}(Q)$

based on	trellis	real operations
original code	bit-level	135,976
	optimal sectionalization	117,772
dual code	bit-level	66,386
	optimal sectionalization	36,342

Table 3: Complexity measures for MAP SISO decoding of the eBCH(32,21) code

We define the total computational complexity  $T$  of an algorithm as the weighted sum of the number of additions and multiplications,  $T = A + W \cdot M$ , where  $W$  is the weighting factor that we will set, based on what was said above, equal to 1. If the algorithm is to be implemented using different hardware,  $W$  must be adjusted accordingly. It is a simple matter to find a trellis sectionalization that minimizes  $T$ . The idea was first mentioned by Lafourcade and Vardy in [5] for Viterbi decoding, but it can be applied to MAP decoding as well.

Denote the weighted number of real operations required to decode one section  $S(a, b)$  with boundary locations  $a$  and  $b$ ,  $a < b$ , by  $T^S(a, b)$ , and the *minimum* weighted number of real operations to process the trellis from point  $a$  to  $b$  by  $T_{min}(a, b)$ . To achieve this  $T_{min}(a, b)$ , we might have to split the trellis from point  $a$  to  $b$  into several sections. The minimum total complexity is given by  $T_{min}(0, n)$ . If we alter the sectionalization of the trellis, but leave the section  $S(a, b)$  unchanged, then this does not influence  $T^S(a, b)$ . Based on this fact, we can write,

$$T_{min}(0, b) = \begin{cases} \min \left\{ T(0, b), \min_{0 < c < b} (T_{min}(0, c) + T(c, b)) \right\}, & 1 < b \leq n \\ T(0, 1) & b = 1 \end{cases} \quad (28)$$

This equation must be applied recursively. The set of all values  $c$  found when computing  $T_{min}(0, n)$  using (28) gives the set of inner boundary locations in the optimum sectionalized trellis.

Table 1 describes the number of floating point operations and storage locations required for SISO MAP decoding based the trellis of the original code,  $T_C(Q)$ , of length-32 Reed-Muller Codes. Since our interest is focused on high rate codes, the sectionalization of  $T_C(Q)$  was not described in this paper. For further information refer to [8]. For decoding in the dual domain the complexity measures given in table 2 are obtained. We always assume an optimum permutation of the columns of  $\mathbf{G}$  that minimizes the trellis complexity.

From the tables we see that for the high rate RM(32,26) code it is most advantageous to use the sectionalized trellis of the dual code. Decoding based on the bit-level trellis of the dual code requires 72% more real operations, and decoding based on the bit-level trellis of the original code, 285% more. For the rate 1/2 RM(32,16) code, however, using  $T_C(Q)$  is slightly better than using  $T_{C^\perp}(Q)$ . When decoding using the dual code, sectionalization is most effective for high rate codes, when decoding using the original code, sectionalization is most effective for low rate codes.

Table 3 shows complexity measures for MAP decoding of an extended BCH(32,21) code. Decoding based on the optimum sectionalized trellis of the dual code saves 73% in real operations compared to decoding based on the original bit-level trellis.

## 4 Simulation Results

The interleaver influences the performance of the turbo code significantly. For small block lengths such as  $K < 1000$  a randomly chosen interleaver may not be the best choice. Fig. 1 shows simulation results for an eBCH(32,21) turbo code using a random, a block, and an improved (semi) random interleaver, all with block size  $K = 441$ . The improved random interleaver is found by randomly selecting an interleaver that does not map more than one bit of the same message of component code 1 into one message of component code 2. From fig. 1 we see that an improved random interleaver performs almost 0.4 dB better in the error floor region than a random interleaver, even outperforming a block interleaver. Both the block interleaver and the improved random interleaver yield a performance below the union bound, indicated by a dot-dash line.

In [7], Nickl *et al.* presented a series of simulation results for block turbo codes using Hamming codes up to length 1023. Decoding of these codes was carried out in the dual domain, but without utilizing a trellis. The complexity of the simulations was so great that a fast neurocomputer had to be used. The Hamming(31,26) turbo code (interleaver size  $K = 676$ , overall rate  $R = 0.722$ ) achieved  $P_b = 10^{-5}$  within 2.3 dB of capacity, whereas the Hamming(1023,1013) turbo code ( $K = 1,026, 169$ ,  $R = 0.981$ ) came within 0.27 dB of capacity.

The question that comes to mind is whether shorter, lower rate block codes can also perform close to capacity if long enough interleavers are employed. Since RM( $2^m, 2^m - m, 4$ ) codes are extended Hamming( $2^m - 1, 2^m - 1 - m, 3$ ) codes, their performance in a turbo coding scheme is very similar to Hamming codes. An advantage of Reed-Muller codes is that they have a more regular trellis structure than Hamming codes.

Figure 2 shows simulation results for the RM(32,26) turbo code ( $R = 0.684$ ) using an improved random interleaver of different sizes. The SNR pertaining to Shannon's capacity limit for this code rate is 1.17 dB. We make the following observations:

- From subplot a) we can see that again the improved random interleaver yields a performance below the union bound.
- The larger the interleaver, the more iterations are necessary for the MAP algorithm to converge in the waterfall region. At low SNRs and high SNRs few iterations suffice.
- Increasing the interleaver size improves the error floor only slightly. This may not come as a surprise, since increasing the interleaver size does not change the minimum distance of the code, which determines its performance at high SNRs. The waterfall region of the curve, however, is shifted to lower SNRs. At  $P_b = 3 \cdot 10^{-4}$ , the gain of the code with interleaver size  $K = 251,992$  over the code with  $K = 676$  is about 0.8 dB. The improvement in the waterfall region is because low weight codewords become less likely for larger interleavers.

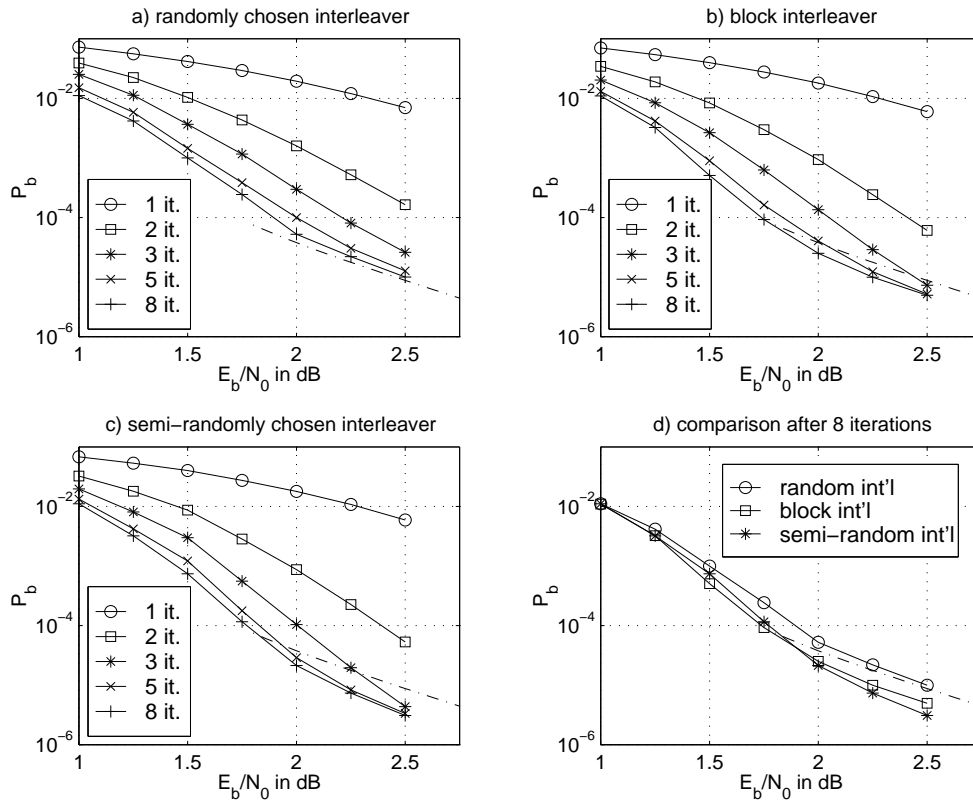


Figure 1: extended BCH(32,21) turbo code, interleaver size  $K = 441$

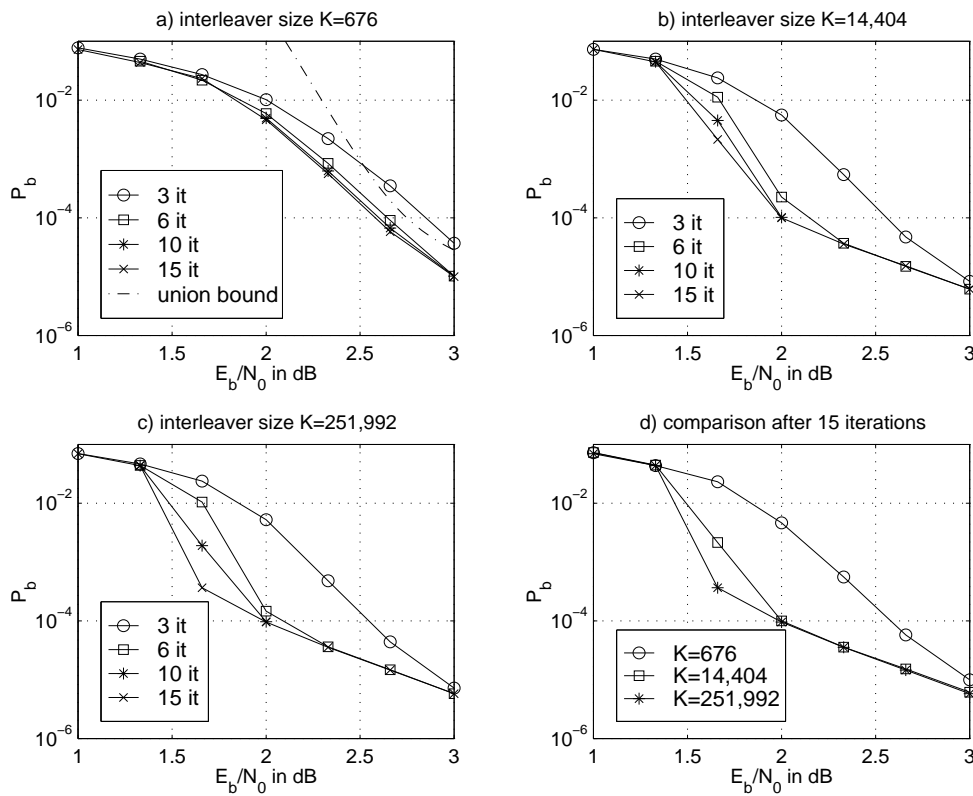


Figure 2: RM(32,26) turbo code, improved random interleaver

The answer to the question if shorter, lower rate codes can also perform close to capacity, depends on where the error floor occurs. Since we have defined the distance from capacity as the distance between the SNR that is required to achieve  $P_b = 10^{-5}$  and the SNR pertaining to Shannon's capacity limit, a lower rate code with a low error floor, i.e., below  $P_b = 10^{-5}$ , might approach capacity. The RM(32,26) turbo code didn't succeed, but a good candidate might be an eBCH(64,51) turbo code ( $R = 0.662$ ) whose error floor seems to be well below  $P_b = 10^{-5}$  [2]. However, this code probably must be decoded using a suboptimum algorithm, since optimal MAP decoding of this code is extremely complex.

## 5 Concluding Remarks

In this paper, an optimal MAP Soft-In Soft-Out decoder was modified to utilize a sectionalized trellis diagram of the dual code. It was shown how to choose a trellis sectionalization that minimizes the computational complexity of the algorithm. For high rate block codes, using the sectionalized trellis diagram of the dual code was found to be most efficient, with the exact advantage depending on the block code used. This modified MAP algorithm can be used to decode component codes in a turbo coding scheme.

The complexity considerations assumed that only a posteriori values for information bits are to be computed. In serial concatenation (product codes), soft-values are needed for all code bits. The extension to this case, however, is straightforward.

Simulation results suggest that it might be possible to achieve a performance close to capacity, even if a reasonably short and not extremely high rate block code such as the extended BCH(64,51) code is employed in a turbo coding scheme using a large improved random interleaver.

## Acknowledgment

The authors would like to thank Dr. M. Fossorier for valuable help and ideas.

This work was supported by the National Science Foundation under Grant NCR95-22939 and by NASA under Grant NAG5-557.

## References

- [1] G. Battail, M. C. Decouvelaere, P. Godlewski: Replication Decoding. *IEEE Transactions on Information Theory*, Vol. IT-25, No. 3, pp. 332–345, May 1979
- [2] M. P. C. Fossorier, S. Lin: Soft-Input Soft-Output Decoding of Linear Block Codes Based on Ordered Statistics. *Proc. IEEE Globecom Conference*, Nov. 1998.
- [3] J. Hagenauer, E. Offer, L. Papke: Iterative Decoding of Binary Block and Convolutional Codes. *IEEE Transactions on Information Theory*, Vol. 42, No. 2, pp. 429–445, March 1996
- [4] C. R. Hartmann, L. D. Rudolph: An Optimum Symbol-by-Symbol Decoding Rule for Linear Codes. *IEEE Transactions on Information Theory*, Vol. 22, pp. 514–517, Sept. 1976
- [5] A. Lafourcade, A. Vardy: Optimal Sectionalization of a Trellis. *IEEE Transactions on Information Theory*, Vol. 42, pp. 689–703, 1996
- [6] S. Lin, T. Kasami, T. Fujiwara, M. Fossorier: *Trellises and Trellis-Based Decoding Algorithms for Linear Block Codes*. Kluwer Academic Publishers, 1998
- [7] H. Nickl, J. Hagenauer, F. Burkert: Approaching Shannon's Capacity Limit by 0.27 dB Using Simple Hamming Codes. *IEEE Communications Letters*, Vol. 1, No. 5, pp. 130–132, Sept. 1997
- [8] P. J. Schreier: *Iterative Decoding of Parallel Concatenated High Rate Linear Block Codes*. M.S.E.E. Thesis, University of Notre Dame, IN, March 1999