<table>
<thead>
<tr>
<th>Title</th>
<th>Symbol-by-symbol APP decoding of the Golay code and iterative decoding of concatenated Golay codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author(s)</td>
<td>Li, P; Yeung, KL</td>
</tr>
<tr>
<td>Citation</td>
<td>IEEE Transactions on Information Theory, 1999, v. 45 n. 7, p. 2558-2562</td>
</tr>
<tr>
<td>Issued Date</td>
<td>1999</td>
</tr>
<tr>
<td>URL</td>
<td><a href="http://hdl.handle.net/10722/42847">http://hdl.handle.net/10722/42847</a></td>
</tr>
<tr>
<td>Rights</td>
<td>©1999 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.</td>
</tr>
</tbody>
</table>
Symbol-by-Symbol APP Decoding of the Golay Code and Iterative Decoding of Concatenated Golay Codes

Li Ping, Member, IEEE, and Kwan L. Yeung, Member, IEEE

Abstract—An efficient coset based symbol-by-symbol soft-in/soft-out APP decoding algorithm is presented for the Golay code. Its application in the iterative decoding of concatenated Golay codes is examined.

Index Terms—Concatenated codes, coset decoding, Golay code, iterative decoding, turbo codes, turbo decoding.

I. INTRODUCTION

The symbol-by-symbol soft-in/soft-out (a posteriori probability), also known as MAP (maximum a posteriori), decoding algorithm [1] plays an important role in the iterative decoding of concatenated codes [2]–[4]. The exact APP decoding of the Golay code C can be accomplished by applying the BCJR algorithm to the 256-state minimum conventional trellis of C [5] or the eigenvector algorithm [6] to the 16-state minimum tail-biting trellis of C [7]. The complexities of both methods are quite high. A lower cost alternative is the approximate method of [6] applied to the 16-state tail-biting trellis [7], resulting in complexity (normalized to operations per information bit) of about twice the BCJR algorithm for a rate 1/2, 16-state conventional convolutional code.1

This correspondence presents an efficient, exact APP decoding algorithm for C based on the coset decoding principle [8]–[12]. The complexity of the algorithm is comparable to the BCJR algorithm for a rate 1/2, 16-state conventional convolutional code. We will examine the application of the proposed algorithm in the iterative decoding of the concatenated Golay codes. We will show that, for short interleaver lengths, the concatenated Golay codes can achieve performance similar to or better than the turbo codes.

II. PRELIMINARIES

In [8], Pless introduced an elegant 4 × 6 matrix construction of C. In this section, we will express this method in a product form of the hexacode and the SPC (single parity check) codes. We will then examine the application of the proposed algorithm in the iterative decoding of the concatenated Golay codes. We will show that, for short interleaver lengths, the concatenated Golay codes can achieve performance similar to or better than the turbo codes.

A. Pless’ Construction of C

Denote by 0, 1, ω, ω−1 the four elements of GF(4), referred to as characters. The hexacode is a [6, 3, 4] linear code over GF(4) with the generator matrix [8], [9]

\[
\begin{bmatrix}
1 & 0 & 0 & 1 & \omega & \omega^2 \\
0 & 1 & 0 & 1 & \omega^2 & \omega \\
0 & 0 & 1 & 1 & 1 & 1
\end{bmatrix}
\]

The following are two mappings from the characters to 4 × 1 vectors over \{+1, −1\}. The parity refers to the number of −1 in a column.

\[
\begin{array}{cccc}
1 & 1 & 1 & 1 \\
1 & 1 & −1 & −1 \\
1 & −1 & 1 & −1 \\
1 & −1 & −1 & 1 \\
0 & 1 & \omega & \omega^2 \\
0 & 1 & \omega^2 & \omega \\
0 & 0 & 1 & 1 & 1 & 1
\end{array}
\]

Applying the even (resp., odd) interpretations to the 64 codewords in the hexacode results in 64 4 × 6 binary arrays, collectively referred to as H+ (resp., H−). Let P+ (resp., P−) be the length-6 even and odd SPC codes over \{+1, −1\} containing an even (resp., odd) number of −1, each with 32 codewords. Let

\[
h = \{h[i, j]\} = [h_1, h_2, h_3, h_4, h_5, h_6]
\]

be a 4 × 6 array, where \{h_i\} are 4 × 1 vectors. Denote by "□" the union of two nonoverlapping sets. It is straightforward to verify the equivalence between the construction in [8] and the following definition of C:

\[
C = C^+ \oplus C^-
\]

(3a)

\[
C^+ \triangleq \{h \circ p; h \in H^+, p \in P^+\}
\]

(3b)

\[
C^- \triangleq \{h \circ p; h \in H^-, p \in P^-\}
\]

(3c)

\[
h \circ p \triangleq [h_1p_1, h_2p_2, h_3p_3, h_4p_4, h_5p_5, h_6p_6].
\]

(3d)

B. The APP Decoding Problem of C

Let the transmitted codeword be a 4 × 6 array \(u = \{u[i, j]\} \subset C\) and its noisy observation be \(x = \{x[i, j]\}\). The output of a soft-in/soft-out APP decoder is [1], [2]

\[
L[i, j] = \frac{1}{2} \log \left( \frac{\Pr\{u[i, j] = +1 | x\}}{\Pr\{u[i, j] = -1 | x\}} \right).
\]

(4)

Here a factor of 1/2 is included for convenience. Let \(v = \{v[i, j]\}\) be a 4 × 6 array of the bit confidence values conditioned on individual received symbol, i.e.,

\[
v[i, j] = \frac{1}{2} \log \left( \frac{\Pr\{u[i, j] = +1 | x[i, j]\}}{\Pr\{u[i, j] = -1 | x[i, j]\}} \right).
\]

(5)

For example, \(v[i, j] = x[i, j]/\sigma^2\) for an additive white Gaussian noise (AWGN) channel of variance \(\sigma^2\). Assuming that the entries of \(v\) are independent and all the codewords have equal probability of occurrence, then (4) can be evaluated as [3]

\[
L[i, j] = \frac{1}{2} \log \left( \sum_{v[i, j] = +1} e^{v[i, j]} + \sum_{v[i, j] = -1} e^{-v[i, j]} \right)
\]

(6)

\[
= \frac{1}{2} \log \left( \sum_{v[i, j] = +1} e^{v[i, j]} + \sum_{v[i, j] = -1} e^{-v[i, j]} \right)
\]

Manuscript received May 4, 1998; revised May 18, 1999.

The authors are with the Department of Electronics Engineering, City University of Hong Kong, Hong Kong (e-mail: {eliping; eekyeung}@cityu.edu.hk).

Communicated by A. M. Barg, Associate Editor for Coding Theory.

Publisher Item Identifier S 0018-9448/99/07313-7.

1This is estimated based on the example in [6], assuming that for a short length tail-biting trellis, the extra wrap length is roughly equal to the trellis length; also see Section III-D.
where
\[ \langle \mathbf{c}, \mathbf{v} \rangle = \sum_{j=1}^{6} c_j v_j = \sum_{j=1}^{4} \sum_{i=1}^{6} c[i, j] v[i, j]. \tag{7} \]

In (7), \( c_j \) and \( v_j \) are the \( j \)-th columns of \( \mathbf{c} \) and \( \mathbf{v} \), respectively. Substitute (3) into (7)
\[ \langle \mathbf{c}, \mathbf{v} \rangle = \langle \mathbf{h} \circ \mathbf{p}, \mathbf{v} \rangle = \sum_{j=1}^{6} \langle h_j p_j, v_j \rangle = \sum_{j=1}^{6} p_j \langle h_j, v_j \rangle = \langle \mathbf{p}, \mathbf{w} \rangle \tag{8} \]

with \( \mathbf{w} = [w_1, \cdots, w_6] = [(h_1, v_1), \cdots, (h_6, v_6)] \).

The summations in (6) then become
\[ \sum_{e \in C^x} e^{\langle \mathbf{c}, \mathbf{v} \rangle} = \sum_{e \in C^x} e^{\langle \mathbf{p}, \mathbf{w} \rangle}, \quad x = \text{e, o}. \tag{9} \]

Equation (9) is over 4096 codewords for every \( e[i, j] \), which represents the bulk of the computation involved in (6). A straightforward summation is apparently very costly. We will explore an improved solution below.

### III. EFFICIENT APP DECODING METHOD FOR C

In this section we will first introduce an \( h \)-coset partitioning of \( C \). We will show that (9) can be evaluated partially over each \( h \)-coset using a simple rule. The partial results can be efficiently combined through a set partitioning hierarchy of \( C \). These form the core parts of the new algorithm.

#### A. Partition of \( C \) using \( h \)-Cosets

Fixing any \( \mathbf{h} \) in \( H^c \), we can obtain a unique subset of 32 codewords \( \{ h \circ \mathbf{p}; p \in P^+ \} \) according to (3b). We will call them collectively as an \( h \)-coset generated by \( \mathbf{h} \). For all the elements in \( H^c \), we obtain 64 \( h \)-cosets. Similarly, another 64 \( h \)-cosets in the form of \( \{ h \circ \mathbf{p}; p \in P^+ \} \), each containing 32 codewords, can be generated by \( \mathbf{h} \) in \( H^c \). It can be shown that these 128 \( h \)-cosets form a coset partition of \( C \). Notice that \( h \circ \mathbf{p} = h \circ \mathbf{p'} \) only if \( p = [1, 1, 1, 1, 1, 1] \). Thus \( \{ e = h \circ \mathbf{p}; p \in P^+ \} \) does not include its generator \( h \) since \([1, 1, 1, 1, 1, 1] \notin P^+ \). This distinguishes an \( h \)-coset generator from a coset leader.

Based on the above definition, (9) can be rewritten as (recall that \( e[i, j] = h[i, j] p_j \), see (3))
\[ \sum_{e \in C^x} e^{\langle x, y \rangle} = \sum_{\mathbf{h} \in H^c} x^{\langle h, p \rangle} \sum_{p \in P^x \cap \langle h, p \rangle} e^{\langle \mathbf{p}, \mathbf{w} \rangle}, \quad x = e, o. \tag{10} \]

The inner summation above is over the 32 codewords in an \( h \)-coset and the outer one is over the 64 \( h \)-cosets for each parity. The following is a two-stage technique for evaluating (10).

#### B. Efficient Method for the Inner Summation in (10)

Recall that \( p \in P^+ \) (resp., \( p \in P^- \)) must contain an even (resp., odd) number of –1. It leads to a simple rule for evaluating the inner

\[ e \circ \mathbf{c} = (h \circ \mathbf{p'}) \circ (h \circ \mathbf{p}) = (h \circ \mathbf{p}) \circ (h \circ \mathbf{p'}) = h \circ (\mathbf{p'} \circ \mathbf{p}) \]
and \( \mathbf{p'} \circ \mathbf{p} \) has the same parity as \( \mathbf{p'} \) (and so \( h \)).
For a fixed \( j'' \neq j, j' \), \( J(h_j, h_{j'}) \) can again be partitioned into four nonoverlapping subsets \( \{J(h_j, h_{j'}, h_{j''}); h_{j''} = 0, 1, \omega, \bar{\omega}\} \). Since the hexacode is MDS (maximum-distance separable, see [13]) with minimum distance \( d = 4 \), any three columns, together with the parity, specify a unique \( h \) and so \( J(h_j, h_j, h_{j''}) \) represents a unique \( h \)-coset. As different \( h \)-cosets cannot overlap, this partition of \( J(h_j, h_{j''}) \) is invariant with respect to the choices of \( j'' \). Thus the inner summation in (13c) can be uniquely expressed as

\[
\sum_{J(h_j, h_{j'})} A^{\pm} = \sum_{h \in \text{coset} \in \mathbb{G}} A^{\pm}.
\]  
(14)

The key to efficiency improvement is avoiding the duplication of the summations in (13) and (14) for different output bits. Some details in evaluating (12)–(14) are given below.

i) We limit the \( (j, j') \) pair in (14) to \( (j, j+1) \), \( j = 1, 3, 5 \), referred to as blocks [9]. We first evaluate (14) for 16 \( (\text{character pairs}) \times 3 \) (blocks) \times 2 \( (\text{parities}) \times 96 \) possibilities of \( (h_j, h_{j+1}) \) pair. Each result of (14) is then used twice in (13c), where we set \( j' = j + 1 \) for \( j = 1, 3, 5 \) and \( j' = j - 1 \) for \( j = 2, 4, 6 \).

ii) We evaluate (13) for 6 \( (\text{columns}) \times 4 \) \( (\text{characters}) \times 2 \) \( (\text{parities}) \times 48 \) possibilities of \( h_j \). Each result of (13) can be repeatedly used in (12) for up to three information bits in a column. The information positions for \( C \) defined in (3) can be chosen as \( (i, j) = (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (2, 1), (4, 1), (2, 2), (4, 2), (2, 3), \) and \( (4, 3) \).

The above discussions are summarized below, with costs listed in brackets.

**D. APP Decoding Algorithm for \( C \)**

**Preparation:**

i) Generate 96 possibilities of \( \{e^{\pm w_j} 2^j; j = 1, 2, \ldots, 6\} \), see the Appendix (24 exponentials and 66 multiplications).

ii) Generate 96 possibilities of \( \{a_{j}^{\pm}; j = 1, 2, \ldots, 6\} \) (96 additions).

iii) Generate 256 possibilities of \( \{A^{\pm}\} \) for 128 \( h \)-cosets. This can be done by first generating all the partial products \( a_{j}^{\pm} a_{j+1}^{\pm} \), and \( a_{j}^{\pm} a_{j+1}^{-\pm} \) for \( j = 1, 3, 5 \) and then multiplying three partial products together for every \( h \) \( (4 \times 4 \times 3 \times 2 \times 2 + 128 \times 2 \times 2 = 704 \) multiplications).

**Step 1.** Evaluate (14) for 96 possibilities of \( (h_j, h_{j+1}) \) pair, with \( j = 1, 3, 5 \) \( (96 \times 2 \times 3 = 576 \) additions).
minimum trellises of block codes can be tail-biting trellis structures [3]. When applied to block codes, the trellis-based decoder needs to store only 448 real values of likelihood (ML) decoder is included as reference. The interleavers used are $D_1[i, j] = D[i, j]$ and $D_2[i, j] = D[i, j]$. It is seen that most coding gain can be achieved with three iterations. In some communication systems, interleaver lengths are restricted by delay constraints [4]. The performance of the concatenated Golay codes and turbo codes are compared in Fig. 2 for short interleaver lengths of $L = 96$ and $L = 192$. The interleavers for the Golay based codes are

$$D_1[i, j] = D[i, j] \quad D_2[i, j] = D[(i + j) \mod I, j + 5],$$

$I = 8, 16$ for $L = 96, 192$, respectively. (15)

The turbo codes are generated by polynomials 37 (denominator) and 21 (numerator) [2], with block interleavers. The turbo code encoders are all terminated [14] and thus their actual rates are slightly less than $1/3$. It is seen that for $L = 96$, the performance of the concatenated Golay codes is better than that of the turbo code. For $L = 192$, the performances of the two methods are very close. Only marginal improvement has been observed for the Golay code-based schemes for $L > 200$.

Based on the discussion at the end of Section III-D, the decoding costs of the concatenated Golay codes and 16-state turbo codes in Fig. 2 are roughly comparable in terms of operation numbers per information bit. For the comparison of storage usage, the BCJR algorithm for the rate 1/2, 16-state convolutional code [1], [2] needs to store $16L$ real values during the forward recursion (e.g., for $L = 192, 16L = 3072$). On the other hand, the algorithm in Section III-D needs to store only 448 real values of $\{e^{x+j}, a^j, A^k\}$ plus some extra buffering space (at most 96 real values).

V. CONCLUSION

The trellis-based BCJR APP decoding algorithm [1] is most suitable for convolutional codes [2] or block codes with simple trellis structures [3]. When applied to block codes, the trellis-based approach may not be the best choice. This is due to the fact that the minimum trellises of block codes can be tail-biting [5, 7, 13], which are more difficult to decode than conventional trellises. Alternative methods have been explored [16], [17] for some block codes. This correspondence has shown that the coset-based technique provides a cost-effective approach to the APP decoding of the Golay code.

APPENDIX

All the 48 possible values of $\{v_i = \{h, v_j\}\}$ can be computed using the Gray code technique of [9] with 60 additions and then $e^{x+j}$ can be calculated with 96 exponentials. The following is an alternative approach that is used for the complexity analysis in Section III-D. Define $t(x) = x$ if $x \geq 0$ and $t(x) = 0$ if $x < 0$ and let

$$T_j = \sum_{i=j}^{i-1} t(v[i, j]).$$

Then (see (8))

$$E_j^\pm = e^{\pm x-j} e^{T_j} = e^{\pm (h, v_j)-T_j},$$

$$= e^{\exp \left( \sum_{i=j}^{i-1} (h[i, j]v[i, j] - t(v[i, j])) \right)}. \quad (16)$$

Since $h[i, j] \in \{+1, -1\}$, $\pm h[i, j]v[i, j] - t(v[i, j])$ results in either 0 or $-2v[i, j]$. Fixing $j$, all the 16 possible values of $E_j^\pm$ can be found as

$$1, e^{-2v[1, j]} \quad e^{-2v[2, j]} \quad e^{-2v[3, j]} \quad e^{-2v[4, j]},$$

$$e^{-2v[1, j]+v[2, j]} \quad e^{-2v[1, j]+v[3, j]} \quad e^{-2v[1, j]+v[4, j]} \quad e^{-2v[1, j]+v[5, j]},$$


which can be generated with 4 exponentials and 11 multiplications (multiply-by-2 ignored). For $j = 1, 2, \cdots, 6$, this amounts to 24 exponentials and 66 multiplications. Instead of calculating $\{e^{x+j}\}$ from $\{E_j^\pm\}$, we can also simply replace $e^{x+j}$ in (11) by $E_j^\pm$. This will not affect the final result of the algorithm in Section III-D since (11) is equivalent to (with $T = T_1 + T_2 + \cdots + T_6$)

$$\sum_{p \in \mathbb{Z}} e^{t(p, w)}$$

for $h$ even: $e^{-T} \sum_{p \in \mathbb{Z}} e^{t(p, w)}$

$$= \sum_{j=1}^{6} \left( \frac{E_j^+ \pm E_j^-}{E_j^+ + E_j^-} \pm \frac{E_j^\pm}{E_j^+ - E_j^-} \right),$$

$$j = 1, 2, \cdots, 6 \quad (18a)$$

for $h$ odd: $e^{-T} \sum_{p \in \mathbb{Z}} e^{t(p, w)}$

$$= \sum_{j=1}^{6} \left( \frac{E_j^+ \pm E_j^-}{E_j^+ + E_j^-} \pm \frac{E_j^\pm}{E_j^+ - E_j^-} \right),$$

$$j = 1, 2, \cdots, 6 \quad (18b)$$

The extra factor $e^{-T}$ above will be canceled out during the division in (6).

ACKNOWLEDGMENT

The authors are very grateful for the detailed comments from the anonymous reviewers.

REFERENCES


IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. 45, NO. 7, NOVEMBER 1999

I. INTRODUCTION

There exist two main principles to terminate convolutional codes into block codes. Assume for simplicity that the generator matrix for a rate \( R = b/c \) convolutional code of memory \( m \) is polynomial and realized in controller canonical form.

In the first method we start in the zero state and encode the \( K \) information symbols followed by \( m \) 6-tuples of zeros. Hence, we reach the zero state and the convolutional code has been terminated into a block code by the so-called \( \text{zero-tail} \) (ZT) method at the cost of a rate loss by a factor \( K/(K + mh) \). If the trellis is short this rate loss might not be acceptable.

A termination method that does not suffer from any rate loss is \( \text{tail-biting} \), which can be used to construct very powerful regular trellis representations of block codes. Assuming a trellis of \( L \) sections (for simplicity we assume here that \( m \leq L \)), the \( \text{tail-biting condition} \) is the restriction that the convolutional encoder state \( \sigma \) at time \( t = 0 \) is equal to the encoder state at time \( L \), i.e., \( \sigma_0 = \sigma_L \). A tail-biting trellis of length \( L \) corresponds to a total of \( K = bL \) information symbols, \( c \) symbols per branch, block length \( N = Lc \), \( 2^b \) branches per trellis node; the number of codewords is

\[
M = 2^K = 2^{blc}
\]

and its rate is

\[
R = K/N = b/c.
\]

Let \( u_{[0,L]} = u_0 u_1 \ldots u_{L-1} \) denote the input (information) sequence, \( v_{[0,L]} = v_0 v_1 \ldots v_{L-1} \) the output sequence (codeword), and

\[
G(D) = G_0 + G_1 D + \cdots + G_m D^m
\]

the generator matrix. The codewords of the tail-biting representation of the block code \( E_L \), that is obtained from the convolutional code \( C \) encoded by the generator matrix \( G(D) \) can be compactly written as

\[
v_{[0,L]} = u_{p,t} \cdot G_{L}^{th}
\]

where

\[
G_{L}^{th} = \begin{pmatrix}
G_0 & G_1 & \cdots & G_m \\
G_0 & G_1 & \cdots & G_m \\
\vdots & \vdots & \ddots & \vdots \\
G_{m-1} & G_m & \cdots & G_1 \\
G_1 & G_2 & \cdots & G_m \\
G_0 & G_1 & \cdots & G_m \\
\end{pmatrix}
\]