TRELLISES AND TRELLIS-BASED DECODING ALGORITHMS FOR LINEAR BLOCK CODES

Part 3

Shu Lin and Marc Fossorier

April 20, 1998

12 AN ITERATIVE DECODING ALGORITHM FOR LINEAR BLOCK CODES BASED ON A LOW-WEIGHT TRELLIS SEARCH

For long linear block codes, maximum likelihood decoding based on full code trellises would be very hard to implement if not impossible. In this case, we may wish to trade error performance for the reduction in decoding complexity. Suboptimum soft-decision decoding of a linear block code based on a low-weight subtrellis can be devised to provide an effective trade-off between error performance and decoding complexity. This chapter presents such a suboptimal decoding algorithm for linear block codes. This decoding algorithm is iterative in nature and based on an optimality test. It has the following important features: (1) a simple method to generate a sequence of candidate codewords, one at a time, for test; (2) a sufficient condition for testing a candidate codeword for optimality; and (3) a low-weight subtrellis search for finding the most likely (ML) codeword.

12.1 GENERAL CONCEPTS

A simple low-cost decoder, such as an algebraic decoder, is used to generate a sequence of **candidate codewords** iteratively one at a time using a set of **test error patterns** based on the reliability information of the received symbols. When a candidate is generated, it is tested based on an **optimality condition**. If it satisfies the optimality condition, then it is the **most likely** (ML) codeword and decoding stops. If it fails the optimality test, a **search** for the ML codeword is conducted in a **region** which contains the ML codeword. The search region is determined by the current candidate codeword (or codewords) and the reliability of the received symbols. The search is conducted through a **purged trellis diagram** for the given code using a trellis-based decoding algorithm. If the search fails to find the ML codeword, a new candidate is generated using a new test error pattern (or any simple method), and the optimality test and search are renewed. The process of testing and searching continues until either the ML codeword is found or all the test error patterns are exhausted (or a stopping criterion is met) and the decoding process is terminated.

The key elements in this decoding algorithm are:

(1) Generation of candidate codewords,

(2) Optimality test,

(3) A search criterion,

(4) A low-weight trellis diagram,

(5) A search algorithm, and

(6) A stopping criterion.

12.2 OPTIMALITY CONDITIONS

Suppose C is used for error control over the **AWGN** channel using **BPSK** signaling. Let $\mathbf{c} = (c_1, c_2, \ldots, c_N)$ be the transmitted codeword. For BPSK transmission, $\mathbf{c}$ is mapped into a bipolar sequence $\boldsymbol{x} = (x_1, x_2, \ldots, x_N)$ with $x_i = (2c_i - 1) \in \{\pm 1\}$ for $1 \leq i \leq N$. Suppose $\boldsymbol{x}$ is transmitted and $\boldsymbol{r} = (r_1, r_2, \ldots, r_N)$ is received at the output of the matched filter of the receiver. Let

$z = (z_1, z_2, \ldots, z_N)$ be the binary **hard-decision** received sequence obtained from r using the **hard-decision function** given by

$$z_i = \begin{cases} 1 & \text{for } r_i > 0 \\ 0 & \text{for } r_i \leq 0 \end{cases} \tag{12.1}$$

for $1 \leq i \leq N$. We use $|r_i|$ as the **reliability measure** of the received symbol r_i since this value is proportional to the log-likelihood ratio associated with the symbol hard-decision, the larger the magnitude the greater its reliability.

For any binary N-tuple $u = (u_1, u_2, \ldots, u_N) \in \{0,1\}^N$, the correlation between u and the received sequence r is given by

$$M(u, r) \triangleq \sum_{i=1}^{N} r_i \cdot (2u_i - 1). \tag{12.2}$$

It follows from (12.1) and (12.2) that

$$M(z, r) = \sum_{i=1}^{N} |r_i| \geq 0, \tag{12.3}$$

and for any $u \in \{0,1\}^N$,

$$M(z, r) \geq M(u, r).$$

Define the following index sets:

$$D_0(u) \triangleq \{i : u_i = z_i \text{ and } 1 \leq i \leq N\}, \tag{12.4}$$

$$D_1(u) \triangleq \{i : u_i \neq z_i \text{ and } 1 \leq i \leq N\} = \{1, 2, \ldots, N\} \backslash D_0(u). \tag{12.5}$$

Let

$$n(u) \triangleq |D_1(u)|. \tag{12.6}$$

Consider

$$\begin{aligned} M(u, r) &= \sum_{i=1}^{N} r_i(2u_i - 1) \\ &= \sum_{i \in D_0(u)} r_i(2u_i - 1) + \sum_{i \in D_1(u)} r_i(2u_i - 1) \end{aligned}$$

$$
\begin{aligned}
&= \sum_{i \in D_0(u)} r_i(2z_i - 1) - \sum_{i \in D_1(u)} r_i(2z_i - 1) \\
&= \sum_{i=1}^{N} r_i(2z_i - 1) - 2 \sum_{i \in D_1(u)} r_i(2z_i - 1) \\
&= M(z, r) - 2 \sum_{i \in D_1(u)} |r_i| . \qquad (12.7)
\end{aligned}
$$

Let

$$L(u, r) \triangleq \sum_{i \in D_1(u)} |r_i| . \qquad (12.8)$$

Then, $M(u, r)$ can be expressed in terms of $M(z, r)$ and $L(u, r)$ as follows:

$$M(u, r) = M(z, r) - 2L(u, r). \qquad (12.9)$$

$L(u, r)$ is called the **correlation discrepancy of** u.

From (12.9), the MLD can be stated in terms of the correlation discrepancies of codewords as follows: The decoder computes the correlation discrepancy $L(c, r)$ for each codeword $c \in C$, and decode r into the codeword c_{opt} for which

$$L(c_{\text{opt}}, r) = \min_{c \in C} L(c, r). \qquad (12.10)$$

From (12.10), we see that if there exists a codeword c^* for which

$$L(c^*, r) \leq \alpha(c^*, r) \triangleq \min_{c \in C, c \neq c^*} L(c, r),$$

then $c^* = c_{\text{opt}}$. It is not possible to determine $\alpha(c^*, r)$ without evaluating $L(c, r)$ for all $c \in C$. However, if it is possible to determine a **tight lower bound** on $\alpha(c^*, r)$, then we have a **sufficient condition** for testing the optimality of a candidate codeword.

Suppose c is a candidate codeword for testing. The index set $D_0(c)$ consists of $N - n(c)$ indices. Order the indices in $D_0(c)$ as follows:

$$D_0(c) = \{k_1, k_2, \ldots, k_{N-n(c)}\} \qquad (12.11)$$

such that for $1 \leq i < j \leq N - n(c)$,

$$|r_{k_i}| < |r_{k_j}|. \qquad (12.12)$$

Let $D_0^{(j)}(c)$ denote the set of first j indices in the ordered set $D_0(c)$, i.e.,

$$D_0^{(j)}(c) \triangleq \{k_1, k_2, \ldots, k_j\}. \tag{12.13}$$

For $j \leq 0$, $D_0^{(j)}(c) \triangleq \emptyset$ and for $j \geq N - n(c)$, $D_0^{(j)}(c) \triangleq D_0(c)$.

Let $W = \{0, w_1, w_2, \ldots, w_m\}$ be the weight profile of code C and w_k be the k-th smallest non-zero weight in W. Define

$$s_k \triangleq w_k - n(c), \tag{12.14}$$

$$G(c, w_k) \triangleq \sum_{i \in D_0^{(s_k)}(c)} |r_i|, \tag{12.15}$$

and

$$R(c, w_k) \triangleq \{c' \in C : d(c', c) < w_k\}, \tag{12.16}$$

where $d(c', c)$ denotes the Hamming distance between c' and c.

Theorem 12.1 For a codeword c in C and a nonzero weight $w_k \in W$, if

$$L(c, r) \leq G(c, w_k), \tag{12.17}$$

then the optimal solution c_{opt} is in the region $R(c, w_k)$ [75].

Proof: Let c' be a codeword in $C \backslash R(c, w_k)$, i.e.,

$$d(c', c) \geq w_k. \tag{12.18}$$

We want to show that $L(c, r) \leq L(c', r)$. Let n_{01} and n_{10} be defined as

$$n_{01} \triangleq |D_0(c) \cap D_1(c')|, \tag{12.19}$$

$$n_{10} \triangleq |D_1(c) \cap D_0(c')|. \tag{12.20}$$

Since

$$d(c', c) = n_{01} + n_{10} \geq w_k, \tag{12.21}$$

we have

$$\begin{aligned} n_{01} &\geq w_k - n_{10} \\ &\geq w_k - |D_1(c)| \\ &= w_k - n(c). \end{aligned} \tag{12.22}$$

From (12.19) and (12.22), we find that

$$\begin{aligned} |D_1(c')| &\geq |D_0(c) \cap D_1(c')| \\ &\geq w_k - n(c). \end{aligned} \tag{12.23}$$

It follows from (12.19), (12.22) and (12.23) that

$$\begin{aligned} L(c', r) &= \sum_{i \in D_1(c')} |r_i| \\ &\geq \sum_{i \in D_0^{(w_k - n(c))}(c)} |r_i| \\ &= G(c; w_k) \\ &\geq L(c, r). \end{aligned} \tag{12.24}$$

Eq.(12.24) implies that the most likely codeword c_{opt} must be in the Region $R(c, w_k)$.

△△

Given a codeword c, Theorem 12.1 simply defines a region in which c_{opt} can be found. It says that c_{opt} is among those codewords in C that are at distance w_{k-1} or less from the codeword c, i.e.,

$$d(c, c_{opt}) \leq w_{k-1}. \tag{12.25}$$

If w_{k-1} is small, we can make a search in the region $R(c, w_k)$ to find c_{opt}. If w_{k-1} is too big, then it is better to generate another candidate codeword c' for testing and hopefully the search region $R(c', w_k)$ is small.

Two special cases are particularly important:

(1) If $k = 1$, the codeword c is the optimal MLD codeword c_{opt}.

(2) If $k = 2$, the optimal MLD codeword c_{opt} is either c or a nearest neighbor of c.

Corollary 12.1 Let $c \in C$.

(1) If $L(c, r) \leq G(c, w_1)$, then $c = c_{opt}$.

(2) If $L(c, r) > G(c, w_1)$ but $L(c, r) \leq G(c, w_2)$, then c_{opt} is at a distance not greater than the minimum distance $d_H = w_1$ from c. △△

The first part of Corollary 12.1 provides a sufficient condition for optimality of a codeword. The second part of Corollary 12.1 gives the condition that c_{opt} is either a nearest neighbor of a tested codeword or the tested codeword itself. We call $G(c, w_1)$ and $G(c, w_2)$ the **optimality** and **nearest neighbor test thresholds**, respectively. They will be used in an iterative decoding algorithm for testing. The sufficient condition on optimality given in Corollary 12.1 was first derived by Taipale and Pursley [94].

12.3 GENERATION OF CANDIDATE CODEWORDS AND TEST ERROR PATTERNS

The iterative decoding algorithm to be presented depends on the generation of a sequence of candidate codewords with a simple low-cost decoder. There are a number of ways of generating these candidate codewords. The simplest way is to use a set of probable test error patterns to modify the hard-decision received vector z and then decode each modified received sequence with an algebraic decoder. The test error patterns are generated in the decreasing likelihood order, one at a time. The most probable test error pattern is generated first and the least probable one is generated last. When a test error pattern e is generated, the sum $e + z$ is formed. Then the algebraic decoder decodes the modified received vector $e + z$ into a candidate codeword c for optimality test based on the two sufficient conditions given in Corollary 12.1.

Let p be a positive integer not greater than N. Let Q_p denote the set of the p least reliable positions of the received sequence r. Let E denote the set of 2^p binary error patterns of length N with errors confined to the positions in Q_p. The set E forms the basic set of test error patterns. The error patterns in E are more likely to occur than the other error patterns. In the Chase decoding algorithm-II [14], $p = \lfloor d_H/2 \rfloor$ is chosen and E consists of $2^{\lfloor d_H/2 \rfloor}$ test error patterns where d_H is the minimum distance of the code to be decoded and $\lfloor d_H/2 \rfloor$ denotes the largest integer equal to or less than $d_H/2$. Using this set of test error patterns, Chase proved that his decoding algorithm achieves asymptotically optimum error performance. In the iterative decoding algorithm to be presented in the next section, the same basic set of test error patterns will be used for generating candidate codewords.

For a test error pattern $e \in E$, let dec(e) denote the decoded codeword of the algebraic decoder with $e + z$ as the input vector. In case of a decoding failure (it may occur in a bounded distance-t decoder), let dec(e) $\triangleq (*)$ (undefined). In this case, the next test error pattern $e' \in E$ is generated for decoding. A test error $e \in E$, is said to be **decodable** if dec(e) $\neq (*)$. Two decodable error patterns, e and e' in E are said to be **equivalent** if dec(e) = dec(e') $\neq (*)$. Let e be a decodable error pattern in E and let $Q(e)$ denote the set of all test error patterns in E which are equivalent to e. $Q(e)$ is called the **equivalence class** containing e, and a test error pattern in $Q(e)$ is chosen as the class representative. Since all the test error patterns in an equivalence class generate the same candidate codeword, only the class representative should be used. How to partition E into equivalence classes and generate equivalence class representatives affects the efficiency of any decoding algorithm that utilizes test patterns.

Let E_{rep} denote the set of all representatives of the equivalence classes of E. Then every decodable error pattern in E_{rep} generates a distinct candidate codeword for testing. To construct E_{rep} for a given received sequence r, preprocessing is needed before decoding. This preprocessing of E is effective only if it is simpler than an algebraic decoding operation. An effective procedure for generating test error patterns in E_{rep} is presented in [75].

12.4 AN ITERATIVE DECODING ALGORITHM

This decoding algorithm is iterative in nature and devised based the reliability measures of the received symbols. It consists of the following key steps:

(1) Generate a candidate codeword c by using a test error pattern in E_{rep}.

(2) Perform the optimality test or the nearest neighbor test for each generated candidate codeword c.

(3) If the optimality test fails but the nearest neighbor test succeeds, a search in the region $R(c, w_2)$ is initiated. The search is conducted through the minimum-weight subtrellis, $T_{\min}(c)$, centered around c using a trellis-based decoding algorithm, say Viterbi or RMLD algorithms.

(4) If both optimality and nearest neighbor tests fail, a new test error pattern in E_{rep} is generated for the next decoding iteration.

Suppose the optimal MLD codeword has not been found at the end of the $(j-1)$-th decoding iteration. Then the j-th decoding iteration is initiated. Let c_{best} and $L(c_{\text{best}}, r)$ denote the best codeword and its correlation discrepancy that have been found so far and are stored in a buffer memory. The j-th decoding iteration consists of the following steps:

Step 1: Fetch e_j from E_{rep} and decode $e_j + z$ into a codeword $c \in C$. If the decoding succeeds, go to Step 2. Otherwise, go to Step 1.

Step 2: If $L(c, r) \leq G(c, w_1)$, $c_{\text{opt}} = c$ and stop the decoding process. Otherwise, go to Step 3.

Step 3: If $L(c, r) \leq G(c, w_2)$, search $T_{\text{min}}(c)$ to find c_{opt} and stop the decoding process. Otherwise, go to Step 4.

Step 4: If $L(c, r) < L(c_{\text{best}}, r)$, replace c_{best} by c and $L(c_{\text{best}}, r)$ by $L(c, r)$. Otherwise, go to Step 5.

Step 5: If $j < |E_{\text{rep}}|$, go to Step 1. Otherwise search $T_{\text{min}}(c_{\text{best}})$ and output the codeword with the least correlation discrepancy. Stop.

The decoding process is depicted by the flow diagram shown in Figure 12.1. The only case for which the decoded codeword may not be optimal is the output from the search of $T_{\text{min}}(c_{\text{best}})$. It is important to point out that when a received vector causes the decoding algorithm to perform $2^{\lfloor d_H/2 \rfloor}$ iterations without satisfying the sufficient conditions for optimality, optimum decoding is not guaranteed. Most of the decoding errors occur in this situation. The main cause of this situation is that the number of errors caused by the channel in the most reliable $N - \lfloor d_H/2 \rfloor$ positions is larger than $\lfloor (d_H - 1)/2 \rfloor$.

12.5 COMPUTATIONAL COMPLEXITY

We assume that the algebraic decoding complexity is small compared with the computational complexity required to process the minimum-weight trellis. The computational complexity is measured only in terms of real operations,

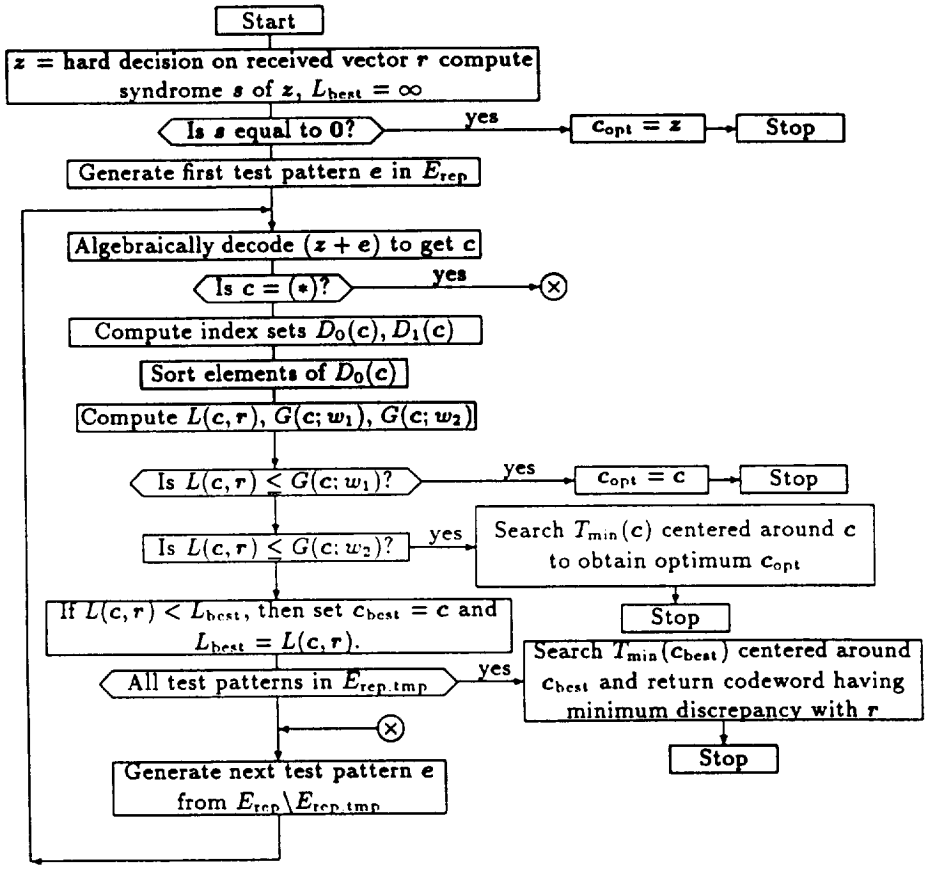


Figure 12.1. Flowchart of the Iterative Decoding Algorithm with minimum weight trellis search, where $E_{\text{rep,tmp}}$ denotes the set of those representative test error patterns that have been generated.

(real additions and comparisons). This number is a variable depending on the SNR. Let C_{max} denote the worst case maximum number of real operations required at any SNR. Then C_{max} can be computed by analyzing the flowchart of Figure 12.1.

Let A_0 denote the fixed number of real operations required in sorting the components of the received vector in increasing order of reliability. Let A_{loop} denote the number of real operations required in:

(1) computing the index sets $D_1(\mathbf{c})$ and $D_0(\mathbf{c})$;

(2) computing the correlation discrepancy $L(\mathbf{c}, \mathbf{r})$, the optimality test threshold $G(\mathbf{c}, w_1)$, and the nearest neighbor test threshold $G(\mathbf{c}, w_2)$; and

(3) comparing $L(\mathbf{c}, \mathbf{r})$ with $G(\mathbf{c}, w_1)$, $G(\mathbf{c}, w_2)$ and $L(\mathbf{c}_{best}, \mathbf{r})$.

Let $A(T_{min})$ denote the fixed number of real operations required to search through the minimum-weight trellis $T_{min}(\mathbf{c})$. Then

$$C_{max} = A_0 + |E_{rep}| \cdot A_{loop} + A(T_{min}) \leq A_0 + 2^{\lfloor d_H/2 \rfloor} \cdot A_{loop} + A(T_{min}). \quad (12.26)$$

Let $\tilde{C}_{max}$ denote the upper bound on C_{max} given by (12.26). Note that $\tilde{C}_{max}$ is independent of SNR. Table 12.1 shows the decoding complexities for some well known codes. Also given in the table are the complexities of the minimum-weight subtrellises and full trellises, $A(T_{min})$ and $A(T)$, of the codes in terms of real operations to be performed. The Viterbi algorithm is used for searching through the subtrellises and full trellises of the codes. We see that $A(T_{min})$ is much smaller than $A(T)$, especially for codes of length 64 or longer. Consider the $(64, 45)$ extended BCH code. Viterbi decoding based on the full code trellis requires $4,301,823$ real operations to decode a received sequence of 64 symbols. However, the worst-case maximum number of real operations required by the iterative decoding based on the minimum-weight trellis is upper bounded by $57,182$ while $A(T_{min}) = 48,830$. We see that there is a tremendous reduction in decoding complexity. Table 12.1 also lists the average number $|E_{rep}|$ of test error pattern representatives for each code at bit-error-rate (BER) of 10^{-5}. We see that $|E_{rep}|$ is much smaller than $|E|$, the size of the basic set of test error patterns.

Table 12.1. Complexity of the minimum-weight trellis iterative decoding algorithm.

Code	$A(T)$	$A(T_{\min})$	L	$\tilde{C}_{\max}$	C_{ave} @ $BER = 10^{-5}$	N_{ave} @ $BER = 10^{-5}$	$\lvert E_{\text{rep}}\rvert$ @ $BER = 10^{-5}$
Golay(23,12,7)	2,559	1,415	3	2,767	< 50	< 1	2
RM(32,16,8)	4,016	1,543	4	5,319	10	0.7	2
ex-BCH(32,21,6)	30,156	5,966	16	6,350	25	0.35	2
RM(32,26,4)	1,295	1,031	8	1,951	< 10	0.1	1
RM(64,22,16)	131,071	11,039	4	146,719	5,000	5	29
ex-BCH(64,24,16)	524,287	11,039	4	146,719	4,000	1.8	20
RM(64,42,8)	544,640	8,111	8	16,495	310	1.3	2
ex-BCH(64,45,8)	4,301,823	48,830	8	57,182	275	0.72	1
ex-BCH(64,51,6)	448,520	50,750	8	52,760	200	0.45	1
RM(64,57,4)	6,951	3,079	8	5,151	< 1	0.09	2

L: Number of section in trellis.

N_{ave}: Average number of iterations.

$|E_{\text{rep}}|$: Average size of the set of representatives of equivalence classes in E.

The average computational complexity of the iterative decoding algorithm can also be analyzed based on the decoding flowchart shown in Figure 12.1. Define the following event:

(1) For $1 \leq i \leq 2^{|E_{\text{rep}}|}$, let $\mathcal{B}_i$ denote the event that the condition $L(\mathbf{c}, \mathbf{r}) \leq G(\mathbf{c}; w_1)$ holds for the first time during the i-th iteration;

(2) For $1 \leq i \leq 2^{|E_{\text{rep}}|}$, let $\mathcal{C}_i$ denote the event that the condition $G(c, w_1) < L(\mathbf{c}, \mathbf{r}) \leq G(\mathbf{c}; w_2)$ is satisfied during the i-th decoding iteration;

(3) Let $\mathcal{B} \triangleq \cup_{\forall i} \mathcal{B}_i$, $\mathcal{C} \triangleq \cup_{\forall i} \mathcal{C}_i$ and $\mathcal{E} \triangleq \mathcal{B} \cup \mathcal{C} \setminus (\mathcal{B}_1 \cup \mathcal{C}_1)$; and

(4) Let $\mathcal{D}$ denote the event that neither $\mathcal{B}$ nor $\mathcal{C}$ occurs during the $|E_{\text{rep}}|$ runs through the decoding loop and the decoding process is terminated.

Let J be the average number of iterations preceding the event $\mathcal{E}$. Let $Pr(X)$ denote the probability of event X. Then the average number of real operations, denoted C_{ave}, required to decode a received word using the above iterative decoding algorithm is a function of the average size of the test error pattern

set E_{rep}, denoted M, and is given by

$$\begin{aligned} C_{\text{ave}} &= A_0 + (Pr(\mathcal{B}_1) + Pr(\mathcal{C}_1))A_{\text{loop}} + (Pr(\mathcal{C}) + Pr(\mathcal{D}))A(T_{\min}) \\ &+ Pr(\mathcal{D})MA_{\text{loop}} + Pr(\mathcal{E})(J+1)A_{\text{loop}}. \end{aligned} \tag{12.27}$$

The probabilities of the above events can be estimated using the Monte-Carlo simulation technique [86].

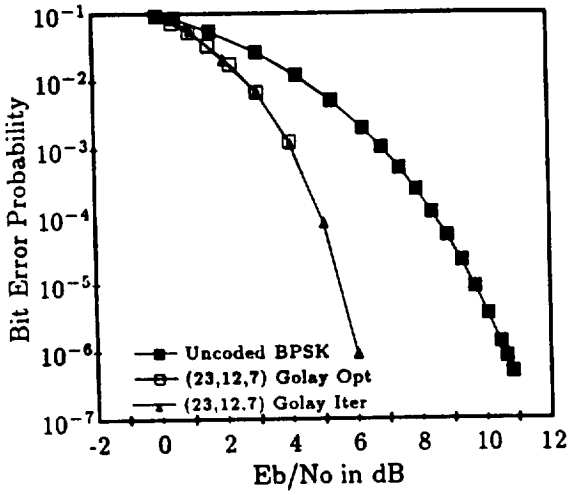
The average computational complexity of the iterative decoding algorithm is very small compared to the worst-case upper bound $\tilde{C}_{\max}$ and the computational complexity $A(T)$ by using the full code trellis with the Viterbi algorithm, as shown in Table 12.1. For example, consider the $(64, 45, 8)$ extended BCH code. At SNR = 2 dB, the average number of real operations required to decode a received sequence of 64 symbols is 40,000 compared to $\tilde{C}_{\max} = 57,182$ and $A(T) = 4,301,823$. At SNR = 5 dB, $C_{\text{ave}} = 750$.

12.6 ERROR PERFORMANCE

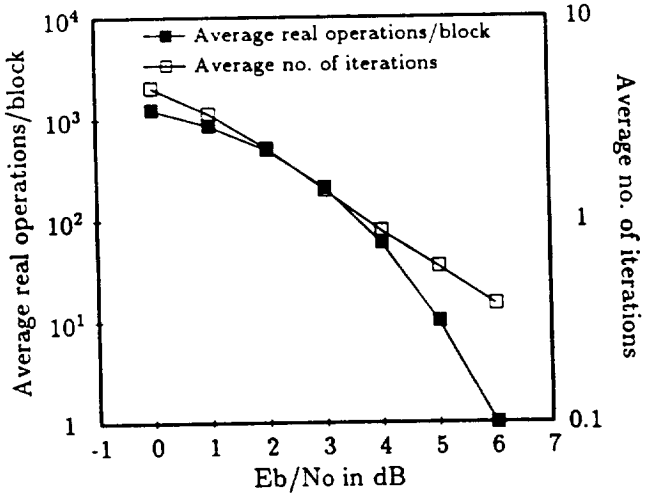
Since the iterative decoding algorithm uses the Chase algorithm-II to generate candidate codewords for optimality test and minimum-weight trellis search, it may be regarded as an improved Chase algorithm-II and hence achieves asymptotically optimal error performance with a faster rate. An upper bound on the block error probability can be found in [101]. Simulation results show that the iterative decoding algorithm achieves near-optimum error performance and significantly outperforms the Chase algorithm-II.

The iterative decoding algorithm has been simulated for all the codes given in Table 12.1. For RM codes, majority-logic decoding is used for generating candidate codewords and otherwise a bounded distance-t decoding algorithm is used. The bit error rates of some codes listed in Table 12.1 are shown in Figures 12.2(a)-12.6(a) and their average computational complexities and average numbers of decoding iterations are shown in Figures 12.2(b)-12.6(b).

Consider the $(32, 16, 8)$ extended primitive BCH code (also an RM code). The iterative decoding algorithm achieves practically the same error performance as the optimum MLD as shown in Figure 12.3(a). To achieve a bit error rate of 10^{-5}, it requires a SNR of 5.6 dB. We also see that the iterative decoding algorithm achieves a 0.55 dB coding gain over the Chase decoding algorithm-II at the bit error rate 10^{-5}. While the Chase decoding algorithm-II requires 16

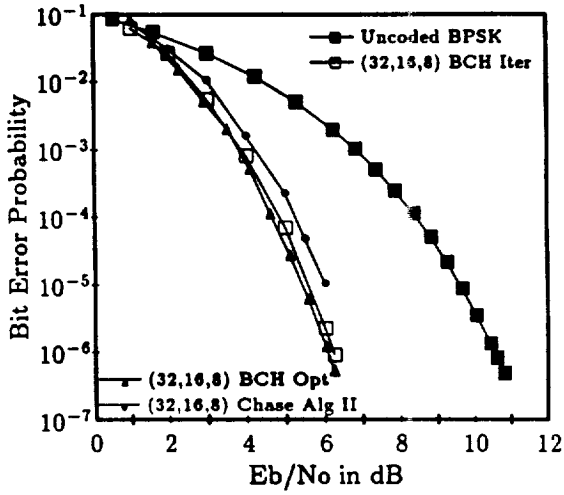


(a) Bit error probability

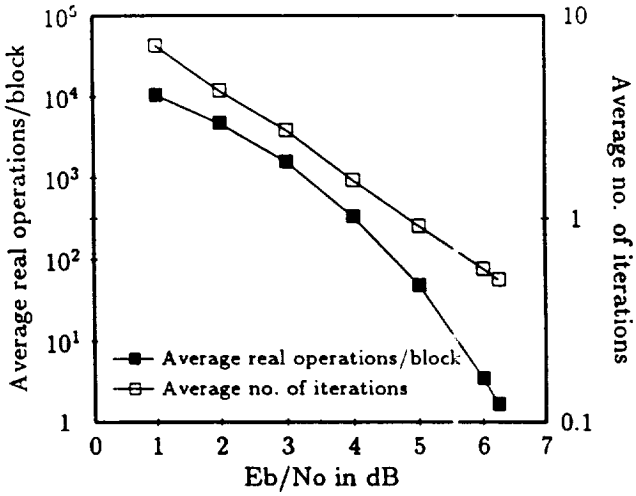


(b) Average computational complexity and average numbers of decoding iterations

Figure 12.2. Error performance and computational complexity of the iterative decoding algorithm for Golay $(23, 12, 7)$ code.

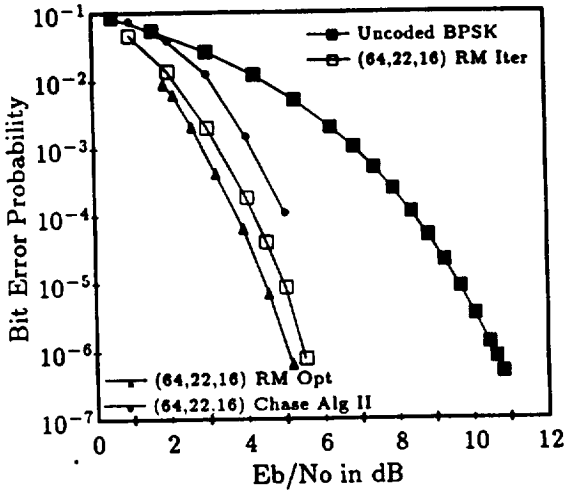


(a) Bit error probability

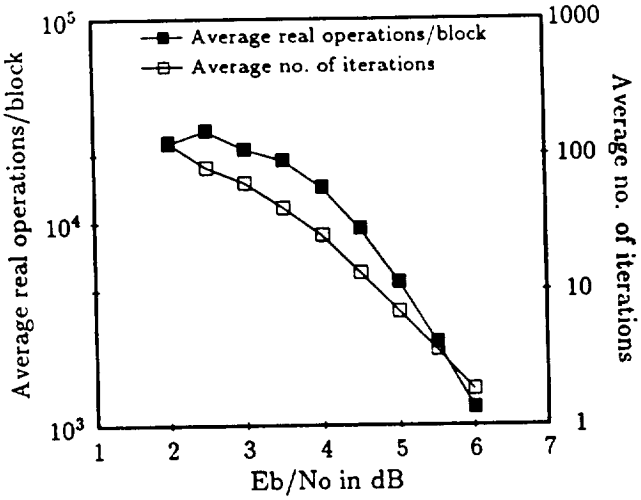


(b) Average computational complexity and average numbers of decoding iterations

Figure 12.3. Error performance and computational complexity of the iterative decoding algorithm for $(32, 16, 8)$ extended primitive BCH code.

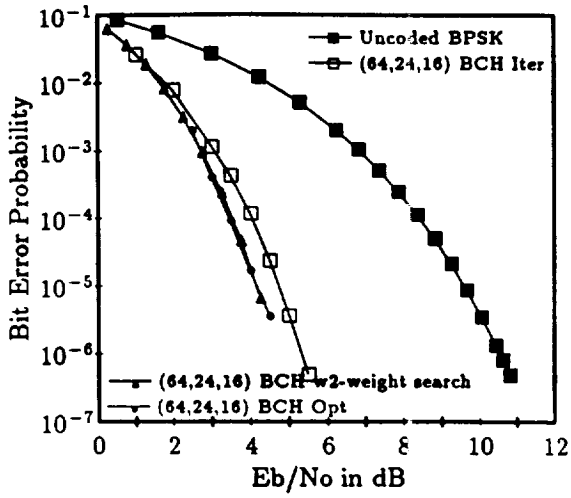


(a) Bit error probability

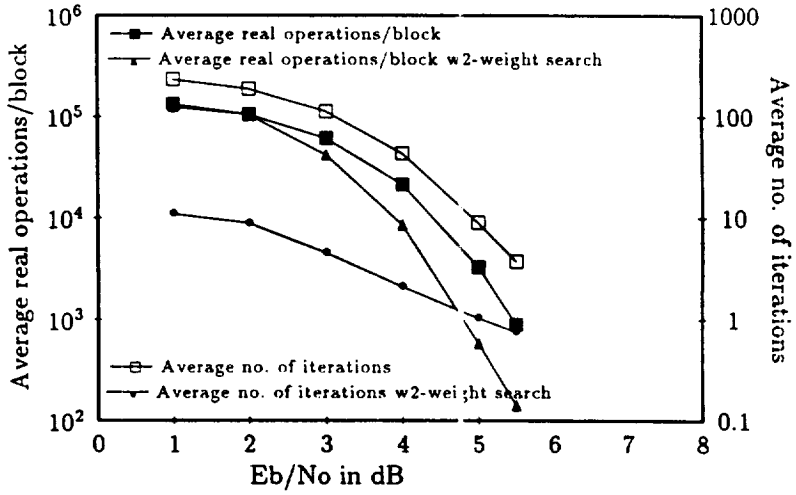


(b) Average computational complexity and average numbers of decoding iterations

Figure 12.4. Error performance and computational complexity of the iterative decoding algorithm for the $(64, 22, 16)$ RM code.

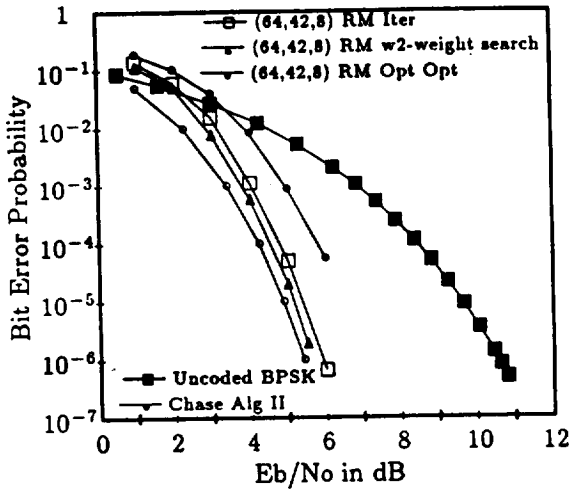


(a) Bit error probability

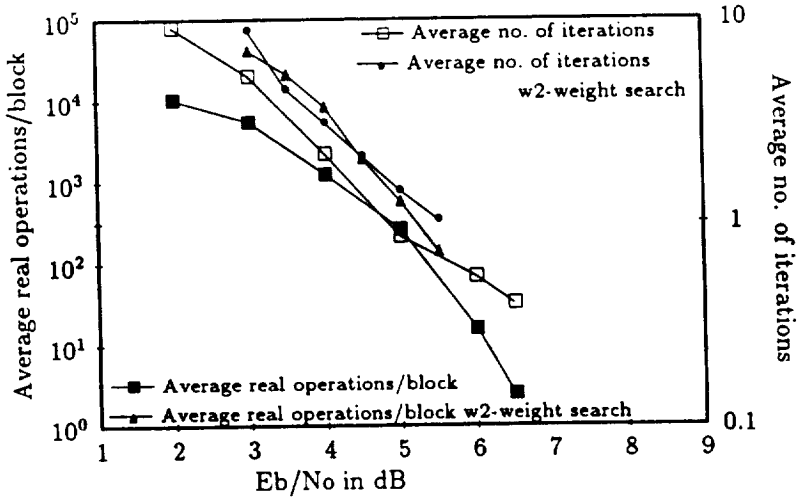


(b) Average computational complexity and average numbers of decoding iterations

Figure 12.5. Error performance and computational complexity of the iterative decoding algorithm for the $(64, 24, 16)$ extended BCH code.



(a) Bit error probability



(b) Average computational complexity and average numbers of decoding iterations

Figure 12.6. Error performance and computational complexity of the iterative decoding algorithm for the $(64, 42, 8)$ RM code.

decoding operations, the iterative decoding algorithm takes only one decoding operation (or iteration) on average at the BER of $= 10^{-5}$.

For all simulated codes of length 64 except the extended Hamming code, the iterative decoding algorithm loses less than 0.5 dB with respect to optimal MLD decoding at the BER of 10^{-6}. However, the iterative decoding algorithm yields a drastic reduction in computational complexity compared with the optimal Viterbi decoding based on the full trellis of the code. As shown in Figures 12.6(a) and 12.6(b), for the $(64, 42, 8)$ RM code, the performance degradation of the iterative decoding algorithm compared with the optimum Viterbi decoding based on the full trellis of the code is 0.5 dB at the BER of 10^{-6}. The SNR required to achieve a BER of 10^{-6} by the iterative decoding algorithm is 5.9 dB. At this SNR, the average number of real operations required to decode a received word is about 25, however, the optimal Viterbi decoding based on the full trellis diagram of the code requires 544, 640 real operations. This is a tremendous reduction in computational complexity with a small degradation in error performance. The average number of iterations required to complete the decoding at this SNR of 5.9 dB is 0.6. At the BER of 10^{-4}, the iterative decoding algorithm achieves a gain of 1.1 dB over the Chase algorithm-II with an average of 400 real operations and an average of 1.2 decoding iterations.

The error performance of the iterative decoding algorithm can be improved by using a larger search region for the optimal MLD solution c_{opt}. For example, we may use the region $R(c, w_3)$ which consists of all the codewords at the distances w_1 (minimum distance) and w_2 (next to minimum distance) from the current tested codeword for searching c_{opt}. In the case that $G(c, w_2) < L(c, r) \leq G(c, w_3)$, the decoder searches the purged trellis diagram $T_{w_2}(c)$ centered around c for finding c_{opt}, where $T_{w_2}(c)$ consists of c and all the paths of the overall trellis diagram of the code which are at distances w_1 and w_2 from c. We call this search the w_2-weight trellis search. Of course, the improvement is achieved with some additional computational complexity.

Consider the $(64, 24, 16)$ extended BCH code. The w_2-weight trellis search achieves practically optimum performance of MLD as shown in Figure 12.5(a). It recovers the 0.6 dB loss in performance based on the minimum-weight trellis search. The SNR required to achieve the BER of 10^{-5} is 4.2 dB. At this SNR, the average number of real operations and the average number of itera-

tions for decoding a received sequence are about 6,000 and 2, respectively as shown in Figure 12.5(b). The maximum number of real operations required for the improved decoding algorithm is 175,232. At the same BER of 10^{-5}, the minimum-weight trellis search algorithm requires an average of 5,000 real operations and 15 iterations. We see that 0.6 dB gain in performance over the minimum-weight trellis search algorithm is achieved at the expense of a modest additional complexity. If the modified algorithm with a larger search region is applied to the $(64, 42, 8)$ RM code, a 0.3 dB coding gain over the minimum-weight trellis search algorithm at the BER of 10^{-5} is achieved with a very modest additional computational complexity on average as shown in Figure 12.6(a) and 12.6(b).

12.7 SHORTCOMINGS

The above iterative decoding algorithm is very simple and provides good error performance with large reduction in decoding complexity. However, it has several major shortcomings and for long codes, there is a significant performance degradation compared to MLD for low to medium SNRs. First, the algebraic decoder used in the algorithm may fail to decode the modified hard-decision received sequence to generate a candidate codeword for testing. Therefore, the decoding algorithm may have a decoding failure. Second, some test error patterns may result in the same candidate codeword and hence useless decoding iterations unless some preprocessing is done to rule out or reduce the repetitions. Third, there is no guarantee that the candidate codewords are generated in the order of increasing improvement, i.e., the next generated candidate codeword has larger correlation metric than the previous ones. This may result in unnecessary decoding iterations and prevent a fast decoding convergence. Fourth, the sufficient conditions for optimality and nearest neighbor tests are based on only the current candidate codeword, and information from previously tested candidate codewords is not being used. This information may help to narrow down the search of the ML codeword and reduce the possibility that it slips through the test without being detected. Finally, the performance degradation is large for codes whose minimum weight codewords do not span the codes. For example, the minimum weight codewords of the extended $(64, 24, 16)$ BCH

code do not span the code and their is a 0.6 dB coding gain loss compared to MLD at the BER of 10^{-5}.