NASA/CR—1999-209402

DTVI–59

Investigation of Near Shannon Limit Coding Schemes

S.C. Kwatra, J. Kim, and Fan Mo
The University of Toledo, Toledo, Ohio

September 1999

The NASA STI Program Office . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program Office plays a key part in helping NASA maintain this important role.

The NASA STI Program Office is operated by Langley Research Center, the Lead Center for NASA's scientific and technical information. The NASA STI Program Office provides access to the NASA STI Database, the largest collection of aeronautical and space science STI in the world. The Program Office is also NASA's institutional mechanism for disseminating the results of its research and development activities. These results are published by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA's counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or cosponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services that complement the STI Program Office's diverse offerings include creating custom thesauri, building customized data bases, organizing and publishing research results . . . even providing videos.

For more information about the NASA STI Program Office, see the following:

- Access the NASA STI Program Home Page at ***http://www.sti.nasa.gov***

- E-mail your question via the Internet to help@sti.nasa.gov

- Fax your question to the NASA Access Help Desk at (301) 621-0134

- Telephone the NASA Access Help Desk at (301) 621-0390

- Write to:
  NASA Access Help Desk
  NASA Center for AeroSpace Information
  7121 Standard Drive
  Hanover, MD 21076

NASA/CR—1999-209402

DTVI–59

Investigation of Near Shannon Limit Coding Schemes

S.C. Kwatra, J. Kim, and Fan Mo
The University of Toledo, Toledo, Ohio

Prepared under Grant NAG3–1718

National Aeronautics and
Space Administration

Glenn Research Center

September 1999

Trade names or manufacturers' names are used in this report for identification only. This usage does not constitute an official endorsement, either expressed or implied, by the National Aeronautics and Space Administration.

Available from

NASA Center for Aerospace Information
7121 Standard Drive
Hanover, MD 21076
Price Code: A05

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22100
Price Code: A05

TABLE OF CONTENTS

Table of contents III

List of tables VI

List of figures VII

Chapter 1: Introduction 1

1.1 Coding 2

1.2 Block coding and decoding 3

1.2.1 Definitions of block codes 3

1.2.2 Block coding 3

1.2.3 Block decoding 4

1.2.4 Common block codes 5

1.2.4.1 Single bit parity-check codes 5

1.2.4.2 Repeated codes 6

1.2.4.3 Hamming codes 6

1.2.4.4 Cyclic codes 6

1.2.4.5 Other block codes 7

1.3 Convolutional coding and decoding 7

1.3.1 Definition of convolutional codes 7

1.3.1.1 Code tree 8

1.3.1.2 Trellis 9

1.3.1.3 State diagram 10

1.3.2 Convolutional encoding 10

1.3.3 Convolutional decoding 11

1.3.3.1 Maximum likelihood decoding of convolutional codes 11

1.3.3.2 Sequential decoding of convolutional codes 12

Chapter 2: Turbo codes 13

2.1 Concepts of turbo codes 14

2.1.1 Turbo encoding system 14
2.1.1.1 Recursive systematic convolutional codes (RSCC) 15
2.1.1.2 Interleaver 16
2.1.1.3 Puncturing pattern 16
2.1.2 Turbo decoding system 19
2.1.2.1 General turbo decoding scheme 19
2.1.2.2 MAP algorithm 20
2.2 Performance of turbo codes 28
2.3 Output weight distribution and performance bounds for turbo codes 32
2.3.1 Output weight distribution 32
2.3.2 Performance bounds 34
2.4 Relation between the system parameters and output weight distribution 36
2.4.1 Generator polynomial 37
2.4.2 Interleaver 41
2.4.3 Puncturing pattern 44
Chapter 3: Iterative decoding of block codes 51
3.1 Construction of trellis for block codes 51
3.1.1 Characteristics of the trellis constructed from block codes 51
3.1.2 The method of construction 54
3.2 Iterative log-likelihood decoding of binary block codes 55
3.2.1 Log-likelihood algebra 55
3.2.2 Soft-in/soft-out decoder 56
3.2.3 Iterative decoding algorithm 57
3.2.4 Optimal and sub-optimal algorithm 58
3.3 Implementation of the algorithm 60
3.3.1 Straight-forward implementation 60
3.3.2 Dual code implementation 61

3.3.3 A decoding example by using straight-forward implementation 62
3.3.3.1 Constructing trellis for information bits 63
3.3.3.2 The decoding system 65
3.3.3.3 The calculation of extrinsic value from constructed trellis 66
3.3.3.4 Simulation result 69

References 70

LIST OF TABLES

Table 2.1	Performance of high rate turbo codes	30
Table 2.2	Factors for the performance of turbo codes	31
Table 2.3	Performance of 4/5 turbo codes with different size interleavers	43
Table 2.4	Floor flaring effect for different interleaver sizes	44
Table 2.5	Puncturing patterns selected for different code rates	45
Table 2.6	Selected bit locations after puncturing for 2/3 rate	48
Table 2.7	Selected bit locations after puncturing for 5/6 rate	48
Table 2.8	The locations selected by selecting different bits in each 10 parity bits for 5/6 rate	50

LIST OF FIGURES

Fig 1.1	Transmission with coding	2
Fig 1.2	An example of a typical convolutional encoder	7
Fig 1.3	The structure of code tree	8
Fig 1.4	A part of trellis between depth k and depth k+1	9
Fig 1.5.	The structure of state diagram	10
Fig 2.1	General encoding scheme of turbo codes	15
Fig 2.2	Punctured 23_31 RSCC1 with rate 4/5	18
Fig 2.3	Punctured 23_31 RSCC2 with rate 4/5	19
Fig 2.4	General decoding scheme of turbo codes	19
Fig 2.5	(5,7) RSCC	24
Fig 2.6	The state diagram of the (5 , 7) RSCC	24
Fig 2.7	Trellis diagram of (5, 7) RSCC	25
Fig 2.8	Soft-decoding bounds at different code rates	29
Fig 2.9	The performance of turbo codes at different code rates	30
Fig 2.10	The performance of turbo codes compared to some convolutional and block turbo codes, also the Shannon limit	31
Fig 2.11	An example of turbo encoder with M=2 and feedback polynomial $1+D+D^2$	38
Fig 2.12	Recursive encoder with generator polynomial (23 , 31)	41
Fig 2.13	Comparison of the (23, 31) and (31, 27) generator polynomials	41
Fig 2.14	The influence of interleaver size on the performance of turbo codes	43
Fig 2.15	Comparisons of different puncturing patterns for high rates at certain Eb/No	46
Fig 2.16	The performance of 5/6 code with different interleaver sizes	47
Fig 2.17	The improvement of the performance with the modified puncturing pattern at different interleaver sizes	49
Fig 2.18	The improvement of the performance with the modified puncturing pattern at code rates 5/6, 10/11, 15/16	50

Fig 3.1	The trellis constructed for a (7,4) Hamming code before expurgation	53
Fig 3.2	Expurgated trellis for (7,4) Hamming code	54
Fig 3.3	Soft-in / soft-out decoder	56
Fig 3.4	Iterative decoding procedure with soft-in / soft-out decoders	57
Fig 3.5	Full trellis for first information bit location	63
Fig 3.6	The final trellis with two ending states for first information bit location	64
Fig 3.7	The final trellis with two ending states for information bit location 2	64
Fig 3.8	The decoding system of (7,4) Hamming code while working on information bit 1	65
Fig 3.9	The decoding system of the (7,4) Hamming code while working on information bit 2	66

CHAPTER 1
INTRODUCTION

Turbo codes, representing the most important breakthrough in coding, are able to operate near Shannon limit. Extensive research results are being reported about this novel technique. The commonly accepted turbo coding is implemented by a system, which consists of two parallel concatenated recursive systematic convolutional encoders separated by an interleaver [1]. The maximum a posteriori probability (MAP) algorithm is applied for decoding because of its improved performance [2]. Since low-rate codes are not appropriate for commonly used applications, there is a need to develop high rate turbo codes [3]. It has been shown that some high rate codes have very good performance but others exhibit poor performance. It is claimed that selection of puncturing patterns has considerable influence on the performance [3]. In this report, performance of high rate turbo codes is analyzed based on the simulation results. For high rates with normal performance, different puncturing patterns have been selected in the simulations and their performance is compared. For special high rate codes with poor performance, an alternative puncturing algorithm is developed which shows significant improvement in the performance.

Iterative decoding of block codes has gained more and more interest recently. Log- likelihood algorithm is used in the decoding and the "symbol by symbol" MAP decoding is the optimal method [4]. The construction of trellis for block codes is the first and a key step in the decoding [5]. By the constructed trellis for each information bit, an extrinsic value can be calculated by using MAP algorithm, which is then used as the a priori value of the next iteration. The procedures for trellis construction, extrinsic value calculation, and iterative algorithm will be discussed in detail in this report.

Before the discussion of turbo codes and iterative block decoding, a review of coding, block codes and convolutional codes is given in this chapter. Turbo convolutional codes are discussed in Chapter 2 and iterative block decoding is introduced in Chapter 3.

1.1 Coding

A cost-effective system transmits information at a rate and a level of reliability that are acceptable. Two parameters are important in the design of a digital communication system. One parameter is the signal energy per bit to noise power spectral density ratio, E_b/N_0. The second parameter is the bandwidth. Practical considerations place a limit on the value of available E_b/N_0; it's followed that under some conditions it is impossible to provide acceptable quality because of inadequate E_b/N_0.

Channel coding is used to provide for the reliable transmission of the digital information over the channel. For a fixed value of E_b/N_0, coding is a good and practical way to improve the data quality. For fixed error rate, with the help of coding, we can decrease the requirement of the E_b/N_0, which will in turn decrease the required transmitted power.

Coding introduces the redundancy into the message based on a prescribed rule to detect the error and to correct the error. Transmission process with coding is shown in Fig 1.1. Channel encoder accepts message bits and adds redundancy according to the coding rule. Channel decoder exploits the redundancy to decide which message bit was transmitted.

The error, effect of the channel impairment, is minimized by coding. However, not all of the errors can be detected and corrected by coding. The correction capacity depends on the similarity between the acceptable and the unacceptable code words. Block coding and convolutional coding are the two most important and widely used methods in coding.

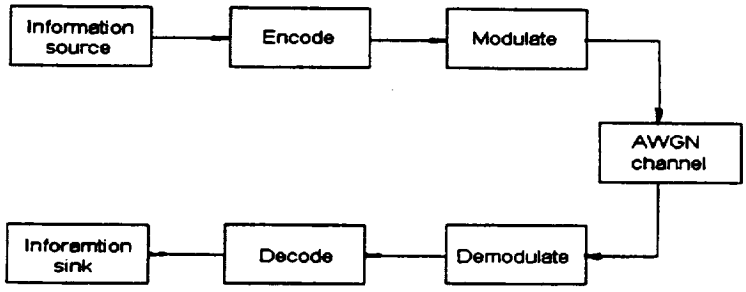

Fig 1.1 Transmission with coding

1.2 Block coding and decoding

1.2.1 Definition of block codes

Codes formed by taking a block of k information bits and the added m redundant bits to form a code word of $n = k + m$ bits are called block codes. These can be represented as (n, k) codes. The n-bit codeword consisting of k information bits and m redundant bits is called systematic code. The code where k information bits are not explicitly present in the codeword is called nonsystematic code.

The k information bits represent the 2^k equally likely messages. The total number of possible n-bit codewords is 2^n. There are $2^n - 2^k$ n-bit codewords that do not represent possible messages.

If we want to maintain the rate of information transmission, the transmitting rate should increase after the coding by $R_c / R_b = n / k$, where R_c, R_b are the coded and uncoded bit rates respectively.

1.2.2 Block coding

Assume that the uncoded word is $u = [u_1\ u_2\ u_3 \ldots\ u_k]$. The generation of a block code starts with a selection of the number m of parity bits to be added. Specify an H matrix,

$$H = \begin{vmatrix} h_{11} & h_{12} & \cdots & h_{1k} & 1 & 0 & 0 & \cdots & 0 \\ h_{21} & h_{22} & \cdots & h_{2k} & 0 & 1 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ h_{m1} & h_{m2} & \cdots & h_{mk} & 0 & 0 & 0 & \cdots & 1 \end{vmatrix} \qquad (1.1)$$

$m \times k$ $\qquad\qquad$ $m \times m$

which is made up of an $m \times k$ sub-matrix h and an $m \times m$ identity submatrix. Each h_{ij} in the matrix is either 1 or 0. Assume the coded words as $v = [u_1\ u_2 \ldots\ u_k\ p_1\ p_2 \ldots\ p_m]$, where v and H should satisfy the equation,

$$Hv^T = 0 \qquad (1.2)$$

To generate a code word v from u, we form a generator matrix G.

$$G = \begin{vmatrix} 1 & 0 & 0 & \cdots & 0 & h_{11} & h_{21} & \cdots & h_{m1} \\ 0 & 1 & 0 & \cdots & 0 & h_{12} & h_{22} & \cdots & h_{m2} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & 1 & h_{1k} & h_{2k} & \cdots & h_{mk} \end{vmatrix} \qquad (1.3)$$

$k \times k$ $\qquad\qquad$ $k \times m$

G consists of an identity submatrix of dimension $k \times k$, and a second sub-matrix, which is the transpose of h (one of the sub-matrics of H). The codeword v corresponding to each uncoded word u is

$$v = uG \qquad (1.4)$$

For each u generated, equation (1.2) should be satisfied.

1.2.3 Block decoding

Decoding the received codewords can be done by evaluating the correlation of the received word with all possible words, and the one that exhibits the closest correlation is determined as the transmitted codeword. This method is not efficient for codewords of large length. Block coding provides an alternate way to reduce the complexity of decoding.

Name the received message as r. It may or may not be the same as the transmitted codeword v. We can determine if r is equal to v by using the equation,

$$Hr^T = 0 \qquad (1.5)$$

If equation (1.5) can not be satisfied, that means there are at least one or even more bits in error. If the equation can be satisfied, we can not absolutely be certain that r is correct and equal to v, because there's the possibility that several errors occurred in the transmission and they happened to change the transmitted codeword into another possible codeword. When $r \neq v$, we assume

$$r = v + e \qquad (1.6)$$

where e is the error pattern. Thus we will also have

$$r^T = v^T + e^T \qquad (1.7)$$

The appearance of a 1 in the error pattern e indicates an error in the corresponding bit position and 0 indicates no error has been made.

We can begin the decoding from the evaluation of syndrome s of the received codeword r. Since Hv^T equals 0,

$$s = Hr^T = H(v^T + e^T) = Hv^T + He^T = He^T \tag{1.8}$$

If s is not equal to zero, that means there are one or more errors. s equals to zero means either there is no error or the error pattern is equal to a valid codeword. If s is not zero, we can calculate s by the equation

$$s = Hr^T \tag{1.9}$$

For single error case, we can compare s with each column of H. If the i_{th} column of H is identical to s, then the i_{th} bit of the codeword is in error. For more than one error case, we must solve (1.8) and identify the error patterns. The error pattern with fewest errors should be selected.

The number of the possible error patterns is 2^k. Thus the number of error patterns will be very large with the increase of k. However, there are a maximum number of errors that a code can correct, thus we can ignore the possibility of errors larger than that number since we can not correct them.

1.2.4 Common block codes

1.2.4.1 Single parity-check bit codes

Single parity-check bit coding is the simplest method in block codes. The theory of this method is as follows:

1) Adding a redundant bit p_1 at the end of the information bits, so

$$n = k + m = k + 1 \tag{1.10}$$

2) If the information bits have odd number of 1's, or equivalently, the addition of the information bits equals to 1, p_1 is set to be 1.

3) Otherwise, p_1 is set to be 0.

This method keeps an even number of 1's in the transmitted message. If the received message shows odd number of 1's, then error must have occurred in the transmission. This method works well only under the condition that the probability of more than 1 errors to occur in a codeword is quite low. Another shortcoming of this method is that it can only detect the error but can not detect which bit is in error. It follows that this method can not be used to correct the error.

1.2.4.2 Simple repetition codes

This method repeats a binary bit $2t+1$ times. Since $k= 1$ and $m = 2t$,

$$n = k + m = 1 + 2t \tag{1.11}$$

Repeated code with length $2t + 1$ can correct as much as t errors. But it will need significant bandwidth because the rate is changed to $1/(2t+1)$. Therefore such codes are inefficient.

1.2.4.3 Hamming codes

Assume d as the distance between each pair of codewords. Hamming distance (d_{min}) is defined as the minimum value of d. The greatest likelihood of confusion between words will be encountered for a codeword pair where d is the minimum. So the Hamming distance establishes the upper limit of the effectiveness of a code.

In Hamming code, we have

$$\text{Block length } n = 2^m - 1 \tag{1.12}$$

$$\text{Number of message bits } k = 2^m - 1 - m \tag{1.13}$$

$$\text{Number of parity bits } n - k = m \tag{1.14}$$

where $m \geq 3$. If $d_{min} = 2t + 1$, then the errors smaller than t bits can be corrected. In a (7,4) Hamming code ($m = 3$), the smallest Hamming weight for nonzero codewords is 3, so $d_{min} = 3$ and $t = 1$, it follows that single error can be corrected.

The parity check matrix $\boldsymbol{H}$ has m rows and n columns. Each column is unique and no column consists of all zeros. To form systematic code, all the columns are arranged to separate submatrix $\boldsymbol{h}$ and the identity submatrix $\boldsymbol{I}$.

1.2.4.4 Cyclic codes

Cyclic codes form a subclass of linear block codes and they have the advantage that they are easily encoded and decoded. Indeed, many of the important linear block codes are either cyclic codes or closely related to cyclic codes. Cyclic codes have two fundamental properties:

1) Linear property: The sum of two code words is also a codeword.
2) Cyclic property: Cyclic shift of codewords forms other valid codewords. Codewords

can be written in a cycle. There are $2^k - 1$ starting points to read the code, each related to the other with a shift.

Hamming code is an example of the cyclic code. Assume a *(7, 4)* Hamming code. The number of information bits $k = 4$. It follows that there are a total of 16 codewords for the Hamming code. Two groups of seven of them are precisely the cyclic-shift related words. The last two codewords, other than these fourteen, are *0000000* and *1111111*. For these two codes, any cyclic shift forms the same codeword.

1.2.4.5 Other block codes

Some other types of block codes include Hadamard code, extended code, Golay code, and BCH code.

1.3 Convolutional coding and decoding

1.3.1 Definition of convolutional codes

In block coding, the encoder generates n- bit codeword from a k-bit message. The code words are produced on block-by-block basis. So there must be a buffer to store the message before the encoding is done. In convolutional coding, the use of buffer is not needed. A convolutional encoder operates on the incoming message sequence continuously in a serial manner. An example of a typical convolutional encoder is shown in Fig 1.2, in which we see that a convolutional code is generated by combining the outputs of an M-stage shift register with the employment of N_A binary adders.

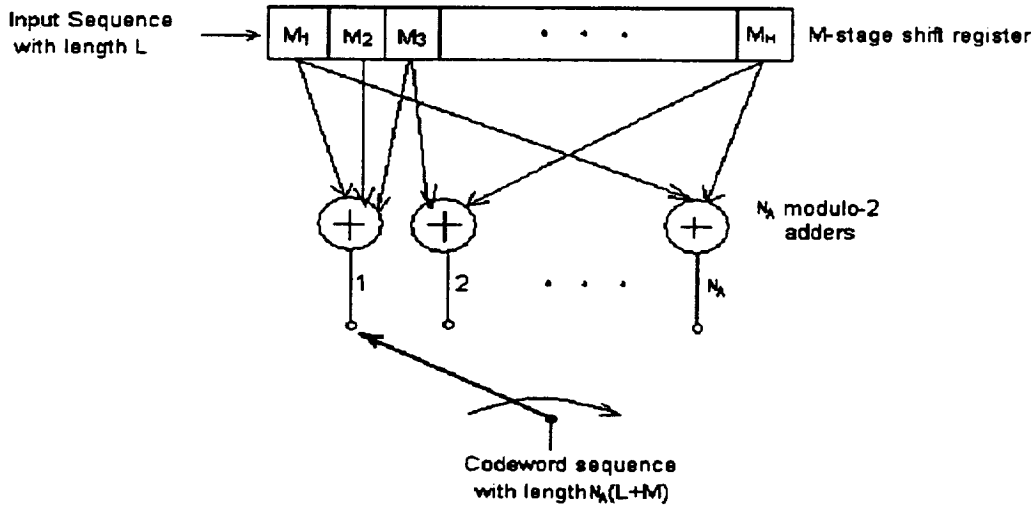

Fig 1.2 An example of a typical convolutional encoder

As shown in Fig 1.2, we assume that the length of the message is L and the system consists of an M-stage shift register and N_A modulo-2 adders, the code rate will be

$$r = \frac{L}{N_A(L+M)} \tag{1.15}$$

Normally, L is much larger than M, so the code rate can be simplified as $1/N_A$.

Constraint length is defined as the number of shifts over which message bit can influence the encoder output. The constraint length K equals $M+1$ in convolutional coding. The structural properties of a convolutional encoder are portrayed in graphical form by using three equivalent diagrams: code tree, code trellis, and state transition diagram.

1.3.1.1 Code tree

Fig 1.3 shows the first several stages of a code tree. Each branch of a tree represents an input symbol. Normally, input 0 specifies the upper branch in a tree, input 1 specifies the lower branch. A specific path in the tree is traced from left to right in accordance with the input sequence. The corresponding coded symbols on the branches of that path constitute the sequence supplied by the encoder to the discrete channel input. The tree becomes repetitive after k branches. The nodes become identical because the first bit has been shifted out of the register.

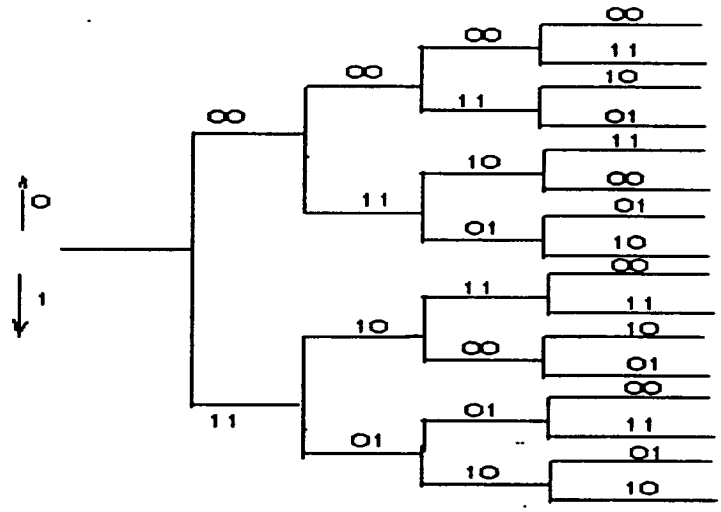

Fig 1.3 The structure of code tree

1.3.1.2 Code Trellis

The code tree can be transformed into a new form, called trellis. Trellis is a tree-like structure with remerged branches.

As in a tree, each input sequence corresponds to a specific path through the trellis. However, a trellis is more instructive than a tree in that it brings out explicitly the fact that the associated convolutional encoder is a finite-state machine.

State is defined as the most recent M message bits shifted into the encoder register. The state of this encoder can assume any one of the 2^{K-1} possible values. The trellis contains $L + K$ levels which are called as depth of the trellis. Trellis is preferred than tree because the number of nodes at any level of the trellis doesn't continue to grow as the number of incoming message bits increases. Fig 1.4 shows part of a trellis between depth i and depth $i+1$. The solid lines represent inputs of 0, and the dashed lines represent inputs of 1.

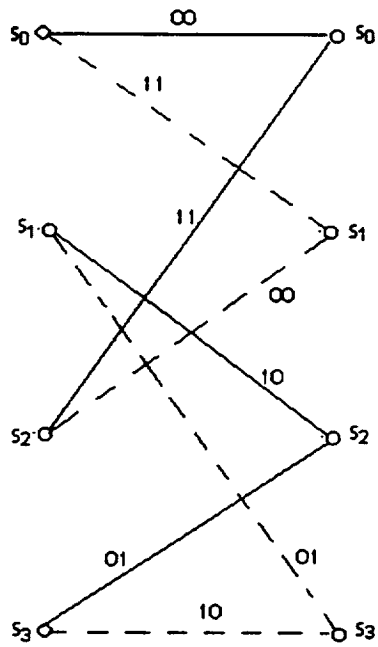

Fig 1.4 A part of trellis between depth i and depth $i+1$

1.3.1.3 State transition diagram

Though it looks very simple, the input- output relation of a convolutional encoder is completely described by its state diagram.

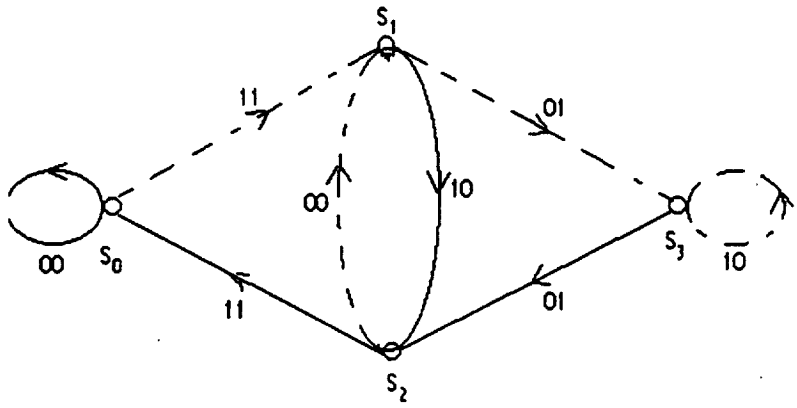

Fig 1.5. The structure of state diagram

The nodes of the state diagram represent the possible states of the encoder. Each node has 2^{M-1} incoming branches and 2^{M-1} outgoing branches. The label on each of the branches represents the encoder's output as it moves from one state to another.

1.3.2 Convolutional encoding

The operation of the encoder (Fig 1.2) proceeds as follows:

1) Assume the shift register is initialized.
2) The first bit of input data enters in the first register M_1.
3) During the message bit interval, the adder calculates N_A outputs.
4) The next message bit moves to M_1, and the first bit transfers from M_1 to M_2, and again, all N_A adder outputs are calculated.
5) This process continues until last bit of the message comes in M_1.
6) Enough 0's are added to the end of the message sequence, to allow the whole encoding process to be completed as the last bit leaves the last register.
7) The shift registers are in the original clear condition again.

1.3.3 Convolutional decoding

1.3.3.1 Maximum likelihood decoding of convolutional codes

Viterbi algorithm for the decoding of convolutional codes is developed. In the development process, first, for the binary symmetric channel (BSC), the maximum-likelihood decoder reduces to a minimum (Hamming) distance decoder. Second, the trellis representation is used to establish the basic concepts of the Viterbi algorithm.

Assume v as the input code vector of the channel, and r denotes the corresponding received vector. Vector r may differ from vector v if error occurs due to the channel noise. However, from the received r, we can estimate v. The decoding rule for choosing the estimate of v, given the received vector r, is said to be optimum when the probability of decoding error is minimized. So the maximum-likelihood decoding rule for the binary symmetric channel is as follows: Choose the estimate x that minimizes the Hamming distance between r and v. Thus for the binary symmetric channel, the maximum-likelihood decoder reduces to a minimum distance decoder.

Thus we may decode a convolutional code by choosing a path in the code tree whose coded sequence differs from the received sequence in the fewest number of places. We may equally limit our choice to the possible paths in the trellis representation of the codes.

Viterbi algorithm makes sequence of decisions when working through the trellis. The algorithm operates by computing a "metric" for every possible path in the trellis. The metrics of the 2^{K-1} possible paths entering the node are compared and the one with the lower metric is retained. The paths that are retained are called survivors. No more than 2^{K-1} survivor paths and their metrics will ever be stored. The relatively small list of paths is always guaranteed to contain the maximum- likelihood choice.

The steps can be described as:

1) Starting at level $i = M$, compute the metric for the single path entering each state of the encoder. Store the survivor and its metric for each state.
2) Increment the level i by 1. Compute the metric for all the paths entering each state by adding the metric of the incoming branches to the metric of the connecting survivor from the previous time unit. For each state, identify the path with the lowest metric as the survivor of step 2. Store the survivor and its metric.

3) If level $i < L + M$, repeat step 2, otherwise, stop.

Viterbi algorithm is a maximum-likelihood decoder, which is optimal for a white noise Gaussian channel.

1.3.3.2 Sequential decoding of convolutional codes

This method is sub-optimal but can avoid the computation of the likelihood, or metric, of every path in the trellis, thereby reducing computational complexity and allowing the constraint length K to take on very large values. Although sequential decoding algorithms are not as good as maximum likelihood decoding algorithms, they are computationally effcient for large K.

Sequential decoding is an intuitive trial-and-error technique for searching out the correct path in a code tree. During the course of this search, the decoder moves forward and backward in the code tree, one node at a time. The decision to move forward or backward is determined by the manner in which the metric of the algorithm varies along the path followed by the decoder.

Several algorithms have been devised for the sequential decoding of convolutional codes. Fano algorithm is probably the most important because it has the useful feature that it uses very little storage. In Fano algorithm, the decoder moves forward and backward, the decision is made by comparing the path's Fano metric at the node with a running threshold maintained by the decoder.

In addition to the computer requirements for executing the Fano algorithm, the decoder contains a buffer to store the received sequence, and a replica of the encoder.

CHAPTER 2
TURBO CODES

There exists a limiting value of E_b/N_0 below which error-free communication is impossible at any information rate. This value of E_b/N_0 is called as Shannon limit. It's not possible in practice to reach Shannon limit, because it will cause the bandwidth requirement and implementation complexity to increase without bound. Shannon's work provided a theoretical proof for the existence of codes that can improve the BER performance, or reduce the E_b/N_0 required. Our aim in coding and decoding is to get as close to the Shannon limit as possible.

Low BER in high noise environment requires the very complex channel coding and decoding schemes. According to Shannon's theorem, performance of long random codes can approach Shannon's limit. However, long random codes are extremely difficult to decode generally.

Turbo coding, defined as the process of using parallel concatenation in conjunction with recursive systematic convolutional codes (RSCC), can produce codes with performance close to the Shannon limit. As mentioned above, Shannon limit can be reached when decoding large random codes; so in addition to a large minimum distance, good codes should have a distance distribution that mimics that of random coding. Turbo codes can be designed to generate a weight distribution similar to that of random codes. It requires encoding the information as well as the interleaved version of the information through a pseudo-random interleaver. The input is presented as blocks of bits.

Turbo coding is regarded as an important new technology developed in recent years because it leads the error control coding techniques finally to get very close to the Shannon limit. The performance of turbo codes is much better than all other ever designed block or convolutional coding techniques. Turbo codes are so efficient because they combine several codes by concatenation, maximize the use of channel information,

and have random like distribution of codewords. This approach has the significant error correcting capacity even at very low E_b/N_0.

Though turbo coding is a newly invented error correcting technique, a large number of research papers have been published. Turbo coding techniques have progressed very rapidly and we can expect several commercial applications in the near future. Most of the research work has been on finding the exact explanation of the extraordinary performance of turbo codes and providing methods to obtain an even further improvement on the performance of turbo codes. In earlier research work, the outstanding performance of turbo codes was shown by computer simulations, and some theoretical explanations for the simulation results were discussed [1][13][17]. Then, some important components and parameters of the coding system became the main concentration. Researchers analyzed the theory about generator polynomial [20], interleaver [22][23], puncturing pattern [3][24] and the decoding algorithm [14][15], and tried to modify them to achieve even better performance. At the same time, factors such as system complexity, execution time, and cost were considered. In the most recent two or three years, output weight distribution has been found to decide the performance of turbo codes [25]. One area of investigation is the influence of the factors such as interleaver and generator polynomial on the output weight of the turbo codes. It is of interest to determine a way to achieve the best estimation of the output weight distribution when all the system parameters have been decided. An accurate estimation will be very helpful for the evaluation of the performance of the system.

2.1 Concept of turbo codes

2.1.1 Turbo encoding system

Turbo codes are encoded by concatenating two RSCC's using an interleaver. When a block of message bits is input to the system, they are encoded directly with one of the two RSCC's, called RSCC1. The same block of message bits are interleaved by a pseudo-random interleaver before encoding with another RSCC, called RSCC2. After the parity sequences are generated by RSCC's, they are punctured by puncturer1 and puncturer2 to increase the code rate. General encoding scheme of turbo codes is shown in

Fig 2.1.

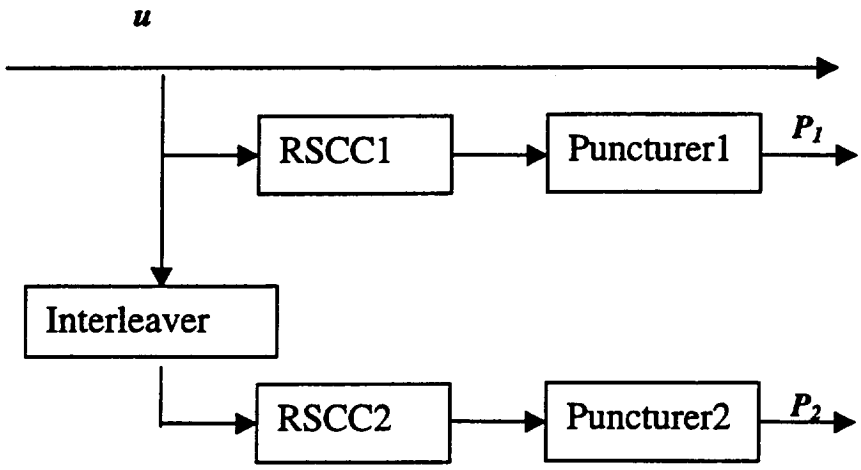

Fig 2.1 General encoding scheme of turbo codes

2.1.1.1 Recursive systematic convolution codes (RSCC)

RSCC's are constructed from NSCC's (Non-systematic Convolution Codes) by using a feed back loop. They perform better than the best NSCC's at any SNR, especially for high code rate. RSCC's generator is called as *a_b* RSCC. '*a*' and '*b*' represent octal numbers that are converted to binary to represent the connections in a generator circuit where *a* is called as FB(feedback) connection and *b* as FF(feed-forward) connection.

Assume the generator matrix of a nonrecursive convolutional code has the form

$$G_{NR}(D) = [g_1(D) \quad g_2(D)], \tag{2.1}$$

the equivalent generator matrix of the recursive systematic encoder is

$$G_R(D) = \left[1 \quad \frac{g_2(D)}{g_1(D)}\right] \tag{2.2}$$

where $g_1(D)$ and $g_2(D)$, respectively, represent the feedback and feedforward connections of the RSC encoder. The impulse response of a well designed memory M RSCC will repeat itself after 2^{M-1} bits [2].

2.1.1.2 Interleaver

To achieve the best possible performance of turbo codes, using a good interleaver is the most important factor. Most of the input sequences, after going through the RSCC's, have a random-like output weight distribution. However, there exist some input sequences which cause low output weights. These low weight codewords cause the codes to perform poorly. The use of interleaver in the encoding of turbo codes is helpful to reduce the number of low output weight codewords generated by the single RSCC. When some of the input words produce low weight output codewords through RSCC1, the interleaver makes most of them to produce higher weight codewords through RSCC2.

The interleaver permutes the information bits in an alternative order to make the output of RSCC2 (P_2) appear to be independent of the information sequence (u) and therefore random-like, but at the same time, still have a structure that permits decoding. Random interleaver is preferred. Size of $L=A\times A$ memory is used where the bits to be interleaved are stored. These bits are always read in through the rows of the memory, then read out by using pseudo-random algorithm to implement interleaving. The correlations between these bits are changed in the process.

The randomness of an interleaver in a turbo-code scheme can be tested by using computer simulations. Deinterleaving, the inverse function of interleaving, is implemented after the decoding.

2.1.1.3 Puncturing pattern

Turbo coding is an important new technology that allows the operation of coded modulation schemes near channel capacity on power-limited channels. So, it can be used to offer near-capacity performance for deep space and satellite channels. However, it is desirable that the performance of turbo-coding schemes be also available for bandlimited channels. Trellis-coded M-PSK schemes have been proposed for bandwidth-efficient modulation and coding, but the carrier recovery faces the problem that the receiver is forced to operate below the recovery loop's threshold. High rate turbo codes, which are both power and bandwidth efficient, may be the solution to this problem.

Puncturing method is used to achieve higher rate codes. Assume that the original code has a rate of R_0. It means that for transmitting each information bit, $1/R_0$ bits are

transmitted through the channel. Also assuming the puncturing period is N_p and in each period the puncturing pattern is similar, we can construct a puncture matrix with dimension $(1/R_0) \times N_p$, with the elements in the matrix either 1 or 0. 1 represents that the corresponding bit is retained and 0 represents that the bit is punctured.

An example to show how to achieve high rate is as follows. Let R_0 = 1/2, we puncture the code with a period 4 and the 2 x 4 puncture matrix is defined as,

$$\begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \tag{2.3}$$

The rate of the original codes is 1/2 because for every 4 information bits, 8 bits are sent through the channel. After the puncturing, the rate of the code is changed into 4/5, because now, only five bits are sent for the 4 information bits.

High rate turbo codes are obtained when we use the concept of puncturing on turbo codes. In turbo coding, the input data go to the RSCC1 directly and go to the RSCC2 after interleaving. RSCC1 and RSCC2 can be identical or not. The systematic information bit u_i is transmitted directly. RSCC1 and RSCC2 will produce the parity bits, denoted as p_{1i} and p_{2i} as shown in Fig 2.1. The rate for the RSCC1 and RSCC2 are both 1/2 when k parity bits are added to the k information bits and transmitted for each of them. In this case, for the whole system, $2k$ parity bits are transmitted through the channel together with the k information bits, so the rate for the system is 1/3. Any code rate higher than 1/3 for turbo codes is called high rate. The code rate of the system can be calculated from the code rates of the 2 RSCC's from the following equation,

$$\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2} - 1 \tag{2.4}$$

R_1 and R_2 can be different, but they should satisfy $R_1 \leq R_2$ for best decoding performance [3].

For turbo codes, in order to obtain good results from iterative decoding, only the parity bits can be punctured. Thus, after puncturing, the range of the code rates of each RSC encoder is between R_0 and N_p / N_p+1. The rate N_p / N_p+1 appears when only one bit in each puncturing period is retained. Generally, high rate codes with a rate

$$R = N_p / N_p+1,\ 2 \leq N_p \leq 16 \tag{2.5}$$

are considered for constituent encoders with memory size $M = 4$. They will achieve code rates from 0.67 to 0.94. Similar rates are always selected for two RSCC's, thus we have $R_1 = R_2$. From (2.6), we can determine that only when

$$R_1 = R_2 = \frac{2N_p}{2N_p + 1}, \tag{2.6}$$

R will have the value N_p / N_p+1. It follows that for each $2N_p$ information bits, only 2 parity bits, each from one of the two RSC encoders, will be transmitted after puncturing. Thus there are $(2N_p)^2$ possible puncturing patterns to be considered in total.

Here we give an example to show how to achieve a high rate turbo code with rate of 4/5. Each of the two RSC encoders should have a rate of 8/9. Thus, we have $R_0 = 1/2$, $N_p = 4$, $2N_p = 8$. The following 2 puncturing matrics (2 x 8) are applied. P_A is for the RSCC1 and P_B is for RSCC2.

$$P_A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.7}$$

$$P_B = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{2.8}$$

Notation $P(c_1, c_2)$ can be used to indicate the puncturing pattern of turbo codes. If the c_{1th} bit in each period of $2N_p$ parity bits is saved for the RSCC1, and the c_{2th} bit is saved for RSCC2, it is described as $P(c_1, c_2)$. So in our example, we have $P(1, 6)$. Figures 2.2 and 2.3 show the punctured turbo codes achieved according to the puncturing matrics (2.7) and (2.8). Generator matrics used here are both 23_31 for the two encoders.

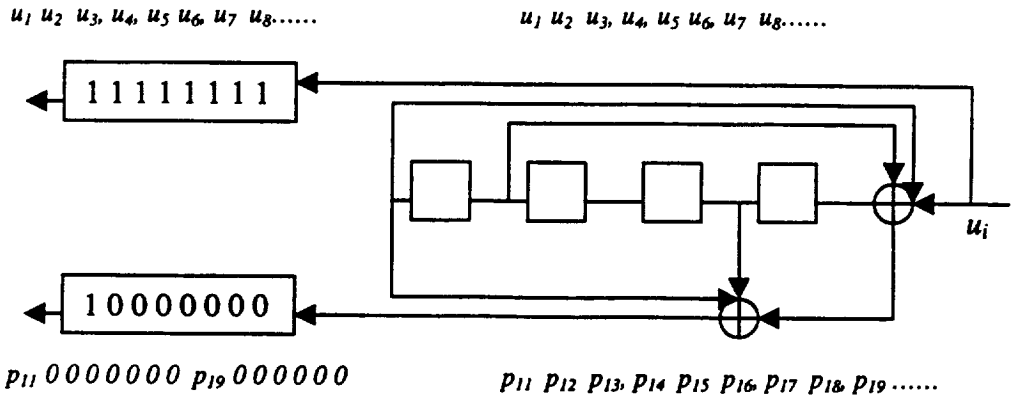

Fig 2.2 Punctured 23_31 RSCC1 with rate 4/5

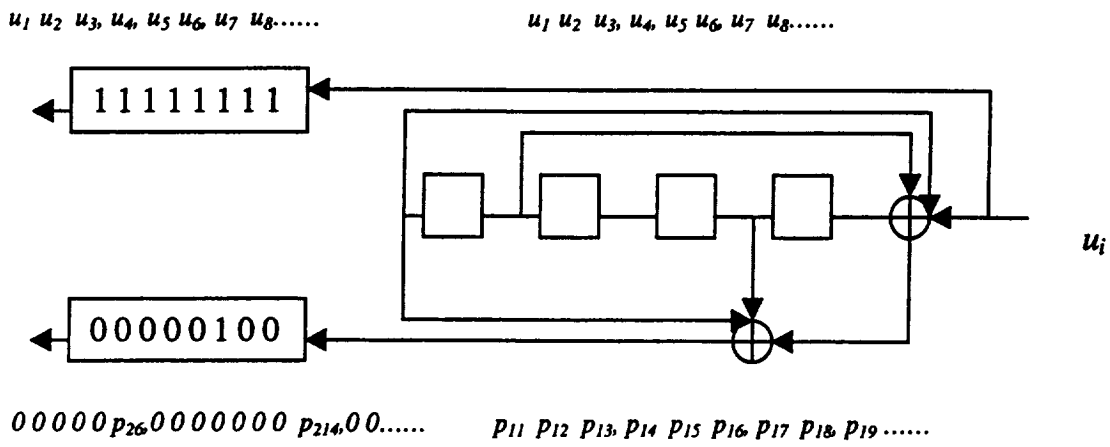

Fig 2.3 Punctured 23_31 RSCC2 with rate 4/5

2.1.2 Turbo decoding system

2.1.2.1 General turbo decoding scheme

The decoding system of turbo codes is much more complicated as compared to the decoding system for convolutional codes. The general scheme for turbo decoding is shown in Fig 2.4.

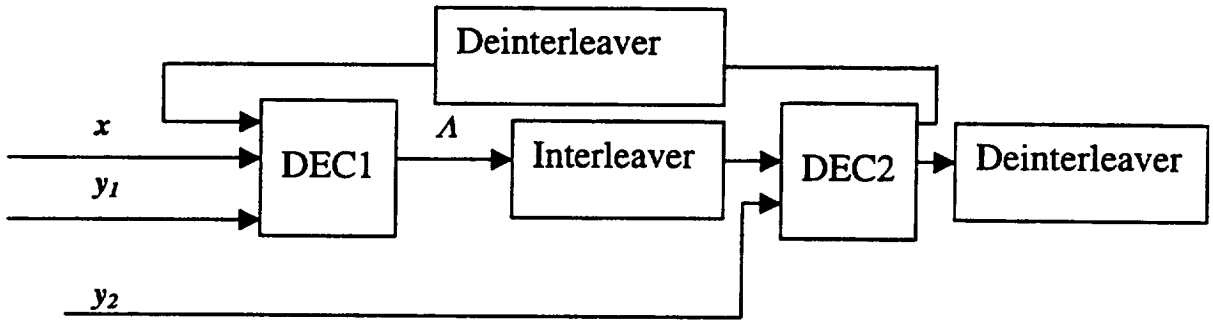

Fig 2.4 General decoding scheme of turbo codes

The system has two decoders. The first soft output decoder is used with the inputs being the systematic information and the output of the RSCC1 (noise added). The output of this decoder is an estimate of the information sequence and is called as reliability value Λ. The input of the second soft output decoder is the interleaved new estimate Λ together with the parity bits form the RSCC2. The second DEC produces a new estimate of the interleaved information bits.

The performance of the system can be improved by iteration, or we say adding a feedback path from the output of the second decoder to the input of the first decoder. Thus the first decoder can use all the information available instead of only using the

systematic information and the output from the first RSCC. The feedback information should be independent of the information generated by DEC1, otherwise it will cause positive feedback and the decoding could be unstable.

2.1.2.2 MAP algorithm

Soft output decoding is applied for turbo codes to improve the performance since all the information from the channel can be used without any loss by this method. Several algorithms are used to implement soft decision. Among them, MAP algorithm (Maximum a Posteriori Probability Algorithm) is the optimal one for decoding. The other algorithms also widely used include Max-log MAP (a simplification of MAP algorithm), and SOVA (Soft Output Viterbi Algorithm).

MAP algorithm gives both the decision for every bit and the reliability value for the bit. This optimal method can minimize the probability of bit error. A value defined as

$$\Lambda(u_i) = \ln \frac{P(u_i = 1)}{P(u_i = 0)} \tag{2.9}$$

is used to determine a soft output value, where P denotes the probability of u_i equal to 1 or 0, the sign of $\Lambda(u_i)$ determines whether the bit is a 0 or 1 while the magnitude determines the reliability of the decoded bit. Natural log base is always used. For derivation of the MAP algorithm, we use the notations below,

$r_{i1}^{\ i2}$: received sequence from states at time $i1$ to time $i2$

r_0^f : the entire received sequence (corrupted by noise)

r_i : the received information at time unit i, $R_i = (x_i, y_i)$

x_i : Information bit at time unit i

y_i : parity bit at time unit i

S_i : the state of encoder at time unit i

s : value of S_i

s': value of S_{i-1}

s & $s' = 0, 1, \ldots M_s - 1$, where M_s is the total number of states

MAP algorithm gives the decision and the reliability value for any bit given that all bits have been received, so we have

$$P(u_i = a) = \sum_{(s',s)\to u_i=a} P\{S_{k-1} = s'; S_k = s \mid r_0^f\} \tag{2.10}$$

In this equation, u_i is the input information bit at time unit i. The value of u_i equals to a which is either 1 or 0. $(s', s) \rightarrow u_i = a$ means the possible state transition at time unit i while the input bit is a. Equivalently, based on Baye's rule we have

$$P\{S_{i-1} = s'; S_i = s \mid r_0^f\} = \frac{P\{S_{i-1} = s'; S_i = s; r_0^f\}}{P\{r_0^f\}} \tag{2.11}$$

Define a

$$\sigma_i(s', s) = P\{S_{i-1} = s'; S_i = s; r_0^f\} \tag{2.12}$$

Then, we have

$$\Lambda(u_i) = \ln \frac{\sum_{(s',s)\to u_i=1} \sigma_i(s', s)}{\sum_{(s',s)\to u_i=0} \sigma_i(s', s)} \tag{2.13}$$

In MAP algorithm, the probability of state transition is split into three portions.

$$\sigma_i(s', s) = \alpha_{i-1}(s') \times \gamma_i(s', s) \times \beta_i(s) \tag{2.14}$$

where $\alpha_i(s)$ represents the portion that developed from the received information prior to the time of the state transition, $\beta_i(s)$ represents the portion that developed from the received information after the state transition and $\gamma_i(s', s)$ represents portion based on the received information at the time of state transition. We have

$$\alpha_{i-1}(s') = P\{S_{i-1} = s'; r_0^{i-1}\} \tag{2.15}$$

$$\beta_i(s) = P\{r_i^f \mid S_i = s\} \tag{2.16}$$

$$\gamma_i(s', s) = P\{S_i = s; r_i \mid S_{i-1} = s'\} \tag{2.17}$$

Next we prove Equation (2.14). Based on Markov property, we know that if the state at time i, S_i, is known, events after time i don't depend on r_0^i.

$$
\begin{aligned}
&\alpha_{i-1}(s') \times \gamma_i(s', s) \times \beta_i(s) \\
&= P\{S_{i-1} = s'; r_0^{i-1}\} \times P\{S_i = s; r_i \mid S_{i-1} = s'\} \times P\{r_i^f \mid S_i = s\} \\
&= P\{S_{i-1} = s'; r_0^{i-1}\} \times P\{S_i = s; r_i \mid S_{i-1} = s'; r_0^{i-1}\} \times P\{r_i^f \mid S_i = s\} \\
&= P\{S_{i-1} = s'; S_i = s; r_0^{i}\} \times P\{r_i^f \mid S_i = s\} \\
&= P\{S_{i-1} = s'; S_i = s; r_0^{i}\} \times P\{r_i^f \mid S_{i-1} = s'; S_i = s; r_0^{i}\} \\
&= P\{S_{i-1} = s'; S_i = s; r_0^{f}\} \\
&= \sigma_i(s', s)
\end{aligned}
$$

Now, to achieve the reliability value, we need to calculate $\alpha_i(s)$, $\beta_i(s)$ and $\gamma_i(s', s)$. $\alpha_i(s)$ and $\beta_i(s)$ can be calculated recursively.

$$
\begin{aligned}
\alpha_i(s) &= \sum_{s'} \alpha_{i-1}(s') \times \gamma_i(s', s) \\
\beta_{i-1}(s') &= \sum_{s} \beta_i(s) \times \gamma_i(s', s)
\end{aligned}
\tag{2.18}
$$

We prove Equations (2.18) here,

$$
\begin{aligned}
&\sum_{s'} \alpha_{i-1}(s') \times \gamma_i(s', s) \\
&= \sum_{s'} P\{S_{i-1} = s'; r_0^{i-1}\} \times P\{S_i = s; r_i \mid S_{i-1} = s'\} \\
&= \sum_{s'} P\{S_{i-1} = s'; r_0^{i-1}\} \times P\{S_i = s; r_i \mid S_{i-1} = s'; r_0^{i-1}\} \\
&= \sum_{s'} P\{S_{i-1} = s'; S_i = s; r_0^{k}\} \\
&= P\{S_i = s; r_0^{i}\} = \alpha_i(s) \\
&\sum_{s} \beta_i(s) \times \gamma_i(s', s) \\
&= \sum_{s} P\{r_i^f \mid S_i = s\} \times P\{S_i = s; r_i \mid S_{i-1} = s'\} \\
&= \sum_{s} P\{r_i^f \mid S_i = s; r_i; S_{i-1} = s'\} \times \frac{P\{S_i = s; r_i; S_{i-1} = s'\}}{P\{S_{i-1} = s'\}} \\
&= \sum_{s} \frac{P\{S_{i-1} = s'; S_i = s; r_{i-1}^f\}}{P\{S_{i-1} = s'\}} \\
&= \sum_{s} P\{S_i = s; r_{i-1}^f \mid S_{i-1} = s'\} \\
&= P\{r_{i-1}^f \mid S_{i-1} = s'\} = \beta_{i-1}(s')
\end{aligned}
$$

We add superscript to $\sigma_i(s', s)$ and $\gamma_i(s', s)$ to show the information bit at time i (0 or 1). Modified equations are shown below

$$\Lambda(u_i) = \ln \frac{\sum_{s'} \sum_{s} \sigma_i^{1}(s', s)}{\sum_{s'} \sum_{s} \sigma_i^{0}(s', s)} \tag{2.19}$$

$$\sigma_i^{a}(s', s) = \alpha_{i-1}(s') \times \gamma_i^{a}(s', s) \times \beta_i(s) \tag{2.20}$$

$$\alpha_i(s) = \sum_{a} \sum_{s'} \alpha_{i-1}(s') \times \gamma_i^{a}(s', s)$$

$$\beta_{i-1}(s') = \sum_{a} \sum_{s} \beta_i(s) \times \gamma_i^{a}(s', s)$$

The only unknown is $\gamma_i^a(s',s)$. We have

$$\begin{aligned}
&\gamma_i^{a}(s', s) \\
&= P\{S_i = s; r_i \mid S_{i-1} = s'\} \\
&= \frac{P\{S_i = s; r_i; S_{i-1} = s'\}}{P\{S_{i-1} = s'\}} \times \frac{P\{S_i = s; S_{i-1} = s'\}}{P\{S_i = s; S_{i-1} = s'\}} \\
&= \frac{P\{S_i = s; r_i; S_{i-1} = s'\}}{P\{S_i = s; S_{i-1} = s'\}} \times \frac{P\{S_i = s; S_{i-1} = s'\}}{P\{S_{i-1} = s'\}} \\
&= P\{r_i \mid S_i = s; S_{i-1} = s'\} \times P\{S_i = s \mid S_{i-1} = s'\} \\
&= P_x \times P_y
\end{aligned} \tag{2.21}$$

Here P_y is a constant since if the $S_{i-1} = s'$ is known, the probability of $S_i = s$ has been decided. Then we only need to obtain P_x. The r_i is made up of x_i and y_i , where x_i represents the i_{th} information bit and y_i represents the i_{th} parity bit. Assume that signals go through an AWGN channel with noise variance $N_0/2$ and BPSK modulation is implemented. Thus we actually transmit 1 for u_i=1and –1 for u_i=0. Thus

$$\begin{aligned}
x_i &= (2u_i - 1) + noise \\
y_i &= (2p_i - 1) + noise
\end{aligned} \tag{2.22}$$

And,

$$
\begin{aligned}
P_x &= P\{r_i \mid S_i = s; S_{i-1} = s'\} \\
&= P\{x_i \mid S_i = s; S_{i-1} = s'\} \times P\{y_i \mid S_i = s; S_{i-1} = s'\} \\
&= \exp[-\frac{(x_i - u_i)^2}{N_0}] \times \exp[-\frac{(y_i - p_i)^2}{N_0}]
\end{aligned} \tag{2.23}
$$

Thus, $\gamma_i^a(s', s) = const \times \exp[-\frac{(x_i - u_i)^2}{N_0}] \times \exp[-\frac{(y_i - p_i)^2}{N_0}]$ (2.24)

Up to now, $\alpha_i(s)$, $\beta_i(s)$ and $\gamma_i(s', s)$ at any time unit i can be achieved. We can get the reliability value based on them,

$$
\Lambda(u_i) = Ln \frac{\sum_s \sum_{s'} \gamma_1(R_i, s', s) \cdot \beta_i(s) \cdot \alpha_{i-1}(s')}{\sum_s \sum_{s'} \gamma_0(R_i, s', s) \cdot \beta_i(s) \cdot \alpha_{i-1}(s')} \tag{2.25}
$$

We give an example here to show the steps of calculations. Assume we have a memory size 2 recursive convolutional encoder as shown in Fig 2.5.

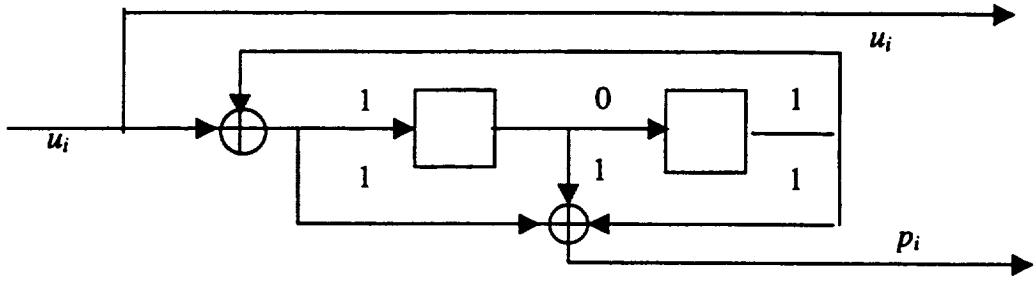

Fig 2.5 (5,7) RSCC

The input - output state diagram of this (5, 7) RSCC is shown in Fig 2.6,

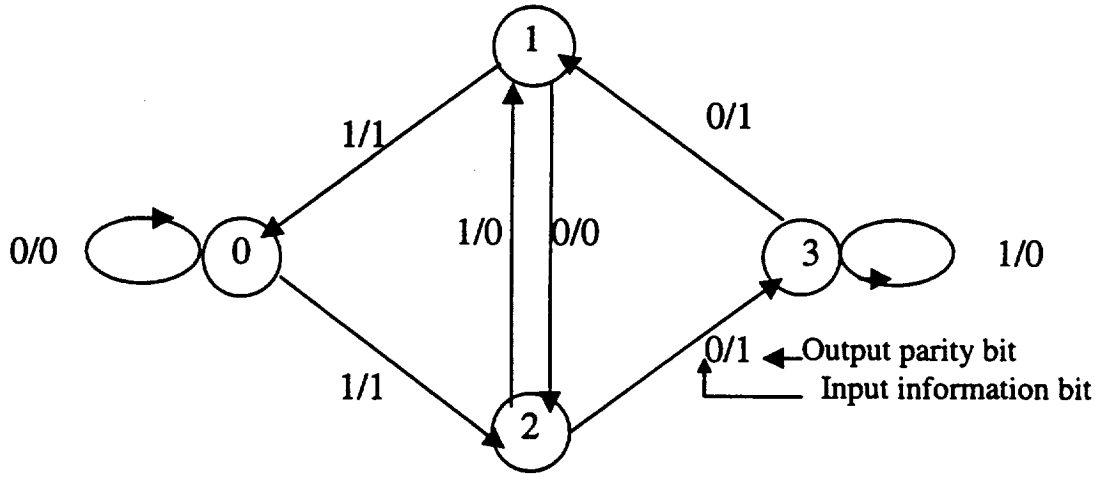

Fig 2.6 The state diagram of the (5 , 7) RSCC

To implement the MAP algorithm, trellis diagram is more important. Trellis diagram of the encoder is shown in Fig 2.7

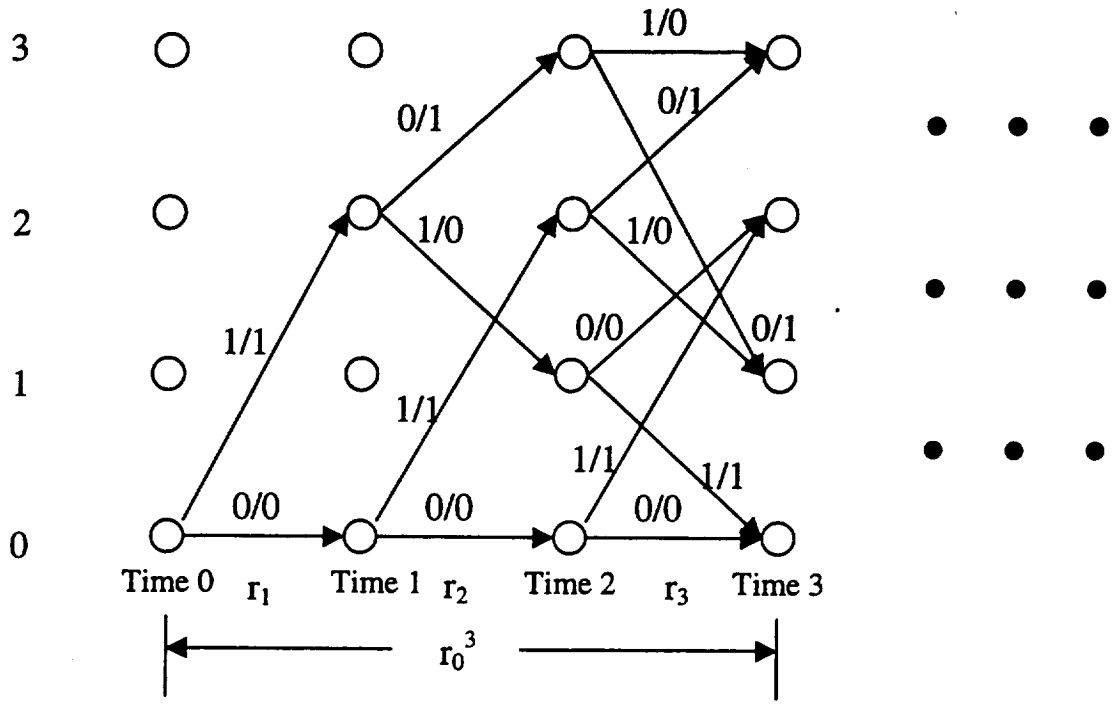

Fig 2.7 Trellis diagram of (5, 7) RSCC

Assume that ten random bits are generated by the encoder. The information sequence is *[0 0 1 0 0 0 1 0 0 0]*. Then the 10 parity bits generated by the encoder are *[0 0 1 1 1 0 0 0 1 1]*. After going through the AWGN channel with noise variance 1.6 and puncturing half of the parity bits to achieve rate 1/2 code (bits deleted by puncturing are inserted as zeros), the received sequences are as follows.

Information bits x_i :

[-1.04 -1.14 1.73 -1.48 -0.02 -1.49 -0.53 -1.71 -1.94 -2.73]

Parity bits y_i :

[-0.70 0 -0.23 0 1.78 0 -0.59 0 1.53 0]

The five main steps in the decoding procedure are as follows:

1. Calculate all $\gamma_i(s',s)$
2. Calculate $\alpha_i(s)$ for all states and times from $\alpha_0(s)$ to $\alpha_f(s)$
3. Calculate $\beta_i(s)$ for all states and times from $\beta_f(s)$ to $\beta_0(s)$
4. Calculate all $\sigma_i^a(s',s)$ in one time unit
5. Calculate $\Lambda(u_i)$

Step 1: Calculation of $\gamma_i(s', s)$

Assume the constant in the equation [2.24] is 1. We show the example for calculating $\gamma_1^0(0,0)$ and $\gamma_1^1(0,2)$ here. We have x_1= -1.04 , y_1= -0.70. *For* $\gamma_1^0(0,0)$, u_1= -1 , p_1= -1 (Transition noted with 0/0), so

$$\gamma_1^0(0,0) = \exp[-\frac{(-1.04-(-1))^2}{N_0}]\times\exp[-\frac{(-0.70-(-1))^2}{N_0}] = 0.98$$

For $\gamma_1^1(0,2)$, u_1= 1, p_1= 1 (Transition noted with 1/1), we have

$$\gamma_1^1(0,2) = \exp[-\frac{(-1.04-1)^2}{N_0}]\times\exp[-\frac{(-0.70-1)^2}{N_0}] = 0.02$$

The rest of $\gamma_i(s', s)$ can be calculated in the same way.

Step 2: Calculation of $\alpha_i(s)$

We assume that the encoder started at $\alpha_0(0) = 1$ and $\alpha_0(s) = 0$ for all $s \neq 0$. All $\alpha_i(s)$ can be achieved by recursive calculation based on the previous $\alpha_i(s)$ and $\gamma_i(s', s)$. We show the example for calculating $\alpha_1(0)$ and $\alpha_1(2)$ here. Since there is no transition available from state 0 to state 1 and state 3 at time 1 yet, so $\alpha_1(1)=0$ and $\alpha_1(3)=0$. And

$\alpha_1(0)=\alpha_0(0)\times\gamma_1^0(0,0)=1x0.98=0.98$

$\alpha_1(2)=\alpha_0(0)\times\gamma_1^1(0,2)=1x0.02=0.02$

Similarly, all $\alpha_i(s)$ can be computed and are as follows:

i=	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
State 3:	*0*	*0*	*.02*	*.03*	*.80*	*.12*	*.03*	*.33*	*.18*	*.63*	*.02*
State 2:	*0*	*.02*	*.07*	*.83*	*.11*	*.03*	*.77*	*.18*	*.36*	*.02*	*.33*
State 1:	*0*	*0*	*.00*	*.11*	*.06*	*.79*	*.12*	*.36*	*.32*	*.33*	*.63*
State 0:	*1*	*.98*	*.91*	*.03*	*.03*	*.06*	*.08*	*.13*	*.14*	*.02*	*.02*

Step 3: Calculation of $\beta_i(s)$

The calculation of $\beta_i(s)$ is implemented as backward recursion. The final state of the encoder is not known. However, there are two methods that can be used for initialization of $\beta_{10}(s)$. $\beta_{10}(s)$ is either initialized as $\alpha_{10}(s)$ or as equal weighting as $1/2^M$.

We use the first method for the initialization and show the example to calculate $\beta_9(0)$ here.

We already know that $\beta_{10}(0)= \alpha_{10}(0)=0.02$, $\beta_{10}(2)= \alpha_{10}(2)=0.33$, $\gamma_{10}^{0}(0,0)=0.18$, $\gamma_{10}^{1}(0,2)=0.0055$, then

$$\beta_9(0)= \beta_{10}(0)\,\gamma_{10}^{0}(0,0)+ \beta_{10}(2)\,\gamma_{10}^{1}(0,2) = 0.038$$

Here we should take care of one more thing. In previous calculation for $\gamma_i(s', s)$, no normalization for the probability distribution has been done. So, we must do a normalization to make the sum of the $\beta_i(s)$ at any time unit i to be 1.

After normalization, all $\beta_i(s)$ are listed below:

$i=$	0	1	2	3	4	5	6	7	8	9	10
State 3:	.23	.17	.10	.07	.26	.44	.23	.01	.34	.63	.02
State 2:	.16	.10	.58	.26	.44	.24	.27	.33	.66	.03	.33
State 1:	.20	.51	.18	.43	.07	.26	.44	.64	.00	.33	.63
State 0:	.41	.22	.19	.24	.23	.06	.06	.02	.00	.19	.02

Step 4: Calculation of $\sigma_i(s', s)$

After all the $\alpha_i(s)$, $\beta_i(s)$ and $\gamma_i(s', s)$ have been obtained, $\sigma_i(s', s)$ can be calculated. Example of calculating $\sigma_1^{1}(0,2), \sigma_1^{0}(0,0)$ is shown here,

$$\sigma_1^{0}(0,0) = \alpha_0(0) \times \gamma_1^{0}(0,0) \times \beta_1(0)$$

$$\sigma_1^{1}(0,2) = \alpha_0(0) \times \gamma_1^{1}(0,2) \times \beta_1(2)$$

Step 5: Obtain $\Lambda(u_i)$.

Then the reliability value of the first information bit is

$$\Lambda(u_1) = \ln \frac{\sum_{s'}\sum_{s} \sigma_1^{1}(s', s)}{\sum_{s'}\sum_{s} \sigma_1^{0}(s', s)} = \frac{\sigma_1^{1}(0,2)}{\sigma_1^{0}(0,0)} = -4.67$$

By using the same method, the reliability values achieved for ten information bits are *[-4.67 -2.2 5.13 -6.15 3.77 -6.15 1.66 -6.0 -7.1 -7.8]*. From these reliability values, we get the complete decoded sequence as

[0 0 1 0 0 0 1 0 0 0]

which is identical to the original information sequence.

Though MAP algorithm is the optimal decoding algorithm, it has some obvious disadvantages. Very large amount of memory is needed for decoding since before $\beta_i(s)$ can be calculated, all the $\alpha_i(s)$ at any state and any time must be stored. The calculation complexity is very high since large amount of multiplications and additions must be implemented.

We have mentioned that, other than MAP algorithm, SOVA and log-MAP algorithms can also be implemented. SOVA compares metric values at each node of trellis to decide which path is the maximum likelihood path, similar to standard Viterbi algorithms. However, for each node, SOVA also compares the maximum likelihood path with the second best path to update a reliability value. This method requires only comparisons of metrics and table lookups and only needs one pass through the information, while MAP algorithm requires both forward $(\alpha_i(s))$ and backward $(\beta_i(s))$ passes. So it is less time consuming than MAP algorithm. Log – MAP algorithm is a simplification of the MAP algorithm. It takes the log of the probability distribution of the transition $\gamma_i(s', s)$ and replaces them by approximations. Log – MAP algorithm is a better approximation than SOVA and there is only a little degradation in performance of log-MAP compared to MAP algorithm.

2.2 Performance of turbo codes

For a bit error rate lower than 10^{-5}, the uncoded binary modulation (BPSK) requires the E_b/N_0 to be larger than 9.6 dB. Form our simulation results, we found that for rate 1/3 turbo code , to reach a bit error rate of 10^{-5}, the E_b/N_0 for turbo codes can be reduced to 0.1dB. The performance improvement of turbo codes can be as large as 9.5 dB.

The soft-decision decoding performance bounds at different code rates are given in Proakis' book, Digital Communications[9, Fig. 5.2.14]. This plot shows the smallest E_b/N_o values to achieve the BER of 10^{-5} with BPSK modulation. At the rate equal to zero, which means infinite parity bits are added in the transmission together with the information bit, the bound is approximately –1.6 dB, which is equal to the Shannon limit.

At the rate equal to 1, which means no parity bits are transmitted and is equivalent to the uncoded transmission, the bound is given as 9.6 dB and matches the performance for uncoded transmission. Fig 2.8 shows the bounds.

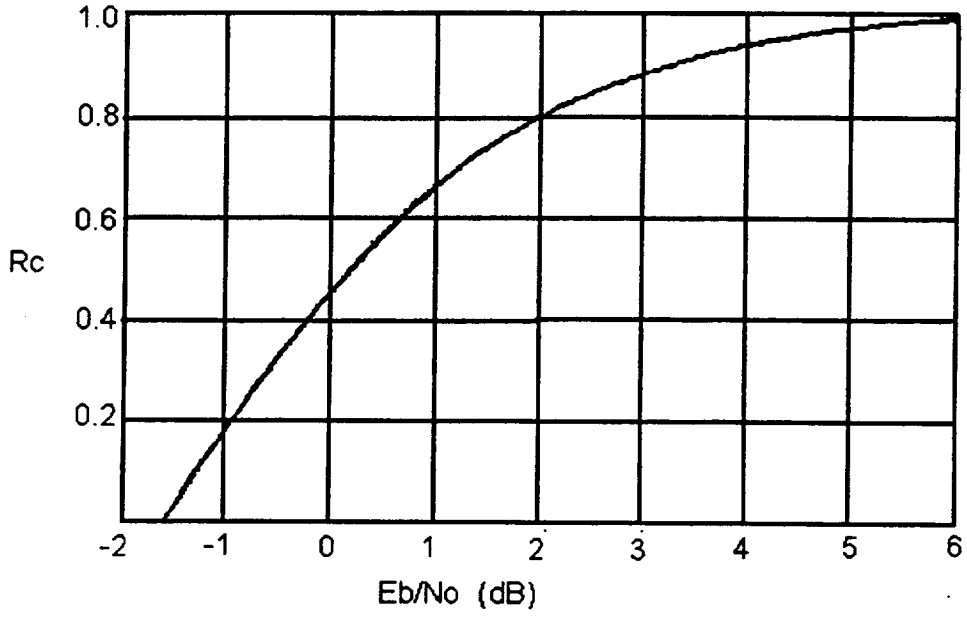

Fig 2.8 Soft-decoding bounds at different code rates

In our research, performance of high rate turbo codes and the bounds are compared at BER of 10^{-5}. Our simulations are done for rates 1/2, 2/3, 3/4, 4/5, 5/6, 10/11, 15/16, and 16/ 17 with the best selection of parameters. A two – encoder parallel - concatenation system with memory size 4 is implemented. We select the generator polynomial as 23_31 since it is the best choice and implement a peusdo-random interleaver with size 256×256. The selection of puncturing patterns is according to the recent paper "High Rate Turbo Codes for BPSK/ QPSK Channels"[3] and our research. In Table 2.1, puncturing patterns selected for different code rates are listed. For rates 5/6, 10/11, 15/16, modified puncturing patterns are applied to achieve the best performance (which we will discuss later). MAP algorithm is applied in decoding since it is the optimal soft decoding algorithm. The number of iterations in the decoding process is set

to be 18. In each set of simulations, the total number of information bits is 10^7. That is, the BER value (bit error rate) we can test is down to 10^{-6} level.

Fig 2.9 shows the performance after 18 iterations for different code rates. The Eb/No values where BER of 10^{-5} can be achieved by different rate codes are also listed in Table 2.1, and they are compared with the Shannon limit. From the results, we see that the performance of turbo codes at all these rates is within 0.5 dB from the Shannon Limit.

Table 2.1 Performance of high rate turbo codes

Rate	1 / 2	2 / 3	3 / 4	4 / 5	5/6	10/11	15/16	16 / 17
Puncturing patterns	P(1,2)	P(3,4)	P(3,5)	P(7,6)	modified	Modified	Modified	P(2,2)
E_b/N_0(dB)	0.75	1.55	2.1	2.5	2.8	3.7	4.2	4.3
Distance(dB)	0.45	0.5	0.5	0.5	0.5	0.45	0.4	0.4

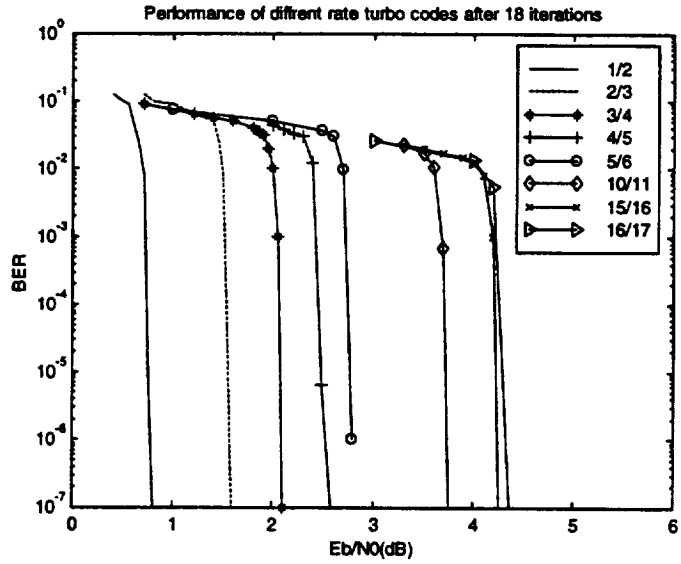

Fig 2.9 The performance of turbo codes at different code rates

Fig 2.10 shows the performance of turbo codes compared to some block and convolutional coding schemes and the performance bounds.

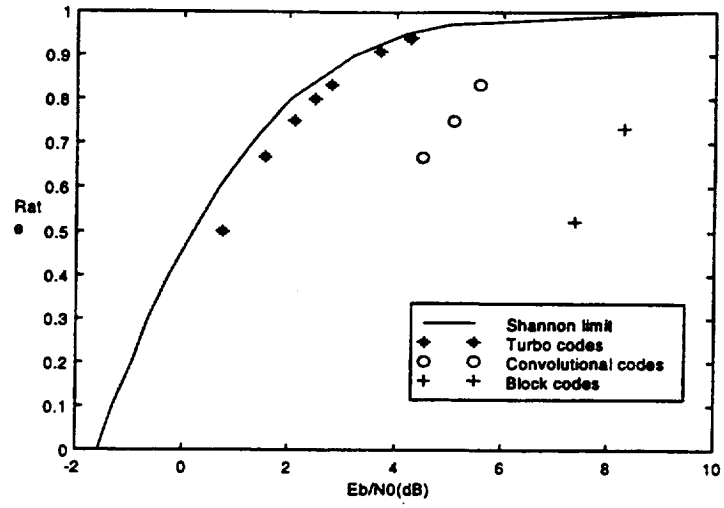

Fig 2.10 The performance of turbo codes compared to some convolutional and block turbo codes, also the Shannon limit

In our simulations, we studied the effects of system components and parameters on the code performance. All possible components or parameters which affect the performance are listed in Table 2.2 . In next section, we will discuss the dominant factor for turbo code performance.

Table 2.2 Factors for the performance of turbo codes

Encoding System	Structure	Parallel concatenation/ Serial concatenation Levels of concatenation
	Convolutional encoder	Memory size (constraint length) Systematic / nonsystematic Recursive / non-recursive Generation polynomial
	Interleaving	Nonrandom / random Algorithm used for random interleaving Size
	Puncturing	Code rate Puncturing pattern
Decoding system	Algorithm	Soft / hard
	iterations	

2.3 Output weight distribution and performance bounds of turbo codes

2.3.1 Output weight distribution

The output weight distribution is the new concern of turbo code researchers. The relation between the turbo code performance and the output weight distribution has been studied extensively. At first, the performance of turbo codes was claimed to be mainly decided by the lowest weight code word (which equals to the free distance of the code) together with the effective multiplicity of these free distance code words [18]. Then new thoughts came out that the whole output weight spectrum should be considered to estimate the code performance [25].

Linear recursive systematic codes are used as the component codes of turbo codes. The minimum output weight of the codes is equal to the free distance of the code. For punctured high rate turbo codes, the minimum output weight decreases, but it should be proportional to the free distance of the codewords that are generated by the component RSCC. Since better error detection and correction capability can be expected when the free distance of the codewords is larger, minimum output weight can be seen as the dominant factor of the code performance.

Low weight output sequence is always generated by the low weight input sequence. The output sequence is made up of the input information sequence and the generated parity sequence. So when the information sequence itself has high weight, the output sequence will definitely have high weight too. Also, high weight information sequence has very little chance to generate low weight parity sequence.

We know that interleaver in the encoding system makes the possibility of all encoders to generate low weight output simultaneously to be very low. We can prove that higher the weight of the input sequence, the lower the possibility will be. For the case of using perfect random interleaver with size $L = A \times A$, we assume a weight-w information sequence. The information sequence has a nonzero-bit distribution which causes the low weight output from the first RSCC. The probability for the interleaved information sequence, as the input to the second RSCC, to also have the nonzero-bit distribution to cause low weight output can be approximately represented by

$$P = \frac{2 \times (w-1)!}{L^{w-1}} \tag{2.26}$$

This probability is achieved approximately when we assume that the interleaver size is large enough so that the block edge effects are negligible.

For example, assume an input sequence *[1 0 0 1 0 0 0 0....]* with weight 2, which can cause the low weight output in the first RSCC. After interleaving, the probability for the interleaved sequence to also have 2 zeros between its two nonzero bits will be roughly *2/L*. Now we explain how this *2/L* is calculated. Interleaver is used to change the permutation of the bits in the information sequence. For the weight 2 information sequence, after the location for the first nonzero bit has been decided, there are *L-1* locations left where the second nonzero bit can stay. Among these *L-1* locations, two locations, which are 3 bits ahead of the first nonzero bit and 3 bits after the nonzero bit, cause low weight output. So the probability of low weight output is *2/(L -1)*. It approximately equals to *2/L* when *L* is a large value. Similarly, if assume a weight 3 input sequence, the probability for the interleaved sequence to cause low weight output is approximately $4/L^2$.

From equation (2.26), we can see that the probability of a weight *w+1* input sequence to cause low weight output is only about *1/L* of the weight w sequence. Thus we can draw the conclusion that the free distance code word is most possibly to be generated by the minimum weight information sequence. Some researchers assume that the weight 2 information sequence is the dominator of the performance of turbo codes because weight 2 is the smallest weight to cause low weight output (weight 1 input sequence will never cause a low weight output). We have suspicion on this assumption. In a practical consideration, for an information sequence with length *L*, it is appropriate to assume each bit in the sequence to have half probability to be 0 and half probability to be 1. So the probability for the weight 2 input sequence to happen, especially when *L* is a large number, should be extremely small. We feel the proper assumption is just that the minimum weight input sequence generates the minimum output weight sequence.

Also with our above assumption for the practical consideration, we can expect the weight distribution of the information sequence to show an approximate Gaussian distribution. And we can expect the output weight distribution spectrum to also have a similar shape. That means large percentage of the information sequences have about middle weight and only a small proportion of information sequences are the low weight

or high weight sequences. So although the low weight information sequences will have comparatively much larger influence on the code performance, their multiplicity is much lower than the middle-weight Information sequence. So, for an accurate evaluation of the code performance, considering only the influence of low weight outputs is not sufficient. That is why the output weight distribution spectrum should be taken into consideration for a better estimation of the code performance.

To achieve an improvement in the turbo code performance, we want the minimum output weight to be higher and the multiplicity of the low weight codewords to be smaller. This aim can only be achieved when the variance of the distribution spectrum is decreased. It follows that when we try to improve the turbo code performance, all we need to do is try to decrease the variance of the output weight distribution.

2.3.2 Performance bounds

Since turbo codes are generated by the parallel concatenation of two or more recursive systematic convolutional encoders, we can achieve the performance bounds of turbo codes from the analysis of the systematic convolutional codes bounds. For a block of information bits with length equal to L, we know there are totally 2^L possibilities of the code words. By the theory that the sum of the probabilities of individual events is no less than the probability of the union of the events, we can state that P_b, the error probability of the convolutional codes, should be no greater than the sum of the error probabilities of each of the 2^L possible code words.

$$P_b \leq \sum_{i=1}^{2^L} P_c \,, \tag{2.27}$$

where P_c represents the error probability of each of the possible code words.

Assume that the signal energy per information bit is E_b, then the received signal energy per code word (information + parity) bit is RE_b. R here represents the code rate. If BPSK modulation is used, it follows that '+1' and '–1' are transmitted. Also assume an additive white Gaussian noise (AWGN) channel. We have the Gaussian noise added with mean at $\pm\sqrt{RE_b}$ and variance equal to $N_0/2$. It is well known that the error probability of each code word is given by

$$P_c = \frac{\omega_l}{L} Q\left(\sqrt{d \frac{E_s}{E_N}}\right) = \frac{\omega_l}{L} Q\left(\sqrt{d \frac{2RE_b}{N_0}}\right) \tag{2.28}$$

In which ,

P_c: error propability of each one of the 2^L possible code words

E_S: signal energy per code word bit

E_N: average energy of Gaussian noise

ω_l weight of the information bits of a certain code word

d: Hamming weight of the codewords

E_b/N_0: signal to noise ratio

R: code rate, the ratio of the number of information bits to the codeword length

L: The size of the block of information bits, or information sequence length.

$Q(x)$: an Gaussian cumulative distribution function. Q-function is defined as the integral of zero mean, unit variance Gaussian density function from certain point x to infinite. Here, Q-function shows the probability of error happening when the total Hamming weight of a code word is d.

Combining equations (2.27) and (2.28), BER performance of a finite length convolutional code with maximum-likelihood decoding (MLD) on an AWGN channel can be upper bounded by using the union bound

$$P_b \le \sum_{i=1}^{2^L} \frac{\omega_l}{L} Q\left(\sqrt{d \frac{2RE_b}{N_0}}\right). \tag{2.29}$$

To make the calculation of the bounds more convenient, we make a small modification to collect the codewords of the same d.

$$P_b \le \sum_{d=dfree}^{d_{max}} \frac{N_d \tilde{\omega}_d}{L} Q\left(\sqrt{d \frac{2RE_b}{N_0}}\right) \tag{2.30}$$

Here, we have

d_{free}: The minimum Hamming weight of all possible codewords, free distance

d_{max}: The maximum of the Hamming weights of all codewords which is equal to L/R.

$\tilde{\omega}_d$: The average weight of the information bits when the Hamming weight of the codeword is d.

N_d: The multiplicity of code words with Hamming weight d.

An effective multiplicity of code words with weight d can be defined as N_d/L.

This procedure of deriving the upper union bound of the performance of convolutional codes can also be used to derive the bound of turbo codes. We know the lowest weight output, which decides the free distance of the code, can be regarded as the dominant factor for the code performance, so further simplification can be applied. We get a performance bound of turbo codes based on the free distance of the code and the multiplicity of all the free distance code words.

$$P_b \leq \frac{N_{d_{free}} \tilde{\omega}_{d_{free}}}{L} Q\left(\sqrt{d_{free} \frac{2RE_b}{N_0}}\right) \tag{2.31}$$

We have mentioned that some researchers assume that the free distance code words are formed by the weight 2 input sequence. In this case the bound is simplified as

$$P_b \leq \frac{N_2 \cdot 2}{L} Q\left(\sqrt{d_{free} \frac{2RE_b}{N_0}}\right) \tag{2.32}$$

in which N_2 represents the multiplicity of free distance code words caused by weight 2 information bits. But from our analysis above, we prefer to say that the free distance codewords are generated from the lowest weight input sequence, but not necessarily to be 2.

From our knowledge of the Gaussian density distribution function, equation (2.32) implies that smaller Hamming weight d causes larger value of the Q function and in turn, larger error probability. That is, smaller the free distance of the codes, the worse the performance.

2.4 Relation between the system parameters and output weight distribution

There are tight relationships between the output weight distribution and the generator polynomials, interleaver and puncturing patterns. Based on these relations, we can find the criterion to select the best parameters that can help to decrease the variance

of the output weight distribution spectrum, and thus to improve the performance of the turbo codes. In our work, extensive simulations have been done to search for the best parameters and help to prove our selection criterion. In these simulations, we implemented a two encoders parallel concatenation system with memory size 4. MAP decoding algorithm is applied with 18 iterations. In each group of simulations, the total number of information bits simulated is 10^7.

2.4.1 Generator polynomial

Our first consideration is the component code generated by the convolutional encoder. Based on our formal analysis, the low weight code words are mainly formed by the low weight information sequence. So our main concern is on the low weight information sequence.

The implementation of RSCC (Recursive Systematic Convolutional Encoder) is important. NSCC (non-recursive systematic convolutional encoder) maps a finite weight input sequence into a finite weight output sequence. The output weight of the NSCC is correlated with its input weight and can not satisfy the requirement of random-like codes. The improvement of RSCC is obtained because a finite weight input sequence can be mapped into an infinite weight output sequence. The output weight of RSCC has the same distribution as that of a random code sequence. RSCC gives the greatest gain when used as parallel concatenated codes.

If NSCC is used, the output weight of the low weight information sequence will always be low. RSCC provides significant improvement in the output weight of parity sequence. The generation of most of the low- weight codewords is avoided by the use of RSCC because of the contribution of the feedback structure of the encoder. This structure makes the previously encoded information bits feed back continuously to the encoder's input. However, for small number of low weight information sequences with certain nonzero bits distribution, low weight output will still be possibly formed even by RSCC. An example to illustrate this is given below.

We use a weight-2 information sequence for the example. Assume that we have a RSCC with memory size $M = 2$ as shown in Fig 2.11 The feed-forward and feedback polynomials of the RSCC are $1+ D^2$ and $1+D +D^2$ respectively. The weight 2 information

sequence is assumed to be *[1 0 0 1 0 0 0 0 0 ...]* and can be described as $1+D^3$. The low weight output parity sequence *[1 1 1 1 0 0 0 0....]* is formed by our encoder. When the first nonzero bit of the information sequence comes, the trellis path of the convolutional codes diverges from the all zero state. Later, when the second nonzero bit inputs to the encoder, it happens to drive the encoder back to the all zero state. After that, all the remaining bits in the information sequence are zeros. None of them can lead the encoder away from the all zero state again. So, other than the first four 1's, all other parity bits are zeros. The weight of the parity bits thus is only 4. As a comparison, we assume another weight 2 sequence, such as *[1 0 0 0 1 0 0 0 0 0....]*, or described as $1+D^4$. This time the second nonzero bit in the information sequence does not drive the encoder back to the all zero state, thus the subsequent zero input and the feedback of the encoder force the encoder to go through a loop of several different states. The parity bits formed by this sequence will be *[1 1 1 0 1 0 1 1 0 1 1 0 1...]*, a high weight output sequence.

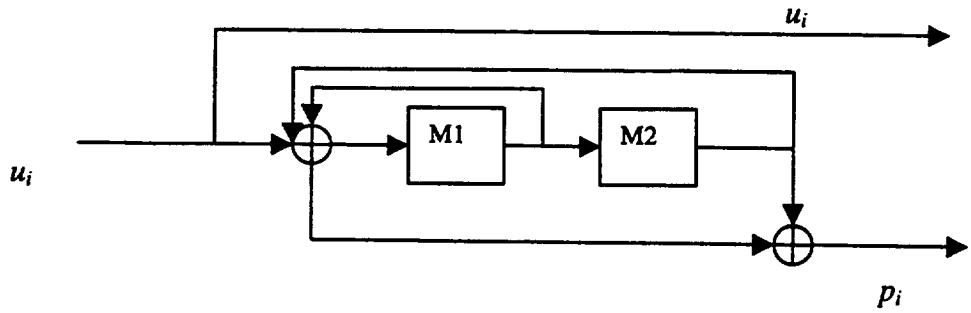

Fig 2.11 An example of turbo encoder with $M=2$ and feedback polynomial $1+D+D^2$

These two examples show that weight-2 information sequences can possibly generate low weight output or high weight output sequences. The difference between two input sequences is the distribution of the nonzero information bits. For the encoder used in the example and any weight-2 input sequences, there are several possibilities of the nonzero bits distribution that can cause low weight output. The first case is the sequence which can be described as $1+D^{3z}$, where z is a small integer larger than 1. All these $1+D^{3z}$ input sequences can be divided by the feed back polynomial $1+D+D^2$, so the second nonzero bit of the information sequence drives the encoder back to the all zero state as was the case for $1+D^3$ sequence. Since when z increases from 1, the weight of the parity bits becomes a little higher than the $z = 1$ case, we should note that z can only be a very

small integer. Otherwise, even though the second nonzero bit finally drives the encoder back to all zero state, the output weight of the parity bits has been large enough before that. Some delayed version of the $1+D^{3z}$ sequences can also give the low weight output. These group of sequences can be described as $D^{z'}(1+D^{3z})$ where z' is also a small integer greater than 1. For example, a delayed version of our input sequence in the first example *[1 0 0 1 0 0 0 0 ...]* is *[0 0 0 1 0 0 1 0 0 0 0....]*, which will also cause low weight parity bits. Other than the first and second cases, low weight output can be generated when the first nonzero information bit appears at the very end of the input sequence. In this case, although the second nonzero bit is not $3z$ bits away from the first nonzero bit, low weight output will be generated. For all weight 2 information sequences, other than these three cases, the output sequence will actually have infinite output weight if no termination is executed at the end to make the parity sequence have the same length as the information sequence. Even with the termination, the weight of the output sequence will still be high.

Input sequence can have an even lower weight than 2, the weight 1 case. Though weight 1 information sequence will definitely cause very low output weight for an NSCC, it will not be the case for RSCC. The code word generated by the weight 1 information sequence will be of infinite length without termination. This is due to the fact that after the only nonzero information bit causes the trellis path to diverge from the all zero state, there will never be another nonzero bit in the information sequence to remerge the path back to the all zero state. Thus for weight 1 input, the only possibility to form low weight output is that the 1 appears at the very end of the sequence. Now let us see what happens to a low weight information sequence that has a weight larger than 2, but still a small value (low weight is the main concern for the performance). Similar to weight 2 case, some of the nonzero bit distributions cause the low weight output while others cause infinite output weight when no termination is done. If a low weight information sequence is made up of several weight 2 sequences that cause low weight output, the information sequence will cause low weight output too. For the encoder in Fig 2.11, we found that some sequences which can be described as $\sum_{z,z'} D^{z'}(1+D^{3z})$, cause low weight output when z is small and there are not too many components to be summed up. For example, low weight parity sequence is generated from weight 4 information sequence *[1 0 0 1 0 ... 0 1 0 0 1 0 0 ....]*.

Encoders with primitive feedback polynomials are the best choices because they help to achieve large free distance of the codes. Assume we have the generator matrix of the RSCC encoder as follows

$$G_R(D) = \left[1 \quad \frac{g_1(D)}{g_0(D)}\right] \tag{2.2}$$

where $g_1(D)$ and $g_0(D)$ are referred to as the feed-forward and feedback polynomials respectively. Again, weight 2 information sequence will be considered here to simplify our analysis. What we want to maximize is the weight of the parity bits, that is

$$p(D) = d(D)\frac{g_1(D)}{g_0(D)} \tag{2.33}$$

Here we name $u(D)$ as the information sequence and $p(D)$ as the parity sequence. For the weight 2 input case, we assume an $u(D) = 1 + D^e$. The e is a finite value selected to be the smallest to make this information sequence to generate low weight output (free distance code word). Then $p(D)$ can be written as

$$p(D) = (1+D^e)\frac{g_1(D)}{g_0(D)} = \frac{g_1(D)}{g_0(D)} + D^e\frac{g_1(D)}{g_0(D)} \tag{2.34}$$

Since the nonzero part of $p(D)$ is also of finite length e, $g_1(D) / g_0(D)$ must be periodic with period e. On average, half of the bits in the e long nonzero subsequence of $p(D)$ will be 1 and counted for output weight. Approximately, we can predict larger value e will mean higher weight for the parity bits. This period e, for a strictly proper rational function of two polynomials such as $g_1(D) / g_0(D)$, is a value no larger than $2^M - 1$, where M is the number of memories used in the encoder. e reaches the maximum when the feedback polynomial $g_0(D)$ is primitive. On average, primitive polynomial results in larger free distance in turbo codes.

In our simulations, we set the memory size of the recursive encoder to be 4. Thus we can expect the maximum period e to reach 15 when the feedback polynomial is primitive. For the case $M=4$, there exist two primitive polynomials, $1 + D + D^4$ and $1 + D^3 + D^4$. Written in the octal number, they are 23 and 31. For the feed-forward polynomial, the criterion to select the best has not been found yet. However, since there are only 2 possibilities of the feedback polynomial and only 8 choices (21,23, 25, 27, 31, 33, 35, 37) for feed-forward polynomial, all the possible combinations are 16. It is not too

large a number, so it's possible to find the best combination from simulations. Our simulations show that, among all the combinations, (23,31) generator polynomial gave the best performance. We take this combination to be the optimal choice of the generator polynomial when the memory size is 4. However, our simulations showed that the performance of the turbo codes is not significantly different with different generator polynomials. Fig 2.12 shows the encoder with the (23, 31) generator polynomial. Fig 2.13 gives the results for one group of comparisons between the (23, 31) and (31, 27) codes.

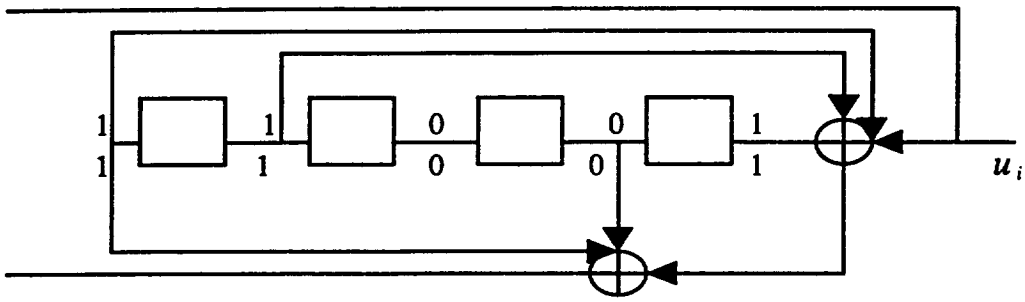

Fig 2.12 Recursive encoder with generator polynomial (23 , 31)

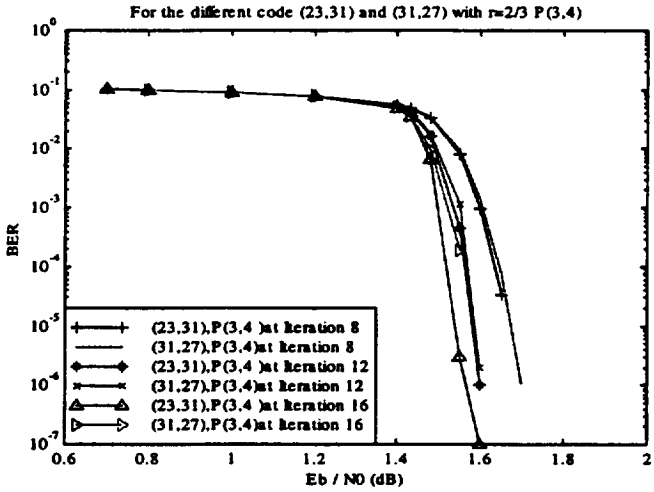

Fig 2.13 Comparison of the (23, 31) and (31, 27) generator polynomials

2.4.2 Interleaver

Interleaver is regarded as the most important component in turbo encoding system. We examine the influence of the interleaver size and the interleaver algorithm on the output weight distribution.

In a two encoders parallel concatenation system, the interleaver is used in turbo coding system to change the distribution of the information sequence before it inputs to the second encoder. So the input sequences to the two encoders in the system are actually different. If the original input sequence has a very low weight, and its nonzero bit distribution happens to cause low weight output parity bits in the first encoder, it's very unlikely that the input sequence to the second encoder, after the interleaving, will still have a nonzero bits distribution which will cause low weight output.

We want the interleaver to make the probability of low weight output, simultaneously from both encoders, to be very small. This probability depends on the algorithm used for the interleaver. Random interleaver is preferred over nonrandom interleaver since the ability of the random interleaver to break the correlation between the bits of the information sequence is much better than the ability of the nonrandom interleaver. It can make the output weight distribution to have a shape similar to the random codes, which makes the performance get very close to the Shannon limit. Also we want the interleaver size (equivalent to the length of the information sequence) to be as large as possible. Larger interleaver size is required by the performance bounds equations to provide low probability of coding error. And it's easy to determine that the probability of the simultaneous low weight output from both encoders is inversely proportional to the interleaver size. The probability can be decreased significantly when the interleaver size is increased.

The interleaving algorithm and the interleaver size are the two considerations in the selection of the interleaver. We have mentioned that for the first concern, random interleaver is preferred than non-random interleaver. Pseudo – random interleaver is selected in our simulation work since it is most commonly used random interleaver. Our main concern is how significant is the influence of the interleaver size on the performance, and what will be the appropriate interleaver size to be used in practical applications. For the appropriate size, two factors should be considered. First, we know the bit error rate (BER) decreases with the increase in the size of the interleaver. This effect is called interleaver gain and demonstrates the necessity of larger interleaver. On the other hand, increasing the interleaver size causes an obvious slow down of the speed of the turbo codes, especially the speed of turbo decoding. This is because a certain

number of iterations are needed in the decoding process to improve the performance. And in each iteration, the interleaving and deinterleaving (inverse function of interleaving) processes must be executed several times. Thus a tradeoff is needed between the better performance of the turbo codes and the real time decoding.

In order to observe the code performance with different interleaver sizes, we set the generator polynomial to be (23, 31) and select the 4/5 code rate to do the simulations. Puncturing pattern is selected to be P (7, 6), which is claimed by [3] to be an optimum choice. The interleaver sizes 256×256, 128×128, 64×64, 32×32, 16×16, are compared. To maintain the number of bits being tested in each case (about 10^7), the numbers of blocks selected for each simulation are 150, 600, 2400, 9600, 38400. The simulation results are shown in Fig 2.14 and Table 2.3.

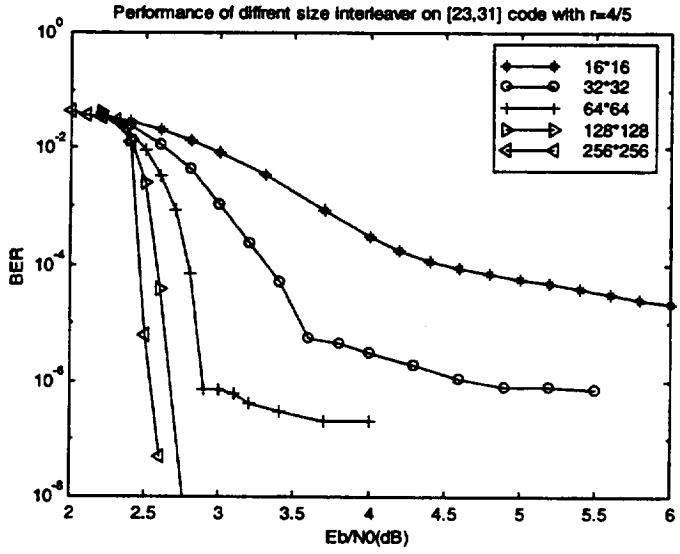

Fig 2.14 The influence of interleaver size on the performance of turbo codes

Table 2.3 Performance of 4/5 turbo codes with different size interleavers

	256*256	128*128	64*64	32*32	16*16
Eb/No(10^{-5})(dB)	2.5	2.6	2.8	3.4	>6.0
Distance to bound	0.5	0.6	0.8	1.4	>4.0
Coding gain	7.1	7.0	6.8	6.2	<3.6

There are several observations from the simulation results. First, with the decrease of interleaver size, the performance of the 4/5 turbo codes decreases quickly. Table 2.3 shows the E_b/N_0 values with different interleaver sizes to reach the BER of 10^{-5}. The performance is also compared with Shannon limit. The coding gain and the distance from the bound at rate 4/5 are given in the table. Second, the run time of the program increases significantly when the interleaver size is increased. The third observation is the so called floor flaring effect, which is a phenomenon that when the E_b/N_0 value increases steadily, the rate of improvement in performance decreases significantly. This is a serious effect because large increase of E_b/N_0 can only achieve very little improvement in the performance of the turbo codes. The error floor effect is found to be caused by the performance union bound of the turbo codes and happens when the performance is near the bound. When it happens, the slope of the curve drops and then keeps the same as the slope of the bound. In our simulation results, no floor flaring effect is found for sizes 256×256 and 128×128. Thus the floor flaring effect happens at lower than 10^{-6} level for these two cases. It can not be observed because it is beyond the capability of our simulation. But for sizes less than 64×64, the floor flaring effect is obvious as a low slope region of the performance curve. The BER values where the error floor effect appears are listed in Table 2.4. If 10^{-6} is set as a level to decide if the error floor flaring effect is significant enough to influence the performance of turbo codes, then for the Pseduo-random interleaver we used, 64×64 is the minimum size which can be accepted.

Table 2.4. Floor flaring effect for different interleaver sizes

	Error floor effect
256*256	$<<10^{-6}$
128*128	$<<10^{-6}$
64*64	Between10^{-7} and 10^{-6}
32*32	Between10^{-6} and 10^{-5}
16*16	Between10^{-4} and 10^{-5}

2.4.3 Puncturing pattern

Puncturing is also an important factor to determine the performance of the turbo codes. The higher the desired code rate, the more parity bits need to be punctured, and the poorer the performance of the turbo codes. From another view, puncturing causes the

decrease of the output weight, which decreases the free distance of the code and degrades the performance of the codes.

Puncturing pattern is used to decide which parity bits should be punctured and which should be kept after puncturing. Notation $P(c_1, c_2)$ is used to indicate the puncturing pattern of turbo codes. In [3], the author claimed that from their simulation results, some of the puncturing patterns, such as $P(3, 4)$ for rate 2/3 and $P(7,6)$ for rate 4/5, are the optimal choice. No theoretical proof was given in the paper. Our opinion is that the selection of the puncturing pattern has some relation with the interleaver algorithm. Different interleavers will have different requirements for the puncturing pattern. For the pseudo – random interleaver, which is used in [3] and also in our simulations, we don't think the value of c_1 or c_2 has any significant influence on the code performance. After the random interleaving, the order of the information bits has been completely changed. There is no reason to say that keeping the c_{1th} bit in the first parity sequence and the c_{2th} bit in the second sequence is better than other choices. To prove our thinking, we did a set of the simulations. Table 2.5 shows the different puncturing patterns in the simulations. The second column of the table gives the patterns that were claimed to be optimal by [3] at four different rates. The other two patterns have been randomly selected for comparisons for each rate and they are listed in the third and fourth columns of the table. In the fifth column of the table, the E_b/N_0 value selected for each rate to do the simulations is listed.

Table 2.5 Puncturing patterns selected for different code rates

Code rate	Puncturing Pattern form [3]	Random Puncturing Pattern (1)	Random Puncturing Pattern (2)	Eb/No (dB)
2/3	P(3,4)	P(1,2)	P(1,1)	1.6
¾	P(3,5)	P(3,3)	P(3,2)	2.1
4/5	P(7,6)	P(1,6)	P(1,1)	2.5
16/17	P(2,2)	P(2,11)	P(11,11)	4.5

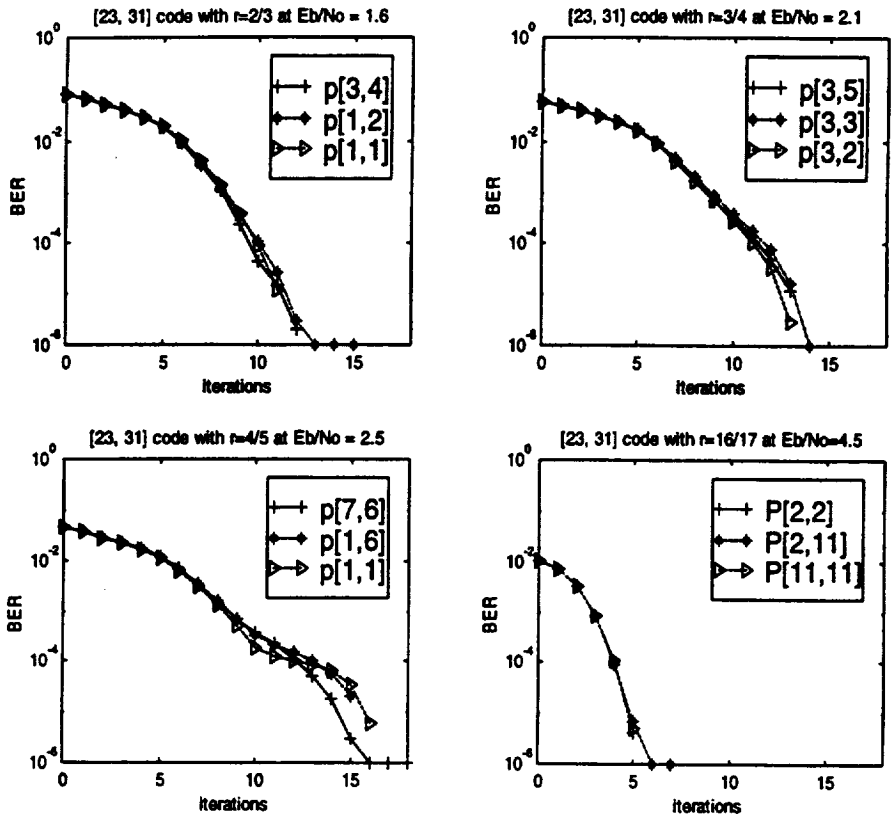

Fig 2.15 Comparisons of different puncturing patterns for high rates at certain Eb/No

Fig 2.15 shows the simulation results for all the four rates at these E_b/N_0 values. We compared the performance for the three different selected puncturing patterns at different iterations. As expected, there's no obvious difference in the performance of the three different puncturing patterns for all of the rates. The puncturing pattern selected by [3] can not be distinguished to be the optimal choice.

In our simulations, we happen to find that for some special rates, turbo codes show very poor performance, and the increase of the interleaver size does not show large improvement as for other rates. Fig 2.16 shows the case for one of the special rates, rate 5/6. In Fig 2.16, we see BER reaches 10^{-5} at E_b/N_0 =8.3. This performance is much worse than other rates and far from what is expected. Other special rates giving poor performances are 10/11 and 15/16 within our concerned range (rate 1/2 to 16/17). We find for these special rates, puncturing pattern is no longer showing insignificant effect on the performance. Next, we give the explanation for the poor performance of these rates,

and then we offer the modified puncturing patterns we designed, which successfully improved the performance of these codes to a level as good as all other rates.

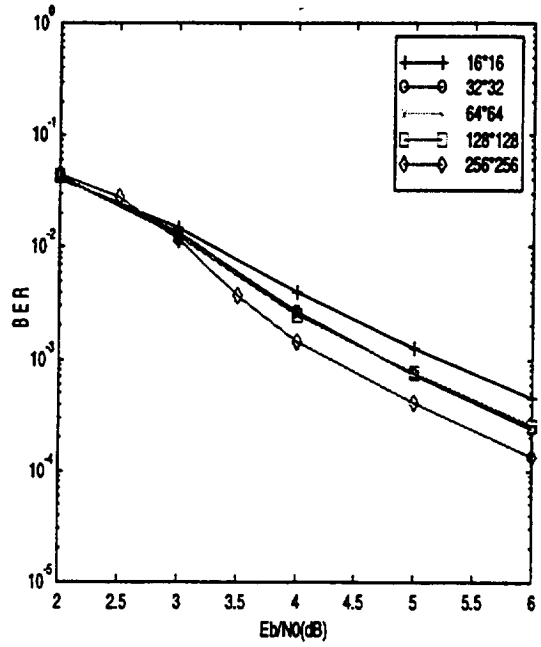

Fig 2.16 The performance of 5/6 code with different interleaver sizes

As we mentioned above the impulse response of single data input shows a periodic pattern of parity bits at output of the recursive encoder because of the feedback structure. And since the encoder is linear, an input of two or more data will yield a sum of shifted versions of periodic patterns and is essentially periodic. Thus a period structure exists in the output parity sequence of the encoder. When the primitive feedback polynomial is implemented, this period can reach the maximum. For our *M=4* case, that period is 15. That is to say, in this period, 15 different locations are possible to be selected as the kept bits. Comparing the influence of this period of 15 on the normal and special rates, we found that for normal rates, the bits on all the 15 locations have the same probability to be kept by puncturing. But for the special rate, only the bits on some of the 15 locations are possible to be kept, and the bits on all other locations will never be selected. To make it more clear, we give an example of the comparison between rate of 2/3 as normal rate and rate of 5/6 as special rate. For rate of 2/3 turbo codes, we keep 1 bit in every 4 parity bits to achieve the desired code rate. Similarly, for rate of 5/6 turbo codes, we keep 1 bit in every 10 parity bits. Without any essential loss of generality, for

2/3 code rate, we choose a puncturing pattern which keeps the third bit in each 4 parity bits, and for 5/6 code rate, we choose the first bit in each 10 parity bits. Table 2.6 and table 2.7 below show how many different locations in the period of 15 can be selected for these two rates.

Table 2.6 selected bit locations after puncturing for 2/3 rate

3^{rd} in each 4 parity bits	3	7	11	15	19	23	27	31
Location in period 15	3	7	11	15	4	8	12	1
3^{rd} in each 4 parity bits	35	39	43	47	51	55	59	...
Location in period 15	5	9	13	2	6	10	14	...

Table 2.7 Selected bit locations after puncturing for 5/6 rate

1_{st} in each 10 parity bits	1	11	21	31	41	51	61	71
Location in period 15	1	11	6	1	11	6	1	11
1_{st} in each 10 parity bits	81	91	101	111	121	131	141	...
Location in period 15	6	1	11	6	1	11	6	...

From Table 2.6, we see that in the case of 2/3 rate, in the first 4 parity bits, the 3^{rd} location is picked. In the second 4 parity bits, the 7^{th} location is picked. Then, the 11^{th}, 15^{th} locations are picked in the 3^{rd} and 4^{th} group of 4 parity bits. Thus, as shown in table 5, we found that all of the 15 locations can be selected with same probability. Then we look at the special 5/6 rate case in Table 2.7. In the first and second group of 10 parity bits, the 1^{st} and the 11^{th} locations are selected. Then in the 3^{rd} group, the 6^{th} location is picked. Then in the following groups, we see from the table that the 1^{st}, 11^{th}, 6^{th} locations are selected over and over again. Thus, only a limited number of locations are picked in this case (This is also true in the other special code cases such as 10/11, 15/16). When the information sequence has low weight, there is large probability that in the long length of consecutive periods of 15, the value of the bits on some locations is always the same. Suppose these bits are all zeros, then after the puncturing, the weight of the output sequence will be very low because the weight of the retained parity bits is too low. Under such circumstances, even if the encoder itself doesn't generate the low weight output

sequence, the final output weight is very low because of the inappropriate puncturing pattern. And this is the reason for the poor performance.

The solution of this problem is to modify the puncturing pattern so that more of different locations can be selected for these rates. Here we show how we designed the alternative pattern for 5/6 code rate to improve the performance. We have calculated that if the first bit in each 10 parity bits is retained, only the 1^{st}, 11^{th} and 6^{th} locations in the 15 locations can be selected. Similarly, if the second bit in each 10 parity bits is retained, it picks the 2_{nd}, 12_{th}, and 7_{th} locations. In Table 2.8, *P(i)* in the first column represents the i_{th} bit retained in each 10 parity bits. The second column shows the locations that can be picked in the period of 15. From this table, it's not difficult to observe that if we select the 1^{st} , 12^{th} , 23^{rd} , 34^{th} , 45^{th} bit in every 50 parity bits, all the locations are selected and we can still maintain the code rate to be 5/6. Simulations were performed to examine this alternative puncturing method. Fig 2.17 shows that with all different interleaver sizes, the performance of the codes improved significantly. For larger interleaver sizes, the improvement is especially significant. The E_b/N_0 value to make BER reach 10^{-5} is decreased from 8.3 dB to 2.8 dB by implementing our modified puncturing pattern.

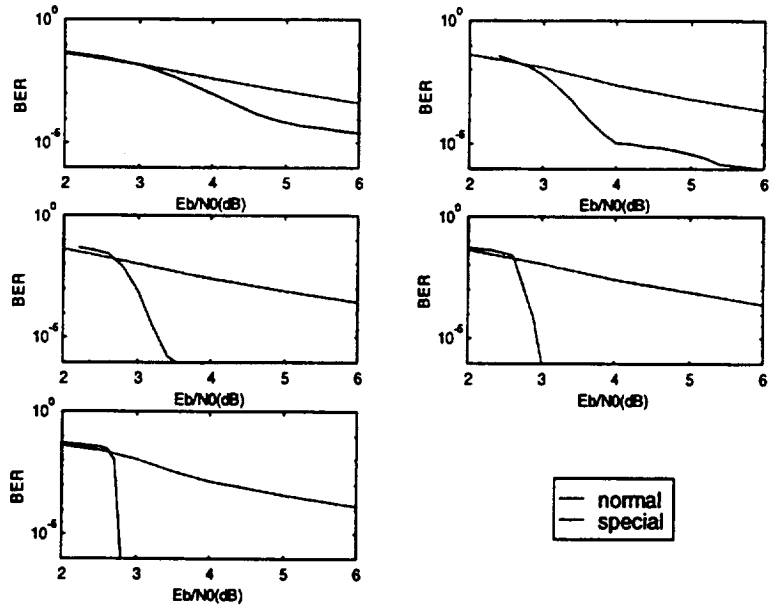

Fig 2.17 The improvement of the performance with the modified puncturing pattern at different interleaver sizes

Table 2.8 The locations selected by selecting different bits in each 10 parity bits for 5/6 rate

P(1)	1 , 11 , 6
P(2)	2 , 12 , 7
P(3)	3 , 13 , 8
P(4)	4 , 14 , 9
P(5)	5, 15, 10

For rate 10/11 and 15/16, same method is used to design the new puncturing patterns. In Fig 2.18, we can see the performance of the three special code rates (5/6, 10/11, 15/16) improved very significantly.

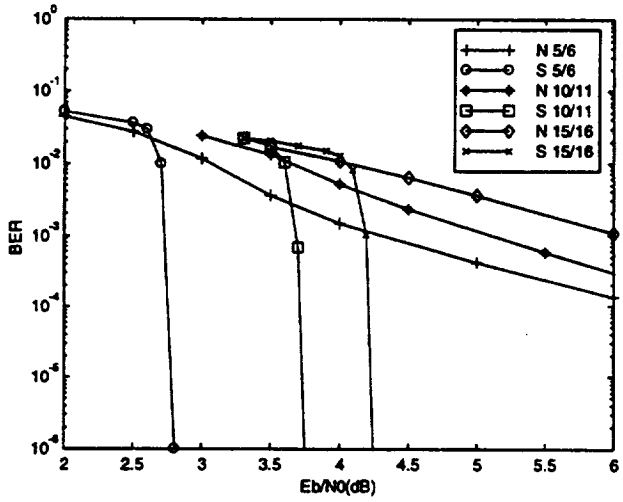

Fig 2.18 The improvement of the performance with the modified puncturing pattern at code rates 5/6, 10/11, 15/16

CHAPTER 3
ITERATIVE BLOCK DECODING

Many efficient algorithms have been found for using channel measurement information (soft decisions) in the decoding of convolutional codes than in the block codes, so researchers are concerned with the maximum likelihood decoding of linear block codes using channel measurement information. This decoding method is particularly useful to decode the high-rate codes because the complexity will increase very fast with the increase of the parity bits. To implement maximum likelihood decoding on linear block codes, it's necessary to construct a trellis for the block code. So in section 3.1, the method to construct trellis from a linear block code will be introduced. The iterative log-likelihood decoding algorithm is given in section 3.2. The implementation of this method using trellis is discussed in section 3.3.

3.1 Construction of trellis from block codes

3.1.1 Characteristics of the trellis constructed from block codes

Soft decision, maximum likelihood decoding of any *(n, k)* linear block code can be accomplished by using the Viterbi algorithm. If the block code is over *GF(2),* the trellis constructed will have these characteristics:

1) The depth of the trellis is n.
2) There are no more than $2^{(n-k)}$ states in the trellis.
3) There are 2^k paths through the trellis, each of the 2^k distinct codewords correspond to a distinct path.
4) Each node in the trellis represents an $(n-k)$ tuple with elements 0 or 1 (the two elements of *GF (2)*).
5) Each transition between two states is labeled with the appropriate codeword symbol v_k, the first k symbols represent the k information bits u_k, the following n-k symbols represent the parity bits.

There are also some other properties for special block codes:

1) For the cyclic code, the trellis is periodic.
2) For a productive code, the number of states in the trellis can be much less than $2^{(n-k)}$ [5].
3) For the single parity check code, the Viterbi algorithm applied to the trellis is the same as the Wagnar decoding.

3.1.2 The method of construction

The general formulation of the trellis for linear block codes uses the systematic H matrix of the code. Compared to the trellis of the convolutional codes, the structure of trellis formed from block codes is irregular. $s_j(i)$ is used here to represent the nodes at depth i, and the subscript 'j' represents the j_{th} state in the total $2^{(n-k)}$ states, r_i is used as the input bit between depth i to depth $i+1$, and h_i is used as the i_{th} column of the H matrix. Then, the steps for constructing a trellis are shown below:

1) The trellis starts at depth $i=0$ with the all zero state, named as $s_0(0)$.
2) At each depth i, the collection of nodes at depth $(i+1)$ is obtained from the collection of nodes at depth i, the formula used is shown below:

$$s_l(i+1) = s_j(i) + r_i h_{i+1} \tag{3.1}$$

3) Nodes and lines that do not end at all zero state at depth n are removed.

Here we give an example of a Hamming code to show how to follow these three steps to construct the trellis.

Hamming codes are block codes with code rate $(2^m - 1, 2^m - 1 - m)$, given by (1.12) and (1.13). For convenience, we choose $m = 3$ and thus we get a Hamming code with (7,4) code rate. The minimum distance of the code is 3. It means that we are able to detect two errors but can correct only 1 error by using this code. Assume we have a systematic H matrix as below

$$H = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

We can construct the trellis with the H matrix. The trellis should be from depth 0 to depth 7, and have at most 8 states in each depth from 000 to 111.

The initial state at depth 0 is 000 as described in step 1. The input between depth 0 and depth 1 has two possible values, 0 and 1. We can calculate the states at depth 1 by using equation (3.1). Then we have as follows

when input is 0, $s_0(1) = s_0(0) + 0 \times h1 = 000 + 0 \times 101 = 000$ (3.3)

when input is 1, $s_5(1) = s_5(0) + 1 \times h1 = 000 + 1 \times 101 = 101$ (3.4)

h1 here is the transpose of the first column of ***H*** matrix.

Thus at depth 1, we will have two states 000 and 101. Then by the same method, at depth 2, four states are obtained. Two of them are obtained from the state 000 at depth 1, the other two from the state 101 at depth 1, by different inputs 0 and 1 (Fig 3.1).

The calculations are shown here:

From state 000 at depth 1,

when input is 0, $s_0(2) = s_0(1) + 0 \times h2 = 000 + 0 \times 111 = 000$ (3.5)

when input is 1, $s_7(2) = s_0(1) + 1 \times h2 = 000 + 1 \times 111 = 111$ (3.6)

From state 101 at depth 1,

when input is 0, $s_5(2) = s_5(1) + 0 \times h2 = 101 + 0 \times 111 = 101$ (3.7)

when input is 1, $s_2(2) = s_5(1) + 1 \times h2 = 101 + 1 \times 111 = 010$ (3.8)

The same method is applied repeatedly for the depths from 3 to 7. And the number of states remains no more than 8 at these depths. The completely constructed trellis is shown in Fig 3.1.

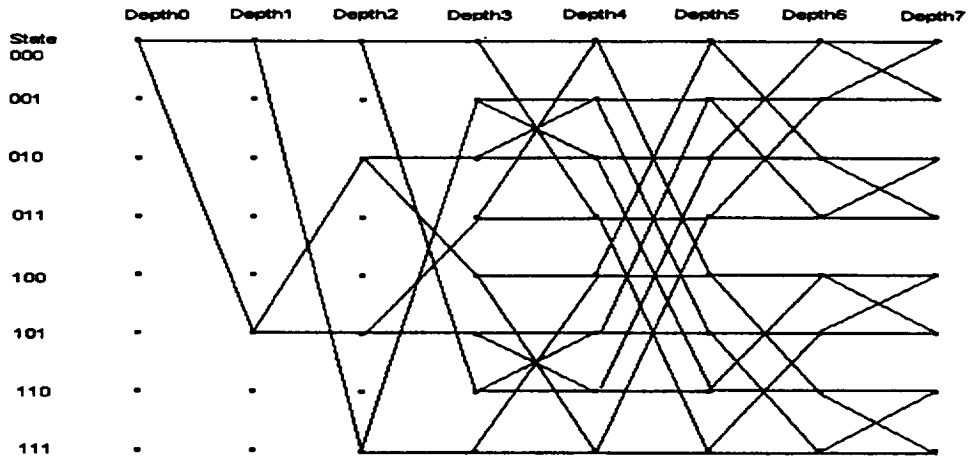

Fig 3.1 The trellis constructed for a (7,4) Hamming code before expurgation

Following step 3, the next step is to remove the nodes and lines that do not end at 000 state at depth n. The trellis after expurgation is shown in Fig 3.2.

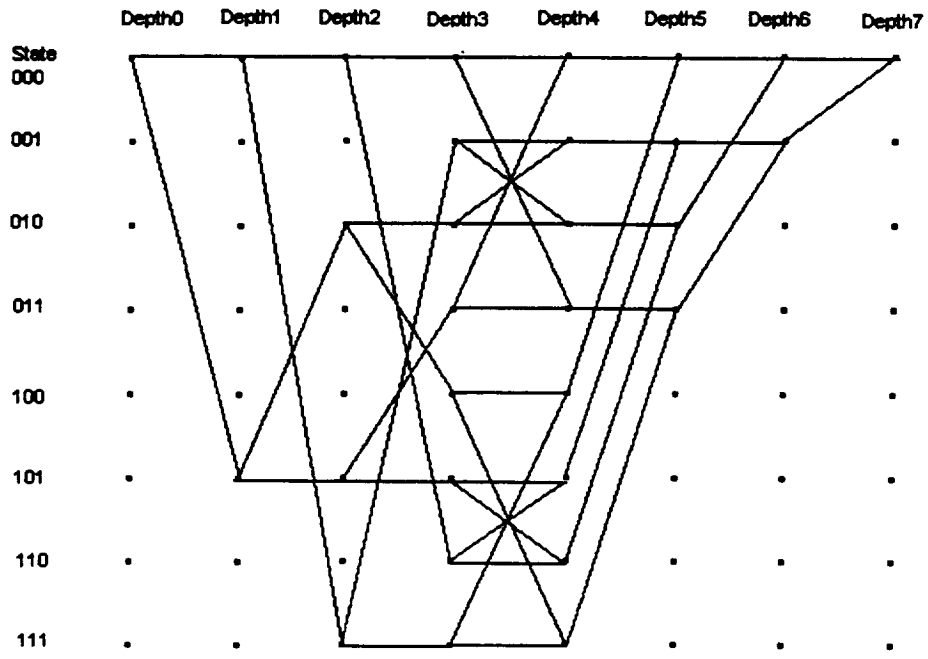

Fig 3.2 Expurgated trellis for (7,4) Hamming codes

For cyclic codes, an alternative method can also be used to form the trellis. It is built by tracing all the possible states of the storage devices for all possible inputs. The number of trellis states at depth i in the expurgated trellis is 2^k in the range *[1, n-k-1]*, $2^{(n-k)}$ in the range *[n-k, k]*, and $2^{(n-k)}$ while i are in the range of *[k, n]*. And the trellis repeats its pattern in the range *[n-k, k]*.

The steps for building the trellis are as follows:

1) The trellis starts at depth $i=0$ with the all zero state.
2) The polynomials at depth $i+1$ are then formed from the polynomials at depth i in accordance with the formula: $s_l(x; i+1) = (xs_j(x; i) + x^q r_i)$ *modulo* $g(x)$
3) Nodes and lines that do not end at all zero state at depth n are removed.

In polynomial notation, each of the states is represented by a polynomial.

3.2 Iterative log-likelihood decoding of binary block codes

In this part, to show how the decoding algorithm works, we will introduce the log-likelihood algebra, the soft in / soft out decoder, the iteration algorithm and some optimal and sub-optimal algorithms being used.

3.2.1 Log-likelihood algebra

The log-likelihood ratio of a binary random variable u is defined as

$$L(u) = \log \frac{P(u = u_1)}{P(u = u_2)} \tag{3.9}$$

$P(u)$ here denotes the probability of the random variable u. This ratio is denoted as the soft value. The sign of $L(u)$ is the hard decision, and the magnitude is the reliability (soft) decision. If the random variable u is conditioned on another random vector, named as y, then the conditioned log-likelihood ratio can be described as:

$$L(u \mid y) = \log \frac{P(u_1 \mid y)}{P(u_2 \mid y)} \tag{3.10}$$

Note that if the probability $P(y) = 1$, the ratio of that term can be canceled out, the joint log likelihood $L(u, y)$ is then equal to the conditioned log-likelihood $L(u \mid y)$, so from equation (3.10), we have,

$$L(u \mid y) = L(u) + L(y \mid u) = \log \frac{P(u_1)}{P(u_2)} + \log \frac{P(y \mid u_1)}{P(y \mid u_2)} \tag{3.11}$$

The "symbol by symbol" MAP (maximum a posteriori probability) is the optimal decoding algorithm [4]. A trellis of finite duration can represent it. The output of a "symbol by symbol" MAP decoder is defined as a posteriori log-likelihood ratio for transmitted +1 and −1 in the information sequence:

$$L(\hat{u}) = L(u \mid y) = \log \frac{P(u = +1 \mid y)}{P(u = -1 \mid y)} \tag{3.12}$$

Assume the transmission is on an AWGN channel, we will have

$$p(y \mid u = 1) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y-1)^2}{2\sigma^2}\right) \tag{3.13}$$

$$p(y \mid u = -1) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y+1)^2}{2\sigma^2}\right) \tag{3.14}$$

Together with equation (3.11), the posteriori log-likelihood ratio of u conditioned on the matched filter output y is:

$$L(\hat{u}) = L(u \mid y) = \log\frac{P(u=+1)}{P(u=-1)} + \log\frac{P(y \mid u=+1)}{P(y \mid u=-1)}$$

$$= \log\frac{P(u=+1)}{P(u=-1)} + \log\frac{\frac{1}{\sqrt{2\pi\sigma^2}}\exp(-\frac{(y-1)^2}{2\sigma^2})}{\frac{1}{\sqrt{2\pi\sigma^2}}\exp(-\frac{(y+1)^2}{2\sigma^2})} \qquad (3.15)$$

$$= L(u) + \log(\exp(4y/2\sigma^2)) = L(u) + \frac{2}{\sigma^2}y = L(u) + L_c y$$

$L(u)$ is the priori ratio. $L_c = \frac{2}{\sigma^2}$ is called as the reliability of the channel. In our research, we will assume the channel with a constant L_c (time- invariant).

3.2.2 Soft-in / soft-out decoder

The log likelihood algebra shows that any decoder can be used which accepts soft inputs (including a priori values), and delivers soft outputs (made up of three terms, the soft channel, the priori input, and the extrinsic value). Any linear binary code in systematic form can be used as the component code and the soft-in/ soft-out algorithms exist for these codes. Fig 3.3 is a soft-in/ soft-out decoder.

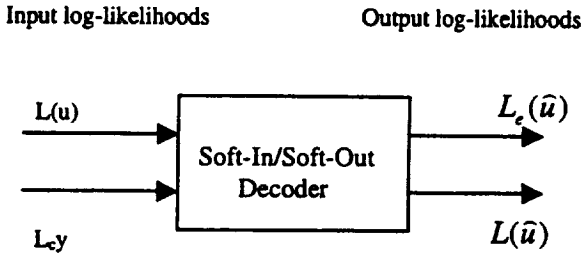

Fig 3.3 Soft-in / soft-out decoder

In Fig 3.3, $L(u)$ represents a priori values for all the information bits, $L_c y$ are the channel values for all code bits. $L_e(\hat{u})$ represents the extrinsic values for all information bits, and $L(\hat{u})$ is the soft output, a posteriori values for all information bits.

The extrinsic information contains the soft output information from all the other coded bits in the code sequence. The $L(u)$ and $L_c y$ value of the current bit do not influence it. Note that the extrinsic values are used as a priori values only for information bits and not for parity bits because codeword probabilities are determined from a priori probabilities of the information bits only.

For systematic codes, we have three independent estimates for the log-likelihood ratio. The soft output of the information bit u can be represented by the three additive terms:

$$L(\hat{u}) = L_c y + L(u) + L_e(\hat{u}) \tag{3.16}$$

3.2.3 Iterative decoding algorithm

Iterative decoding of systematic convolutional codes has been termed as turbo coding. However, it can also be used for linear binary systematic block codes.

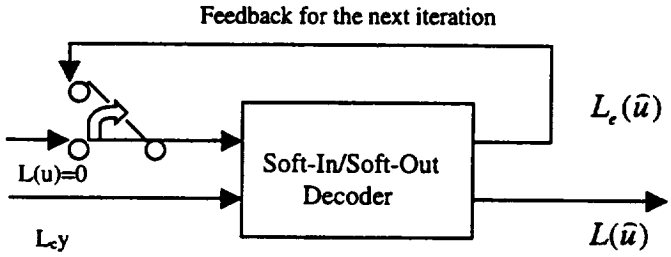

Fig 3.4 Iterative decoding procedure with soft-in / soft-out decoders

For the first iteration, no a priori value exists, thus we can initialize it to be 0. After that, the extrinsic values are used as the a priori value of next iteration step as shown in Fig 3.4.

At first, the L-values are statistically independent but after several iterations, because they use the same information indirectly over and over again, they will be more and more correlated. For the final decision after the last iteration, the last extrinsic pieces of information are combined with the received value as the output.

The iterations can be controlled by a stop criterion derived from cross entropy, t here represents the number of iterations[4].

$$T(t)=\sum_{k}\frac{|\Delta L_e^{-(t)}(\hat{u}_i)|^2}{\exp(|L_Q^{(t)}(\hat{u}_i)|)}<threshold \tag{3.17}$$

3.2.4 Optimal and sub-optimal algorithms

As we have mentioned above, the "symbol by symbol" MAP algorithm is the optimal method. If we use "symbol by symbol" MAP decoding rule for systematic convolutional codes in feedback form with binary trellis, the formula (3.16) can be represented as

$$L(\hat{u}_i)=L_c y_{i,1}+L(u_i)+\log\frac{\sum_{\substack{(s',s)\\u_i=+1}}\gamma_i^{(e)}(s',s)\cdot\alpha_{i-1}(s')\cdot\beta_i(s)}{\sum_{\substack{(s',s)\\u_i=-1}}\gamma_i^{(e)}(s',s)\cdot\alpha_{i-1}(s')\cdot\beta_i(s)} \tag{3.18}$$

In which, s and s' represent the indexes at level i-1 and i respectively. We have the forward recursion

$$\alpha_i(s)=\sum_{s'}\gamma_i(s',s)\cdot\alpha_{i-1}(s') \tag{3.19}$$

and the backward recursion

$$\beta_{i-1}(s')=\sum_{s}\gamma_i(s',s)\cdot\beta_i(s') \tag{3.20}$$

The forward and backward recursion are initialized with $\alpha_{start}(0)=1$, and $\beta_{end}(0)=1$. The branch transition probabilities between s' and s are,

$$\gamma_i^{(e)}(s',s)=\exp(\frac{1}{2}\sum_{v=2}^{n}L_c y_{i,v}x_{i,v}) \tag{3.21}$$

The calculation of actual probabilities can be avoided by using the logarithm of the probabilities and the approximation $\log(e^{L1}+e^{L2})\approx\max(L_1,L_2)$. This sub-optimal realization of the "symbol by symbol" MAP rule is called s Log-MAP rule. It has been proved that the performance of the log–MAP algorithm is close to the optimal "symbol by symbol" MAP algorithm [4].

The "soft-in/soft-out " Viterbi Algorithm (SOVA) for systematic convolutional codes in feedback form with a binary trellis can also be used. The SOVA output in its approximate version has the format:

$$L_{SOVA}(\hat{u}_i) = L_c y_{i,1} + L(u_i) + L_e(\hat{u}_i) \tag{3.22}$$

In which, $L_e(\hat{u}_i)$ is the product of the $\hat{u}_i$ and the first three terms in the formula,

$$M_i(s^{(j)}) = M_{i-1}(s'^{(j)}) + \frac{1}{2}L(u_i)u_i^{(j)} + \frac{1}{2}\sum_{v=1}^{n} L_c y_{i,v} x_{i,v}^{(j)} \tag{3.23}$$

This method preserves the desired additive structure. Consequently, we subtract the input values from the soft output of the SOVA and obtain the extrinsic information to be used in the matrices of the succeeding decoder. The extrinsic term is weekly correlated in this case. For small memories, the SOVA is about half of the complexity of the Log-MAP algorithm.

When MAP decoding rule is used for linear binary block codes, the branch transition probability for systematic block codes with statistically independent information bits can be written as

$$\begin{aligned}&\gamma_i(s', s) = P(s \mid s') \cdot P(y_i \mid s', s) = p(x_i, y_i) \\ &= \begin{cases} p(y_i \mid x_i) \cdot p(u_i) \ldots\ldots 1 \le i \le k \\ p(y_i \mid x_i) \ldots\ldots k+1 \le i \le n \end{cases}\end{aligned} \tag{3.24}$$

Also the log likelihood ratio,

$$\begin{aligned}&L(x_i \mid y_i) \\ &= \begin{cases} L_c y_i + L(u_i) \ldots\ldots 1 \le i \le k \\ L_c y_i \ldots\ldots k+1 \le i \le n \end{cases}\end{aligned} \tag{3.25}$$

Thus the soft output of the "symbol by symbol" MAP algorithm for block codes can be written as

$$L(\hat{u}_i) = L_c y_i + L(u_i) + \log \frac{\sum_{\substack{(s',s) \\ u_i=+1}} \alpha_{i-1}(s') \cdot \beta_i(s)}{\sum_{\substack{(s',s) \\ u_i=-1}} \alpha_{i-1}(s') \cdot \beta_i(s)} \tag{3.26}$$

If the Log-MAP algorithm is used, the formula can be simplified as

$$L_{Log-MAP}(\hat{u}_i) = L_c y_i + L(u_i) + \max_{\substack{(s',s)\\ u_i=+1}}(\log \alpha_{i-1}(s') + \log \beta_i(s)) - \max_{\substack{(s',s)\\ u_i=-1}}(\log \alpha_{i-1}(s') + \log \beta_i(s)) \tag{3.27}$$

3.3 Implementation of the algorithm

Two methods are considered to implement the MAP decoding rule for linear block codes. One of the methods implements the original code and is closely related to the "symbol by symbol" MAP algorithm, the other uses dual code. These two algorithms lead to the same result.

3.3.1 Straightforward implementation

Omitting the terms which are equal for all transitions from time *i-1* to time *i* and using the preceding definition of $L(x_i, y_i)$, the branch transition operation used in (3.19) and (3.20) can be written as $exp(L(x_i, y_i)x_i/2)$, so (3.16) can be described as

$$L(\hat{u}_i) = L_c y_i + L(u_i) + \log \frac{\sum_{\substack{x \in C\\ u_i=+1}} \prod_{\substack{j=1\\ j \neq i}}^{N} \exp(L(x_j, y_j)x_j / 2)}{\sum_{\substack{x \in C\\ u_i=-1}} \prod_{\substack{j=1\\ j \neq i}}^{N} \exp(L(x_j, y_j)x_j / 2)} \tag{3.28}$$

This equation separates the codewords in two groups. One with all the codewords having a "+1" at the i_{th} position, the other with all the codewords having a "-1 " at the kth position.

This separation can be implemented into trellis by small changes in the construction principle. In general, *i* different trellises are constructed to obtain the soft output $L(\hat{u}_i)$ for all information bits.

The trellis is built by using all the columns of the H matrix excluding the i_{th} one, and additionally by storing every path ending at time *n* at the state $S_n = h_i$.

$S_{end1}=0$ and $S_{end2}=h_i$ are two possible ending states. The time steps in the trellis are named after the corresponding column of the H matrix, thus the i_{th} time instant will not appear in the trellis any longer.

The paths ending in the zero state S_{end1} represent the codewords with a "+1" at the i_{th} position. The paths ending in the state S_{end2} represent the codewords with a "-1" at the i_{th} position. For the class of cyclic codes the trellises for the different information bits are obtained by simply shifting the indices.

3.3.2 Dual code implementation

If $n - k < k$, dual code will have fewer codewords than the original code. So, under such situation, the use of dual code will result in the reduction of the decoding complexity. The dual code C' can be presented as a trellis with at most 2^k states.

The forward recursion can be written as: $\tilde{\alpha}_i(s) = \sum_{s'} \tilde{\gamma}_i(s', s) \cdot \tilde{\alpha}_{i-1}(s')$ (3.29)

The backward recursion: $\tilde{\beta}_{i-1}(s') = \sum_{s} \tilde{\gamma}_i(s', s) \cdot \tilde{\beta}_i(s')$ (3.30)

The recursions are initialized with $\tilde{\alpha}_0(0) = 1$, and $\tilde{\beta}_n(0) = 1$. The branch transition probabilities between the states s, s' are defined here as,

$$\tilde{\gamma}_i(s', s) = (\tanh(L(x_i; y_i)/2))^{(1-x_i)/2} \quad (3.31)$$

Two methods are used to implement "symbol by symbol" MAP rule using the dual code. Method 1 builds up the full trellis for the dual codes and implements one forward and one backward recursion. The soft output for each information bit is calculated by the formula,

$$L(\hat{u}_i) = L_c y_i + L(u_i) + \log \frac{\sum_{(s',s)} \tilde{\alpha}_{i-1}(s') \cdot \tilde{\beta}_i(s)}{\sum_{\substack{(s',s) \\ x_i=+1}} \tilde{\alpha}_{i-1}(s') \cdot \tilde{\beta}_i(s) - \sum_{\substack{(s',s) \\ x_i=-1}} \tilde{\alpha}_{i-1}(s') \cdot \tilde{\beta}_i(s)} \quad (3.32)$$

Method 2 is to construct the modified trellis for the dual codewords to perform one forward recursion for each information bit. The soft output can be written as:

$$L(\hat{u}_i) = L_c y_i + L(u_i) + 2ar\tanh(\tilde{\alpha}_n(S_{end2}) / \tilde{\alpha}_n(S_{end1})) \quad (3.33)$$

Dual code of a cyclic code is still a cyclic code. So the modified trellis for every information symbol can still be built one from the other by simply shifting the indices.

3.3.3 A decoding example by using straight-forward implementation

For convenience, here we still choose the (7, 4) Hamming code and the same systematic H matrix as in 3.1.2.

$$H=\begin{bmatrix}1 & 1 & 1 & 0 & 1 & 0 & 0\\ 0 & 1 & 1 & 1 & 0 & 1 & 0\\ 1 & 1 & 0 & 1 & 0 & 0 & 1\end{bmatrix} \tag{3.2}$$

Thus we will have a corresponding G matrix (see in Section 1) as:

$$G=\begin{bmatrix}1 & 0 & 0 & 0 & 1 & 0 & 1\\ 0 & 1 & 0 & 0 & 1 & 1 & 1\\ 0 & 0 & 1 & 0 & 1 & 1 & 0\\ 0 & 0 & 0 & 1 & 0 & 1 & 1\end{bmatrix} \tag{3.34}$$

Assume we have the information bits u= 1, 1, 0, 1, then the codeword is

$$v=u\cdot G=[1\ \ 1\ \ 0\ \ 1]\cdot\begin{bmatrix}1 & 0 & 0 & 0 & 1 & 0 & 1\\ 0 & 1 & 0 & 0 & 1 & 1 & 1\\ 0 & 0 & 1 & 0 & 1 & 1 & 0\\ 0 & 0 & 0 & 1 & 0 & 1 & 1\end{bmatrix}=[1\ \ 1\ \ 0\ \ 1\ \ 0\ \ 0\ \ 1] \tag{3.35}$$

The first four bits in v are information bits, and the last three bits are parity bits.

In BPSK transmission, we actually transmit the signal sequence as *[1, 1 –1, 1, -1, -1,1]*. The signals received are simulated using SPW software. In the simulation, in order to make L_c = *1*(for the simplification of calculation), we make the variance of the noise to be 2. The simulated results are *y=[1.6 2.7 -1.2 -0.8 -0.6 0.08 1.1]*.

If the hard decoding is applied directly to these received signals, we will have a sequence *[1 1 0 0 0 1 1]*. 2 errors have occurred, one in the fourth information bit and second in the second parity bit. Though (7, 4) Hamming code can detect two errors, it can only correct one error, so the normal decoding failed.

Now we will see how our iterative block decoding works step by step, and what it can do to decrease the error probability.

3.3.3.1 Constructing trellis for information bits

From the discussion in section 3.3.1, we know that four trellises should be built, one for each of the information bit locations. And from each trellis, a soft output $L(\bar{u}_i)$ should be obtained.

We build the trellis for the first information bit as an example. The trellis should be built following these rules:

1) The trellis is built by using all the columns of $\boldsymbol{H}$ matrix except the i^{th} one.
2) There are two ending states of the trellis, S_{end1} represents the codewords with a "+1" at i^{th} position, S_{end2} represents the codewords with a "-1" at i_{th} position.
3) The trellises of cyclic codes are obtained by simply shifting the $\boldsymbol{H}$ matrix.

For information bit location1, to build a trellis that satisfies the above rules, we first build a trellis by shifting to the left each column of the $\boldsymbol{H}$ matrix and the first column becomes the last column.

$$H_{SHIFT1} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{3.36}$$

The expurgated trellis constructed (same method as in 3.1.2) is shown in Fig 3.5.

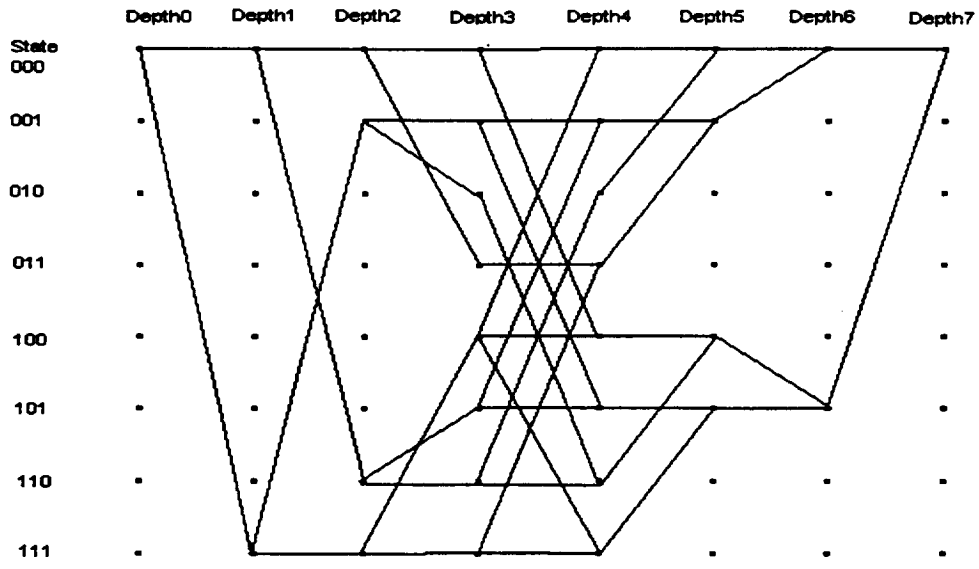

Fig3.5 Full trellis for first information bit location.

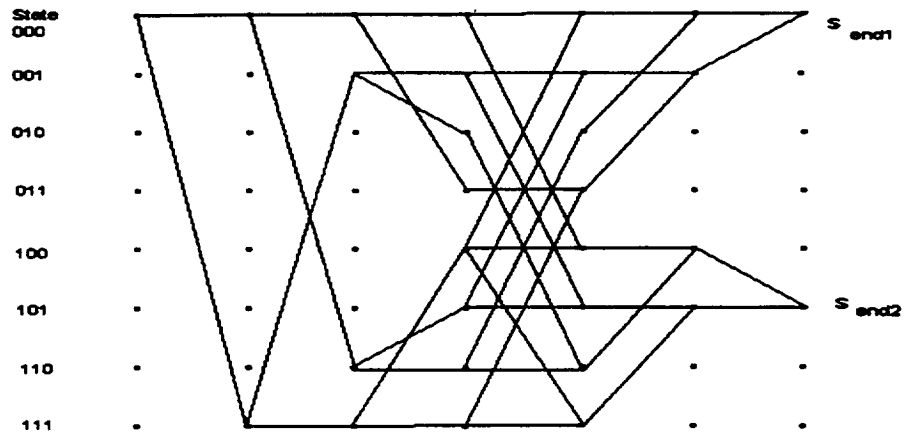

Fig 3.6 The final trellis with two ending states for first information bit location.

This trellis is not the final trellis we want yet. In the decoding system, we only use the trellis between depth 0 and depth 6. Thus we will have the final trellis with two ending states as in Fig 3.6.

For information bit location 2, the same method can be applied. First a full trellis is built by using the shifted matrix $\boldsymbol{H}$,

$$H_{SHIFT2} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \tag{3.37}$$

Then, the part of trellis between depth 6 and depth 7 is discarded. (Fig 3.7)

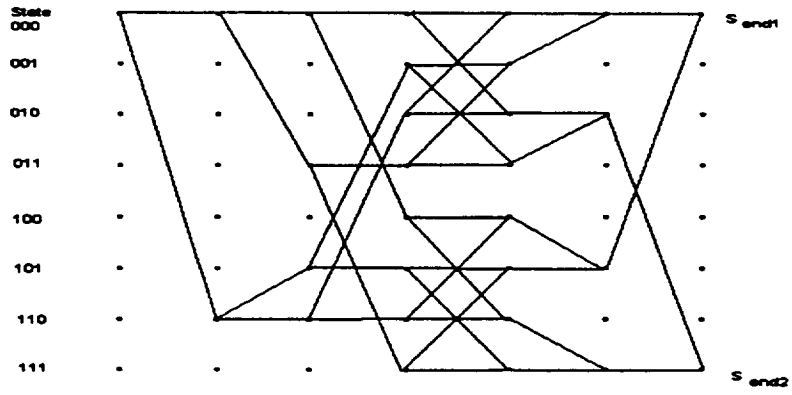

Fig 3.7 The final trellis with two ending states for information bit location 2

We see two different structured trellises for information bit location 1 and information bit location 2. Trellises for information bit location 3 and information bit location 4 can be constructed from further shifting of H matrix.

3.3.3.2 The decoding system

Fig 3.8 shows the basic decoding system for straightforward implementation.

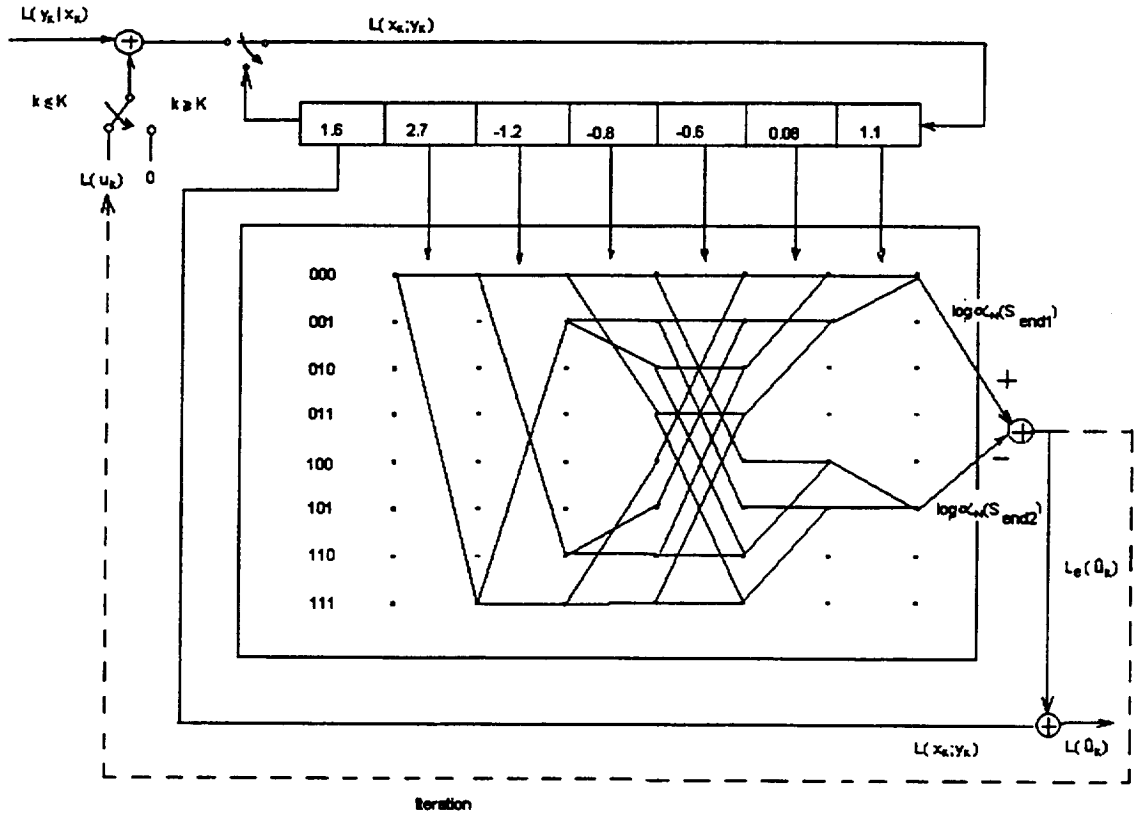

Fig3.8 The decoding system of (7,4) Hamming code while working on information bit 1

In this system, $L(y_k|x_k)=L_c y$ is the input to the system. $L(u_k)$ is added to information bits but not to the parity bits. All the information and parity bits are stored in the seven buffers. While the system works for information bit 1, the system calculates the extrinsic value by using the trellis we have constructed for the bit. And this $L_e(\hat{u}_1)$ can be feedback to $L(u_1)$ as the priori value of the next iteration. At last, the extrinsic value is added to the $L(y_1; x_1)$ as the final soft output $L(\hat{u}_1)$.

For information bit 2, the system shifts the buffers and changes the corresponding trellis. Fig 3.9 shows the system while working for information bit 2.

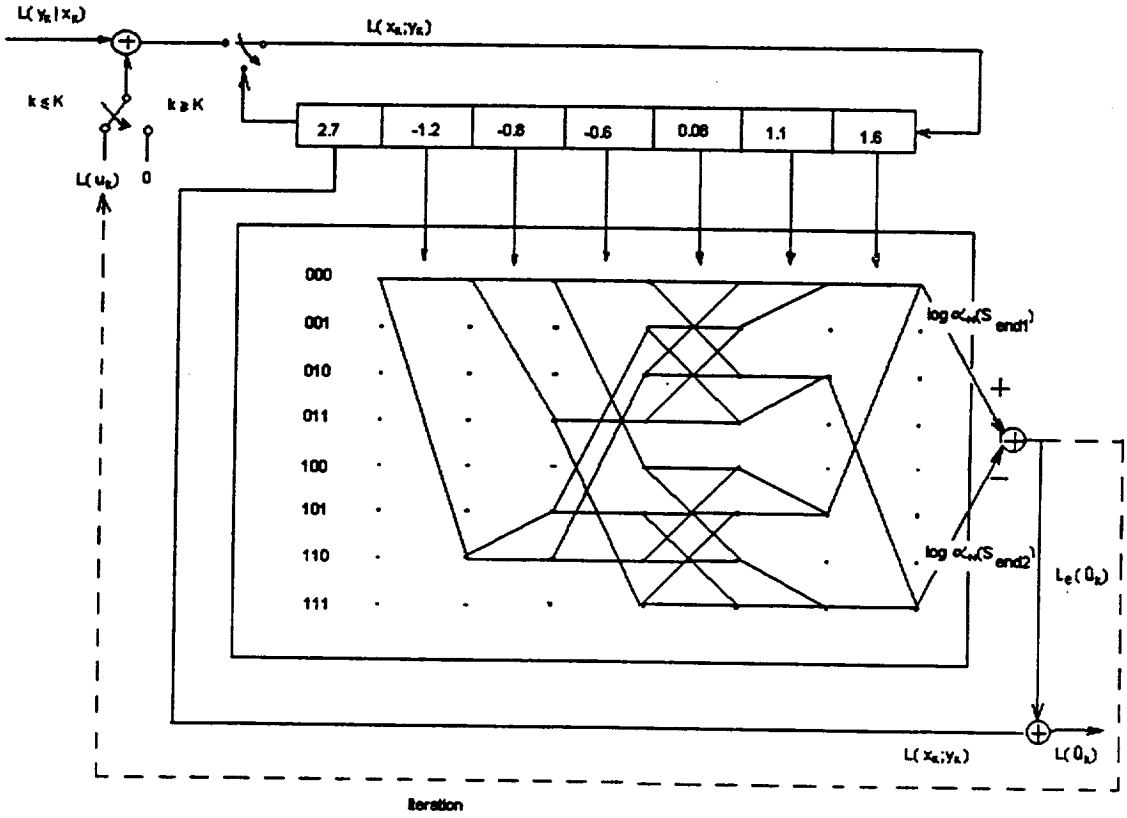

Fig3.9 The decoding system of the (7,4) Hamming code while working on information bit 2

3.3.3.3 Calculation of extrinsic value from the constructed trellis

The calculation of extrinsic value is done by using (3.28). We go through the trellis for information bit location 1 to show the steps to calculate $L_e(\hat{u}_1)$. We separate the calculation in (3.28) for each depth in the trellis. The equation for each depth can be described as

$$\log \alpha_j(s_1) = \log(\alpha_{j-1}(s_1)\exp(-L(x_j, y_j)/2) + \alpha_{j-1}(s_2)\exp(L(x_j, y_j)/2)) \quad (j \neq i) \tag{3.38}$$

Between depth *j-1* and depth *j*, the *exp(-L(x_j, yj)/2)* used in the equation is the branch transition operation being used form state s_1 to state s_1, while input is 0. The *exp(L(x_j,*

$yj)/2)$ used in the equation is the branch transition operation being used form state s_2 to state s_1, while input is 1.

Assume $L(u)=0$ before the first iteration. From (3.25) and the simulated received signals, we can calculate the value of each $L(x_j, y_j)$ as follows,

$$L(x_1, y_1)=1.6 \quad L(x_2, y_2)=2.7 \quad L(x_3, y_3)=-1.2 \quad L(x_4, y_4)=-0.8$$
$$L(x_5, y_5)=-0.6 \quad L(x_6, y_6)=0.08 \quad L(x_7, y_7)=1.1 \tag{3.39}$$

Now we start from the beginning of the trellis (Fig 3.6).

Depth 0:

We set an initial condition $\alpha_0(0)=1$.

Depth 1:

From depth 0 to depth 1, the input is $L(x_2, y_2)=2.7$, so $exp(-L(x_2, y_2)/2)=0.26$ and $exp(L(x_2,y_2)/2)=3.9$. There are 2 states at depth 1, and each state has only one previous state.

$$\text{For state 0, } \alpha_1(0)= \alpha_0(0)\, exp(-L(x_2, y_2)/2)=0.26;$$
$$\text{For state 7, } \alpha_1(7)= \alpha_0(0)\, exp(L(x_2, y_2)/2)=3.9; \tag{3.40}$$

Depth 2:

There are four states at depth 2, and each state still has only one previous stage. Calculating by the same method for depth 1, and have $L(x_3, y_3)=-1.2$, we get,

$$\alpha_2(0)= \alpha_1(0)\, exp(-L(x_3, y_3)/2)=0.26\times1.82=0.47;$$
$$\alpha_2(1)= \alpha_1(7)\, exp(L(x_3, y_3)/2)=3.9\times0.55=2.15;$$
$$\alpha_2(6)= \alpha_1(0)\, exp(L(x_3, y_3)/2)=0.26\times0.55=0.14;$$
$$\alpha_2(7)= \alpha_1(7)\, exp(-L(x_3, y_3)/2)=3.9\times1.82=7.10; \tag{3.41}$$

Depth 3:

8 states, each state with one previous state, $L(x_4, y_4)=-0.8$, we can get,

$$\alpha_3(0)=0.47\times1.5=0.71 \quad \alpha_3(1)=2.15\times1.5=3.23$$
$$\alpha_3(2)=2.15\times0.67=1.44 \quad \alpha_3(3)=0.47\times0.67=0.31$$
$$\alpha_3(4)=7.10\times0.67=4.76 \quad \alpha_3(5)=0.14\times0.67=0.09$$

$$\alpha_3(6)=0.14\times1.5=0.21 \qquad \alpha_3(7)=7.10\times1.5=10.65$$

(3.42)

Depth 4:

8 states, each state with 2 previous states, $L(x_5, y_5) = -0.6$, we get,

$$\alpha_4(0)= \alpha_3(0)\exp(-L(x_4, y_4)/2)+ \alpha_3(4)\exp(L(x_4, y_4)/2)=0.71\times1.35+4.76\times0.74=4.48$$

$$\alpha_4(1)= \alpha_3(1)\exp(-L(x_4, y_4)/2)+ \alpha_3(5)\exp(L(x_4, y_4)/2)=3.23\times1.35+0.09\times0.74=4.43$$

$$\alpha_4(2)= \alpha_3(2)\exp(-L(x_4, y_4)/2)+ \alpha_3(6)\exp(L(x_4, y_4)/2)=1.44\times1.35+0.21\times0.74=2.10$$

$$\alpha_4(3)= \alpha_3(3)\exp(-L(x_4, y_4)/2)+ \alpha_3(7)\exp(L(x_4, y_4)/2)=0.31\times1.35+10.65\times0.74=8.30$$

$$\alpha_4(4)= \alpha_3(4)\exp(-L(x_4, y_4)/2)+ \alpha_3(0)\exp(L(x_4, y_4)/2)=4.76\times1.35+0.71\times0.74=6.96$$

$$\alpha_4(5)= \alpha_3(5)\exp(-L(x_4, y_4)/2)+ \alpha_3(1)\exp(L(x_4, y_4)/2)=0.09\times1.35+3.23\times0.74=2.51$$

$$\alpha_4(6)= \alpha_3(6)\exp(-L(x_4, y_4)/2)+ \alpha_3(2)\exp(L(x_4, y_4)/2)=0.21\times1.35+1.44\times0.74=1.35$$

$$\alpha_4(7)= \alpha_3(7)\exp(-L(x_4, y_4)/2)+ \alpha_3(3)\exp(L(x_4, y_4)/2)=10.65\times1.35+0.31\times0.74=14.61$$

(3.43)

Depth 5:

4 states, each state with 2 previous states, $L(x_6, y_6) = 0.08$, we get,

$$\alpha_5(0)= \alpha_4(0)\exp(-L(x_5, y_5)/2)+ \alpha_4(2)\exp(L(x_5, y_5)/2)=4.48\times0.96+2.10\times1.04=6.49$$

$$\alpha_5(1)= \alpha_4(1)\exp(-L(x_5, y_5)/2)+ \alpha_4(3)\exp(L(x_5, y_5)/2)=4.43\times0.96+8.30\times1.04=12.89$$

$$\alpha_5(4)= \alpha_4(4)\exp(-L(x_5, y_5)/2)+ \alpha_4(6)\exp(L(x_5, y_5)/2)=6.96\times0.96+1.35\times1.04=8.08$$

$$\alpha_5(5)= \alpha_4(5)\exp(-L(x_5, y_5)/2)+ \alpha_4(7)\exp(L(x_5, y_5)/2)=2.51\times0.96+14.61\times1.04=17.60$$

(3.44)

Depth 6:

2 states as ending states of the trellis, each state with 2 previous states, $L(x_7, y_7) = 1.1$, we get,

$$\alpha_6(0)= \alpha_5(0)\exp(-L(x_6, y_6)/2)+ \alpha_5(1)\exp(L(x_6, y_6)/2)=6.49\times0.57+12.89\times1.73=26.00$$

$$\alpha_6(5)= \alpha_5(5)\exp(-L(x_6, y_6)/2)+ \alpha_5(4)\exp(L(x_6, y_6)/2)=17.60\times0.57+8.08\times1.73=24.01$$

(3.45)

Finally, we obtain the extrinsic value $L_e(\hat{u}_1) = \log 26.00 - \log 24.01 = 0.03$.

(3.46)

3.3.3.4. Simulation result

We have mentioned above that, without the iterative log-likelihood decoding, 2 bits are in error and the decoding of Hamming codes failed. Using the program of Guo in which the iterative log-likelihood algorithm is applied, we find that the error in information bit 4 has been corrected.

REFERENCES

1. William E Ryan (wryan@nmsu.edu), "A Turbo Code Tutorial".
2. S.C. Kwatra, Peter Curry, "Investigation of Different Constituent Encoders in a Turbo-code Scheme for Reduced Decoder Complexity", *Technical Report, EECS Department, The University of Toledo*,1998.
3. Omer F. Acikel (oacikel@nmsu.edu), william E. Ryan (wryan@nmsu.edu), "High Rate Turbo Codes for BPSK / QPSK Channnels".
4. J. Hagenauer, "Iterative Decoding of Binary Block and Convolutional Codes", *IEEE Trans. on Information Theory*, VOL. 42, NO. 2, March, 1996.
5. Jack K. Wolf, "Efficient Maximum Likelihood Decoding of Linear Block Codes Using a Trellis", *IEEE Trans. on Information Theory*, Vol. IT-24, 1978.
6. Herbert Taub, Donald L. Schilling, "Principles of Communication Systems", 2nd Edition .
7. Simon Haykin, "Digital Communication" .
8. Andrew J. Viterbi, Jim K Omura, "Principals of Digital Communication and Coding".
9. John G. Proakis, "Digital Communications".
10. Bernard Sklar," Digital Communications: Fundamentals and Applications".
11. Claude Berrou, "Some Clinical Aspects Of Turbo Codes", *International Symposium on Turbo Codes,* Brest-France, 1997.
12. Qinyu Chen, J.Kim, S.C. Kwatra, "Investigation on Higher Rate Turbo-Code", *Internal Report, EECS department, The University of Toledo.*
13. Claude Berrou, "Near Optimal Error Correcting Coding and Decoding: Turbo-Codes", *IEEE Trans. On Communications* , VOL. 44, NO. 10, Oct., 1996.
14. Rainer Licas, Martin Bossert, Markus Breitbach, "On Iterative Soft-Decision Decoding of Linear Binary Block Codes and Product Codes", *IEEE Journal on Selected Areas in Communications*, VOL. 16, NO. 2, Feb., 1998.
15. Hari T. Moorthy, Shu lin, Tadao Kasami, "Soft-Decision Decoding of Binary Linear Block Codes Based on an Iterative Search Algorithm", *IEEE Trans. on Information Theory*, VOL. 43, NO. 3, May, 1997.

16. Yulin Guo, J. Kim and S. C. Kwatra, "Implementation of Iterative 1-Dim Block Coding", *Internal Report, EECS department, The University of Toledo.*
17. Claude Berrou, A. Glavieux, P. Thitimajshima, "Near Optimal Error Correcting Coding and Decoding: Turbo-Codes (1)", *IEEE* 1993.
18. D. Divsalar, S. Dolinar, "Performance Analysis on Turbo Codes", *IEEE* 1995.
19. C. Wang, "On the Performance of Turbo Codes", *proceedings of IEEE MILCOM '98*, Bosten, MA. Oct, 1998.
20. J. Segher, L.C. Perez, "On Selecting Code Generators for Turbo Codes".
21. S.Benedetto R. Garello G. Montorsi, "A Search for Good Convolutional Codes to be Used in the Construction of Turbo Codes", *IEEE Trans. on Communications* , VOL. 46, NO. 9, Sept., 1998.
22. G. Battail, "Pseudo-Random Turbo codes", *IEEE* 1995.
23. J. D. Anderson, V. V. Zyablov, "Interleaver Design for Turbo Codes", *International Symposium on Turbo Codes*, Brest, France, 1997.
24. M. Oberg, P.H. Siegel, "The Effect of Puncturing in Turbo Encoders", *International Symposium on Turbo Codes*, Brest, France, 1997.
25. S. Dolinar, D. Divsalar, "Weight Distribution for Turbo Codes Using Random and Nonrandom Interleaver", *TDA Prograss Report*, 42-122, pp. 56-65, Aug 1995.
26. W.J. Blackert, E.K. Hall, "An Upper Bound on Turbo Code Free Distance", *IEEE* 1996.
27. M. Oberg, P.H. Siegel, "Lowering the Error Floor Flaring for Turbo Codes", *International Symposium on Turbo Codes,* Brest, France, 1997.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1999	Final Contractor Report

4. TITLE AND SUBTITLE

Investigation of Near Shannon Limit Coding Schemes

5. FUNDING NUMBERS

WU–632–50–5C–00
NAG3–1718

6. AUTHOR(S)

S.C. Kwatra, J. Kim, and Fan Mo

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

The University of Toledo
Department of Electrical Engineering and Computer Science
College of Engineering
Toledo, Ohio 43606

8. PERFORMING ORGANIZATION REPORT NUMBER

E–11924

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
John H. Glenn Research Center at Lewis Field
Cleveland, Ohio 44135–3191

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

NASA CR—1999-209402
DTVI–59

11. SUPPLEMENTARY NOTES

Project Manager, R.E. Jones, Communications Technology Division, NASA Glenn Research Center, organization code 5650, (216) 433–3457.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Categories: 33 and 61 Distribution: Nonstandard

This publication is available from the NASA Center for AeroSpace Information, (301) 621–0390.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

Turbo codes can deliver performance that is very close to the Shannon limit. This report investigates algorithms for convolutional turbo codes and block turbo codes. Both coding schemes can achieve performance near Shannon limit. The performance of the schemes is obtained using computer simulations. There are three sections in this report. First section is the introduction. The fundamental knowledge about coding, block coding and convolutional coding is discussed. In the second section, the basic concepts of convolutional turbo codes are introduced and the performance of turbo codes, especially high rate turbo codes, is provided from the simulation results. After introducing all the parameters that help turbo codes achieve such a good performance, it is concluded that output weight distribution should be the main consideration in designing turbo codes. Based on the output weight distribution, the performance bounds for turbo codes are given. Then, the relationships between the output weight distribution and the factors like generator polynomial, interleaver and puncturing pattern are examined. The criterion for the best selection of system components is provided. The puncturing pattern algorithm is discussed in detail. Different puncturing patterns are compared for each high rate. For most of the high rate codes, the puncturing patten does not show any significant effect on the code performance if pseudo - random interleaver is used in the system. For some special rate codes with poor performance, an alternative puncturing algorithm is designed which restores their performance close to the Shannon limit. Finally, in section three, for iterative decoding of block codes, the method of building trellis for block codes, the structure of the iterative decoding system and the calculation of extrinsic values are discussed.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Coding; Modulation; Communications	82
	16. PRICE CODE: A05

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	