AFRL-IF-RS-TR-2002-50
Final Technical Report
March 2002

HIGH DIMENSIONAL TRELLIS CODED MODULATION

Ohio University

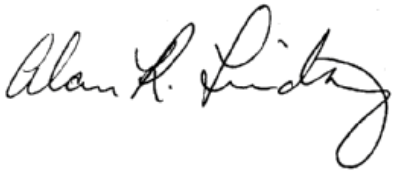
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2002-50 has been reviewed and is approved for publication.

APPROVED:



ALAN R. LINDSEY
Project Engineer

FOR THE DIRECTOR:

WARREN H. DEBANY, Technical Advisor
Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	MARCH 2002	Final Mar 99 – Jun 01

4. TITLE AND SUBTITLE
HIGH DIMENSIONAL TRELLIS CODED MODULATION

5. FUNDING NUMBERS
C - F30602-99-C-0064
PE - 62702F
PR -: 2304
TA - G9
WU - P1

6. AUTHOR(S)
Jeffrey C. Dill, Alan R. Lindsey, and Xiangyu Song

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Ohio University
Electrical and Computer Engineering Department
Athens Ohio 45701

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/IFGC
525 Brooks Road
Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2002-50

11. SUPPLEMENTARY NOTES
AFRL Project Engineer: Alan R. Lindsey/IFGC/(315) 330-1879/Alan.Lindsey@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***
This work develops a high dimensional circular trellis-coded modulation with permuted state structure (HDCTCM) for power-limited spread spectrum channels. High dimensional simplex signal constellation is systematically developed to achieve the optimal energy efficiency and maximize the minimum distance among error events for any size trellis. By systematically analyzing the error events, sets of state transitions that should be assigned simplex are identified. Butterfly structure of circular trellis coding is successfully related to those state transitions and perfectly aligned into a multidimensional matrix. An algebraic representation of a simplex in a high dimensional space is introduced, and a corresponding signal constellation and symbol assignment procedure are built.
A circular trellis code has an unknown starting state; conventional algorithms for calculating the minimum distance of a trellis code always assume a known starting state. This work develops a computational algorithm to calculate minimum distance of circular trellis codes. The coding gain of HDCTCM is evaluated after the minimum distance is obtained. Tight upper and lower bonds on the minimum distance of HDCTCM codes are derived. Furthermore, a method to build codes that have the bounds of the minimum distance is developed. Whereas, in most coding schemes, the optimal distance codes can only be obtained through exhaustive search.

14. SUBJECT TERMS
Trellis Coding, Tail-Biting, Multi-Dimensional Modulation, Signal Assignment, Simplex Signals, Butterfly

15. NUMBER OF PAGES
194

16. PRICE CODE

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

Table of Contents

List of Tables

List of Figures

Executive Summary

This research project has successfully investigated the use of high-dimensional M-ary orthogonal signal sets with trellis-coded modulation (TCM) in order to achieve significant improvement in the performance of spread-spectrum communication systems.

The advantages derived from this work can be applied directly to improve the performance of anti-jam (AJ) and/or low probability of intercept (LPI) communication systems for military applications. In addition, this work would also has potential advantages in commercial code-division multiple access (CDMA) networks.

Trellis Coded Modulation (TCM) is a technique which combines the error correction and modulation functions of the communication system, to achieve significant gains in system performance. This is accomplished by using an m-ary signal set (termed a constellation of signals), which is partitioned into sub-constellations, and representing the transmitter as a finite state machine. The data symbols at the transmitter are mapped onto a sub-constellation in order to select the transmitted symbol. The sub-constellation is determined by the current "state" of the transmitter, and state transitions occur as each symbol is transmitted. While all symbols in the constellation are used with equal probability over time (assuming random data), only certain sequences of symbols can occur, due to the structure of the finite state machine model. Knowledge of these sequential restrictions is used by the receiver to choose the most likely valid sequence, which enables significant error correction to occur at the receiver. The improved performance is primarily due to the fact that decisions are made according to a Euclidean distance criteria, as opposed to a Hamming distance criteria. This is inherent in a system which combines modulation and coding, and is somewhat analogous to soft decision decoding.

Traditionally, TCM has been used on band-limited channels, in order to achieve higher data rates. The classic example of this is high-rate telephone modems (56 Kbits/sec over a 3 KHz. channel). This is accomplished with a large constellation of symbols in a 2 dimensional signal space (i.e. inphase and quadrature). This places the signals in the constellation "close" together (i.e. they are not orthogonal) and thus more power is required to achieve acceptable error performance. Thus, TCM is used very effectively in this case to improve bandwidth efficiency, at the expense of power efficiency (a good trade-off for the band-limited telephone modem channel).

In general, there are two widely used algorithms for decoding convolutional or trellis codes: the well known Viterbi algorithm, and the BCJR algorithm. In a generalized sense, both of these algorithms are mathematically equivalent, and in fact both algorithms belong to a larger class of algorithms referred to as “dynamic programming” algorithms.

The Viterbi algorithm has enjoyed wide commercial success, as the decoder of choice for convolutional codes in satellite systems, digital cell phones, and telephone modems, to name only a few of its applications.

The BCJR algorithm, after having been ignored for years, has become popular recently for the decoding of turbo codes (or parallel concatenated codes) which require an iteration between two permuted code sequences.

The decoding algorithm which has been implemented for CTCM is based on the BCJR algorithm, with several key modifications which I will describe below. A tutorial of the standard BCJR algorithm, as well as detailed modifications, is included in the Appendix.

I should point out that at this point, my initial implementation of the BCJR algorithm achieves excellent bit-error-rate performance, but is not optimized for execution speed, as would need to be done in a commercially viable system. Work on improving the speed of this implementation is underway, and I believe that some additional patentable ideas will result.

The following are enhancements to the standard BCJR algorithm successfully implemented and tested under this contract.

Dealing with an Unknown Initial State

Conventional trellis decoding algorithms assume that the encoder and decoder both start in an agreed upon initial state. With the unknown initial state of TBCM, and the strong tailbiting property, this is not possible. The solution is to simply set all initial state probabilities as equally likely for the first iteration of the decoder, and allow the decoder to converge naturally on the correct initial state.

Dealing with non-binary data symbols.

The trellis used for TBCM is non-binary in a general sense. The current implementation uses a 4-ary source alphabet, but other sizes could easily be accommodated. In general, m-ary source alphabets are accommodated by structuring the encoder/decoder with a look-up table implementation which includes both the state transitions and the the channel symbols, and can be easily modified to accommodate any new trellis structure or symbol constellation.

Dealing with the absence of the data stream in the encoded bit stream (i.e. nonsystematic constituent codes)

Published descriptions of the implementation of turbo decoders refer to the permuted "common" or "extrinsic" information which is used in the feedback iteration process to improve the state estimates of the parallel decoders. In this code, the common information is not explicit in the data stream, and must be inferred indirectly by the

algorithm. This is accomplished by summing the individual transition probabilities at each stage of the trellis prior to the permuted/depermuted feedback stage, in order to obtain an estimate of the extrinsic information.

Permutation of symbol sequences

Rather than permuting individual symbols at the encoder, groups of symbols, of size M are permuted. The decoder is then structured to estimate the states between sequences of M symbols, rather than the states between individual symbols. This results in an improvement in bit-error performance, but the cost is greater computational complexity.

Summary of Results

This work has resulted in 1 Journal publications, 8 conference papers, 1 Ph. D. Dissertation, and 2 MS Theses, as listed below.

Journal Publications

"Bounds on the Minimum Distance of High Dimensional Circular Trellis Coded Modulation," S. Song, J. Dill, and A.R. Lindsey, submitted to *Signal Processing*.

Conference Publications

"Butterfly Structure in Trellis Coded Modulation", with Y. C. Lo, and Alan Lindsey, Proceedings of Globecom 99, Rio De Janeiro, Brazil, December, 1999, pp. 707-711 .

"Circular TCM with Simplex Signals," with Y. Lo and A.R. Lindsey, *Proceedings of International Conference on Signal Processing for Advanced Telecommunications,* Toronto, CANADA, Sep. 1998.

"Circular Trellis Coding for Wideband Channels," with A. R. Lindsey, and Y. Lo, *Proceedings of AIAA Space & Technology Conference,* Albuquerque, NM, Sep. 1999.

"On a Trellis Coded Modulation Design with a Strong Tail Biting Constraint," with S. Lopez-Permouth, A. R. Lindsey, *Proceedings of International Conference on Telecommunications 2000*, Acapulco, MEXICO, 21-24 May, 2000.

"Iterative Decoding of Circular Trellis Coded Modulation," with A.R. Lindsey, SPIE Conference on Digital Wireless Communications (OR36), Orlando, FL, Apr. 2000.

"Bounds on the Minimum Distance of High Dimensional Circular Trellis Coded Modulation," X. Song, J. Dill, and A.R. Lindsey, accepted for publication, *Defense Applications of Signal Processing*, Adelaide, Australia, September, 2001.

"High Dimensional Circular Trellis Coded Modulation", X. Song, J. Dill, and A. Lindsey, accepted for publication, MILCOM, Washington, DC, October, 2001.

Analysis of Minimum Error Events in Circular Trellis-Coded Modulation with Simplex Signaling Scheme," X. Song, J. Dill, and A. Lindsey, accepted for publication, *IEEE Emerging Technology Symposium on Broadband Communications for the Internet Era,* Dallas, TX, September, 2001.

Ph.D. Dissertations

12-00	Song, Xiangyu	High Dimensional Circular Trellis-Coded Modulation in Spread Spectrum Communications

MS Theses

3-00	Zhou, Biyun	Non-Binary Cyclic Codes and its Application in Decoding of High Dimensional Trellis-Coded Modulation
3-00	Cui, Xiao Xiao	Modified Viterbi Decoding Algorithm for Circular Trellis Coded Modulation

Patent Application

This work has also resulted in the filing of a patent application, entitled

"Apparatus and Method of CTCM Encoding and Decoding for a Digital Communication System"

The co-inventors of this patent are:

J. Dill
A. Lindsey
S. Lopez-Permouth
X. Song
F. Alder
Y. Lo

Simulation Code

This project has resulted in a detailed simulation of the CTCM transmitter and receiver on a Gaussian noise channel. The code for this simulation has been delivered in electronic format.

Chapter 1 INTRODUCTION

1.1 History of Research on CTCM

Trellis coding is a popular error correcting technology in modern digital communications. An important technique used in trellis coding is trellis termination that forces the encoded trellis path to satisfy the state constraint--the starting state and ending state should be the same [1]. In shift-register based trellis coding environment, the initial state of encoder is always assumed to be 0 state and zero tailing [1] [2] is used for trellis termination. In this method, a certain number of zeros are padded to the end of the terminated input data sequence to force the encoded trellis path to end at 0 state. However, code rate is reduced due to these padding zeros. It becomes significant when the input information sequence has short length.

An alternative to trellis termination without code rate loss is tail biting, first introduced by Solomon and Van Tilborg [3] and generalized later by Howard H. Ma *et al*. [2]. The state constraint is satisfied by forcing every trellis path to be a circular path--a path having the same starting state and ending state, which can be any of the possible states in the trellis. Its implementation involves initializing the encoder to a certain state with the last certain number of information bits in the input information sequence. For a long time, zero tailing has been the only practically used trellis termination method because of the simplicity and the fact that many efficient decoding algorithms were invented based on that condition.

With the recent development of turbo codes [4] and the requirement of short frame transmission [5] [6], trellis termination began to be addressed again [7] [8]. A novel circular trellis coding with a permuted state structure was invented [9] [10] [11]. A permuted state transition table is built to guarantee the state constraint. Any input data

block is uniquely mapped to a circular trellis path under this state transition table. This is done without code rate loss or initializing the encoder.

Since Ungerbock introduced trellis-coded modulation (TCM) in 1982 [12], coded modulation has been extensively researched and has achieved remarkable commercial success in its applications in band-limited channels. TCM combines a state-oriented trellis coding scheme--usually a convolutional coding scheme--and a multi-level/phase modulation into one single process. Coding gain is achieved without bandwidth expansion. Ungerbock's work eventually led to coded modulation schemes capable of operation near capacity in a band-limited channel [13].

Applying the concept of conventional TCM, circular trellis coded-modulation (CTCM) with the permuted state structure for power-limited spread spectrum channels has been investigated in [9] [10] [11] [14]. In these literatures, orthogonal, bi-orthogonal, and simplex signal constellations have been researched for very small trellises. Since there is no systematic procedure for signal constellation construction and symbol assignment, those works cannot be extended to large trellis for practical applications. Neither can systematic analysis be done on the distance property, which is the most importance parameter for an error correcting code.

Efficient decoding algorithms were developed for TCM [15] [16] [17] [18]. The most used is Viterbi algorithm [15], which is the optimum decoding to minimize the code word error [19] [20]. The other optimum algorithm, which minimizes the symbol (bit) error, is BCJR algorithm [21]. These algorithms require that the starting trellis state or the distribution of the starting states be known *a priori*. The unknown starting state in CTCM makes these decoding algorithms not directly applicable. Maximum likelihood (ML) decoding can always be used, but only for input information sequences with small length. No efficient decoding algorithm exists in the literature for current CTCM with permuted state structure. A circular BCJR algorithm was developed recently for tail biting codes [22], which can be useful in CTCM with permuted state structure.

1.2 Contributions of this Research

In this research, a systematic high dimensional CTCM with permuted state structure (HDCTCM) is developed. The systematic designs of a high dimensional simplex signal constellation and symbol assignment procedure are invented to achieve optimal energy efficiency for power-limited channels [23].

Minimum distance of a code is a primary parameter in determining its error correcting capacity [24]. Conventional algorithms [25] [26] to calculate the minimum distance of a trellis code always assume that the starting state is known *a prior*, which is not true for circular trellis coding. This research develops a practical computational algorithm to calculate the minimum distance of circular trellis coding [27]. The minimum distance of HDCTCM is obtained using this algorithm. The coding gain of HDCTCM is evaluated after the minimum distance is obtained.

Due to the systematic construction of the signal constellation and symbol assignment, upper and lower bonds are derived for HDCTCM codes [28]. Also, this research proves these bounds can be reached. And a method to build codes that have the bounds of minimum distance is developed [29], whereas in most coding scheme, the optimal distance codes can only be obtained through exhaustive search.

Moreover, the circular BJCR algorithm is explored for decoding HDCTM. Several implementation issues are solved. Tentative hard decision, along with the reliability information, is used to statistically evaluate the starting state. The circular property is incorporated into this algorithm to select the most reliable symbol as the starting point in the decoding.

1.3 Outline of this Dissertation

In this dissertation, the background knowledge on trellis coding, TCM, CTCM with permuted state structure, Viterbi and BCJR algorithms are given in Chapter 2. In Chapter 3, the systematic designs of signal constellation and symbol assignment for HDCTCM are introduced. The proof of uniformity of this coded modulation scheme is presented as well. Chapter 4 derives the computational algorithm to calculate the minimum distance for circular trellis codes. In addition, the distance property shown by

HDCTCM using this algorithm is illustrated. Chapter 5 derives the upper and lower bounds of the minimum distance of HDCTCM and designs the procedure to build the codes that achieve the bounds of minimum distance. The possible distance distributions of optimum codes are also discussed in this chapter. Chapter 6 explores iterative circular BCJR for decoding HDCTCM. Implementation issues are solved in this chapter. An iterative circular shift BCJR is developed for decoding HDCTCM. The properties of this decoding algorithm are demonstrated through simulations and comparisons with other decoding algorithms. The error performance of HDCTCM using this decoding algorithm is analyzed as well. Chapter 7 presents the conclusions and recommends some further research in a number of specific directions.

CHAPTER 2 BACKGROUND

This chapter provides the background materials on trellis coding, conventional TCM, circular trellis coding with permuted state structure, and decoding algorithms for TCM.

2.1 Error Control Coding

This dissertation work falls into the area of error control coding. In this section, the related stages to this research in digital communication system are discussed, and then the representation of error control codes is given.

2.1.1 Communication System Overview

A digital communication system can be described by the simplified Figure 2.1.

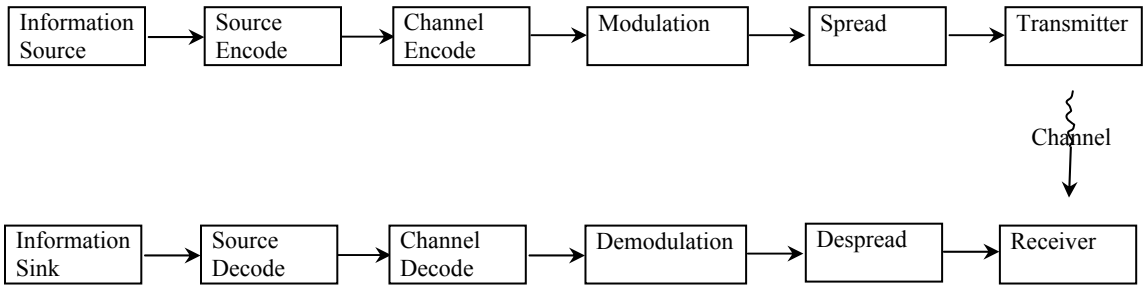


Figure 2.1 Simplified digital communication system

The stages related to this research are Channel Encode/Decode, Modulation/Demodulation and Spread/Despread. The input to the channel encoder is assumed to be the output of the source encoder and is called the information sequence.

Channel coding transforms the information sequence to a sequence of code symbols called code sequence or code word. The transformation is designed to improve communication performance by enabling the code sequence to better withstand various channel impairments, such as noise, fading, and jamming. The modulator maps the code sequence to a sequence of channel symbols drawn from a signal constellation. Channel symbols are realized by analog waveforms to be transmitted over the channel. In this dissertation, the channel is assumed to be a discrete memoryless channel (DMC) with additive white Gaussian noise (AWGN). The demodulator makes a definite decision (hard decision) or a quantized approximation (soft decision) for each received symbol to one of the symbols in the signal constellation. Those symbol values are fed into the decoder to decide the code sequence and the corresponding information sequence. Spreading maps the signals into an extended time or frequency range according to a certain mapping function before transmission. This has advantages in many aspects, such as anti-jamming. Despreading extracts the signal before spreading. More detailed structure and functionality of each stage can be referred to in [1].

2.1.2 Representation of Error Control Codes

A basic kind of channel coding is error control coding. It introduces a controlled redundancy to the information sequence to provide the transmitted code sequence with improved ability to combat error and provide a way for the receiver to detect or correct error during transmission over a noisy channel. Error control codes can be divided into two general categories: block codes and convolutional codes.

In block coding, the information sequence is segmented into blocks of fixed length k. The encoder transforms each block of k information symbols to a r-tuple vector known as code word according to a certain rule defined by a generator matrix or polynomial. k / r is defined as code rate--a measure of information bit (symbol) per code bit (symbol). $(r - k)$ bits are the redundancy bits (symbols) added to perform error detection or correction. The encoder encodes each k information symbols independently. Most decoding methods for block codes are algebraic operations such as syndrome decoding for linear block codes [30].

Convolutional codes were first introduced by Elias in 1955 [31]. Convolutional codes differ from block codes in that the encoder contains memory so that the r-tuple encoder output at any given time depends not only on the k-tuple input at that time instance but also on the previous K input k-tuples. K is defined as constraint length (sometimes termed as memory order). An (k, r, K) convolutional code can be implemented with a k-input, r-output linear sequential circuit with input memory K, with $K = \max_{1 \le i \le k} K_i$, where K_i is the number of the delay elements in the i_{th} shift register. The rate k / r has the same code rate significance as block codes. A binary $(2,1,2)$ encoder is shown in Figure 2.2.

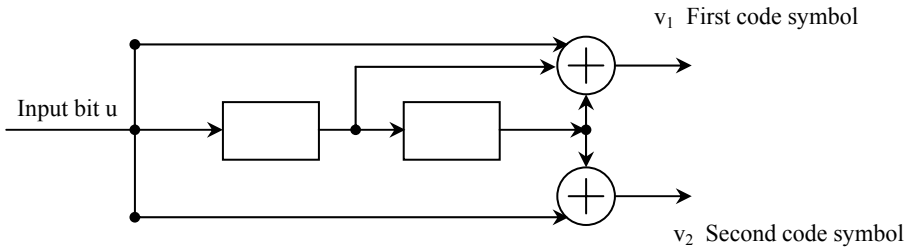


Figure 2.2 **A (2,1,2) binary convolutional encoder**

A set of polynomials indicating the connections between the adders and the stages of the registers completely describes the encoder. It is called the polynomial representation. The other two frequently used representations are state diagram and trellis diagram. Viewing the convolutional encoder as a finite state machine driven by the information sequence, a state diagram illustrates the state transitions related to each possible k-tuple input. Figure 2.3 is the state diagram for the encoder shown in Figure 2.2. The branch is labeled by the k-tuple input string together with the output code word. Denote the encoder state at time t_i as S_i, then the encoder is said to be Markov in the sense that the probability, $P(S_{i+1} \mid S_i, S_{i-1}, ..., S_0)$, of being in state S_{i+1}, given all the previous states, depends only on the most recent state S_i, that is, the probability is equal to $P(S_{i+1} \mid S_i)$.

By extending a state diagram along the time dimension and merging the same state at the same time instance, a trellis diagram can be obtained. Figure 2.4 is the trellis diagram for the encoder in Figure 2.2.

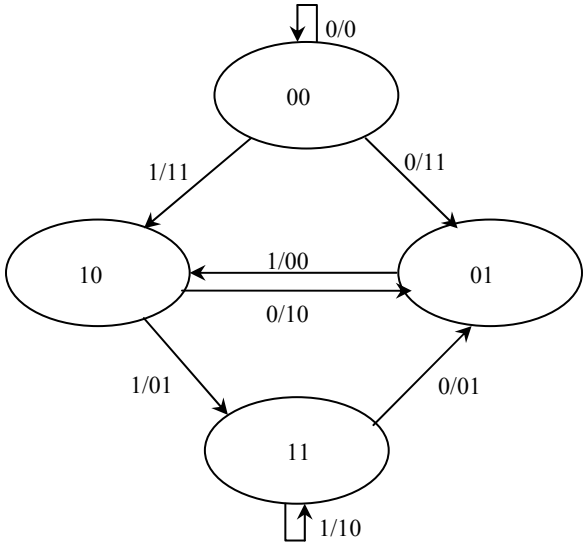


Figure 2.3 **State diagram for the encoder in Figure 2.2**

A node in a trellis denotes a state. A branch represents a state transition driven by a certain input information string. It is labeled by a code symbol the encoder outputs when that state transition occurs. The level of trellis at time t_0 is 0, and it increases by one at each time instance. From a starting state, all the states will be reached after a certain trellis level and then a fixed trellis section prevails. This trellis section completely describes the codes. A code word corresponding to an information sequence is represented by a sequence of state transitions along the time dimension, which is named as a trellis path.

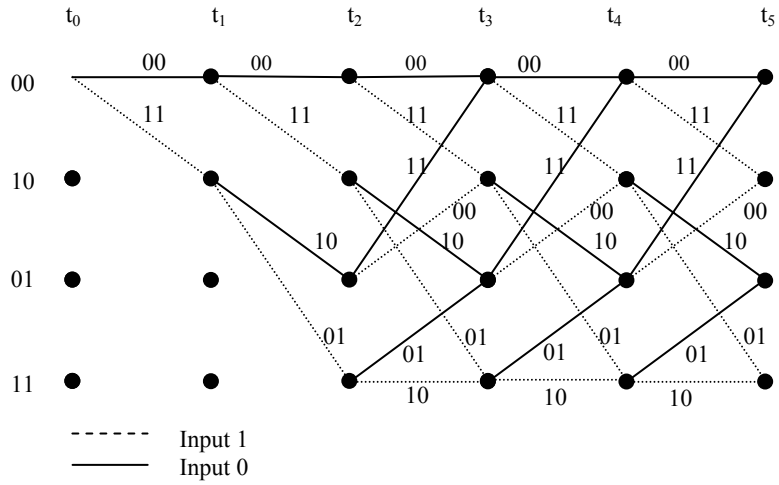


Figure 2.4 Trellis diagram for the encoder in Figure 2.2

Any encoder described by a finite state machine can be represented by a trellis and is named trellis encoder. Convolutional codes are the earliest linear codes in the class of trellis codes. We will investigate the circular trellis codes in this dissertation. Circular trellis encoder refers to a type of trellis encoder, in which each information sequence is mapped to a circular trellis path--a path having the same starting state and ending state. This state can be any of the total states in this trellis and is completely determined by the information sequence itself.

All the information conveyed by state diagram and trellis diagram can be recorded in a table, called state transition table. It indicates the next state of the state transition originating from any of the total states and driven by all the possible k-tuple inputs. Building a trellis codes is to specify its state transition table.

2.2 Trellis Termination

In trellis coding, the information sequence is truncated periodically with length L to have a block structure. In order to provide the same error protection to the last information symbol, a technique called trellis termination is employed to force the encoded trellis path to satisfy a state constraint--the starting and ending states of a trellis path should be the same. Zero tailing, tail biting, and circular trellis with permuted state

structure are the three ways to do this. They are introduced in the following two subsections.

2.2.1 Zero Tailing and Tail Biting

For a trellis code, usually 0 state is assumed. 0 state corresponds to the situation that the contents of the shift register are all zeros initially. Zero tailing is used for trellis termination. A certain number of zeros are appended to the truncated information sequence to force the encoded trellis path to go back to 0 state. For ease of understanding, consider a binary $(1, r, K)$ convolutional code. In this case, K zeros are needed to append. For an information sequence of length L, the resultant code is a $((L+K)r, L)$ block code of rate $(1/r)(L/(L+K)) = (1/r)(1-K/(L+K))$. The term $K/(L+K)$ is called the rate loss due to zero tailing. To reduce the rate loss, the truncation period L is generally made as long as permissible.

An alternative that does not introduce code rate loss is tail biting. For a binary $(1, r, K)$ encoder, first initialize the encoder by inputting the last K information bits of the information sequence into the encoder and ignore the output, then, input all L information bits and take the resultant $L \cdot r$ output bits as code word. The last K information bits will force the encoded path to go back to the initial state of the encoder. This state is fully determined by the input data sequence itself. It is a type of circular trellis coding. The resulting code is a $(L, L \cdot r)$ block code of rate $1/r$.

For a long time, zero tailing is the dominant practical method for trellis termination because of its simplicity and the fact that many efficient decoding algorithms were developed based on this assumption. Tail biting is only used for research to associate quasi-cyclic block codes with convolutional codes in order to apply convolutional decoding algorithm to block codes [3].

2.2.2 Circular Trellis Coding with Permuted State Structure

A novel circular trellis coding with permuted state structure was invented recently. The state constraint is satisfied without code rate loss or initializing the encoder. It is done by designing a permuted state transition table for the encoder.

Denote a circular trellis as $\mathrm{T}(n, D)$, where n is the number of exiting branches from each state. D is called trellis depth; it is the number of stages needed to reach all the possible states from a starting state. This starting state can be any of the total states in this trellis. When the encoder accepts one q-ary information symbol at one time (this is the case for circular trellis coding with permuted state structure), $n = q$, i.e., the size of the information symbol alphabet. When it accepts k-tuple q-ary inputs at one time, $n = q^k$. In either case, the total number of states in this trellis is $S = n^D$.

The state transition table is built using Zech's logarithm operation in finite field--$\mathrm{GF}(S)$. Number the total S states as ∞, 1, 2, ..., $S-1$.

Then in $\mathrm{GF}(S)$, "a nature $(1,\ \mathrm{S}-1)$-type permutation" is expressed as

$$\sigma_\infty = (\infty)\,(1\ \ 2\ \ 3\ \ ...\ \ S-2\ \ S-1) \tag{2.1}$$

It represents the state transition $\infty \to \infty$, $1 \to 2 \to 3 \to ... \to S-2 \to S-1 \to 1$; they are named as 1-cycle and $(S-1)$-cycle ordered state permutation, respectively.

The left $S-1$"$(1,\ \mathrm{S}-1)$-type permutation" in $\mathrm{GF}(S)$ can be built by using Zech's logarithm operation. Denote them as

$$\Theta_i = (i)\ (z(1)+i\ \ \ z(2)+i\ \ \ z(3)+i\ \ \ ...\ \ \ z(S\text{-}3)+i\ \ \ z(S\text{-}2)+i\ \ \ \infty),\ i = 1,2,...,S-1 \tag{2.2}$$

The state transition it represents is

$i \to i$, $z(1)+i \to z(2)+i \to z(3)+i \to ... \to z(S-3)+i \to z(S-2)+i \to \infty \to z(1)+i$. By properly choosing n out of the S "$(1,\ \mathrm{S}-1)$-type permutation" and letting each of them associate with the state transitions driven by one of the n information symbol alphabets, the state transition table is constructed. This state transition table guarantees that each information sequence of length $L \geq D+1$ will be mapped to a circular trellis path, and the starting and ending state of this path is a function of the symbols in the information sequence. No zero padding or encoder initialization is needed in this coding scheme.

2.3 Trellis-Coded Modulation

The goal of channel coding is to achieve coding gain. Coding gain is defined as the reduction in the required E_b / N_0 to achieve a specified error probability using the coded system versus an uncoded system. Where E_b is the average bit energy of the input data, N_0 is the one-sided power spectrum density of channel noise.

In conventional communication, coding and modulation are separate processes. The coding gain is achieved at the price of bandwidth expansion by transmitting r-tuple code word instead of k-tuple data block in the same time slot. TCM combines them into one process. Conventional TCM has been widely used in band-limited channels. With the development of spread spectrum communications, coded modulation for power-limited spread spectrum channels becomes the latest research interest. CTCM with permuted state structure was introduced. We will give a brief review on these two schemes.

2.3.1 Conventional TCM

TCM was invented to achieve coding gain without bandwidth expansion. It combines a multilevel/phase signal constellation with a trellis coding scheme--usually a convolutional code. A larger signal constellation is used to provide the needed coding redundancy while keeping the same channel symbol rate (bandwidth) and average power as those of an uncoded system. Figure 2.5 shows two multilevel/phase signal constellations used by TCM, illustrated along with the corresponding uncoded system.

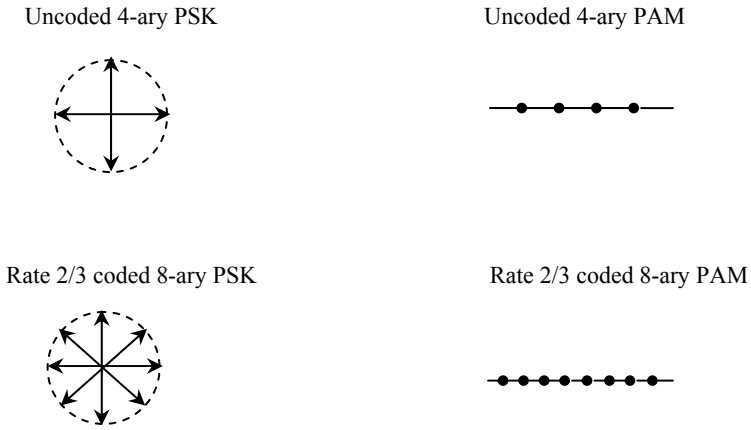


Figure 2.5 Increase of signal set size for trellis-coded modulation

The expanded signal set does result in reduced distance between adjacent signal points for a given signal power. However, because of the redundancy introduced by the code, this reduced distance no longer determines the error performance. Instead, the minimum Euclidean distance (ED) between all pairs of code sequences $\{a_n\}$ and $\{a'_n\}$,

which the encoder can produce determines the error performance. It is defined in Equation (2.2.1).

$$d_{\min} = \min_{\{a_n\} \neq \{a'_n\}} \left[\sum_n d^2 \left(a_n, a'_n \right) \right]^{1/2} \tag{2.3}$$

where $d(a_n, a'_n)$ denotes the ED between channel signals a_n and a'_n.

If soft-decision ML decoding is used, the error-event probability will asymptotically approach the lower bound [19] at high signal-to-noise ratio.

$$\Pr(e) \geq N(d_{\min}) Q(d_{\min} / 2\sigma_n) \tag{2.4}$$

where $N(d_{\min})$ denotes the average number of error events with distance $d_{\min}$, and $Q(\cdot)$ is the Gaussian error probability function. σ_n^2 is the variance of the channel noise.

Therefore, assigning signal points to the coded bits (or the state transitions in a trellis codes) in a way that maximizes the minimum ED is the key to improve system performance. Ungerbock [12] devised an assignment procedure called "mapping by set partitioning", which can always make the maximum miminum ED larger than the minimum distance between signal points in an uncoded system with the same data rate and average power; i.e., it can always get coding gain. The coding gain can be calculated as [32]

$$G = \frac{(d_{\min}^2 / S_{av})_{coded}}{(d_{\min}^2 / S_{av})_{uncoded}} \tag{2.5}$$

where S_{av} is the average signal power.

"Mapping by set partitioning" first successively partitions a channel signal constellation into subsets with increasing minimum distance between symbols in the subsets (see Figure 2.6), and then maps the signal constellation to code bits according to the following rules: All channel symbols are assigned with equal frequency and symmetry.

1. All parallel transitions in the trellis structure are assigned the maximum possible ED.
2. All transitions diverging from or merging into a trellis state are assigned the next maximum possible ED separation.

Applying the symbol assignment rules to an 8-state trellis as shown in Figure 2.7, gives $(d_{\min})_{coded} = 2.141$. Taking uncoded 4-PSK as reference, $(d_{\min})_{uncoded} = 1.414$. Here, the coding gain for 2/3 rate coded 8-PSK is 3.6 dB.

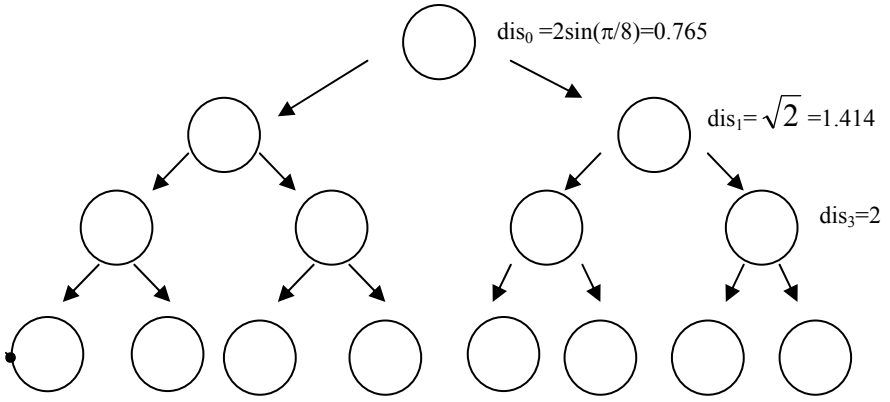


Figure 2.6 A partition of an 8-PSK signal constellation into subsets

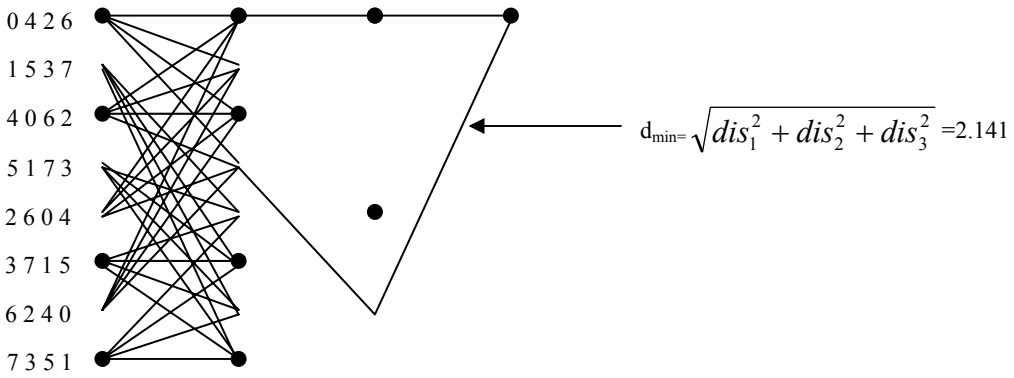


Figure 2.7 Symbol assignment for 8-PSK modulation

2.3.2 CTCM with Permuted State Structure

CTCM with permuted state structure [9] [10] [11] was investigated recently for power-limited spread spectrum channels. The goal is to achieve coding gain, processing gain [1], and power efficiency at the same time. Orthogonal and bi-orthogonal signal constellations were researched, but they are not optimal for power limited channels. A

simplex signal set achieves the same error probability as an equally likely orthogonal set while using the minimum energy. It is the optimum signal set to achieve power efficiency [1] [33]. A simplex is a set of M signals in an N-dimensional space with $N \geq M-1$, satisfying that for all $i, j \in \{1,2,...,M\}$, the cross-correlation between signals s_i and s_j is

$$s_i \bullet s_j = \begin{cases} 1, & i = j \\ \dfrac{-1}{M-1}, & i \neq j \end{cases}, \tag{2.6}$$

for normalized signal energy. It can be verified that these M signals have equal distance from each other.

CTCM using simplex signaling was explored for very small trellises [14]. But the way to build the signal constellation and the symbol assignment are not systematic yet. As a result, the distance property of these codes cannot be analyzed systematically. Efficient decoding was not investigated for these codes before. Those are the problems that will be solved in this dissertation research.

2.4 Decoding of TCM

There are several efficient decoding algorithms developed for conventional TCM, such as sequential decoding proposed by Wozencraft [18] in 1961, threshold decoding introduced by Massey [17] in 1963, Viterbi decoding proposed by Viterbi [15] in 1967 and BCJR algorithm propose by L. R. Bahl *et al*. in 1974 [21]. Among them, the most frequently used in practice is Viterbi algorithm, which has been proved to be a dynamic ML decoding algorithm. It is optimal in the sense that it minimizes the probability of code word error. Another optimal one is BCJR algorithm, which minimize the probability of symbol (or bit) error. These two algorithms are discussed below.

2.4.1 Viterbi Algorithm

For a conventional (k, r, K) convolutional encoder, denote the information sequence containing L k-tuple information symbols as $U = (U_1, U_2, ..., U_t, ..., U_L)$. U_i belongs to the finite information symbol alphabet. The corresponding code sequence is denoted as $V = (V_1, V_2, ..., V_t, ..., V_{L+K})$. V_i is a code symbol the encoder can output. In coded modulation scheme, it is a channel symbol assigned to the state transition occurring at stage i. Suppose the sequence $R = (R_1, R_2, ..., R_t, ..., R_{L+K})$ is received. The

decoder produces an estimate $\hat{V}$ based on the observation of R. To minimize the code word error, the decoder choose $\hat{V}$ such that the probability of $\hat{V}$ being sent, given the received sequence R, is maximized among all possible V's, i.e., [1, see Appendix B]

$$P(\hat{V} \mid R) = \max_{\text{all V}} P(V \mid R) \tag{2.7}$$

If all V's are equally likely distributed, (2.3.1) can be replaced by

$$P(R \mid \hat{V}) = \max_{\text{all V}} P(R \mid V) \tag{2.8}$$

This is called maximum likelihood (ML) decoding criterion. The decoder selects $\hat{V}$ among all possible V's such that $P(R \mid \hat{V})$ is maximized.

$P(R \mid \hat{V})$ can be computed from channel specification. In a case of discrete memoryless channel, we have

$$P(R \mid V) = \prod_{j=1}^{L+m} P(R_j \mid V_j) \tag{2.9}$$

The likelihood function $P(R \mid V)$ is called the path metric associated with the path V. $P(R_j \mid V_j)$ called the branch metric. For AWGN channels one-sided noise power spectrum density N_0,

$$P(R \mid V) = \frac{1}{\sqrt{\pi N_0}} e^{-\frac{\|R-V\|^2}{N_0}}, \tag{2.10}$$

where $\|R - V\|^2$ is the squared ED between R and V.

This is equivalent to choosing one V that has the minimum ED to R.

Viterbi algorithm reduced the number of comparisons among all the possible V's by removing those trellis paths that could not possibly be considered for the maximum likelihood choices at each trellis stage. Only the path with the largest metrics entering each state is stored at each stage. This path is called the survivor. The final survivor is the decoded path.

For TCM with a (k, r, K) convolutional encoder, the basic steps in Viterbi algorithm are

1. Beginning at trellis stage $j = K$, compute the metric for single path entering each state. Store the path and its metric for each state.

2. Increase j by 1. For each state, compute the path metrics for all the paths entering a state by adding the branch metric entering that state to the metric of the connecting survivor at the preceding trellis stage. Store the path with the largest metric (the survivor), together with its metric, and eliminate all other paths.
3. If $j < L + K$, repeat step 2. Otherwise, stop.

2.4.2 BCJR algorithm

The BCJR algorithm is restated here without derivation. The source is assumed to be a discrete-time finite-state Markov process. The S distinct states of the Markov source are indexed by the integer m, $m = 0,1,...,S-1$. The state of the source at time t is denoted by S_t and its output by X_t. A state transition sequence extending from time t to t' is denoted by $S_t^{t'} = S_t, S_{t+1}, ..., S_{t'}$, and the corresponding code sequence is $X_t^{t'} = X_t, X_{t+1}, ..., X_{t'}$, where X_t belongs to some finite discrete alphabet. In coded modulation scheme, it is the channel symbol assigned to state transition $S_{t-1} \to S_t$. The Markov source starts in the initial state S_0 and produces an output sequence X_1^L, ending in the terminal state S_L. X_1^L is the input to a noisy DMC channel whose output is the sequence $Y_1^L = Y_1, Y_2, ..., Y_L$. For all $1 \le t \le L$,

$$P(Y_1^t \mid X_1^t) = \prod_{j=1}^{t} P(Y_j \mid X_j) \tag{2.11}$$

where $P(Y_j \mid X_j)$ is the transition probability of this DMC.

The objective of BCJR decoder is to examine Y_1^L and estimate the a posterior probability (APP) of the states and state transitions of the Markov source, i.e., the conditional probabilities

$$P(S_t = m \mid Y_1^L) = P(S_t = m,\ Y_1^L) / P(Y_1^L) \tag{2.12}$$

and

$$P(S_{t-1} = m', S_t = m \mid Y_1^L) = P(S_{t-1} = m', S_t = m, Y_1^L) / P(Y_1^L) \tag{2.13}$$

In reality, BCJR calculates the joint probabilities

$$\lambda_t(m) = P(S_t = m, Y_1^L) \tag{2.14}$$

and

$$\sigma_t(m', m) = P(S_{t-1} = m', S_t = m, Y_1^L) \tag{2.15}$$

$P(Y_1^L)$ is the probability that $\beta_t(m) = \sum_{m'} \beta_{t+1}(m')\gamma_{t+1}(m, m')$ is observed. For a given Y_1^L, $P(Y_1^L)$ is a constant. Dividing Equations (2.15) and (2.14) by this constant which is available from the decoder will always give Equations (2.12) and (2.13). Alternatively, we can normalize $\lambda_t(m)$ and $\sigma_t(m', m)$ to obtain the same result.

Three sets of probabilities are defined as follows:

$$\alpha_t(m) = P(S_t = m, Y_1^t) \tag{2.16}$$

$$\beta_t(m) = P(Y_{t+1}^L \mid S_t = m) \tag{2.17}$$

$$\gamma_t(m', m) = P(S_t = m, Y_t \mid S_{t-1} = m') \tag{2.18}$$

Then

$$\lambda_t(m) = \alpha_t(m)\beta_t(m) \tag{2.19}$$

$$\sigma_t(m', m) = \alpha_{t-1}(m')\gamma_t(m', m)\beta_t(m) \tag{2.20}$$

And $\alpha_t(m)$ and $\beta_t(m)$ can be formed by forward recursion in Equation (2.21) and backward recursion in Equation (2.22).

$$\alpha_t(m) = \sum_{m'} \alpha_{t-1}(m')\gamma_t(m', m) \qquad t = 1, \ldots, L \tag{2.21}$$

$$\beta_t(m) = \sum_{m'} \beta_{t+1}(m')\gamma_{t+1}(m, m') \qquad t = L-1, \ldots, 0 \tag{2.22}$$

$\gamma_t(m', m)$ can be obtained from Markov data source and channel property.

$$\begin{aligned} \gamma_t(m', m) &= P(Y_t, S_{t-1} = m' \to S_t = m) \\ &= \sum_X P(S_t = m \mid S_{t-1} = m') \cdot P(X_t = X \mid S_{t-1} = m', S_t = m) \cdot P(Y_t \mid X) \end{aligned} \tag{2.23}$$

From the λ_t or σ_t

$$P(Y_1^L) = \sum_m \lambda_t(m) = \sum_{m,m'} \sigma_t(m, m') \tag{2.24}$$

For the recursions in Equations (2.21) and (2.22) to apply at $t = 1$ and $t = L - 1$ respectively, $\alpha_0(m)$ must be the probability of the encoder starting in state m before stage 1 and $\beta_L(m)$ the probability of the encoder ending in state m after stage L. The standard application of the BCJR algorithm assumes $S_0 = S_L = 0$, so

$$\alpha_0(0) = 1 \text{ and } \alpha_0(m) = 0, \text{ for } m \neq 0 \tag{2.25}$$

and

$$\beta_L(0) = 1 \text{ and } \beta_L(m) = 0, \text{ for } m \neq 0. \tag{2.26}$$

The operations of the decoder for computing λ_t and σ_t are outlined below:

1. $\alpha_0(m)$ and $\beta_L(m)$ are initialized according to Equations (2.15) and (2.16).
2. As soon as Y_t is received, the decoder computes $\gamma_t(m', m)$ using Equation (2.23) and $\alpha_t(m)$ using Equation (2.21) for all t and m.
3. After the complete sequence Y_1^L has been received, the decoder recursively computes β_t using Equation (2.22). Then compute $\lambda_t(m)$ and $\sigma_t(m', m)$ using Equation (2.19) and (2.20).

After Equations (2.14) and (2.15) are obtained, the probability of any event that is a function of states or state transitions can be obtained by summing the appropriate $\lambda_t(m)$ or $\sigma_t(m', m)$. Then decode the event as the one having the largest probability.

2.4.3 Circular BCJR Algorithm

BCJR algorithm is much less popular than Viterbi algorithm and is almost never applied in practical systems. The reason for this is that it yields performance in terms of symbol error probability only slightly superior to the Viterbi algorithm, yet it presents a much higher complexity. Both Viterbi and BCJR algorithm require the starting state or the distribution of the starting state be known *a priori*. This is not the case for circular trellis coding. The optimum ML decoding for circular trellis is to run the Viterbi algorithm the number of total states times, each for one possible starting state. This is obviously not desirable for large trellis. Hence, BCJR is investigated again to be used in this case. First, the probability of the distribution of starting state needs to be solved.

John B. Anderson [22] extended BCJR algorithm to tail biting codes. We refer to it as circular BCJR algorithm. Rewrite BCJR in vector and matrix format. Define α_t as a row vector and β_t as a column vector. Each of them has S elements defined in Equations (2.16) and (2.17), respectively. Define γ_t as an $S \times S$ matrix with each element defined in Equation (2.18). For tailing biting codes,

$$S_0 = S_L, \tag{2.27}$$

then [22] shows that

$$\alpha_0 = \frac{\alpha_0 \gamma_1 ... \gamma_L}{P(Y_1^L)} \tag{2.28}$$

This means α_0 is the normalized left eigenvector of the matrix $\gamma_1 ... \gamma_L$ corresponding to the largest eigenvalue. Similarly, the right eigenvector of $\gamma_1 ... \gamma_L$ is the proper starting vector for β_L in most applications. [22] employs successive normalization to control the precision of the calculation without changing the results. This augment depends on the fact that scaling of any or all of the α_t, β_t or γ_t during recursions in Equations (2.21) and (2.22) simply leads to scaled $\lambda_t(m)$ and $\sigma_t(m', m)$ in Equations (2.19) and (2.20), but Equations (2.12) and (2.13) can always be obtained by normalizing $\lambda_t(m)$ and $\sigma_t(m', m)$ to have unit sum. This circular BCJR is referred to as circular BCJR with eigenvectors. Using superscript "*o*" to indicate a unit-sum vector, the basic steps are

Given $Y_1^L = Y_1, Y_2, ..., Y_L$

1. Find the left eigenvector corresponding to the largest eigenvalue of $\gamma_1 ... \gamma_L$. This is α_0.

2. Calculate the normalized α's by forward recursion

$$\alpha_t = \alpha_{t-1}^o \gamma_t \tag{2.29}$$

$$\alpha_t^o = \alpha_t / \sum_i \alpha_t(i), \; t = 1, ..., L. \tag{2.30}$$

4. Starting from the right eigenvector of $\gamma_1 ... \gamma_L$, form the normalized β's by

$$\beta_t = \gamma_{t+1} \beta_{t+1}^o \tag{2.31}$$

$$\beta_t^o = \beta_t / \sum_i \beta_t(i), \qquad t = L-1, ..., 0 \tag{2.32}$$

4. Calculate the normalized λ_t and $\sigma_t(m', m)$ by

$$\lambda_t(m) = \alpha_t^0(m) \beta_t^0(m), \tag{2.33}$$

$$\lambda_t^0(m) = \lambda_t(m) / \sum_i \lambda_t(i) \tag{2.34}$$

$$\sigma_t(m', m) = \alpha_{t-1}^o(m') \gamma_t(m', m) \beta_t^o(m), \tag{2.35}$$

$$\sigma_t^o = \sigma_t(m', m) / \sum_{m', m} \sigma_t(m', m) \qquad t = 1, ..., L \tag{2.36}$$

[22] also argues that if we iterate the forward recursion enough times and properly normalize the outcomes, then the resulting sequence of outcomes $\alpha_1 \ldots \alpha_L, \alpha_{L+1} \ldots \alpha_{2L}, \alpha_{2L+1} \ldots$ will converge to repetitions of $\alpha_1^o \ldots \alpha_L^o$, where $\alpha_1^o \ldots \alpha_L^o$ is the normalized α obtained from circular BCJR with eigenvectors . The speed of convergence (the number of iterations required by the forward recursion) is affected by the choice of initial α_0. Similar argument holds for backward recursion. This gives rise an iterative circular BCJR as described below.

Given $Y_1^L = Y_1, Y_2, ..., Y_L$

1. Do not calculate the eigenvectors of $\gamma_1 \ldots \gamma_L$. Set initial α_0 in some way.
2. Calculate a set of normalized $\alpha_1^o \ldots \alpha_L^o$ by the recursion in Equation (2.29) and (2.30), continues this recursion to find α_t^o, $t = L+1, L+2, \ldots$. Take $\gamma_t = \gamma_l$ where $l = t \bmod L$. When $\left\| \alpha_t^o - \alpha_l^o \right\|$ is sufficiently small by a suitable measure, stop. A complete set of α is available in the last L round.
3. Execute a similar procedure backward along the trellis circle, to find the set $\beta_1^o \ldots \beta_L^o$. Set $\beta_L = \alpha_0$ initially.
4. Same as in circular BCJR with eigenvectors.

2.5 Spread Spectrum and HDCTCM

Spread spectrum (SS) communication system is a system satisfying two criteria [34]. First, the bandwidth of the transmitted signal must be much larger than the message bandwidth. Second, this bandwidth must be determined by some function that is independent of the message, known as PSEUDONOISE (PN). The two most prevalent types of SS are direct sequence (SS) and Frequency Hopping (FH). By using PN to shift the phase of message in DS or to select the carrier frequency in FH pseudo randomly, the effect of spreading the message spectrum over a larger frequency range is achieved. Both systems can be coupled with HDCTCM for transmission over a power-limited channel. The coding gain and processing gain can be achieved at the same time.

CHAPTER 3 DESIGN OF HDCTCM

The design of a trellis-coded modulation involves the state transition table construction for the encoder, the design of the signal constellation, and mapping rules between the symbols in the signal constellation and the state transitions. HDCTCM is designed for power-limited spread spectrum channels. The encoder employs the circular trellis coding with permuted state structure. The state transition table is built as described in Chapter 2. This chapter develops a systematic way to design a high dimensional simplex signal constellation and to do the symbol assignment. In Section 3.1, we give the design criterion. In Section 3.2, we apply this criterion to HDCTCM and build a matrix to record all sets of state transitions that should be assigned simplexes. In Section 3.3, the simplex signal constellation and the procedure of symbol assignment are designed.

Uniformity is a desirable property for a coding scheme [35]. Analysis on error performance of this code will be greatly simplified under this condition. In Section 3.4 we prove the uniformity of HDCTCM.

3.1 Design Criterion

The goal of designing a coding scheme is to obtain coding gain, i.e., to get reduced error probability compared with the uncoded system at a specific signal-to-noise ratio. For a trellis coded modulation scheme, suppose a particular code sequence (trellis path) is transmitted, then the decoder will select one code sequence that has the minimum distance to the received sequence in order to minimize the code word error. Denote the probability of making an incorrect decision on the code sequence as $\Pr(e)$. For AWGN channel with noise variance σ_n^2, $\Pr(e)$ will asymptotically approach the lower bound at high signal to noise ratio [19]

$$\Pr(e) \geq N(d_{\min}) Q(d_{\min} / 2\sigma_n) \tag{3.1}$$

where $d_{\min}$ is defined as the minimum Euclidean distance (ED) between all pairs of code sequences, $N(d_{\min})$ denotes the number of paths having distance $d_{\min}$, and $Q(\cdot)$ is the Gaussian error probability function.

We can see the minimum distance of the code is of primary importance to its error performance. Usually, $d_{\min}$ will correspond to the minimum distance produced by a path that splits and remerges with the transmitted path, since we would expect paths that do not remerge to keep accumulating distance. This path is called the error event of this transmitted path. And the distance from this error event to the transmitted path is called free distance, denoted as d_{free}. In order to reduce $\Pr(e)$, we want to maximize d_{free}. From Equation (3.1), we also desire that the number of code sequences having distance d_{free} be small. The symbol error probability, denoted as $\Pr(s)$, is usually of more interest than the code word probability $\Pr(e)$, and is defined as the ratio of the number of symbol errors over the symbols in the information sequence. In order to minimize the symbol error, we also want the information sequence corresponding to the decode path to have a small number of symbols differing from the information sequence corresponding to the transmitted path. This is to say, we want the unmerged stages between the error event and the transmitted path as small as possible. The error event having the minimum unmerged stage with the transmitted sequence is called the minimum error event. For a coding scheme with uniformity, taking any code sequence as the transmitted sequence will give the same arguments as above.

3.2 Build State Transition Matrix

In this section, by analyzing the error events in HDCTM we identify sets of state transitions that should be assigned simplexes. Then a multi-dimensional matrix is designed to record them. First, the butterfly structure of trellis codes is given to facilitate the discussion.

3.2.1 Butterfly Structure in HDCTCM

We denote HDCTCM that employs a circular trellis $\mathrm{T}(n, D)$ with permuted state structure as $\mathrm{HDCTCM}(n, D)$. n and D are originally defined in Section 2.2. For HDCTCM, n is the size of information symbol alphabet and D is the trellis depth. Some

other related system parameters are information sequence length L, the number of space dimensions N of the signal constellation we will build, and the total states $S = n^D$.

For all trellis codes, we can always identify sets of p originating states and sets of q next states that have complete intra-connectivity, but no connectivity with other state sets. Such structure is called a (p,q) butterfly. The set of the originating states or the next states in a butterfly is named the originating state set or the next state set of that butterfly. In a butterfly, each of the p originating states has transitions to each of the q next states--the total number of $p \cdot q$ transitions in a butterfly. In HDCTCM(n, D), the state transition table built using the method described in Section 2.2 shows (n,n) butterflies. Since each state can be a member of one and only one butterfly's originating states set and one and only one butterfly's next states set, there are a total number of $S / n = n^{D-1}$ butterflies representing all the $n^{D-1} \cdot n^2 = n^{D+1}$ state transitions in the state transition table.

Example 3.1 Take HDCTCM$(4,3)$ as an example. The state transition table is shown in Table 3.1. The states are numbered by integer from 1 to 64. It can be found that any member of the set of states $\{1, 64, 4, 33\}$ can transit to state 1, 2, 5, and 34. Then the set of states $\{1, 64, 4, 33\}$ and the set of states $\{1, 2, 5, 34\}$ build a butterfly; they are the originating states set and the next states set of this butterfly, respectively. We write this butterfly as butterfly $\{\{1, 64, 4, 33\}, \{1, 64, 4, 33\}\}$. All 16 butterflies can be identified and are illustrated in Figure 3.1, where each box represents a butterfly. The originating state set and next state set are in the left side and right side of a butterfly, respectively. By this representation, we regard a butterfly as an ordered structure and refer a particular arrangement of the states in a butterfly as the internal order of a butterfly. The location of the state (or transition) in a butterfly is referred to as the internal location of the state (or transition). The internal location of an originating state or a next state from top to bottom in a butterfly is 1, 2, 3, and 4. Number the butterflies arbitrarily by integer from 1 to 16.

Table 3.1 State Transition Table for HDCTCM (4,3)

State	Input 0	Input 1	Input 2	Input 3
1	1	2	5	34
2	3	8	15	37
3	4	14	10	50
4	5	34	1	2
5	6	26	11	47
6	7	64	17	25
7	8	3	37	15
8	9	28	29	20
9	10	50	4	14
10	11	47	6	26
11	12	63	31	62
12	13	27	53	55
13	14	4	50	10
14	15	37	3	8
15	16	54	30	43
16	17	25	7	64
17	18	35	40	51
18	19	49	57	42
19	20	29	28	9
20	21	58	38	56
21	22	61	52	24
22	23	44	32	48
23	24	52	61	22
24	25	17	64	7
25	26	6	47	11
26	27	13	55	53
27	28	9	20	29
28	29	20	9	28
29	30	43	16	54
30	31	62	12	63
31	32	48	23	44
32	33	36	46	39

State	Input 0	Input 1	Input 2	Input 3
33	34	5	2	1
34	35	18	51	40
35	36	33	39	46
36	37	15	8	3
37	38	56	21	58
38	39	46	36	33
39	40	51	18	35
40	41	45	59	60
41	42	57	49	19
42	43	30	54	16
43	44	23	48	32
44	45	41	60	59
45	46	39	33	36
46	47	11	26	6
47	48	32	44	23
48	49	19	42	57
49	50	10	14	4
50	51	40	35	18
51	52	24	22	61
52	53	55	13	27
53	54	16	43	30
54	55	53	27	13
55	56	38	58	21
56	57	42	19	49
57	58	21	56	38
58	59	60	41	45
59	60	59	45	41
60	61	22	24	52
61	62	31	63	12
62	63	12	62	31
63	64	7	25	17
64	2	1	34	5

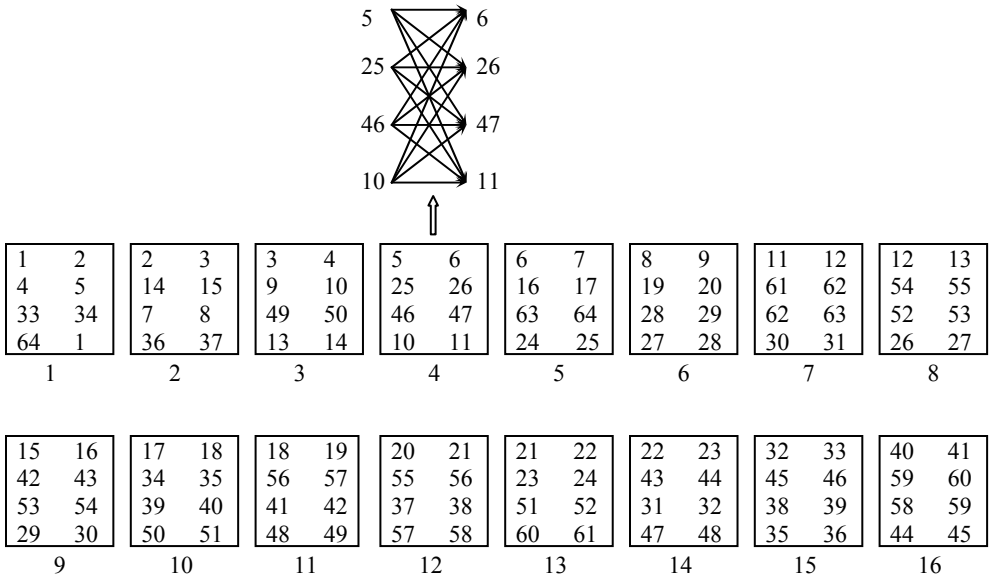


Figure 3.1 Butterflies in HDCTCM (4,3)

3.2.2 Minimum Error Events and Simplex-Transitions

Let us consider the error events. As shown in Section 2.2, for a circular trellis $\mathrm{T}(n, D)$ with permuted state structure, the smallest information sequence length that can be employed is $L = D + 1$. The minimum error events occur in this situation. There are a total number of n^{D+1} paths in the trellis. Each starting state associates with n paths diverging at the beginning and remerging at stage $D + 1$ (see Figure 3.2). Given that each state transition occurs equally likely and there is no better knowledge to differentiate these n paths from each other, we will make these n paths equally distant from each other in our design. Take one of them as the transmitted path; the minimum error events are the other $n - 1$ paths. In order to make d_{free} as large as possible and achieve energy efficiency in power-limited channel, at each trellis stage the n state transitions on these n paths should be assigned a simplex. We name the set of such n state transitions as a simplex-transition. A simplex-transition at stage i is denoted as simplex-transition$_i$, $i = 1, 2, \ldots, D + 1$ (see Figure 3.2).

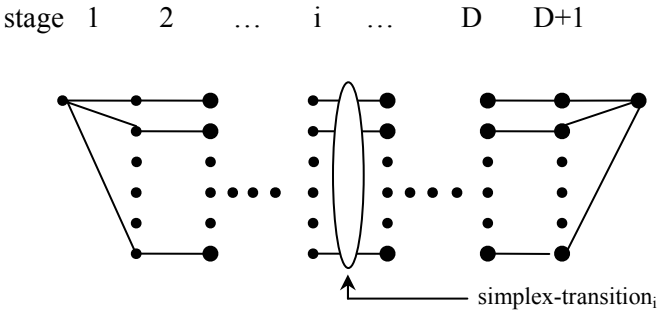


Figure 3.2 The n paths associated with a certain starting state for HDCTCM(n, D) with $L = D + 1$

For each stage i, the n paths associated with a certain starting state define a simplex-transition$_i$; then, for all the possible starting states, there is a total number of n^D simplex-transition$_i$; for each i, $i = 1, 2, \ldots, D+1$. A particular state transition appears exactly once in a simplex-transition$_i$; for all i, $i = 1, 2, \ldots, D+1$.

The set of originating states or next states of the n transitions in a simplex-transition is called the originating state set or next state set of this simplex-transition.

Example 3.2 For HDCTCM$(4, 3)$, Appendix A gives all the paths at $L = D + 1 = 4$. The set of 4 state transitions at stage i on the paths associated with a starting state constitute a simplex-transition$_i$; for each i, $i = 1, 2, 3, 4$. There are 64 simplex-transition$_i$'s; for each i, $i = 1, 2, 3, 4$.

For instance, on the 4 paths associated with starting state 3, the set of state transitions at stage 2 constitute a simplex-transition$_2$ denoted as simplex-transition$_2\{4 \to 1,\ 14 \to 3,\ 10 \to 6,\ 50 \to 35\}$. For a particular state transition--say $2 \to 3$--it can be verified that it appears exactly once in

one simplex-transition$_1\{2 \to 3,\ 2 \to 8,\ 2 \to 15,\ 2 \to 37\}$,

one simplex-transition$_2\{1 \to 1,\ 2 \to 3,\ 5 \to 6,\ 34 \to 35\}$,

one simplex-transition$_3\{26 \to 13,\ 18 \to 49,\ 2 \to 3,\ 8 \to 9\}$, and

one simplex-transition$_4$ $\{2 \to 3,\ 14 \to 3,\ 7 \to 3,\ 36 \to 3\}$.

3.2.3 Illustrate Simplex-Transitions in Butterfly Structure

In HDCTCM(n, D), all n^{D+1} distinct state transitions are represented in its n^{D-1} butterflies. So, we can analyze how the simplex-transitions are reflected in the butterflies. From Figure 3.2 we can see that the n transitions in a simplex-transition$_1$ have a same originating state, so they are contained in the butterfly whose originating state set contains this originating state. They are the state transitions from this originating state in this butterfly to each of all the next states in this butterfly. The n transitions in a simplex-transition$_{D+1}$ have a same next state and are, therefore, contained in the butterfly whose next state set contains this next state. They are the state transitions from each of all the originating states to this next state in this butterfly. Each butterfly contains n simplex-transition$_1$'s and n simplex-transition$_{D+1}$'s. A total number of n^{D-1} butterflies contain all the n^D simplex-transition$_1$'s and simplex-transition$_{D+1}$'s. For the n transitions in a simplex-transition$_i$, $2 \le i \le D$, each transition has a different originating state and a different next state. Compared with the butterflies, it can be seen that each transition is contained in a different butterfly. We say there are n butterflies associated with a simplex-transition$_i$.

Example 3.3 For HDCTCM (4,3), compare all the simplex-transition$_i$'s from Appendix A with all its 16 butterflies in Figure 3.1. For instance, on the 4 paths associated with starting state 3, the simplex-transition$_1$ $\{3 \to 4,\ 3 \to 14,\ 3 \to 10,\ 3 \to 50\}$, is contained in butterfly 3, as shown in Figure 3.3 (a). The simplex-transition$_4$ $\{2 \to 3,\ 14 \to 3,\ 7 \to 3,\ 36 \to 3\}$ is contained in butterfly 2, as shown in Figure 3.3 (b). The 4 transitions in simplex-transition$_2$ $\{4 \to 1,\ 14 \to 3,\ 10 \to 6,\ 50 \to 35\}$ are contained in butterfly 1, 2, 4, and 10, respectively as shown in Figure 3.3 (c). It can be verified that the other 15 transitions in each of the butterfly 1, 2, 4, and 10 constitute the other 15 simplex-transition$_2$'s.

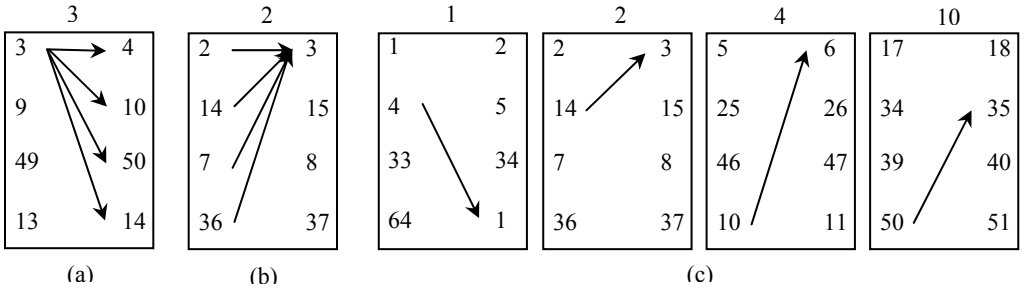


Figure 3.3 Illustrate simplex-transitions in butterflies for HDCTCM$(4,3)$

In fact, for HDCTCM(n, D), the following property is observed for a simplex-transition$_i$, $2 \le i \le D$.

Property 3.1 Consider two simplex-transition$_i$'s: if the originating state (or next state) of a transition in one simplex-transition$_i$ is the same as the originating state (or next state) of a transition in the other simplex-transition$_i$, then the originating state set (or next state set) of one simplex-transition$_i$ will be identical to that of the other simplex-transition$_i$.
From this property, we have the following statement.

Statement 3.1 For a simplex-transition$_i$, find the n butterflies associated with it. Then the other n^2-1 transitions in each of these n butterflies constitute the other n^2-1 simplex-transition$_i$'s.

Proof: Number these n butterflies associated with the first simplex-transition$_i$ as butterfly m, $m = 1,2,...,n$. The originating states and next states in butterfly m will be labeled using g_{mj} and r_{mj}, $j = 1,2,...,n$ as our proof continues.
Each of the n transitions in a simplex-transition$_i$ is contained in a different butterfly. So, for the first simplex-transition$_i$, we can label the originating state and next state of the

transition contained in butterfly m as g_{m1} and r_{m1}, and then this first simplex-transition$_i$ is $\{g_{m1} \to r_{m1} | m = 1,\ldots,n\}$ (see Figure 3.4 (a)).

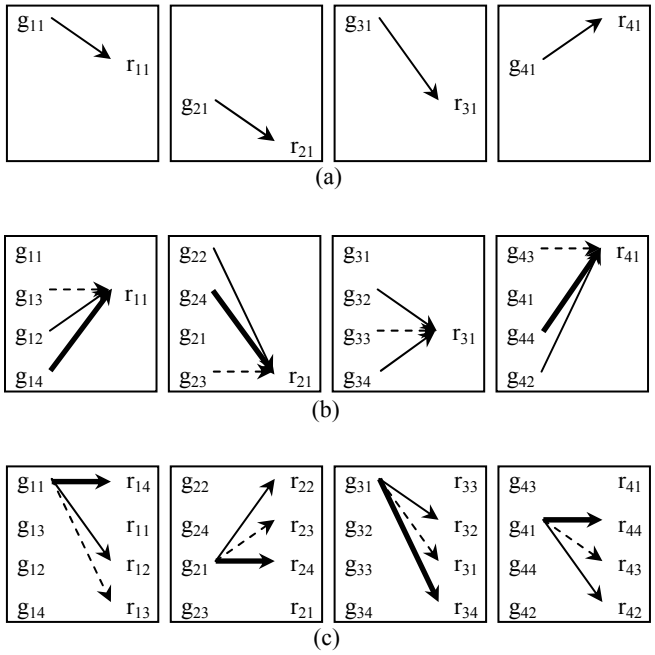


Figure 3.4 Simplex-transition$_i$'s in the four associated butterflies

Consider the other transitions in these n butterflies. For a particular butterfly m', $m' = 1,2,...,n$, there are other $n-1$ transitions having $r_{m'1}$ as next state. Label them as $g_{m'k} \to r_{m'1}$, $k = 2,3,...,n$ (see Figure 3.4 (b)). For each k, consider a new simplex-transition containing transition $g_{m'k} \to r_{m'1}$. The next state of transition $g_{m'k} \to r_{m'1}$ is $r_{m'1}$. $r_{m'1}$ is also the next state of the transition $g_{m1} \to r_{m1}$ in the first simplex-transition$_i$. From Property 3.1, the next state set of this new simplex-transition$_i$ will be identical to that of the first simplex-transition$_i$; it is the set of states $\{r_{m1} | m = 1,\ldots,n\}$. This indicates that this new simplex-transition$_i$ is contained in these n butterflies. For this new simplex-transition$_i$, we can label the transition contained in butterfly m, $m \neq m'$, as $g_{mk} \to r_{m1}$. Then this new simplex-transition$_i$ is $\{g_{mk} \to r_{m1} | m = 1,\ldots,n\}$.

Similarly, there are other $n-1$ transitions originating from $g_{m'1}$ in butterfly m'. Label them as $g_{m'1} \to r_{m'k}$, $k = 2,3,...,n$ (see Figure 3.4 (c)). For each k, a new simplex-transition containing the transition $g_{m'1} \to r_{m'k}$ will have the originating state set identical to that of the first simplex-transition$_i$. It is the set of states $\{g_{m1} | m = 1,\ldots,n\}$. This indicates that this new simplex-transition$_i$ is also contained in these n butterflies. For this new simplex-transition$_i$, we can label the transition contained in butterfly m, $m \neq m'$, as $g_{m1} \to r_{mk}$. Then this new simplex-transition$_i$ is $\{g_{m1} \to r_{mk} | m = 1,\ldots,n\}$.

So far, $(n-1)+(n-1) = 2n-2$ more transitions in each of these n butterflies have been considered and they constitute $2n-2$ new simplex-transition$_i$'s. All the originating and next states in these n butterflies have been labeled as g_{mj} and r_{mj}, $j = 1,2,...,n$.

The transitions yet to be considered in butterfly m' are $g_{m's} \to r_{m't}$, where $s,t = 2,3,...,n$. For each s and t, consider a new simplex-transition$_i$ containing the transition $g_{m's} \to r_{m't}$. From Property 3.1, it will have the originating state set identical to that of the simplex-transition$_i$ $\{g_{ms} \to r_{m1} | m = 1,\ldots,n\}$. It is the set of states $\{g_{ms} | m = 1,\ldots,n\}$. Also, it will have the next state set identical to that of the simplex-transition$_i$ $\{g_{m1} \to r_{mt} | m = 1,\ldots,n\}$. It is the set of states $\{r_{mt} | m = 1,\ldots,n\}$. This shows that the new simplex-transition$_i$ is $\{g_{ms} \to r_{mt} | m = 1,\ldots,n\}$. So, we have shown the remaining $(n-1)\cdot(n-1) = n^2 - 2n + 1$ unconsidered transitions in each butterfly also constitute the other $(n-1)\cdot(n-1) = n^2 - 2n + 1$ new simplex-transition$_i$'s. So far, we have shown that set of transitions $\{g_{mj} \to r_{mk} | m = 1,\ldots,n\}$ is a simplex-transition$_i$; for all j and k, $j,\ k = 1,2,...,n$.

In summary, we have proved that in the n butterflies associated with a simplex-transition$_i$, all the $n\cdot(n\cdot n)$ transitions will constitute a number of $n\cdot n$ simplex-transition$_i$'s. Later in this dissertation, we may say these n butterflies constitute simplex-transition$_i$. Also, we name a set of n butterflies constituting simplex-transition$_i$ as a simplex-transition$_i$-butterfly-set.

For each i, $2 \le i \le D$, there are number of n^D simplex-transition$_i$'s. The total number of n^{D-1} butterflies can be grouped into n^{D-2} sets of n butterflies constituting simplex-transition, each set containing n^2 simplex-transition$_i$'s. Any particular butterfly is a member of a set of n butterflies constituting simplex-transition$_i$ for all i, $2 \le i \le D$.

Example 3.4 For HDCTCM (4,3), all the butterflies and simplex-transitions are solved in Examples 3.1 and 3.2. We can identify the 4 simplex-transition$_2$-butterfly-sets. They are: butterfly$\{1, 2, 4, 10\}$, butterfly$\{3, 6, 9, 12\}$, butterfly$\{5, 8, 7, 14\}$, and butterfly$\{15, 11, 13, 16\}$; The 4 simplex-transition$_3$-butterfly-sets are butterfly$\{1, 3, 5, 15\}$, butterfly$\{2, 6, 8, 11\}$, butterfly$\{4, 9, 7, 13\}$, and butterfly$\{10, 12, 14, 16\}$.

Note that the butterflies and all the paths at $L = D+1$ are completely determined by the state transition table, so all sets of n transitions constituting simplex-transition$_i$'s where $i = 1,2,...,D+1$ and all sets of n butterflies constituting simplex-transition$_i$'s where $i = 2,3,...,D$ are fixed in HDCTCM for a given n and D.

3.2.4 Arrange the Internal Orders of Butterflies in a Simplex-Transitioni-Butterfly-Set

In the next two sections, we will build a convenient structure to record all the simplex-transitions (butterflies). The proof of Statement 3.1 has shown that in a simplex-transition$_i$-butterfly-set, by proper labeling the set of transitions $\{g_{mj} \to r_{mk} | m = 1,\ldots,n\}$ constitute a simplex-transition$_i$ for all j and k, $j, k = 1,2,...,n$. Since we have represented a butterfly as an ordered structure, if we place $g_{m'j}$ in butterflies m' the same internal location as $g_{m''j}$ in butterfly m'', m', $m'' = 1,2,...,n$, then the set of n transitions constituting a simplex-transition$_i$ will be in the same internal location of these n butterflies. The steps to do this are described below in Procedure 3.1.

Procedure 3.1 Arrange the Internal Orders of butterflies in a simplex-transition$_i$-butterfly-set, $i = 2,3,...,D$.

Number these n butterflies as butterfly m, $m = 1,2,...,n$. The originating states and next states in butterfly m will be labeled using g_{mj} and r_{mj}, $j = 1,2,...,n$.

1. Select any one of these n butterflies, say butterfly m'. Arrange the originating states and next states in the left side and right side in this butterfly arbitrarily. Label them as $g_{m'j}$ and $r_{m'j}$ arbitrarily, $j = 1,2,...,n$.
2. For each k, $k = 1,2,...,n$, identify the simplex-transition$_i$ containing transition $g_{m'k} \to r_{m'1}$. For this simplex-transition$_i$, we can label the transition contained in butterfly m, $m \neq m'$, as $g_{mk} \to r_{m1}$. Place g_{mk} and r_{m1} in butterfly m in the same internal location as $g_{m'k}$ and $r_{m'1}$ in butterfly m'.
3. For each k, $k = 2,3,...,n$, identify the simplex-transition$_i$ containing transition $g_{m'1} \to r_{m'k}$. For this particular simplex-transition$_i$, we can label the transition in this simplex-transition contained in butterfly m, $m \neq m'$, as $g_{m1} \to r_{mk}$. Place r_{mk} in butterfly m in the same internal location as $r_{m'k}$ in butterfly m'.

Following the proof of Statement 3.1, it can be verified that all the other set of n transitions $\{g_{ms} \to r_{mt} | m = 1,\ldots,n\}$, for all s and t, $s, t = 2,3,...,n$ constitute simplex-transition$_i$, and are in the same internal location of these n butterflies.

From this procedure, we can see one of the n butterflies' internal orders will fix all the other $n-1$ butterflies' internal orders in a simplex-transition$_i$-butterfly-set.

Example 3.5 For HDCTCM (4,3), from Example 3.4, we know butterfly {1, 2, 4, 10} is a simplex-transition$_2$-butterfly-set. We will arrange the internal order of them according to Procedure 3.1.

1. First choose any one of these four butterflies--say butterfly 2--and arrange the states in an arbitrary way (see Figure 3.5 (a)).

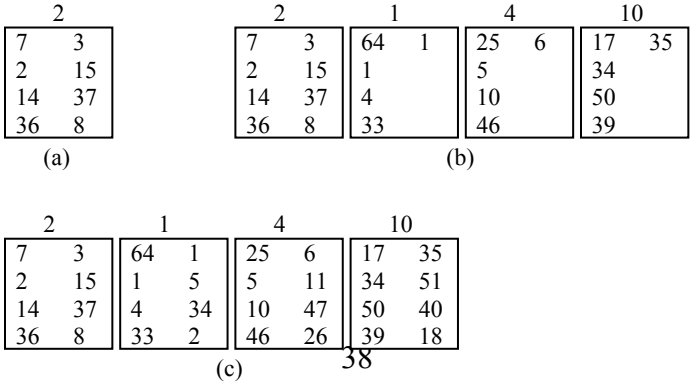


Figure 3.5 Arrange the internal orders of the butterflies in a simplex-transition$_2$-butterfly-set for HDCTM(4,3)

2. For each of the transitions $7 \to 3$, $2 \to 3$, $14 \to 3$ and $36 \to 3$, identify the simplex-transtition$_2$ containing this transition from Appendix A; they are

   simplex-transition$_2\{7 \to 3,\ 64 \to 1,\ 25 \to 6,\ 17 \to 35\}$,

   simplex-transition$_2\{2 \to 3,\ 1 \to 1,\ 5 \to 6,\ 34 \to 35\}$,

   simplex-transition$_2\{14 \to 3,\ 4 \to 1,\ 10 \to 6,\ 50 \to 35\}$ and

   simplex-transition$_2\{36 \to 3,\ 33 \to 1,\ 46 \to 6,\ 39 \to 35\}$.

   Then, the originating states 64, 1, 4, 33 and next state 1 should be placed in the same internal location in butterfly 1 as originating state 7, 2, 14, 36 and next state 3 in butterfly 2. Similarly, the locations of the originating states and one next state of butterflies 4 and 10 will be fixed (see Figure 3.5 (b)).

3. For each of the transitions $7 \to 15$, $7 \to 37$ and $7 \to 8$, the simplex-transition$_2$ containing this transition is

   simplex-transition$_2\{7 \to 15,\ 64 \to 5,\ 25 \to 11,\ 17 \to 51\}$,

   simplex-transition$_2\{7 \to 37,\ 64 \to 34,\ 25 \to 47,\ 17 \to 40\}$ and

   simplex-transition$_2\{7 \to 8,\ 64 \to 2,\ 25 \to 26,\ 17 \to 18\}$.

   Then, in butterfly 1, the next state 5, 34, 2 should be placed in the same internal location as the next state 15, 37, 8 in butterfly 2. Similarly, the locations of the three other next states of butterfly 4 and 10 will be fixed (see Figure 3.5 (c)).

3.2.5 Build State Transition Matrix

We have seen that for HDCTCM(n, D), a total number of n^{D-1} butterflies exist. Simplex-transition$_1$ and simplex-transition$_{D+1}$ are within a butterfly. Simplex-transition$_i$, where $2 \le i \le D$ are constituted by sets of n butterflies. We will regard a butterfly as a unit element with ordered structure, and place all the butterflies in a $D-1$ dimensional matrix with n location indexes along each matrix dimension. Name this matrix as state

transition matrix (STM). A location in STM is referred to using $\text{STM}(x_1,x_2,\ldots,x_i,...,x_{D-1})$, where x_k is the location index along the k_{th} dimension of STM, $x_k \in \{1,2,...,n\}$, $k = 1,2,...,D-1$. When $x_k =:$, it means x_k can take any possible value. $\text{STM}(x_1,x_2,\ldots,x_{k-1},:,x_{k+1},...,x_{D-1})$ will represent the n locations along the k_{th} dimension. These n locations have the same location index x_j along the j_{th} dimension, where $j \neq k$. When $x_k =: /m$, where $m \in \{1,2,...,n\}$, it means x_k can take any possible value except m.

Since any particular butterfly is a member of a set of n butterflies constituting simplex-transition$_i$, for all i, $2 \le i \le D$. We will design a way to place all the butterflies into STM such that the set of n butterflies constituting simplex-transition$_i$ are placed in the n locations along the $(i-1)_{th}$ dimension of STM, i.e., STM $(x_1,x_2,\ldots,x_{i-2},:,x_i,...,x_{D-1})$, for all i, $2 \le i \le D$. There are n^{D-2} possible combinations for x_j, $j \neq i-1$. A total number of n^{D-2} sets of n locations along the $(i-1)_{th}$ dimension are used for placing all the n^{D-2} sets of n butterflies constituting simplex-transition$_i$ for all i, $2 \le i \le D$. Also, the internal orders of the set of n butterflies constituting simplex-transition$_i$, where $2 \le i \le D$, can be arranged according to Procedure 3.1 such that the n transitions constituting a simplex-transition$_i$ are placed in the same internal location of these n butterflies. Then finally, in STM, a simplex-transition$_i$, $2 \le i \le D$, will be the n transitions located in the same internal locations of a set of n butterflies along the $(i-1)_{th}$ dimension.

Note that Procedure 3.1 only arranges the internal orders of the set of n butterflies constituting simplex-transition$_i$. When placing these n butterflies in $\text{STM}(x_1,x_2,\ldots,x_{i-2},:,x_i,...,x_{D-1})$, the order of placing these n butterflies in these locations is not fixed in Procedure 3.1. We will see shortly in the procedure of building STM that the first set of n butterflies constituting simplex-transition$_i$ can be placed into $\text{STM}(x_1,x_2,\ldots,x_{i-2},:,x_i,...,x_{D-1})$ in an arbitrary order, but a particular choice on this order will fix the order of placing all the other sets of n butterflies constituting simplex-transition$_i$ along the $(i-1)_{th}$ dimension of STM. We will define a choice set called STM-FREE, whose member elements indicate a particular choice made for arranging the internal orders or the order of placing the n butterflies constituting simplex-transition$_i$

along the $(i-1)_{th}$ dimension when building STM. The procedure of building STM is stated as below.

Procedure 3.2 Build STM

For HDCTCM(n, D), number its butterflies from 1 to n^{D-1}.

1. First, select any butterfly, record this choice as STM-FREE (first butterfly). There are n^{D-1}choices. Take n=4, D=4 as an example. For instance, butterfly 1 is chosen. Secondly, place this selected butterfly into any location in STM, say--STM$(l_1, l_2, \ldots, l_{D-1})$. Record a particular choice in STM-FREE (first location). There are n^{D-1}choices (see Figure 3.6 (a), take $l_1 = 2$, $l_2 = 3$ and $l_3 = 1$). Then place the n originating states in the left side of this butterfly in an arbitrary way and record a particular arrangement in STM-FREE (left side butterfly). There are $n!$ different arrangements (see Figure 3.6 (b)). Finally, place the n next states in the right side of this butterfly in an arbitrary way and record a particular arrangement in STM-FREE (right side butterfly). There are $n!$ different arrangements (see Figure 3.6 (c)). STM-FREE (left side butterfly, right side butterfly) completely fixes the internal order of this butterfly. Name this first selected butterfly as the anchor butterfly.
2. Build the first dimension. Identify the other $n-1$ butterflies that constitute simplex-transition$_2$ with the anchor butterfly. The internal orders of these $n-1$ butterflies are fixed with respect to the anchor butterfly using Procedure 3.1. They should be placed into locations along the first dimension of STM and should have the location indexes along all the other dimensions the same as anchor butterfly, i.e., the locations STM$(x_1, l_2, \ldots, l_{D-1})$ where $x_1 =: / l_1$. The order of placing them is arbitrary. There are (n-1)! different ways. Record a particular arrangement in STM-FREE (1_{st} dimension). See Figure 3.6 (d), assuming that butterfly {1, 2, 3, 4} constitute simplex-transition$_2$.
3. Build the second dimension. Identify the other $n-1$ butterflies that constitute simplex-transition$_3$ with the anchor butterfly. The internal orders of these $n-1$ butterflies are fixed with respect to the anchor butterfly using Procedure 3.1. They should be placed into locations along the 2_{nd} dimension of STM and have the same location indexes along all the other dimensions as the anchor butterfly, i.e., the locations

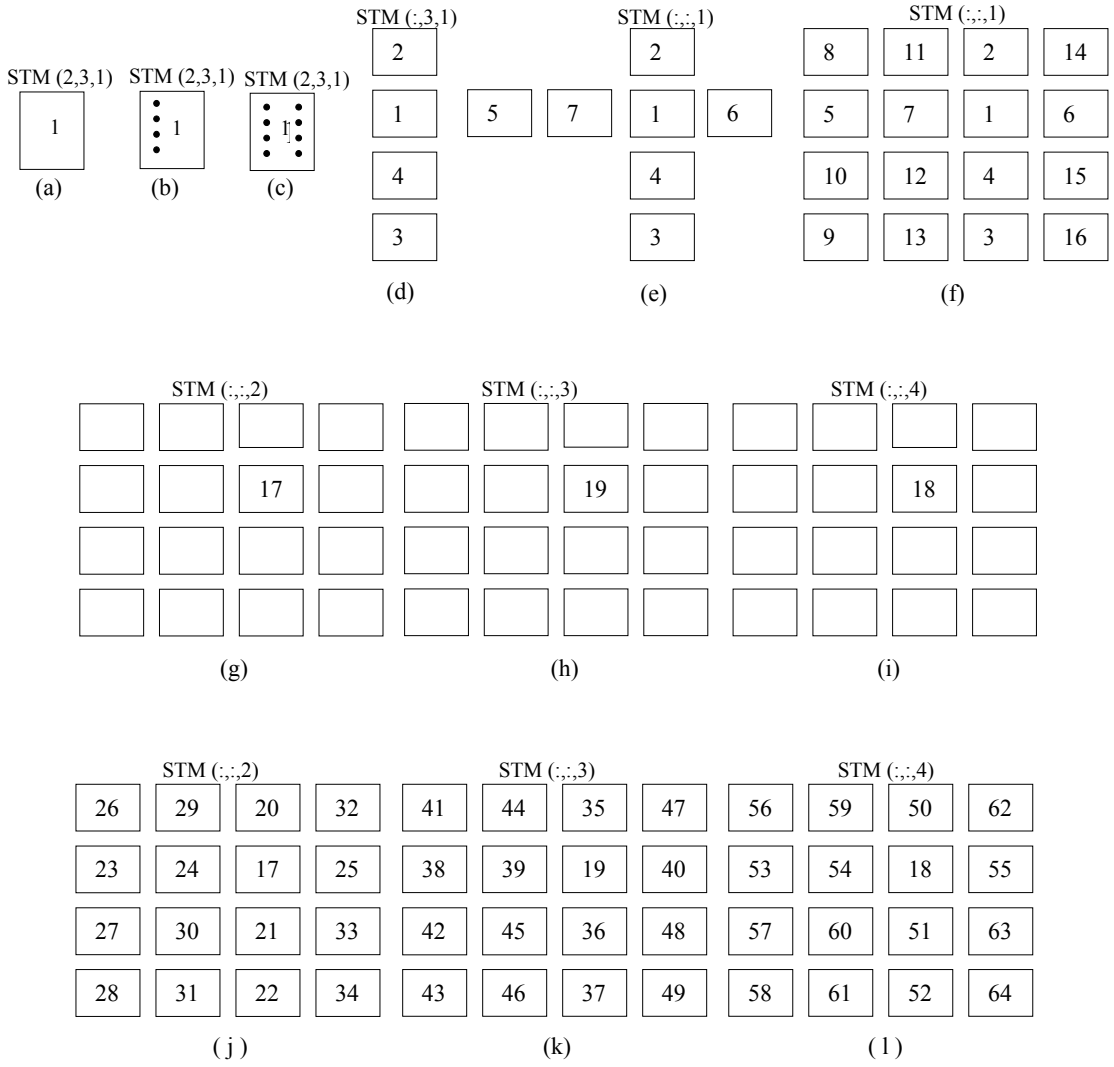


Figure 3.6 Build STM for HDCTCM$(4,4)$

STM$(l_1, x_2, l_3, \ldots, l_{D-1})$ where $x_2 =: / l_2$. The order of placing them is arbitrary. There are $(n\text{-}1)!$ different ways to place them. Record a particular arrangement in STM-FREE

(2_{nd} dimension). See Figure 3.6 (e), assuming that butterfly $\{5, 1, 6, 7\}$ constitute simplex-transition$_3$. For each x_2 where $x_2 \neq l_2$, regard the butterfly in $\mathrm{STM}(l_1, x_2, l_3, \ldots, l_{D-1})$ as the anchor butterfly. Repeat step 2 and the other $n-1$ butterflies constituting simplex-transition$_2$ with this butterfly will be identified. Their internal order is fixed with respect to this butterfly using Procedure 3.1 and they should be placed in locations along the first dimension—$\mathrm{STM}(x_1, x_2, l_3, \ldots, l_{D-1})$ where $x_1 =: / l_1$. Denoted these $n-1$ butterflies as b_k, $k = 1,2,3$. This time, the order of placing them is now fixed. Suppose butterfly b_k is to be placed in $\mathrm{STM}(j, x_2, l_3, \ldots, l_{D-1})$ where $j \in \{1,2,...,n\}$ and $j \neq l_1$; that means b_k is a member of the simplex-transition$_3$-butterfly-set containing the butterfly in $\mathrm{STM}(j, x_2, l_3, \ldots, l_{D-1})$ which has been placed in the previous stage. Since a particular butterfly is a member of one and only one simplex-transition$_3$-butterfly-set, so by cross-checking the membership of butterfly b_k in the $n-1$ simplex-transition$_3$-butterfly-sets containing butterflies in $\mathrm{STM}(x_1, l_2, l_3, \ldots, l_{D-1})$ where $x_1 =: / l_1$, the location index of butterfly b_k along the first dimension is obtained (see Figure 3.6 (f)). In Figure 3.6 (f), assume that butterfly 8, 9, and 10 constitute simplex-transition$_2$ with butterfly 5, and the three of simplex-transition$_3$-butterfly-set containing butterfly 2, 3, 4 are butterfly $\{8, 11, 2, 14\}$, butterfly $\{10, 12, 4, 15\}$, and butterfly $\{9, 13, 3, 16\}$, respectively. Then butterfly 8, 9 and 10 should all have the same location index along the 2_{nd} dimension as butterfly 5 and have the same location index along the 1_{st} dimension as butterflies 2, 4, 3, respectively. Similarly, All the rest of the locations are filled with the proper butterflies. We see the order of placing the other $n-1$ butterflies in the simplex-transition$_2$-butterly-set and simplex-transition$_3$-butterfly-set containing the anchor butterfly fix all the butterflies' location in this two-dimensional place where the anchor butterfly is located.

4. Build the third dimension. Identify the other $n-1$ butterflies constituting simplex-transition$_4$ as the anchor butterfly; their internal order will be fixed using Procedure 3.1. They should be placed in $\mathrm{STM}(l_1, l_2, x_3, \ldots, l_{D-1})$ where $x_3 =: / l_3$. The order of placing them is arbitrary. There are $(n\text{-}1)!$ different ways to place them. Choose a particular arrangement and record it in STM-FREE (3_{rd} dimension). See Figure 3.6

(g), (h), and (i), assuming that butterfly{1, 17, 18, 19} constitute simplex-transition$_4$. For each x_3 where $x_3 \neq l_3$, regard the butterfly in STM$(l_1,l_2,x_3,\ldots,l_{D-1})$ as the anchor butterfly. Repeat step 2 and 3; butterflies from certain simplex-transition$_2$-butterfly-sets and simplex-transition$_3$-butterfly-sets will be identified and placed into the rest of locations in the two-dimensional plane containing this butterfly--STM$(:,:,x_3,l_4,\ldots,l_{D-1})$ and their locations are also fixed. Because for each x_3 where $x_3 \neq l_3$, when repeating step 2 and 3, the order of placing the butterflies constituting simplex-transition$_2$ with the butterfly in STM$(l_1,l_2,x_3,\ldots,l_{D-1})$ along the first dimension into STM$(x_1,l_2,x_3,l_4,\ldots,l_{D-1})$ where $x_1 =: /l_1$ will be fixed by cross-checking the memberships of these butterflies in the simplex-transition$_4$-butterfly-sets containing the butterflies in STM$(x_1,l_2,l_3,l_4,\ldots,l_{D-1})$ where $x_1 =: /l_1$. The order of placing the butterflies constituting simplex-transition$_3$ with the butterfly in STM$(l_1,l_2,x_3,\ldots,l_{D-1})$ along the second dimension into STM$(l_1,x_2,x_3,l_4,\ldots,l_{D-1})$, where $x_2 =: /l_2$ will be fixed by cross-checking the memberships of these butterflies in the simplex-transition$_4$-butterfly-sets containing the butterflies in STM$(l_1,x_2,l_3,l_4,\ldots,l_{D-1})$ where $x_2 =: /l_2$. Then, from the previous steps, this will consequently fix all other butterflies in this two-dimensional plane--STM$(:,:,x_3,l_4,\ldots,l_{D-1})$ (see Figure 3.6 (j)). In Figure 3.6 (j), assume that butterfly 20, 21, and 22 constitute simplex-transition$_2$ with butterfly 17 and also assume that butterfly 23, 24, and 25 constitute simplex-transition$_3$ with butterfly 17. Then the locations of butterfly 20, 21, 22, 23, 24, and 25 will be fixed by cross-checking with simplex-transition$_4$-butterfly-sets containing butterfly 2, 3, 4 and 5, 6, 7. Assume butterfly 20 is in the simplex-transition$_4$-butterfly-set containing butterfly 2, butterfly 23 is in the simplex-transition$_4$-butterfly-set containing butterfly 5, then they should have the same location indexes along first and second dimension in STM$(:,:,2)$ as butterfly 2 and 5 in STM$(:,:,1)$, respectively. By this way, proper sets of butterflies will be placed in the rest locations in STM $(:,:,2)$. Similarly, the rest locations in STM$(:,:,3)$ and STM$(:,:,4)$ will be filled (see Figure 3.6 (k) and (l)). The internal order of all these butterflies will be fixed using Procedure 3.1.

5. Continue this way, each time fill the rest of locations along a new dimension of STM, until completely filling the $D-1$ dimensional matrix. At step i, the $n-1$ butterflies constituting simplex-transition$_i$ with the anchor butterfly will be placed into locations along the $(i-1)_{th}$ dimension, i.e., the locations STM$(l_1,l_2,...,x_{i-1},l_i,\ldots,l_{D-1})$ where $x_{i-1} =: / l_{i-1}$. The order of placing them in these locations is arbitrary. Record a particular arrangement in STM-FREE ($(i-1)_{th}$ dimension). There are (n-1)! different ways. Then, for each x_{i-1} where $x_{i-1} \neq l_{i-1}$, regard the butterfly in STM$(l_1,l_2,...,x_{i-1},l_i,\ldots,l_{D-1})$ as the anchor butterfly. Repeat step 2 until step $i-1$, the unfilled locations in STM$(:,:,...,:,x_{i-1},l_i,...,l_{D-1})$ will be filled by butterflies from certain simplex-transition$_2$-butterfly-sets, simplex-transition$_3$-butterfly-sets, …. , simplex-transition$_{i-1}$-butterfly-sets. The locations of all these butterflies used to fill are fixed; because when repeating step 2 until step $i-1$, the order of placing the butterflies constituting simplex-transition$_k$ where $k = 2,3,...,i-1$ with this butterfly into STM$(l_1,l_2,...,x_{k-1},l_k,\ldots,l_{i-2},x_{i-1},l_i,...,l_{D-1})$ where $x_{k-1} =: / l_{k-1}$ will be fixed by cross-checking the memberships in the simplex-transition$_i$-butterfly-sets containing the butterflies in STM$(l_1,l_2,...,x_{k-1},l_k,\ldots,l_{i-2},l_{i-1},l_i,...,l_{D-1})$ where $x_{k-1} =: / l_{k-1}$. The locations of those butterflies have been fixed in the previous steps. Then the locations of all the rest of the butterflies that should be placed in STM$(:,:,...,:,x_{i-1},l_i,...,l_{D-1})$ are fixed. The internal orders of all butterflies are fixed using Procedure 3.1.

We can see from Procedure 3.2, the $D+3$ times choice defined in STM-FREE totally fix an STM. Changing one butterfly's internal order or the order of placing one set of the n butterflies along the i_{th} dimension will cause corresponding change on all the other butterfly's internal order or the order of placing all the other sets of the n butterflies along the i_{th} dimension, $i = 1,2,...,D-1$. This implies that one butterfly's internal order will fix the internal order of all the other butterflies, i.e., define STM (left side butterfly, right side butterfly). The order of placing one set of the n butterflies along the i_{th} dimension of STM will fix the order of placing all the other sets of n butterflies along the i_{th} dimension, i.e., define STM-FREE (i_{th} dimension). Placing the first selected butterfly in a location other than STM$(1,1,...,1)$ and keeping the other members in STM-

FREE the same will result in a different STM. It can be verified that this resultant STM can also be built by first choosing the butterfly in the this resultant STM$(1,1,...,1)$ as the first selected butterfly, then placing it in STM$(1,1,...,1)$ and defining other members in STM-FREE accordingly. This shows that we can always place the first selected butterfly in STM$(1,1,...,1)$ and still can get all the different STM's by defining the other members in STM-FREE. This gives the total number of different STM's for HDCTCM(n, D) as $n^{D-1} \cdot n! \cdot n! \cdot ((n-1)!)^{D-1}$.

3.2.6 Example of Building STM

Example 3.6 Build STM for HDCTCM$(4,3)$

All sets of butterflies constituting simplex-transition$_2$ and simplex-transition$_3$ are listed in Example 3.4. The 2-dimensional STM will be built as follows:

1. Select any butterfly, arrange its internal order arbitrarily, and place it into any location In Figure 3.7(a), butterfly 1 is chosen and placed in STM$(2,3)$.

2. Build the first dimension.

   Simplex-transition$_2$-butterfly-set containing butterfly 1 is butterfly$\{1, 2, 4, 10\}$. So, butterfly 2, 4, and 10 are placed in locations STM$(1,3)$, STM$(3,3)$ and STM$(4,3)$ in an arbitrary order. The internal orders of them are fixed with respect to butterfly 1 using Procedure 3.1 (see Figure 3.7 (b)).

3. Build the second dimension.

   Simplex-transition$_3$-butterfly-set containing butterfly 1 is butterfly$\{1, 3, 5, 15\}$. So, butterfly 3, 5, and 15 are placed into locations STM$(2,1)$, STM$(2,2)$, and STM$(2,4)$ in an arbitrary order. Their internal orders are fixed with respect to butterfly 1 using Procedure 3.1 (see Figure 3.7(c)). For butterflies 3, 5, and 15, the other 3 butterflies constituting simplex-transition$_2$ with them are butterfly 6, 9, and 12, butterfly 7, 8, and 14 and butterfly 11, 13, and 16. They will be placed into the rest locations in STM$(:,2)$, STM$(:,1)$, and STM$(:,4)$, respectively. Their location indexes along the first dimension will be fixed by cross-checking the membership in three sets of simplex-transition$_3$-butterfly-set containing butterflies in STM$(1,3)$, STM$(3,3)$, and

STM(4,3). They are butterfly{2, 6, 8, 11}, butterfly{10, 12, 14, 16}, and butterfly{4, 9, 7, 13}. So, butterfly 6, 9, and 12 should be placed into STM(:,2) and

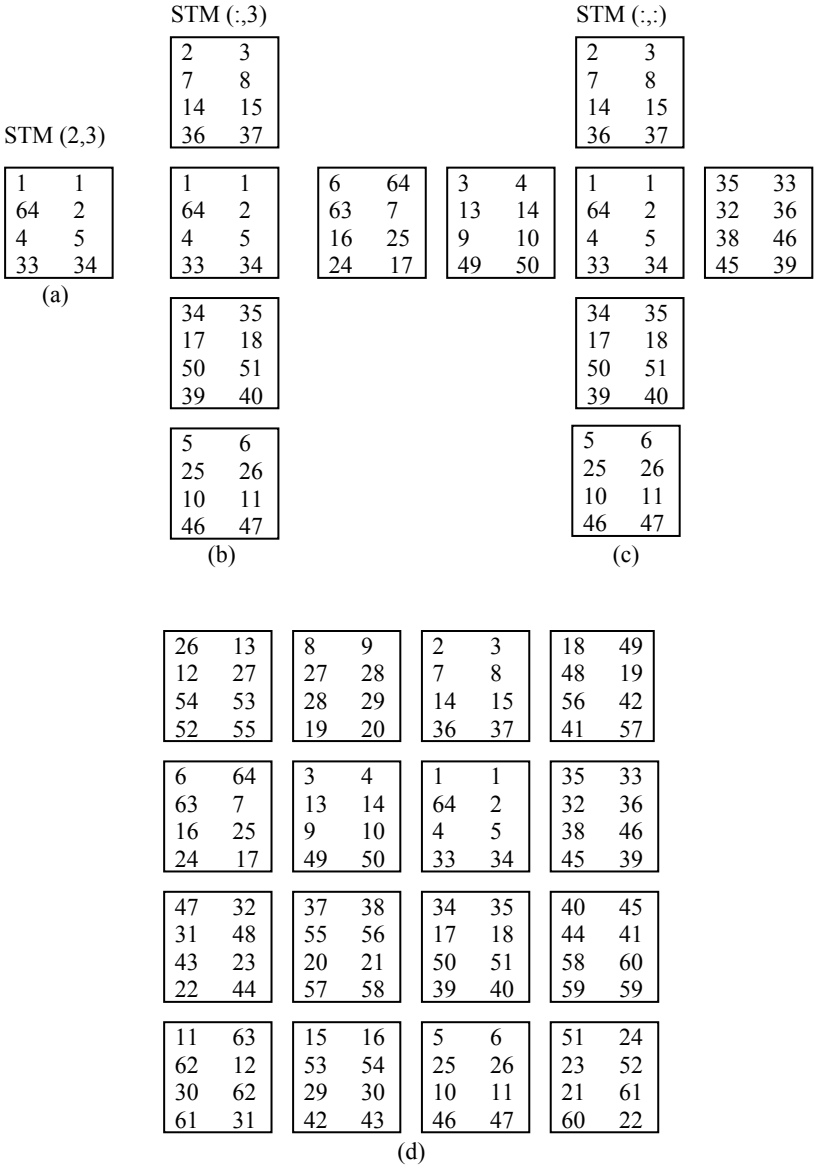


Figure 3.7 Build STM for HDCTCM (4,3)

have the same location index along the first dimension as butterfly 2, 4, and 10 in $\mathrm{STM}(:,3)$. And their internal orders are fixed with respect to butterfly 2, 4, and 10 using Procedure 3.1. Similarly, $\mathrm{STM}(:,1)$ and $\mathrm{STM}(:,4)$ are filled properly (see Figure 3.7(d)).

3.3 Designs of Signal Constellation and Symbol Assignment

In this section, the simplex signal constellation is constructed in a high dimensional space and the mapping rule between the signal constellation and the sate transitions is designed.

3.3.1 Algebraic Representation of Simplex

A simplex is a set of M signals in an N-dimensional space with $N \geq M-1$, satisfying (2.6). For $\mathrm{HDCTCM}(n, D)$, a simplex is to be assigned to a set of n state transitions, that is to say we need a simplex that contains $M = n$ signals. In circular trellis with permuted state structure, n is the size of information symbol alphabet size. Non-binary transmission are often more efficient than binary transmission [36], and when $M > 4$, the energy saving of simplex signaling over orthogonal signaling is not significant [33]. So, in this dissertation, $M = 4$ is chosen for the most practical $\mathrm{HDCTCM}(4, D)$. When $M = 4$, $N \geq 3$, this means that a three-dimensional space is the smallest signal space to build a simplex that contains four member signals.

We introduce an algebraic representation for simplex. A simplex signal s_i, where $i = 1,2,3,4$, is represented as a vector $(\pm a, \pm b, \pm c)$, where $a, b, c \in \{1,2,...,N\}$ and N is the dimension of the signal space. This representation means signal s_i has negative (-) or positive (+) unit pulses in the a_{th}, b_{th}, and c_{th} space dimension and 0's in all the other space dimensions. We name the element in this vector representation as pulse-location and the i_{th} element where $i = 1,2,3$ as pulse-location$_i$. The value of pulse-location$_i$ is a signed integer. In signal s_i, the $\pm a$, $\pm b$ and $\pm c$ are the values of pulse-location$_1$, pulse-location$_2$, and pulse-location$_3$ respectively. We call this signaling scheme the 3-out-of-N simplex signaling scheme. Call a, b, and c the space dimensions occupied by a signal

(or a simplex, or pulse-locations). The corresponding normalized signal s_i is $\sqrt{\frac{1}{3}}(\pm a, \pm b, \pm c)$. A simplex will be written as simplex$\{s_i \mid i = 1,2,3,4\}$.

Two types of simplexes are designed for the signal constellation.

1. Type A simplex--also referred to as source simplex--in which all its four member signals occupy the same three dimensions.

There are only two subtypes; each has either odd or even number of negatives in all its member signals referred to as even or odd source simplex (see Figure 3.8). When we write a simplex in the way as in this figure, we regard a simplex as an ordered structure and the location of a signal in the simplex is referred to as the internal location of the signal in this simplex. Number 1, 2, and 3 refer to the three different dimensions, not necessarily the first three dimensions in an N-dimensional space.

1 2 3	1 2 -3
1 -2 -3	1 -2 3
-1 2 -3	-1 2 3
-1 -2 3	-1 -2 -3

Figure 3.8 Two subtypes of type A simplex

2. Type B simplex, in which no two member signals occupy the same three dimensions.

The total number of the occupied dimensions of a type B simplex is 6 (see Figure 3.9 (a)). Notice a "copy" rule in this representation. By "copy" we mean, in a pair of signals, one pulse-location has the same absolute value but opposite sign. For instance, in Figure 3.9 (a) in the first two signals, pulse-location$_1$ is 1, and –1 respectively. This means these two signals related to each other by "copy" rule for pulse-location$_1$. In a type B simplex, a pair of signals related to each other by "copy" rule for pulse-location$_i$, where $i = 1,2,3$, is called a signal pair for pulse-location$_i$. Their locations in this simplex are called a location pair for pulse-location$_i$. It can be verified from Figure 3.9 (a) that any two signals can constitute a signal pair for a certain pulse-location$_i$. For each pulse-location$_i$, the four member signals in a type B simplex constitute two signal pairs. Define the location of a signal in a simplex from top to bottom as 1, 2, 3, and 4. In Figure 3.9 (a), the first two signals and the second two signals are the two signal pairs for pulse-locatoin$_1$. The two

location pairs for pulse-location$_1$ are locations $(1,2)$ and locations $(3,4)$. All location pairs for all pulse-location$_i$'s, where $i = 1,2,3$, are listed in a matrix called pair.

$$pair = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 3 & 4 \\ 3 & 2 & 2 \\ 4 & 4 & 3 \end{bmatrix} \tag{3.2}$$

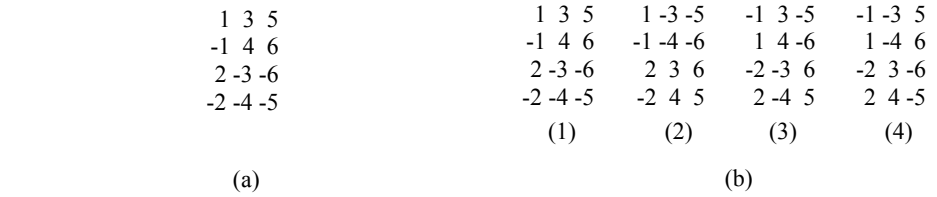


Figure 3.9 (a) A type B simplex (b) 4 type B simplexes formed from 4 source simplexes related to the four signals in the type B simplex in (a)

The two location pairs for pulse-location$_i$ are listed in the i_{th} column of this matrix. They are locations (pair$(1,i)$, $(2,i)$) and locations (pair$(3,i)$, $(4,i)$). Each of the two locations in a location pair is called a pair location of the other location. From Equation (3.2), it can be verified that for a given signal location, its pair location for pulse-location$_i$ and pulse-location$_j$, where $i \neq j$, are different.

Each member signal in a type B simplex is a member signal in a type A simplex that occupies the same three space dimensions as this signal. Call this type A simplex the related source simplex to this signal. For instance, a member signal (1, 3, 5) in type B simplex, shown in Figure 3.9 (a), is related to the source simplex {(1, 3, 5), (1, -3, -5), (-1, 3, -5), (-1, -3, 5)}. Four member signals in a type B simplex will relate to four source simplexes. It can be verified that there is one and only one way to arrange these four source simplexes to form four type B simplexes with each member signal of a formed type B simplex drawn from a different source simplex. For the four source simplexes related to the four signals in the type B simplex shown in Figure 3.9 (a), the only arrangement to form four type B simplexes is shown in Figure 3.9 (b). The set of four signals in each row come from a same source simplex, and the four signals in each column form a type B simplex.

3.3.2 The Idea of Symbol Assignment

For HDCTCM$(4, D)$, our goal is to build a signal constellation with total $S \cdot n = 4^{D+1}$ symbols and assign them to the 4^{D+1} state transitions in STM such that all sets of 4 transitions constituting simplex-transition$_i$'s, $i = 1,2,\ldots, D+1$ are assigned simplexes. In STM, a simplex-transition$_1$ is in a butterfly; the four transitions in it have a same originating state. A simplex-transition$_{D+1}$ is in a butterfly with each of its transitions originating from a different originating state in this butterfly. A simplex-transition$_i$, $i = 2,3,\ldots, D$, is the set of 4 transitions in the same internal location of the four butterflies along the $(i-1)_{th}$ dimension of STM with each of its transition originating from one of the four originating states in the same internal location of these four butterflies.

The idea is to relate each of the member signals s_j, $j = 1,2,3,4$ of a type B simplex to each of the originating state in a butterfly in STM or to each of the four originating states, which are in the same internal location of the set of four butterflies along the $(i-1)_{th}$ dimension of STM, $i = 2,3,\ldots, D$. Denote the originating state to which the signal s_j is related as g_k. In our symbol assignment, the four state transitions originating from state g_k (a simplex-tarnstinon$_1$, denoted as the simplex-transition$_1$ originating from state g_k) will be assigned the member signals from the related source simplex to signal s_j, denoted as ϖ_j. In this way, a simplex-tarnstinon$_1$ is assigned a source simplex. We call this source simplex, ϖ_j, the related source simplex to this originating state g_k or the related source simplex to the simplex-transition$_1$ originating from state g_k. Then the 4 simplex-transtion1's originating from g_k, for all k, $k = 1,2,3,4$ will be assigned using the four related source simplexes ϖ_j, $j = 1,2,3,4$.

As known from the illustration of simplex-transitions in butterflies, when the four originating state g_k, for all k, $k = 1,2,3,4$, are in a butterfly in STM, the four simplex-transition$_1$'s originating states g_k, also constitute four simplex-transition$_{D+1}$'s, with each transition in a simplex-transition$_{D+1}$ drawn from a different simplex-transition$_1$. When the four originating states g_k are in the same internal location of four butterflies along the $(i-1)_{th}$ dimension in STM, the four simplex-transition$_1$'s originating states g_k will

constitute four simplex-transition$_i$'s with each transition in a simplex-transition$_i$ drawn from a different simplex-transition$_1$. Shown in Figure 3.10 (a) are the transitions in a butterfly—butterfly 1 in STM built for HDCTCM(4,3) in Example 3.6**.** Figure 3.10 (b) shows the transitions originating from the four originating states in one set of four butterflies along the second dimension in that STM. The set of four transitions, each having a different line style, is a simplex-transition$_1$. The set of four transitions having the same line style is a simplex-transition$_4$ in Figure 3.10 (a) and a simplex-transition$_3$ in Figure 3.10 (b).

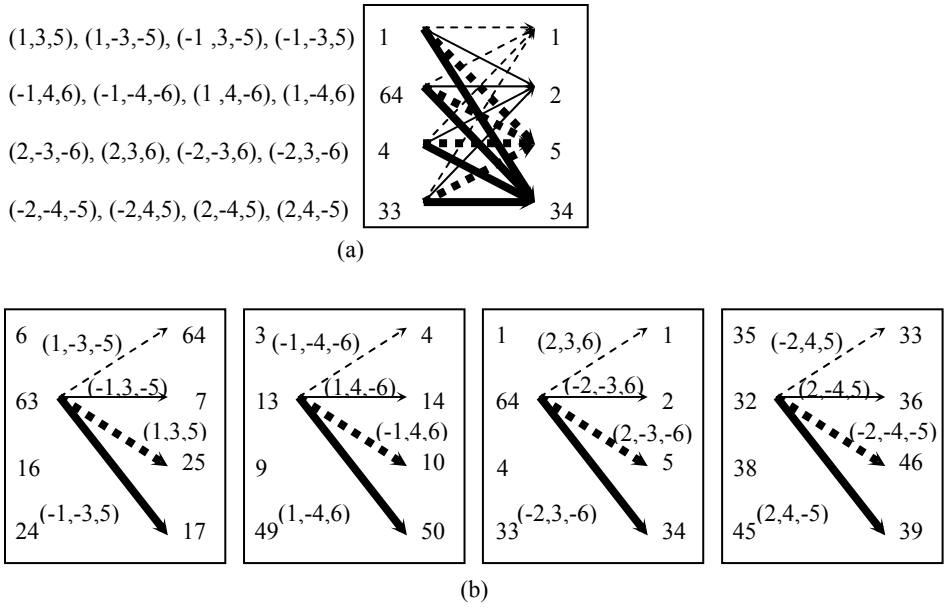


Figure 3.10 Symbols assignment for the transitions originating from four originating states which are (a) in a butterfly in STM and (b) in the same internal location of a simplex-transtition$_i$-butterfly-set in STM

Each of these four simplex-transition$_{D+1}$'s or simplex-transition$_i$'s , $i = 2,3,\ldots,D$, should also be assigned a simplex. We have shown that there is one and only one way to arrange these four related source simplex ϖ_j, $i = 1,2,3,4$, to form four type B simplexes, denoted as ξ_l, $l = 1,2,3,4$, with each member signal of ξ_l drawn from a different source simplex ϖ_j. This says we can arrange the order of assigning the signals in source simplex ϖ_j to the transitions originating from state g_k such that these four simplex-transition$_{D+1}$'s

or simplex-transition$_i$'s be assigned the four type B simplexes ξ_l. Let us see how to do this in the following example.

Example 3.7 Symbols assignment for the transitions originating from four originating state in a butterfly in STM or four originating states in the same internal location of the set of four butterflies along the $(i-1)_{th}$ dimension in STM

In Figure 3.10 (a), we relate the four signals from top to bottom in the type B simplex shown in Figure 3.9 (a) to the four originating state 1, 64, 4, and 33 respectively. Then the source simplex in each row from row 1 to row 4 in Figure 3.9 (b) will be assigned to the simplex-transition$_1$'s from originating state 1, 64, 4, and 33 respectively. The only combination of arranging these four source simplexes to form four type B simplexes is in Figure 3.9 (b). First, we can choose any simplex-transition$_1$ and assign the related source simplex to it arbitrarily. Say we can assign the simplex-transition$_1$ from originating state 1 from top to bottom in this butterfly the signals (1, 3, 5), (1, -3, -5), (-1, 3, -5), and (-1, -3, 5), respectively. Then the symbols assigned to the other simplex-transition$_1$'s will be fixed. The reason is as follows. We have known the symbols assigned to the simplex-transition$_1$ from one of the originating states. That is to say, for each simplex-transition$_4$, we have known the symbol assigned to one transition in this simplex-transition$_4$. This symbol is a member signal of one of the four type B simplexes; say, ξ_m, $m = 1,2,3,4$ in Figure 3.9 (b). Then in order to make this simplex-transition$_4$ to be assigned a type B simplex, the other three transitions in this simplex-transition$_4$ should be assigned the other three member signals in this type B simplex ξ_m. In this example, knowing transition $1 \to 1$ is assigned signal (1, 3, 5), which is a member signal of type B simplex 1 in Figure 3.9 (b), the other three transitions in the simplex-transition$_4$ having the next state 1, $64 \to 16$, $4 \to 1$and $33 \to 1$should be assigned the rest of the signals in type B simplex 1 in Figure 3.9 (b), i.e., (-1, 4, 6), (2, -3, -6) and (-2, -4, -5). In this way, all the transitions from originating state 64, 4, and 33 are assigned proper symbols from the related source simplexes. In Figure 3.10 (a), the signals assigned to the four transitions from an originating state from top to bottom in this butterfly are listed beside that originating state from left to right. The four simplex-transition$_4$'s having the next

state 1, 2, 5, and 34 are assigned type B simplex 1, 2, 3, and 4, as shown in Figure 3.9 (b), respectively.

Similarly, we related the signals in the type B simplex in Figure 3.9 (a) from top to bottom to the four originating states 63, 13, 64 and 32 in Figure 3.10 (b). Arbitrarily assign one simplex-transition$_1$ from an originating state using the related source. For example, assign the four transitions originating from state 63 from top to bottom the signals (1, -3, -5), (-1, 3, -5), (1, 3, 5), (-1, -3, 5), respectively. Then similarly as in Figure 3.10 (a), all the transitions from originating state 13, 64, and 32 will be assigned proper symbols from the related source simplexes. The four simplex-transition$_4$'s from top to bottom are assigned type B simplex 2, 3, 1, and 4, as showed in Figure 3.9 (b).

This example shows that after knowing the signals related to the four originating states, the symbols assigned to the simplex-transition$_1$'s from any one originating state will fix the symbols assigned to the other simplex-transition$_1$'s from the other originating states.

In summary, we want to build a total number of S type B simplex signals and relate each of the originating state g_k in a butterfly in STM, $k = 1,2,3,4$, to a signal s_j in a type B simplex, $j = 1,2,3,4$. In the mean time, we want the signals related to the four originating states, which are in the same internal locations of the set of four butterflies along the $(i-1)_{th}$ dimension, are also member signals from a type B simplex for $i = 2,3,\ldots,D$. Then by assigning the related source simplex to s_j to the four transitions originating from g_k in a proper way, each of all the simplex-transition$_1$'s in STM will be assigned a source simplex; and each of the simplex-transition$_i$'s, $i = 2,3,\ldots,D+1$, will be assigned a type B simplex. This gives the idea of building these S signals.

We will regard a type B simplex as a unit element with an ordered structure and replace a butterfly in STM with a type B simplex. Name the new matrix as initial input simplex (IIS). Similarly, a location in IIS is referred to using IIS$(x_1, x_2, \ldots, x_i, \ldots, x_{D-1})$. If we can make the signals in the same internal location of the four type B simplexes along the $(i-1)_{th}$ dimension of IIS also a type B simplex, then we can build a one-to-one mapping between the originating state in STM and signals in IIS, such that a set of four signals in the same internal locations of the four type B simplexes along the

$(i-1)_{th}$ dimension of IIS--IIS$(x_1, x_2, \ldots, x_{i-2}, \vdots, x_i, \ldots, x_{D-1})$ is related to the set of four states in the same internal locations of the set of four butterflies along the $(i-1)_{th}$ of STM--STM$(x_1, x_2, \ldots, x_{i-2}, \vdots, x_i, \ldots, x_{D-1})$. Also, make the four originating states in a butterfly in STM related to a type B simplex in IIS. Then, by assigning the simplex-transition$_1$ originating from a state the related source simplex properly, all the simplex-transition$_i$, $i = 1, 2, \ldots, D+1$, will be assigned simplex. In this way, the signal constellation for HDCTCM$(4, D)$ consists of all the source simplexes related to all the signals in IIS.

3.3.3 Build Initial Input Signal

The section will build IIS. First, we will describe how to build a type B simplex in Procedure 3.3.

Procedure 3.3 Build a type B simplex

A type B simplex occupies six distinct space dimensions. Number them from 1 to 6.

1. Select any three of these six dimensions and give them any sign to build a signal. And place it in location 1 of this simplex (see Figure 3.11 (a)). From Equation (3.2), location 1 is the first location in the first location pair for all pulse-location$_i$, $i = 1,\ 2,\ 3$.

1	3	5				1	3	5				1	3	5
						-1						-1	4	6
							-3					2	-3	-6
								-5				-2	-4	-5
	(a)						(b)						(c)	

Figure 3.11 Build a type B simplex

2. For each pulse-location$_i$, $i = 1,\ 2,\ 3$, from Equation (3.2), find the other location in the first location pair; it is location pair$(2, i)$. "Copy" the exiting value to that location (see Figure 3.11 (b)). For pulse-location$_i$, $i = 1,\ 2,\ 3$, its value is copied to location 2, 3, and 4, respectively, in this simplex.
3. For each pulse-location$_i$, "fill" the first location in the second location pair, i.e., location pair$(3, i)$, using a new dimension and "copy" to its pair location pair$(4, i)$. The sign of pulse-location$_i$ used to "fill" is the same as the sign of pulse-location$_i$ in the first location of the first location pair (see Figure 3.11 (c)). For pulse-location$_1$, the second location pair is location (3, 4), "fill" location 3 using a new space dimension 2; "copy" it to location 4 as –2.

This procedure will be extended to build the IIS. It starts from a type B simplex placed in the first location in IIS, and then applies "copy" and "fill" operations along each matrix dimension of IIS. The location 1, 2, 3, and 4 in matrix pair will be the location index along the i dimension, $i = 1, 2, \ldots, D-1$.

Procedure 3.4 Build IIS

1. Put the first type B simplex in IIS$(1,\ 1,\ ...,\ 1)$. Take HDCTCM$(4, 4)$ trellis as an example (see Figure 3.12 (a)).

2. Regard this type B simplex as a unit. Extend along the first matrix dimension of IIS. First, for each pulse-location$_i$, $i = 1,\ 2,\ 3$, "copy" the existing values to the other location of the first location pair, i.e., IIS$(pair(2, i),\ 1,\ ...,\ 1)$ (see Figure 3.12 (b)). Then

IIS(:,1,1)

1	3	5
-1	4	6
2	-3	-6
-2	-4	-5

(a)

IIS(:,1,1)

1	3	5
-1	4	6
2	-3	-6
-2	-4	-5
-1		
1		
-2		
2		
	-3	
	-4	
	3	
	4	
		-5
		-6
		6
		5

(b)

IIS(:,1,1)

1	3	5
-1	4	6
2	-3	-6
-2	-4	-5
-1	9	11
1	10	12
-2	-9	-12
2	-10	-11
7	-3	-11
-7	-4	-12
8	3	12
-8	4	11
-7	-9	-5
7	-10	-6
-8	9	6
8	10	5

(c)

IIS(:,:,1)

1	3	5	-1	-3	-5
-1	4	6	1	-4	-6
2	-3	-6	-2	3	6
-2	-4	-5	2	4	5
-1	9	11	1	-9	-11
1	10	12	-1	-10	-12
-2	-9	-12	2	9	12
2	-10	-11	-2	10	11
7	-3	-11	-7	3	11
-7	-4	-12	7	4	12
8	3	12	-8	-3	-12
-8	4	11	8	-4	-11
-7	-9	-5	7	9	5
7	-10	-6	-7	10	6
-8	9	6	8	-9	-6
8	10	5	-8	-10	-5

(d)

IIS(:,:,1)

1	3	5	-1	17	21	13	-3	-21	-13	-17	-5
-1	4	6	1	18	22	-13	-4	-22	13	-18	-6
2	-3	-6	-2	-17	-22	14	3	22	-14	17	6
-2	-4	-5	2	-18	-21	-14	4	21	14	18	5
-1	9	11	1	19	23	-13	-9	-23	13	-19	-11
1	10	12	-1	20	24	13	-10	-24	-13	-20	-12
-2	-9	-12	2	-19	-24	-14	9	24	14	19	12
2	-10	-11	-2	-20	-23	14	10	23	-14	20	11
7	-3	-11	-7	-17	-23	15	3	23	-15	17	11
-7	-4	-12	7	-18	-24	-15	4	24	15	18	12
8	3	12	-8	17	24	16	-3	-24	-16	-17	-12
-8	4	11	8	18	23	-16	-4	-23	16	-18	-11
-7	-9	-5	7	-19	-21	-15	9	21	15	19	5
7	-10	-6	-7	-20	-22	15	10	22	-15	20	6
-8	9	6	8	19	22	-16	-9	-22	16	-19	-6
8	10	5	-8	20	21	16	-10	-21	-16	-20	-5

(e)

IIS(:,:,2)

-1	1	-13	13
1	-1	13	-13
-2	2	-14	14
2	-2	14	-14
1	-1	13	-13
-1	1	-13	13
2	-2	14	-14
-2	2	-14	14
-7	7	-15	15
7	-7	15	-15
-8	8	-16	16
8	-8	16	-16
7	-7	15	-15
-7	7	-15	15
8	-8	16	-16
-8	8	-16	16

(f)

IIS(:,:,3)

-3	-17	3	17
-4	-18	4	18
3	17	-3	-17
4	18	-4	-18
-9	-19	9	19
-10	-20	10	20
9	19	-9	-19
10	20	-10	-20
3	17	-3	-17
4	18	-4	-18
-3	-17	3	17
-4	-18	4	18
9	19	-9	-19
10	20	-10	-20
-9	-19	9	19
-10	-20	10	20

(g)

IIS(:,:,4)

-5	-21	21	5
-6	-22	22	6
6	22	-22	-6
5	21	-21	-5
-11	-23	23	11
-12	-24	24	12
12	24	-24	-12
11	23	-23	-11
11	23	-23	-11
12	24	-24	-12
-12	-24	24	12
-11	-23	23	11
5	21	-21	-5
6	22	-22	-6
-6	-22	22	6
-5	-21	21	5

(h)

Figure 3.12 Build IIS for HDCTCM$(4,4)$

IIS(:,:,2)

-1 33 41	1 37 45	-13 -33 -45	13 -37 -41
1 34 42	-1 38 46	13 -34 -46	-13 -38 -42
-2 -33 -42	2 -37 -46	-14 33 46	14 37 42
2 -34 -41	-2 -38 -45	14 34 45	-14 38 41
1 35 43	-1 39 47	13 -35 -47	-13 -39 -43
-1 36 44	1 40 48	-13 -36 -48	13 -40 -44
2 -35 -44	-2 -39 -48	14 35 48	-14 39 44
-2 -36 -43	2 -40 -47	-14 36 47	14 40 43
-7 -33 -43	7 -37 -47	-15 33 47	15 37 43
7 -34 -44	-7 -38 -48	15 34 48	-15 38 44
-8 33 44	8 37 48	-16 -33 -48	16 -37 -44
8 34 43	-8 38 47	16 -34 -47	-16 -38 -43
7 -35 -41	-7 -39 -45	15 35 45	-15 39 41
-7 -36 -42	7 -40 -46	-15 36 46	15 40 42
8 35 42	-8 39 46	16 -35 -46	-16 -39 -42
-8 36 41	8 40 45	-16 -36 -45	16 -40 -41

(i)

IIS(:,:,3)

25 -3 -41	-25 -17 -45	29 3 45	-29 17 41
-25 -4 -42	25 -18 -46	-29 4 46	29 18 42
26 3 42	-26 17 46	30 -3 -46	-30 -17 -42
-26 4 41	26 18 45	-30 -4 -45	30 -18 -41
-25 -9 -43	25 -19 -47	-29 9 47	29 19 43
25 -10 -44	-25 -20 -48	29 10 48	-29 20 44
-26 9 44	26 19 48	-30 -9 -48	30 -19 -44
26 10 43	-26 20 47	30 -10 -47	-30 -20 -43
27 3 43	-27 17 47	31 -3 -47	-31 -17 -43
-27 4 44	27 18 48	-31 -4 -48	31 -18 -44
28 -3 -44	-28 -17 -48	32 3 48	-32 17 44
-28 -4 -43	28 -18 -47	-32 4 47	32 18 43
-27 9 41	27 19 45	-31 -9 -45	31 -19 -41
27 10 42	-27 20 46	31 -10 -46	-31 -20 -42
-28 -9 -42	28 -19 -46	-32 9 46	32 19 42
28 -10 -41	-28 -20 -45	32 10 45	-32 20 41

(j)

Figure 3.12 (cont.)

IIS(:,:,4)

-25	-33	-5	25	-37	-21	-29	33	21	29	37	5
25	-34	-6	-25	-38	-22	29	34	22	-29	38	6
-26	33	6	26	37	22	-30	-33	-22	30	-37	-6
26	34	5	-26	38	21	30	-34	-21	-30	-38	-5
25	-35	-11	-25	-39	-23	29	35	23	-29	39	11
-25	-36	-12	25	-40	-24	-29	36	24	29	40	12
26	35	12	-26	39	24	30	-35	-24	-30	-39	-12
-26	36	11	26	40	23	-30	-36	-23	30	-40	-11
-27	33	11	27	37	23	-31	-33	-23	31	-37	-11
27	34	12	-27	38	24	31	-34	-24	-31	-38	-12
-28	-33	-12	28	-37	-24	-32	33	24	32	37	12
28	-34	-11	-28	-38	-23	32	34	23	-32	38	11
27	35	5	-27	39	21	31	-35	-21	-31	-39	-5
-27	36	6	27	40	22	-31	-36	-22	31	-40	-6
28	-35	-6	-28	-39	-22	32	35	22	-32	39	6
-28	-36	-5	28	-40	-21	-32	36	21	32	40	5

(k)

Figure 3.12(cont.)

for each pulse-location$_i$, for the locations in the second location pair, "fill" the first location, IIS($pair(3,i)$, 1, ..., 1), using two new dimensions and "copy" to its pair location, IIS($pair(4,i)$, 1, ..., 1). The values used to "fill" comply with "copy" rule and have the same sign pattern as pulse-location$_i$ in the first location of the first location pair (see Figure 3.12 (c)).

3. Extend along the second matrix dimension. We can take all the four type B simplexes along the first dimension as a whole from now on. When filling a pulse-location$_i$, four new dimensions are needed, and "copy" rule and sign pattern for that pulse-location$_i$ are the same as pulse-location$_i$ in IIS(:, 1, ..., 1). First, for each pulse-location$_i$, $i = 1, 2, 3$ "copy" the existing values to the second location in the first location pair, IIS(:, $pair(2,i)$, 1, ..., 1)(see Figure 3.12 (d)). Then for each pulse-location$_i$, "fill" the

first location in the second location pair, IIS$(:, pair(3,i),\ 1,\ ...,\ 1)$, and "copy" to its pair location, IIS$(:, pair(4,i),\ 1,\ ...,\ 1)$(see Figure 3. 12 (e)).

4. Extend along the third matrix dimension. First, for each pulse-location$_i$, "copy" all the existing values to its pair location; it is IIS$(:,:, pair(2,i),\ 1,\ ...,\ 1)$(see Figure 3.12 (f), (g), and (h)). Then, for each pulse-location$_i$, "fill" the first location in the second location pair, IIS$(:,:, pair(3,i),\ 1,\ ...,\ 1)$ and "copy" to its pair location, IIS$(:,:, pair(4,i),\ 1,\ ...,\ 1)$. This time, the "fill" is on a two-dimensional plane with no existing pulse-location$_i$'s on it, so the first location from both location pairs needs to be filled, then "copy" them to their pair location (see Figure 3.12 (i), (j), and (k)).
5. Continue this way until extending IIS to the $(D-1)_{th}$ matrix dimension. Each time, extend IIS to one more dimension, in step j, "copy" the existing values to its pair location along the $(j-1)_{th}$ dimension, i.e., IIS$(\underbrace{:,:,...,}_{j-2} pair(2,i), \underbrace{1,...,1}_{D-j})$. Then, for each pulse-location$_i$, for the locations in the second location pair along the $(j-1)_{th}$ dimension, "fill" the first location, IIS$(\underbrace{:,:,...,}_{j-2} pair(3,i), \underbrace{1,...,1}_{D-j})$ and "copy" to its pair location IIS$(\underbrace{:,:,...,}_{j-2} pair(4,i), \underbrace{1,...,1}_{D-j})$. The "fill" is on a $j-2$-dimensional matrix with no existing pulse-location$_i$'s on it, so, it contains "fill" the first locations in both the location pairs along the $(j-2)_{th}$ dimension, IIS$(\underbrace{:,:,...,}_{j-3} pair(1,i), pair(3,i), \underbrace{1,...,1}_{D-j})$ and IIS$(\underbrace{:,:,...,}_{j-3} pair(3,i), pair(3,i), \underbrace{1,...,1}_{D-j})$ and "copy" to their pair location, IIS$(\underbrace{:,:,...,}_{j-3} pair(2,i), pair(3,i), \underbrace{1,...,1}_{D-j})$ and IIS$(\underbrace{:,:,...,}_{j-3} pair(1,i), pair(3,i), \underbrace{1,...,1}_{D-j})$. Recursively, that will contain "fill" and "copy" along the $(j-3)_{th}$ dimension, until the "fill" and "copy" along the second dimension of IIS in step 3.

In this way, for HDCTCM$(4, D)$ trellis, the total distinct signal dimensions needed to build IIS is $N = 6 \cdot 2^{D-1}$. The signal dimension can be frequency or time slot when the signal is actually transmitted.

3.3.4 The Procedure of Symbol Assignment

After IIS is built, the mapping between the originating states in STM and signals in IIS has been stated in the previous section, which serves as the guide for building the IIS. As seen in Example 3.7, the symbol assigned to a particular transition will change if we change the order of mapping the four signals s_j in a type B simplex to the four originating states in a butterfly or the four originating states in the same internal locations of the set of four butterflies along the $(i-1)_{th}$ dimension of STM. The same symbol assignment can be obtained by keeping a fixed mapping order and changing the order of the four originating states in a butterfly (defined by STM-FREE (left side butterfly)) or the order of the four butterflies along the $(i-1)_{th}$ dimension (defined by STM-FREE ($(i-1)_{th}$ dimension). That is to say, we can get the same symbol assignment if we fix the mapping order and use different STM built by changing the STM-FREE (left side butterfly) and STM-FREE ($(i-1)_{th}$ dimension). In our symbols assignment, we will fix the mapping order and leave the different symbol assignments to employing different STM-FREEs. The originating state having an internal location in the butterfly located at $\mathrm{STM}(l_1, l_2, \ldots, l_{D-1})$ will be mapped to the signal that has the same internal location in the type B simplex located at $\mathrm{IIS}(l_1, l_2, \ldots, l_{D-1})$. Then the four transitions from each originating state in STM will be assigned using the related source simplex and, as shown in Example 3.7, the order of assigning the four signals in this related source simplex to these four transitions can be arranged properly such that each of all the simplex-transition$_i$'s, $i = 1, 2, \ldots, D+1$, is assigned a simplex. We will work out an example to see how to make the symbol assignment after the mapping order between the originating states in STM and signals in IIS is fixed.

Example 3.8 Symbol assignment for $\mathrm{HDCTCM}(4,3)$

The IIS is built as in Figure 3.12(e). Use the STM built in Example 3.6.

1. Start the symbol assignment from the first originating state in the first butterfly in STM, i.e., butterfly located at $\mathrm{STM}(1,1)$. The originating state 26 is related to signal (1, 3, 5). Then the four transitions originating from the state 26 (simplex-transition$_1$ having state 26 as originating state) should be assigned the related source simplex {(1,

3, 5), (1, -3, -5), (-1, 3, -5), (-1, -3, 5)}. The order of assigning these four signals to these four transitions is flexible. But if a particular one is chosen, as we will see, it will fix the order of assigning all the simplex-transition$_1$'s from all the other originating states in STM using the related source simplexes. We assign the four transitions originating from state 26 from top to bottom in this butterfly, i.e., $26 \to 13$, $26 \to 27$, $26 \to 53$ and $26 \to 55$ the signals (1, 3, 5), (1, -3, -5), (-1, 3, -5), and (-1, -3, 5), respectively. This order can be identified from the locations of negative signs in these four signals.

2. Then consider the other three originating states 12, 54 and 52 in this butterfly. They relate to signal (-1, 4, 6), (2, -3, -6), and (-2, -4, -5), respectively. The related source simplex to each of these signals will be assigned to the four transitions originating from state 12, 54, and 52, respectively. From Example 3.7, the symbols assigned to the simplex-transition$_1$'s originating from state 26 will fix the symbols assigned to the other simplex-transition$_1$'s from the other originating states in this butterfly. Then, all the transitions in the butterfly located at STM(1,1) are assigned proper symbols.
3. Then we will consider transitions in the other $n-1$ butterflies constituting simplex-transition$_2$ with the first butterfly. They are located along the first dimension of STM, i.e., at locations STM(x_1,1), where $x_1 =: /1$. For each of the four internal locations that an originating state can have, consider the four originating states in that internal location of the four butterflies located at STM(:,1). From the previous step, we know the symbols assigned to the simplex-transition$_1$ originating from the originating state in the butterfly located at STM(1,1). Then from Example 3.7, the symbols assigned to all the transitions originating from the originating state in butterflies located at STM(2,1) and STM(3,1) and STM(4,1) will be fixed. For instance, the four originating states 12, 63, 31, and 62 along the first dimension, knowing the symbols assigned to the simplex-transition$_1$ from originating state 12 will fix the symbols assigned to the simplex-transition$_1$'s from originating state 63, 31, and 62. In this way, all the other transitions in butterflies located in STM(2,1), STM(3,1), and STM(4,1) are assigned proper symbols.
3. After knowing the symbols assigned to the transitions in butterflies located in STM(:,1), we can assign symbols to transitions in the set of four butterflies along the

second dimension of STM. Consider the four originating state in the same internal location of these four butterflies. From the previous step, we have known the symbols assigned to the simplex-transition$_1$ from the originating state in $\mathrm{STM}(:,1)$. Then from Example 3.7, the symbols assigned to all the transitions originating from the originating states in $\mathrm{STM}(:,2)$ $\mathrm{STM}(:,3)$ and $\mathrm{STM}(:,4)$ will be fixed. So far, all the state transitions have been assigned proper symbols.

For $\mathrm{HDCTCM}(4, D)$ with $D \geq 3$, continue the above symbols assignment procedure. At each step, extend the symbol assignment procedure to a new dimension of STM. Consider the set of four originating states in the same internal location of the four butterflies along that dimension. There is one originating state located in the butterfly at the first location index along that dimension. From the previous step, the symbols assigned to the simplex-transition$_1$ from this originating state are known; then from Example 3.7, the simplex-transition$_1$'s from the other three originating states will be fixed. In this way, all the transitions in the set of butterflies along that dimension will be assigned proper symbols at that step.

For a particular state transition, different symbol assignments will be obtained if we start the symbol assignment procedure from a different butterfly or a different originating state in the first butterfly in STM, or if we change the order of assigning the related source simplex to the simplex-transition$_1$ from the first originating state. Those resultant symbol assignments can also be obtained by using the symbol assignment procedure described above but employing different STMs built by choosing different STM-FFREE (first butterfly) (or STM-FREE (first location)) and different STM-FREE (left side butterfly), respectively. In our symbol assignment, we will always take the way we did in this example and leave the different symbol assignments to employing different STM-FREEs.

In summary, for a $\mathrm{HDCTCM}(4, D)$ system, the IIS is built using Procedure 3.4. The signal constellation consists of all the source simplexes related to all the signals in IIS. Different symbol assignments (different codes) for $\mathrm{HDCTCM}(4, D)$ result from applying the symbol assignment procedure described above to different STMs built by choosing different STM-FREEs.

We have seen that there is one to one mapping between each originating state in STM and the symbols in IIS. We also see from the symbol assignment procedure that for a state transition in STM, knowing its location in STM (the location of the butterfly containing this transition and the internal location of this transition in this butterfly), the symbol assigned to this transition will be known. That is to say, there is also a one to one mapping between each state transition and each symbol in the signal constellation.

3.4 Uniformity of HDCTCM

This section proves that HDCTCM using 3-out-of-N simplex signaling scheme is uniform.

3.4.1 Uniformity in coding

In coding scheme, the set of pair-wise squared Euclidean Distance SED $\{d_{ij}^2\}$ between code word C_i and C_j, where $i, j = 1,2,\ldots,W$ and W is the total number of legal code words, can be arranged into a $W \times W$ matrix E

$$E = \begin{bmatrix} d_{11}^2 & d_{12}^2 & \cdots & d_{1M}^2 \\ d_{21}^2 & d_{22}^2 & \cdots & d_{2M}^2 \\ \vdots & \vdots & \ddots & \vdots \\ d_{M1}^2 & d_{M2}^2 & \cdots & \vdots \end{bmatrix} \tag{3.3}$$

Because of the commutativity of SED, $d_{ij}^2 = d_{ji}^2$. The matrix is a symmetric matrix with all diagonal elements being zero. If the matrix E has such a symmetry that the sum of all the elements in any row (column) of E does not depend on the row (column) index, then the distance distribution calculation can be greatly simplified. Such symmetry corresponds to having all the code words on an equal footing and allows the consideration of a single reference code word rather than all the pairs of code words for computation of error probabilities and distance distribution. Such symmetry is called uniformity [35].

3.4.2 Algebraic Definition of Uniformity

Denote I as a $W \times 1$ column vector whose elements are all ones. For a $W \times W$ square matrix O, if I is an eigenvector of its transpose O^T, then $I^T O = \phi I^T$ holds,

where ϕ is the eigenvalue with respect to the eigenvector I. Then, the sum of the elements in a column of O does not depend on the column index. O is called column-uniform [35]. Similarly, if I is an eigenvector of the square matrix z, then $zI = I\chi$ holds, where χ is the corresponding eigenvalue. Then, the sum of the elements in a row of z does not depend on the row index. z is called row-uniform. It is easy to prove that the product or the sum of two column- (row-) uniform matrices is itself column- (row-) uniform [35]. Also, for a $W \times W$ matrix, O, that is either row- or column- uniform, we have $I^T OI = W\phi$, where κ is a constant.

3.4.3 Conditions for Uniformity

Assume code word C_i can be expressed as a vector of L symbol elements as $C_i = [C_{i1},...,C_{ik},...,C_{iL}]$. The SED between C_i and C_j is equal to the sum of the SED between the corresponding symbols,

$$d^2(C_i, C_j) = d_{ij}^2 = \sum_{k=1}^{L} d^2(C_{ik}, C_{jk}) = \sum_{k=1}^{L} d_{ijk}^2 \tag{3.4}$$

Let $E_k = \left\{ d_{ijk}^2 \right\}$ $i, j = 1,2,\ldots,W$, $k = 1,2,\ldots,L$, then $E = \sum_{k=1}^{L} E_k$.

Since E and E_k are symmetric, row- (column-) uniform implies column- (row-) uniform. We do not differentiate them later. If we can show E_k is uniform for all k, then E is uniform.

Define a code word matrix (CM) containing all the code words as

$$CM = \begin{bmatrix} C_{11} & C_{12} & \cdots & C_{1L} \\ C_{21} & C_{22} & \cdots & C_{2L} \\ \vdots & \vdots & \ddots & \vdots \\ C_{W1} & C_{W2} & \cdots & C_{WL} \end{bmatrix} \tag{3.5}$$

Denote the total number of symbols in the signal constellation as TOT-SYM. If a symbol assignment method guarantees that each symbol in the signal constellation is used equally often, which is the case for most coding/modulation scheme such as HDCTCM, then in each column of CM, each symbol will occur $W / TOT - SYM$ times. d_{ijk}^2 is the SED between the i_{th} and j_{th} symbols in column k of CM, then in each row (column) of E_k ,

there are $W / TOT - SYM$ duplicate terms of d_{ijk}^2 because of the duplicate appearance of each symbol. The sum of each row (column) of E_k is equal to $W / TOT - SYM$ times the sum of the single appearance of the duplicate terms d_{ijk}^2 in that row (column).

If we cross out the duplicate d_{ijk}^2 in E_k along rows and columns and keep the first appearance of the duplicate terms of d_{ijk}^2, then a new matrix G_k is formed. It is a $TOT - SYM \times TOT - SYM$ pair-wise SED between TOT-SYM distinct symbols. Obviously, G_k is also symmetric. Because the sum of each row (column) of E_k is equal to $W / TOT - SYM$ times that of G_k, if the sum of each row (column) in G_k is independent of row (column) index--i.e., G_k is uniform--then E_k is uniform. So, the proof of the uniformity of E_k is transferred to the proof of the uniformity of G_k.

3.4.4 Uniformity of HDCTCM

For all k, the sum of a particular row (column) in G_k is the sum of SED from a particular symbol to all the other symbols in the signal constellation. What we need to prove is that sum does not depend on that particular symbol.

In HDCTCM $(4, D)$ using 3-out-of-N simplex signaling scheme, there are 4^D symbols in IIS; any two of them occupy at least two different space dimensions. All the source simplexes related to all the symbols in IIS constitute the signal constellation. So, there are 4^D source simplexes in the signal constellation. Any two source simplexes occupy at least two different space dimensions. Each symbol in the signal constellation is a member of a source simplex.

Pick any symbol in this signal constellation as the reference symbol. Suppose this reference symbol is a member of source simplex I. All the other symbols can be categorized to the following types:

1. Symbols in the source simplex I. They are the other three member signals in source simplex I. It can be verified that in an even or odd source simplex, the SED profile from any member signal to the other three member signals is 8/8/8.
2. Symbols in the source simplexes other than source simplex I. Those simplexes occupy at least two different dimensions from source simplex I. According to the different

dimensions occupied by those simplexes from that occupied by source simplex I, these source simplexes can be divided into two sub-categories here.

2a. The number of different dimensions is two. Suppose source simplex I occupies dimensions 1, 2, and 3. Source simplex J occupies dimensions 1, 4, and 5. Simplex I can be an even or odd source simplex, as does simplex J. There are four combinations of the source simplex types for I and J. They are (I even, J even), (I even, J odd), (I odd, J even), and (I odd, J odd). Take (I even, J odd) as an example, then simplex I is $\begin{pmatrix} 1 & 2 & 3 \\ 1 & -2 & -3 \\ -1 & 2 & -3 \\ -1 & -2 & 3 \end{pmatrix}$, simplex J is $\begin{pmatrix} 1 & 4 & -5 \\ 1 & -4 & 5 \\ -1 & 4 & 5 \\ -1 & -4 & -5 \end{pmatrix}$.

The SED from any symbol in simplex I to the four symbols in simplex J is 4/4/8/8. It can be verified that for any of these four combinations, the SED profile from any symbol in simplex I to the four symbols in simplex J is 4/4/8/8, as is the SED profile from any symbol in simplex J to the four symbols in simplex I.

2b. The number of different dimensions is three. Suppose source simplex I occupies dimensions 1, 2, and 3. Simplex J occupies dimension 4, 5, and 6. In each of the four source simplex type combinations, the SED profile from any symbol in simplex I to the four symbols in simplex J is 6/6/6/6, as is the SED profile from any symbol in simplex J to the four symbols in simplex I.

From the reference symbol to all the other symbols in the signal constellation, the distance profile is a certain number of the distance profiles in these difference categories. There is always one and only one distance profile of category 1. Suppose the number of source simplexes in category 2a and 2b are n_1 and n_2 respectively, then $1 + n_2 + n_2 = 4^D$. If from any symbol, looking at all the other symbols in the signal constellation, there are same amount of n_1 and n_2, then the SED profile from any symbols is the same, but this is not necessary because all we want is the sum of the SED profile from the reference symbol. Note that the sum of the distance profile in each category is the same, i.e.,

$$8 + 8 + 8 = 4 + 4 + 8 + 8 = 6 + 6 + 6 + 6 = 24 \tag{3.6}$$

Therefore, the sum of the SED profile from the reference symbol is

$$1 \cdot (8+8+8) + n_1 \cdot (4+4+8+8) + n_2 \cdot (6+6+6+6) = 24 \cdot (1+n_1+n_2) = 24 \cdot 4^D$$

(3.7)

This sum does not depend on which symbol being chosen as the reference symbol. We have proved that the sum of SED from any symbol in the signal constellation to all the other symbols is the same; i.e., the sum of a row (column) of G_k is the same regardless the row (column) index. That is, G_k is uniform for all k, and then, E_k is uniform for all k. This will give that E is uniform.

In conclusion, HDCTCM$(4, D)$ using 3-out-of-N simplex signaling scheme is uniform.

CHAPTER 4 Computational Algorithm to Calculate the Minimum Distance

For an error control coding scheme, the minimum distance among all the code words plays an important role in determining the error correcting capacity of the code. This was discussed in Section 3.1. Computational algorithms [25] [26] [37] were developed to obtain the minimum distance for convolutional codes. The assumption made in these algorithms is the known starting state for a code word. This assumption is changed in the circular trellis situation. This chapter will develop a new algorithm to calculate the minimum distance for circular trellis coding. Section 4.1 gives the general description of this algorithm. Sections 4.2 and 4.3 give the derivation of the algorithm in detail. Section 4.4 analyzes the computational complexity of this algorithm. Section 4.5 shows the minimum distance structure of HDCTM obtained using this algorithm.

4.1 General Description of the Algorithm

Consider a circular trellis code $\mathrm{T}(n, D)$, as defined in Section 2.2. The encoder can be implemented using tail biting or circular trellis coding with permuted state structure. The total number of states in this trellis is $S = n^D$. For information sequence with length L (having $L\,k$-tuple information symbols when the encoder accept k-tuple inputs at one time), the corresponding code sequence is a trellis path with L stages. There are total n^L legal paths representing all the code words, $n^{(L-D)}$ paths for each possible starting state. A $\mathrm{T}(2,2)$ trellis is shown in Figure 4.1 with the paths starting from a certain state; the information sequence length is $L = 5$.

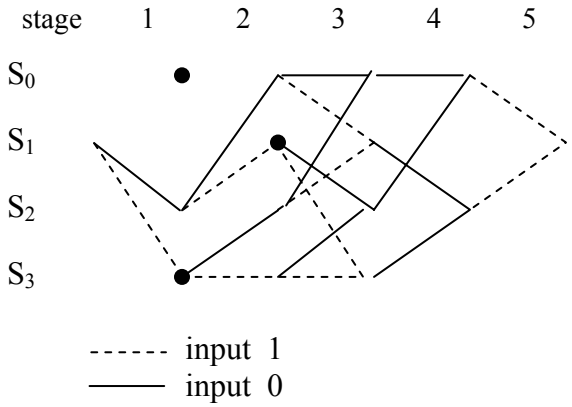


Figure 4.1 A circular trellis T$(2,2)$ with $L = 5$

For a coded modulation scheme, a branch in the trellis is labeled with a channel symbol assigned to the state transition represented by this branch. In this algorithm, distance is calculated as squared Euclidean distance (ED). Consider the uniform circular trellis codes. This kind of code allows consideration of a single reference path rather than all the pairs of paths for computation of distance.

From one path in a circular trellis, look at all the other paths; they can be divided into two categories:

1. The paths having the same starting state as the starting state of this path.
2. The paths having different starting state from the starting state of this path.

We are going to develop an algorithm with two parts to calculate the minimum distance related to these two kinds of paths. Part I (see Section 4.2) calculates the minimum distance among the paths that have a same starting state and identifies the pair of paths having this minimum distance between them. Part II (see Section 4.3) calculates the minimum distance between a path and all the other paths that have different starting state. Then the algorithm to calculate the minimum distance among all the legal paths has the following steps:

Algorithm: calculate the minimum distance of circular trellis T(n, D).

1. Select an arbitrary state from all S states. Using Algorithm Part I, calculate the minimum distance among the paths having this state as starting state and identify a pair of paths having this minimum distance between them.
2. Select either path from the identified pair of paths in step 1. Using Algorithm Part II, calculate the minimum distance between this selected path and all the other paths that have different starting states from the starting state of this selected path.
3. From the selected path in step 2, the minimum distance to the paths in the first category (defined above) has been obtained from step 1, the minimum distance to the paths in the second category has been obtained from step 2. Therefore the smaller one of these two is the minimum distance from the selected path to all the other legal paths in this trellis. For trellis codes with uniformity, it is the minimum distance among all the legal paths.

4.2 Algorithm--Part I: Calculate the Minimum Distance Among the Paths Having a Same Starting State

This algorithm is inspired by the application of Bellman's principle of optimality [24] in Viterbi algorithm and in the computational algorithm for calculating the free distance for convolutional codes [25] [37]. Bellman's principle of optimality is introduced in this section, followed by the derivation of the algorithm.

4.2.1 The Principle of Optimality

Denote this starting state as start_state. Name the summation of the distance of a trellis path at each stage up to stage i as the partial distance at stages i, where $1 \le i \le L$.

Bellman's principle of optimality pertains to a sequential decision problem whose global objective function is an additive function of costs of transitions between intermediate stages. The basic idea is that the global optimal path must be an extension of an optimal path to some state at an intermediate stage i, and this must hold for all stage indexes i. Here our globally optimal objective is a pair of paths having minimum distance between them among all the paths having start_state as their starting state. By applying Bellman's principle of optimality to a pair of paths, we claim that they must be the pair of paths that have minimum partial distance between them at stage i, among all

pairs of paths starting from start_state and reaching the same states as those reached by the globally optimal pair of paths at stage i. This holds for all i. The proof is by contradiction. Suppose the globally optimal pair of paths reach states p and q at stage j; if there is a second pair of paths reaching states p and q having less partial distance between them at this stage, then by replacing the segment of the global optimal pair of paths by the segments on the second pair of paths up to this stage, a new pair of paths is built which have less distance between them than that of the i globally optimal pair of paths. This contradicts with the definition of the globally optimal paths.

Denote the minimum partial distance between all pairs of paths starting from start_state and reaching states p and q at stage i as $\delta^i_{p,q}$, p, q can be any of the total states. The history of the pair of paths having distance $\delta^i_{p,q}$ between them is stored in $\lambda^i_{p,q}$. It is a string consisting of sequences of information symbols causing the state transitions on this pair of paths until stage i.

The argument above says, at each stage, for each pair of states p and q, all but one pair of paths having distance $\delta^i_{p,q}$ can be pruned away. Until the last stage L, the pair of paths kept for state pair p, q with $p = q = start\_state$ is the globally optimum pair of paths we seek.

4.2.2 Derivation of Algorithm--Part I

Let us show how to update those δ and λ. In general, there are n states from which a one-stage transition to state p is possible; call these states the predecessors of p denoted as p_l, $l = 1,\ldots,n$. In the case of parallel transitions, $p_l \to p$, for all l denotes all the n transitions that are caused by the n possible k-tuple information symbols. So in general, there are a total of n^2 pairs of paths reaching states p and q at stage i and they pass through the state pair $(p_l,\ p_m)$, $l,m = 1,\ldots,n$ at stage $i-1$. Then the minimum partial distance between all pairs of these paths is

$$\delta^i_{p,q} = \min\{\delta^{i-1}_{p_l,q_m} + d^2(p_l \to p, q_m \to q)\},\quad l,m = 1,\ldots,n. \tag{4.1}$$

The distances δ^{i-1} come from previous stage, $d^2(p_l \to p, q_m \to q)$ is the squared Euclidean distance between the two channel symbols associated with the transitions

$p_l \to p$ and $q_m \to q$. Suppose the minimum value is $\delta^{i-1}_{p_g,q_h} + d^2(p_g \to p, q_h \to p)$ in Equation (4.1) (whenever there is a tie, arbitrary choice applies); then $\lambda^i_{p,q}$ is updated by appending the k-tuple information symbols causing the state transitions denoted as $u(p_g \to p)$ and $u(q_h \to q)$, to $\lambda^{i-1}_{p_g,q_h}$; i.e.,

$$\lambda^i_{p,q} = \{\lambda^{i-1}_{p_g,q_h}, u(p_g \to p), u(q_h \to q)\} \tag{4.2}$$

Due to the pair-wise symmetry for distance, we only need to consider state pair (p,q) with $q \geq p$.

δ is initialized as

$$\delta^0_{p,q} = \begin{cases} 0, & q = p = start\_state \\ \infty, & p \neq start\_state, q \geq p \end{cases} \tag{4.3}$$

When $\delta^i_{p,q} = \infty$, there is no pair of paths reaching states p and q at stage i. All $\lambda^0_{p,q}$ are set as null sequence.

From start_state, D stages state transitions are needed to reach all the n^D states. For the first D-1 stages, at each stage the states that can be reached from start_state is a subset of the total states in this trellis. We only need to update $\delta^i_{p,q}$ and $\lambda^i_{p,q}$ where p and q belong to those reached states. There is no pair of paths that can reach states p and q, where p and q belong to those states that cannot be reached at these stages; i.e., $\delta^i_{p,q}$ is always ∞ and $\lambda^i_{p,q}$ is always a null sequence for those p and q. At stage D or later, all the states can be reached, so $\delta^i_{p,q}$ and $\lambda^i_{p,q}$, where p and q belong to all the states, need to be updated .

Unfortunately, this will not select the pair of paths we want. Let us see what we will get going through the trellis. Initially, $\delta^0_{start\_state, start\_state} = 0$. At stage 1, for any state p that can be reached at this stage, the term $\delta^0_{start\_state, start\_state} + d^2(start\_start \to p, start\_start \to p) = 0 + 0 = 0$ in Equation (4.1) will give $\delta^1_{p,p} = 0$. This means the pair of paths having minimum partial distance between them--among all the pair of paths starting from start_state and reaching state p--have the same route till this stage. At stage 2, for any state p that can be reached, the term

$\delta^0_{p_l,p_l} + d^2(p_l \to p, p_l \to p)$ in Equation (4.1) is equal to $0+0=0$, where p_l is one of the reached states in stage 1. This leads to $\delta^2_{p,p} = 0$. Continuingly in this way, we will get $\delta^i_{p,p} = 0$ for any state p that can be reached at stage i from start_state. When $i \geq D$, all the total states can be reached; i.e., for any state p of the total states, $\delta^i_{p,p} = 0$. Finally, $\delta^L_{s\_state,s\_state} = 0$; i.e., the minimum distance among all the paths starting from start_state is 0. The pair of paths having this minimum distance between them have the same route. It is one path that is regarded as a pair of paths. This happens, because when we derive Equation (4.1), we assume that a pair of paths contains two different paths. For the pair of states p and q with $p \neq q$, the two paths reaching different states are always two different paths. However, for $p=q$ there is no restriction regarding one path as a pair of paths reaching state p and q with $p=q$ at each stage in Equation (4.1), and that leads to the result aforementioned. We must put some conditions on updating $\delta^i_{p,q}$ with $p=q$ to eliminate this problem.

Consider how all the n^{L-D} paths are built in this trellis. (See Figure 4.1.) From start_state, it branches out n branches at stage 1, forming n different paths. Then each of the reached states at stage 1 branches out n branches at stage 2, forming n^2 different paths. In this way, each of the reached states at each stage will branch out n branches at the next stage, forming n times more different paths. So, at stage $L-D-1$, n^{L-D-1} different paths will be formed, and each of the reached state at this stage will branch out n branches at stage $L-D$, forming the entire n^{L-D} different paths. From stage $L-D$ until L, at each stage the reached state will not branch out to form new different paths but will have only one state transition to the next stage. Consequently, at each stage i, among all the pairs of paths both reaching state p, where $1 \leq i \leq L-D-1$ and p is any of the reached states in stage i, there is always a pair of paths having the same route till this stage. We know they are different paths, because all reached states will always branch out at the next stage; then, two paths having the same route till this stage will always have different route paths later. This pair of paths is represented by the term $\delta^{i-1}_{p_l,p_l} + d^2(p_l \to p, p_l \to p)$, where $\delta^{i-1}_{p_l,p_l} = 0$, in Equation (4.1). But at each stage i, where $L-D \leq i \leq L$, among all the pairs of paths both reaching state

p where p is any of the reached states in stage i, there is no pair of paths having the same route till this stage. Since each reached state at this stage will have only one state transition to the next stage, if they have the same route till stage i, they will have the same route afterwards, and they will just be the same path. So, all pairs of paths both reaching p at stage i must have different partial paths until this stage. They must either pass through different states at stage $i-1$ or, if they pass through the same state at stage $i-1$, the distance between them should be nonzero at that stage. This means the term $\delta^{i-1}_{p_l,p_l} + d^2(p_l \to p, p_l \to p)$ in Equation (4.1), where $\delta^{i-1}_{p_l,p_l} = 0$, does not correspond to a pair of different paths, and it should be eliminated. The correct updating equation for $\delta^i_{p,q}$, where $p=q$, will be

$$\delta^i_{p,q} = \min\left\{\delta^{i-1}_{p_l,q_m} + d^2(p_l \to p, q_m \to q), \quad p_l \neq q_m \quad or \quad if \quad p_l = q_m, \quad \delta^{i-1}_{p_l,q_m} \neq 0\right\} \quad (4.4)$$

Using this equation at stage $L-D$ will give $\delta^{L-D}_{p,p} \neq 0$ for all states p that can be reached at this stage, and will give $\delta^{L-D}_{p,p} = \infty$ for those p that cannot be reached. Then at stage $L-D+1$, when updating $\delta^{L-D+1}_{p,p}$ there will be no such term as $\delta^{L-D}_{p_l,p_l} + d^2(p_l \to p, p_l \to p)$ with $\delta^{L-D}_{p_l,p_l} = 0$ need to be eliminated. As a result, Equation (4.3) becomes Equation (4.1). So do later stages until stage L. In summary, the updating equation for $\delta^i_{p,q}$ with $p=q$, is Equation (4.1), when $i \neq L-D$, and Equation (4.4), when $i = L-D$.

4.2.3 Basic Steps for Algorithm--Part I

Now, we can give the basic steps for Algorithm--Part I:

1. Initialize δ^0 using Equation (4.1). All λ^0 are initialized as null sequences.
2. For each stage i, where $1 \le i \le L$, for any pair of states p and q with $q \ge p$, p and q both belong to the states that can be reached from start_state at this stage

   If $p \neq q$, update $\delta^i_{p,q}$ using Equation (4.1),

   else, if $i \neq L-D$, update $\delta^i_{p,q}$ using Equation (4.1), if $i = L-D$, using Equation (4.4).

   In either case, update $\lambda^i_{p,q}$ accordingly using Equation (4.2).

3. $\delta^{L}_{start\_state,start\_state}$ is the minimum distance among all the paths having start_state as starting state and ending states and $\lambda^{L}_{start\_state,start\_state}$ stores the pair of paths having this minimum distance between them.

4.3 Algorithm--Part II: Calculate the Minimum Distance Between a Path and All the Paths Having Different Starting State from the Starting State of This Path

Denote this particular path as base_path, and its starting state as base_state.

As shown in Section 2.4, Viterbi algorithm finds the path having the minimum distance to the received sequence. The Viterbi algorithm works for a trellis in which all the paths have the same stating state and ending state known *a priori*. Using Viterbi algorithm in circular trellis codes $\mathrm{T}(n, D)$ is to run Viterbi algorithm a total of n^D times--one run for each possible starting state. Regarding the base_path as the received sequence, each Viterbi run will yield the path having the minimum distance to the base_path, among all the paths having a certain starting state and the same ending state. The Viterbi run for base_state will always find the base_path with distance zero because the base_path is, in fact, a legal path among all the paths having base_state as starting and ending state. For our purpose, we do not need that since we already solved the problem of finding the minimum distance among all the paths having a same starting state in Algorithm--Part I. The other $n^D - 1$ Viterbi runs give the minimum distance between the base_path and the paths having different starting state from base_state. That is what we want.

So, we have the following steps for Algorithm--Part II:

1. Regard the base_path as the received sequence. For each of the states other than base_state, denoted as ε_j, where $1 \le j \le n^D - 1$, run Viterbi algorithm on the trellis having ε_j as starting state and ending state. We will get the minimum distance between the base_path and the paths having ε_j as starting state and ending state denoted as d_j.
2. $\min_j(d_j)$, $j = 1,...,n^D - 1$, is the minimum distance between the base_path and all the paths having different starting and ending state from base_state.

The Viterbi algorithm for ε_j is stated here. Denote the minimum partial distance between the base_path and all the paths entering state p at stage i as Λ_p^i, p is any of the states that can be reached from ε_j at this stage. We do not need to store the path history for our purpose. In general, there are n paths entering p at stage i, they pass through p_l at stage $i-1$, where p_l is one of the predecessors of state p. The updating equation for Λ_p^i is

$$\Lambda_p^i = \min\{\Lambda_p^{i-1} + d^2(p_l \to p, \textit{state transion on base\_path}), l = 1,..., n\} \tag{4.5}$$

where $d^2(p_l \to p, \textit{state transion on base\_path})$ is the squared Euclidean distance between the two channel symbols associated with the state transition $p_l \to p$ and the state transition on the base_path from stage $i-1$ to stage i.

The Viterbi run for state ε_j has the following steps:

1. Set $\Lambda_p^0 = \begin{cases} 0, & p = \varepsilon_j \\ \infty, & \textit{otherwise} \end{cases}$. (4.6)
2. At each stage i, where $1 \le i \le L$, for each of the state p that can be reached at this stage, update Λ_p^i using Equation (4.5).
3. $\Lambda_{\varepsilon_j}^L$ is the minimum distance between the base_path and all the paths having ε_j as starting and ending state.

4.4 Computation Complexity of the Algorithm

This section analyzes the computation complexity of the algorithm. Ways to reduce the computation load are also given.

4.4.1 Analyze the Computation Complexity of the Algorithm

In Algorithm--Part I, for a pair of states p and q each updating using Equations (4.1) and (4.4) involves the add-compare-select operations. Regard these operations as a basic operation. At each stage, at most $(S^2 + S)/2$ state pairs need to be updated, when p and q can be any of the total states and $q \ge p$. Therefore, the total number of operations needed is upper bounded by $(S^2 + S)/2 \cdot L$. It is linear on the second order of total state number S and the information sequence length L. In Algorithm--Part II, for a

state p, regard the add-compare-select operations in each updating using Equation (4.5) as a basic operation; the total number of operations needed is upper bounded by $S \cdot L$. It is linear on the first order of total number of states S. For the entire algorithm, the total number of operations is upper bounded by $(S^2 + 3S)/2 \cdot L$, although different computation loads may be involved in the basic operations for part I and part II. In practical implementation, the terms $d^2(p_l \to p, q_m \to q)$ in Equations (4.1) and (4.4) can be calculated only once and stored in a table, for all p, q, and p_l, q_m. So can the terms $d^2(p_l \to p$,state transion on ref_path) in Equation (4.5). Later, for each stage the d^2 terms can be retrieved through quick table lookups. Then if we only consider the additions as the computation load in each basic operator, the total number of computation is upper bounded by $(S^2 + S)/2 \cdot L \cdot n^2 + S \cdot L \cdot n$. The path memory needed for this algorithm is $(S^2 + S)/2 \cdot 2\log_2 n \cdot L$ bits.

4.4.2 Computation Reduction

Still some computation reduction can be done. For Algorithm--Part I, our goal is to get $\delta^L_{start\_state, start\_state}$ at stage L. From Equation (4.1), only $\delta^{L-1}_{p_l, q_m}$ from the previous stage are needed, where p_l and q_m belong to the predecessors of state start_state. In order to get these δ^{L-1}, only the δ^{L-2} of pair of states p and q are needed, where p and q both belong to the predecessors of p_l and q_m. p and q are the second level predecessors of start_state, i.e., the states from which a two-stage state transition to start_state is possible. The i_{th} level predecessors of start_state are all the states from which an i-stage state transition to start_state is possible. In general there are n^i states. The D_{th} level predecessors are the total S states in this trellis. At stage i, only δ^i of pair of states both belonging to the $(L-i)_{th}$ predecessors of start_state needs to be calculated (0_{th} predecessor is start_state itself) to get $\delta^L_{start\_state, start\_state}$. When $L - i < D$; i.e., $i > L - D$, the $(L-i)_{th}$ predecessors are a subset of the set of total states, so, for each stage i where $L - D + 1 \le i \le L$, only δ^i of a reduced number of pairs of states needs to be updated. The same is true for updating λ^i. Also, we have seen for stage i<L-D, there is always a pair of paths both reaching states p and having the same route until this stage.

That is to say, $\delta^i_{p,p} = 0$, for any state p that can be reached at this stage. Equation (4.1) will always give 0 if one actually calculates it. The updating of $\lambda^i_{p,p}$ for these p is in Equation (4.3), where $p_g = q_m$ and p_g is any one of the predecessors of p and was reached at stage $i-1$.

The revised version of Algorithm--Part I to reflect these reductions is:

1. Initialize as before.
2. For each stage i, where $1 \le i \le L$, get the set of states that will be considered.

   If $i \ge L - D + 1$, the set of states considered is the $(L-i)_{th}$ predecessors of start_state,

   else if $i < D$, it is the set of all the states that can be reached by a i-stage state transition from start_state.

   else it is the set of the total S states

   Then for each pair of states p and q with $q \ge p$, p and q both belong to the set of states considered,

   If $p \ne q$, update $\delta^i_{p,q}$ using Equation (4.1),

   else {

   if $i < L - D$, set $\delta^i_{p,q} = 0$,

   else if $i = L - D$, update $\delta^i_{p,q}$ using Equation (4.4) ,

   else update $\delta^i_{p,q}$ using Equation (4.1)

   }

   In each of these cases, update $\lambda^i_{p,q}$ accordingly using Equation (4.2).
3. Same as before.

The reduction of the computation for the last D stages is also applied for Algorithm--Part II. Then for each stage i, where $L - D + 1 \le i \le L$, only Λ^i_p, where p belongs to the $(L-i)_{th}$ predecessors of base_state, needs to be updated. The revision of it is obvious.

4.5 Minimum Distance of HDCTCM

Apply this algorithm to HDCTCM $(4, D)$. Denote the minimum distance obtained from step 1 in Algorithm (Section 4.1) as min_dis_1, the one obtained from step 2 as

$\min\_dis_2$, the minimum distance of this code is $\min\{\min\_dis_1, \min\_dis_2\}$. For each D, $D = 2,3,4,5$, for all different codes (different STM-FREE), $\min\_dis_1$ and $\min\_dis_2$ are calculated at different information length L. In Figure 4.2 to 4.5, the $\min\_dis_1(\min\_dis_2)$ versus L is plotted for some for some different codes for each D, $D = 2,3,4,5$. Distance is calculated as squared ED, signal has unit plus in each of the three dimensions it occupies, out of the total dimensions of the signal space.

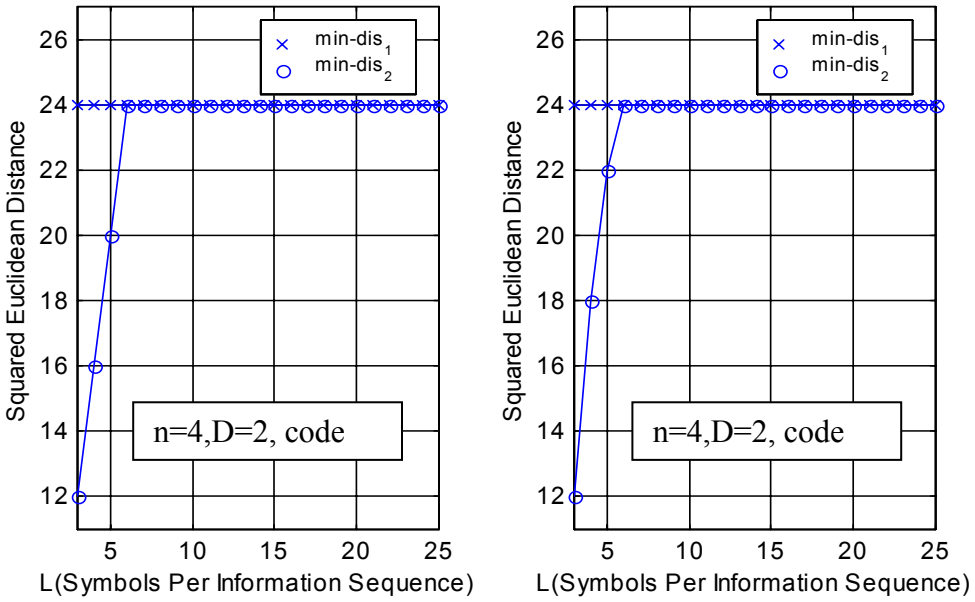


Figure 4.2 Minimum distance of HDCTCM$(4,2)$

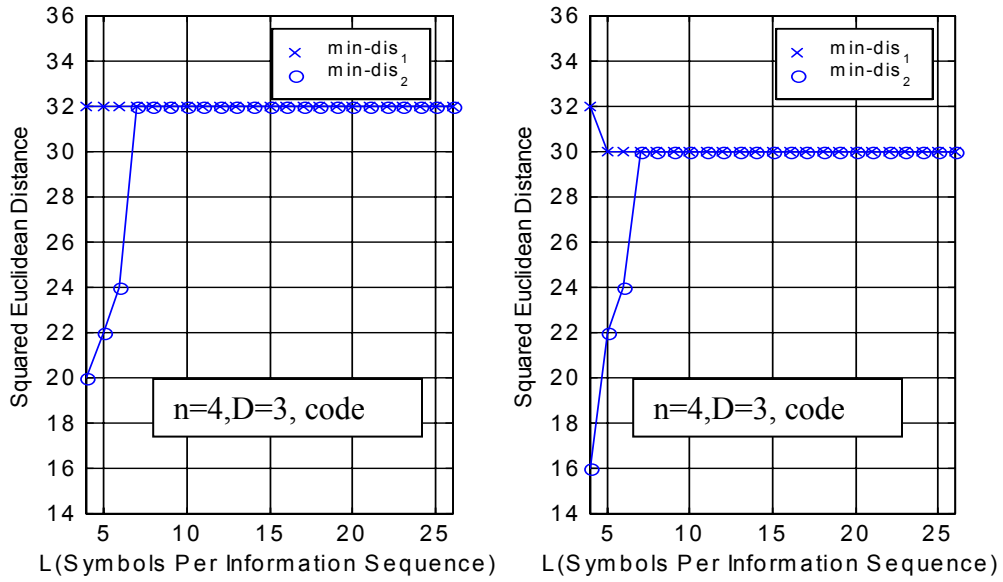


Figure 4.3 Minimum distance of HDCTCM(4,3)

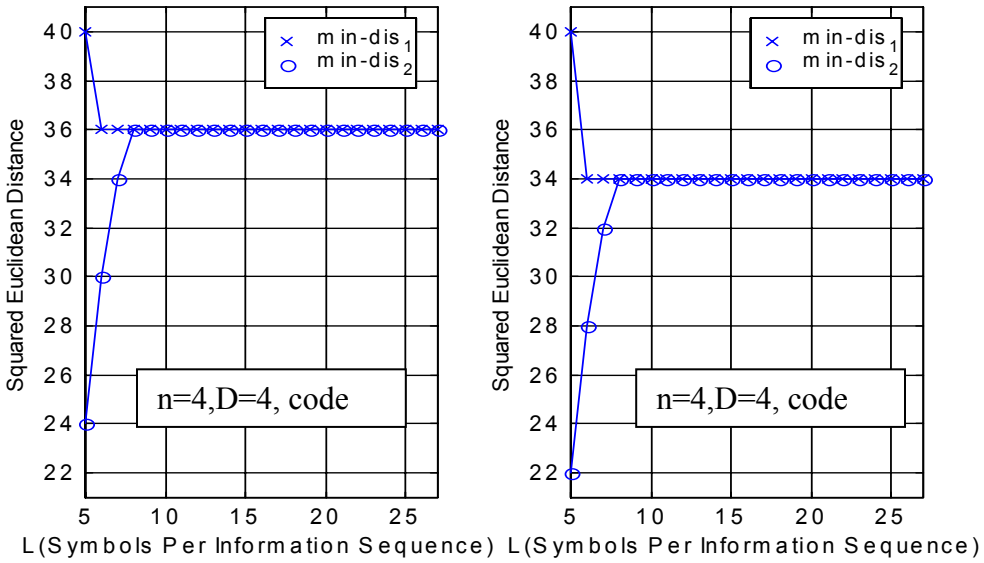


Figure 4.4 Minimum distance of HDCTCM(4,4)

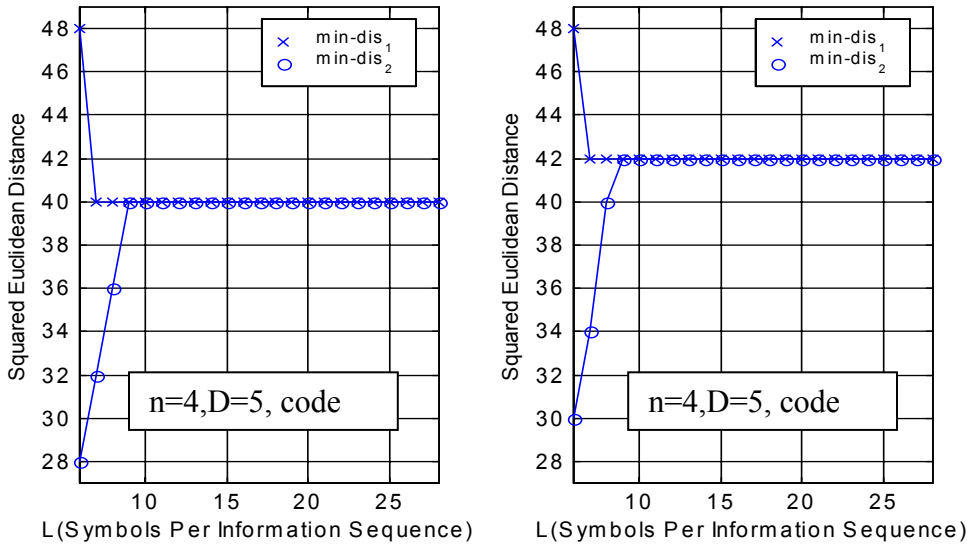


Figure 4.5 Minimum distance of HDCTCM(4,5)

For all the cases calculated, the following trend is observed. For min_dis_1, when $L = D+1$, it is $8\cdot(D+1)$. This verifies the correctness of this algorithm. At $L = D+1$, the state transitions on the n paths related to a certain starting state at each stage are assigned symbols that construct simplex; this is how we build the codes. The distance between simplex signals is 8, so the distance between these paths is $8\cdot(D+1)$. At $L = D+2$, min_dis_1 is another value that can be different from $8\cdot(D+1)$. For $L > D+2$, it keeps the same value as the value at $L = D+2$.

For min_dis_2, it increases when L increases from the smallest value D+1 and eventually reaches the value same as min_dis_1 at $L = D+2$. This happens when $L \approx 2D+(0\sim 2)$, where $0\sim 2$ denotes an integer between 0 and 2. For larger L, min_dis_2 remains at this value.

In practical implementation, $L > 2D+(0\sim 2)$. Therefore, for HDCTCM$(4, D)$ with a given D and $L > 2D+(0\sim 2)$, the minimum distance for a code is equal to min_dis_1 at $L = D+2$. From these figures, different codes (corresponding to different STM-FREE) of HDCTCM$(4, D)$ for a given D can give different values for min_dis_1 at $L = D+2$. But a tight upper and lower bounds on this value--the minimum distance of

HDCTCM$(4, D)$--exist. The validity of this claim will be proved in the following chapter.

CHAPTER 5 BOUNDS ON THE MINIMUM DISTANCE OF HDCTCM AND OPTIMUM DISTANCE CODE

One central problem of coding theory has been the seeking of codes with large (or largest) minimum distance for a given code parameters [24]. For most coding schemes, the optimum distance codes are found only by exhaustive search of all the possible cases. For a code with a given code parameter, if there are a large number of codes, exhaustive search is forbidden and an upper and lower bounds on the minimum distance is desired to evaluate the error correcting capacity of this code. In this chapter, an upper and lower bounds on the minimum distance of practical HDCTCM $(4, D)$ codes are derived in Section 5.1. Section 5.2 proves that the bounds can be reached. Moreover, Section 5.3 designs a procedure other than the exhaustive search method to obtain the optimum distance code for HDCTCM$(4, D)$.

5.1 Bounds on the Minimum Distance of HDCTCM

The last chapter shows that for a given trellis depth D, the minimum distance of a HDCTCM$(4, D)$ code with information sequence length L, where $L > 2D + (0 \sim 2)$ and $0 \sim 2$ denotes an integer between 0 and 2, is equal to the minimum distance among the paths associated to a certain starting state at $L = D + 2$. For a given D, the signal constellation for HDCTM$(4, D)$ codes is fixed; different codes correspond to mapping the signal constellation to different STMs built by taking different STM-FREEs. From Figures 4.2 to 4.5, different codes will give different minimum distances. Bounds on the minimum distance of practical HDCTCM $(4, D)$ codes for a given D are derived in this section.

5.1.1 Distance Structure of Signal Constellation

In this chapter, all distances refer to the squared Euclidean distances (SED). From building IIS in Chapter 3, we can see that for two symbols in IIS, if they are related by "copy" rule, they will occupy the same space dimension for one pulse-location and occupy different space dimensions for the other two pulse-locations. If they are not related by "copy" rule, they will occupy different space dimensions for all three pulse-locations. All source simplexes related to all the symbols in IIS constitute the signal constellation. It can be verified that the distance between any two member signals in a source simplex or a type B simplex is 8. So, any two symbols in the signal constellation can be members of a source simplex, and then the distance between them is 8. Or they can come from two different source simplexes; then they will occupy at least two different dimensions for two pulse-locations. For the other pulse-location, they can be the same absolute value with opposite sign, the same absolute value with same sign, or different absolute value. For the first case, these two symbol are members of a type B simplex, then the distance between them is 8; for the second case, such as signal (1, 3, 5) and signal (1, -2, -4), the distance between them is 4; for the third case, such as signal (1, 3, 5) and signal (2, 4, 6), the distance between them is 6.

Each of all the states in HDCTCM$(4, D)$ is related to a unique symbol in IIS. The source simplex related to this symbol is assigned to the state transitions originating from this state. So, the symbols assigned to state transitions originating from different states are drawn from different source simplexes. Therefore, these symbols occupy at least two different dimensions for two pulse-locations. A simplex-transition$_1$ is assigned a source simplex. A simplex-transition$_i$, $2 \le i \le D+1$, is assigned a type B simplex.

5.1.2 Path Groups

For HDCTCM $(4, D)$ with $L = D + 2$, there are $4^2 = 16$ paths related to each possible starting state. (See Figure 5.1.) These paths can be divided into four groups and each group has four paths, which have a same state transition at stage 1. Denote each of these groups as an initial-group. These paths can be divided into four other groups at the same time, and each group has four paths which have the same state transition at stage $D + 2$. Denote each of these groups as a terminal-group. From stage 2 till stage D, each

path reaches different states. At stage $D+1$, the four paths in a terminal-group will reach one of the four states that can transit to the ending state, which is the same as the starting state at stage $D+2$. Each of the four paths in a terminal-group belongs to a different initial-group and vice visa. Number these paths using integers from 1 to 16 according to the following principle: each initial-group is given the four consecutive numbers $\{4 \cdot m+1,\ ....,\ 4 \cdot (m+1)\}$, $m = 0,\ 1,\ 2,\ 3$, and each terminal-group is given an arithmetic progression sequence with an increment of 4, $\{l,\ l+4,\ l+4 \cdot 2,\ l+4 \cdot 3\}$, $l =$ 1, 2, 3, 4. (See Figure 5.1.) By this numbering, each path in an initial-group belongs to a different terminal-group and vice visa. The path number will identify the state transition on this path at each stage as well as the butterfly containing this state transition. Therefore, we will simply express "the state transition on a path at a certain stage" as "a path at a certain stage". Similarly, express "the state transitions on the four paths in a group at a certain stage" as "the state transitions in a group at a certain stage" or just as "a group at a certain stage".

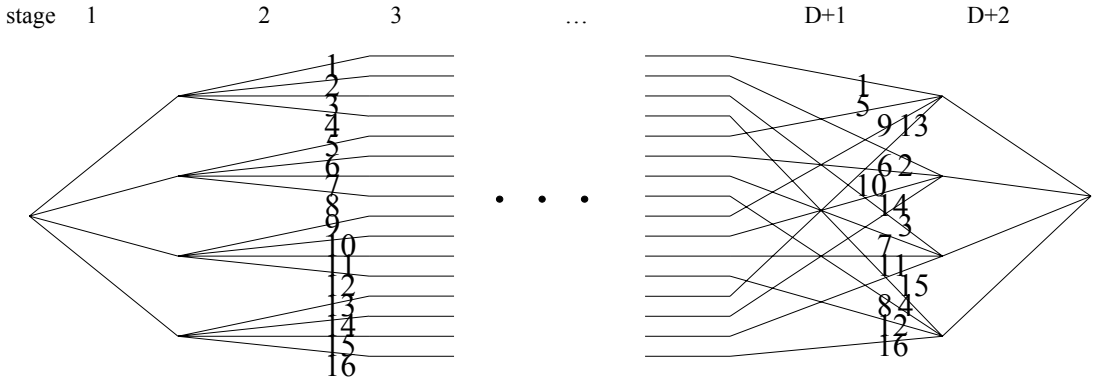


Figure 5.1 The 16 paths related to a certain starting state for HDCTCM$(4, D)$ at $L = D+2$

Comparing the 4 transitions in a group at each stage with the sets of 4 transitions constituting simplex-transitions, the following pattern is observed:

For the 4 transitions in an initial-group: At stage 1, they are the same from definition. At stage 2, they have a same originating state; i.e., they constitute a simplex-transition$_1$. At stage i, where $3 \le i \le D+1$, they are one set of 4 transitions constituting simplex-transition$_{i-1}$. At stage $D+2$, they have a same next state; i.e., they constitute a simplex-transition$_{D+1}$.

For the 4 transitions in a terminal-group: At stage 1, they have a same originating state; i.e., they constitute a simplex-transition$_1$. At stage i, where $2 \le i \le D$, they are one set of 4 transitions constituting simplex-transition$_i$. At stage $D+1$, they have a same next state; i.e., they constitute a simplex-transition$_{D+1}$. At stage $D+2$, they are the same from definition.

5.1.3 Distance Property of the Paths

The distance of a path always refers to the distance between this path and a reference path. The distance of a path at a certain stage is the distance between the symbols assigned to the state transitions at this stage on this path and the reference path. The distance of a path is the summations of its distances at all stages.

Recall that all the state transitions are recorded in STM, a $D-1$ dimensional matrix. The unit element in STM is a butterfly with an internal order. A simplex-transition$_1$ or a simplex-transition$_{D+1}$ is within a butterfly. A simplex-transition$_i$, where $2 \le i \le D$, is the set of the 4 transitions in the same internal locations of the 4 butterflies along the $(i-1)_{th}$ dimension of STM. Each of the originating states g in STM is related to a symbol s in IIS that has the same location as the location of state g in STM. The source simplex related to symbol s is assigned to the transitions orientating from state g. Knowing the state transition's location in STM, the symbol from the signal constellation assigned to this transition will be uniquely known. So, these 16 paths can be depicted in STM stage by stage, and the symbols assigned to these paths at each stage will be known. Then the distances of these paths at each stage can be analyzed.

Pick any of these 16 paths as the reference path, denoted as reference-path. Then the initial-group and terminal-group containing the reference path are known, denoted as reference-initial-group and reference-terminal-group, respectively. Denote the symbol assigned to the reference-path at a certain stage as reference-symbol. Among the 15 paths other than the reference-path, there are 3 paths in the reference-initial-group, denoted as Γ_i, $i = 1, 2, 3$, and there are 3 paths in the reference-terminal-group, denoted as Π_i, $i = 1, 2, 3$. For the remaining 9 paths, neither in the reference-initial-group nor in the reference-terminal-group, there are 3 paths in each of the initial-groups (terminal-groups) other than the reference-initial-group (reference-terminal-group), denoted as P_i, $i = 1, 2, \ldots, 9$.

Stage 1: These 16 paths originate from a same state and reach 4 different states. They are within a butterfly. (See Figure 5.2(a).) A box represents a butterfly. The 4 transitions in an initial-group are the same while those in a terminal-group constitute a simplex-transition$_1$. The originating state of these 16 paths is related to a unique symbol in IIS, the related source simplex to that symbol is assigned to the simplex-transition$_1$ at this stage. Denote the symbols assigned to the four transitions from top-to-bottom in STM at this stage as s_1, s_2, s_3 and s_4. Then $\{s_i \mid i = 1,2,3,4\}$ is a source simplex.

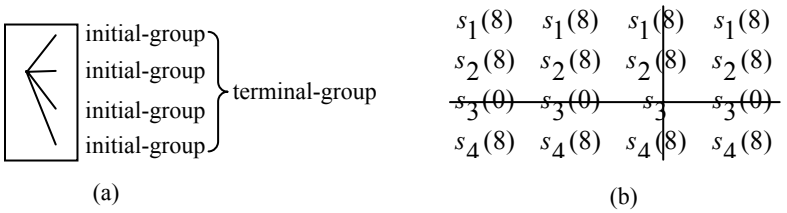


Figure 5.2 The 16 paths in STM and corresponding symbol matrix at stage 1 for HDCTCM$(4, D)$ with $L = D + 2$

To illustrate the distances of these 16 paths, we place the symbols assigned to these paths at each stage into a 4×4 matrix. A symbol is a unit element in this matrix. Place the symbols assigned to an initial-group into a row and place the symbols assigned to a terminal-group into a column in this matrix. The symbols assigned to the four initial-groups (terminal-groups) located in STM from top-to-bottom or from left-to-right are placed into the first to fourth row (column). This matrix is called the symbol matrix. We see that there is one to one correspondence between the locations of these 16 paths in STM and the locations of the symbols assigned to these paths in the symbol matrix. Later, we will say a path is at a certain location in the symbol matrix, as well. That means the symbol assigned to this path is at that location in the symbol matrix. The distance from a symbol to the reference-symbol in the symbol matrix will give the distance from the corresponding path to the reference-path.

The symbol matrix at this stage is shown in Figure 5.2(b). The distance between s_j and s_k, $j, k = 1, 2, 3, 4$ is 0 when $j = k$, and 8 when $j \neq k$. Pair-wise distances

between symbols are all known. So, we do not need to know the actual values in the symbol s_j. The location of the reference-symbol in the symbol matrix will fix the distance from each of the other 15 symbols to the reference-symbol. (See Figure 5.2(b).) Later, the distance between a symbol and the reference symbol will be referred to as the distance of a symbol. Denote the row and column containing the reference symbol as the reference-row and reference-column. In Figure 5.2(b), the crossed out row and column are the reference-row and reference-column, respectively. The intersection is the reference-symbol. The distance of a symbol is written in the bracket besides the symbol. The matrix resulting by replacing a symbol in a symbol matrix with its distance is called a distance matrix. No matter where the reference-symbol is located at--among the 15 symbols other than the reference-symbol--the distance is 0 for all the three symbols in the reference-row. For all three symbols in the reference-column, the distance is 8. For the other 9 symbols, the distance is 8.

Therefore, the distances of the 15 paths other than the reference-path have the following property: the distances of the 3 paths in the reference-initial-group are all 0; the distances of the 3 paths in the reference-terminal-group are all 8; and the distances of all the other 9 paths are all 8.

Stage 2: These 16 paths relate to 4 originating states; an initial-group constitutes a simplex-transition$_1$. They are in a butterfly. A terminal-group constitutes a simplex-transition$_2$; they are located in the same internal locations of a set of 4 butterflies along the first dimension of STM. (See Figure 5.3(a).) The set of 4 paths having the same line style is a terminal-group.

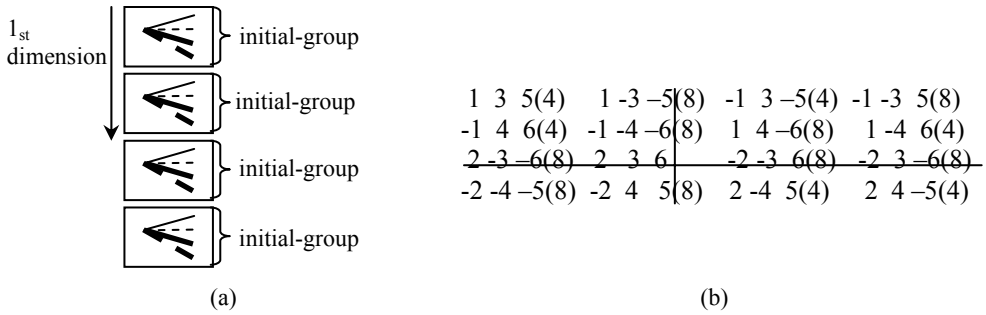


Figure 5.3 The 16 paths in STM and corresponding symbol matrix at stage 2 for HDCTCM$(4, D)$ with $L = D + 2$

As shown in Chapter 3, the set of the four signals s_i, $i = 1, 2, 3, 4$, related to these four originating states g_k, $k = 1, 2, 3, 4$, is a type B simplex θ. The symbol assignment is made such that four transitions in the simplex-transition$_1$ originating from state g_k are assigned member signals from the related source simplex to signal s_i, denoted as ϖ_i. Also, each of these four simplex-transition$_2$'s is assigned a type B simplex ξ_l, $l = 1,2,3,4$ formed by the four source simplexes ϖ_i. So in the symbol matrix for this stage, each row will be a source simplex ϖ_i and each column a type B simplex ξ_l. The "prototype" symbol matrix is shown in Figure 5.3 (b). Numbers 1 to 6 are used to indicate different space dimensions. By "prototype" we mean, an actual symbol matrix can be any row or (and) column exchanged. That variation results from different absolute locations of these 16 paths located in different STMs. Correspondingly, signals in different type B simplex θ (identified by the negative sign locations in its member signals) will be related to the four originating states g_k, that will give different ϖ_i and ξ_l. For a given type B simplex θ, there is one and only one arrangement for ϖ_i to form four ξ_l. It can be verified that for all different type B simplexes θ, the symbol matrixes are the "prototype" symbol matrix with some row or (and) column exchanged.

In order to obtain the distances of the symbols in the symbol matrix, we need to know the actual symbol matrix and the reference-symbol's location. For the "prototype" symbol matrix, no matter where the reference-symbol is located--among the other 15

symbols other than the reference symbol--all three symbols in the reference-row or the reference-column have distance 8, because the symbol assigned to each of these paths is a member signal of the source simplex or the type B simplex containing the reference-symbol. Cross out the reference-row and reference-column; for the symbols in the left 3×3 matrix, their distances are either 4 or 8. There is one and only one distance 8 in each row (column), and the column (row) index of distance 8 in each row (column) is different. (See Figure 5.3(b).) Denote each of these three locations where the symbol located at these locations have distance 8 as a special-8-location. For a given row and column indexes of the reference-symbol in a symbol matrix, exchange some rows or (and) columns in this "prototype" symbol matrix; the resultant distance matrix will be the distance matrix obtained from the "prototype" symbol matrix with those rows or (and) columns exchanged. We can see the distance property described above will not change for these resultant distance matrixes. That is to say, any actual symbol matrix has the distance property described above.

Therefore, at this stage, the following property of the 15 paths other than the reference-path holds: the distances of the 3 paths in the reference-initial-group and the 3 paths in the reference-terminal-group are all 8. Among the other 9 paths P_i, $i = 1, 2, \ldots, 9$, each initial-group (terminal-group) contains one path having distance 8 and two paths having distance 4. Each of the three paths having distance 8--denoted as a special-8-path--belongs to a different terminal-group (initial-group). We name the distance property at this stage as Distance Rule 1.

Stage 3 to D: At stage i, where $3 \le i \le D$**,** these 16 paths originate from 16 different originating states and reach 16 different next states. An initial-group constitutes a simplex-transition$_{i-1}$; they are located in the same internal locations of the set of 4 butterflies along the $(i-2)_{th}$ dimension in STM and have the same location indexes along all the other dimensions. Four initial-groups are in 4 sets of 4 butterflies along the $(i-2)_{th}$ dimension. A terminal-group constitutes a simplex-transition$_i$. They are located in the same internal locations of the set of 4 butterflies along the $(i-1)_{th}$ dimension in STM and have the same location indexes along all the other dimensions. Four terminal-groups are in 4 sets of 4 butterflies along $(i-1)_{th}$ dimension. Each of the 4 paths from an

initial-group belongs to a different terminal-group and vice visa. So, these 16 paths are in 16 different butterflies. These butterflies are located in a two-dimensional plane along the $(i-1)_{th}$ and $(i-2)_{th}$ dimension of STM. (See Figure 5.4(a).)

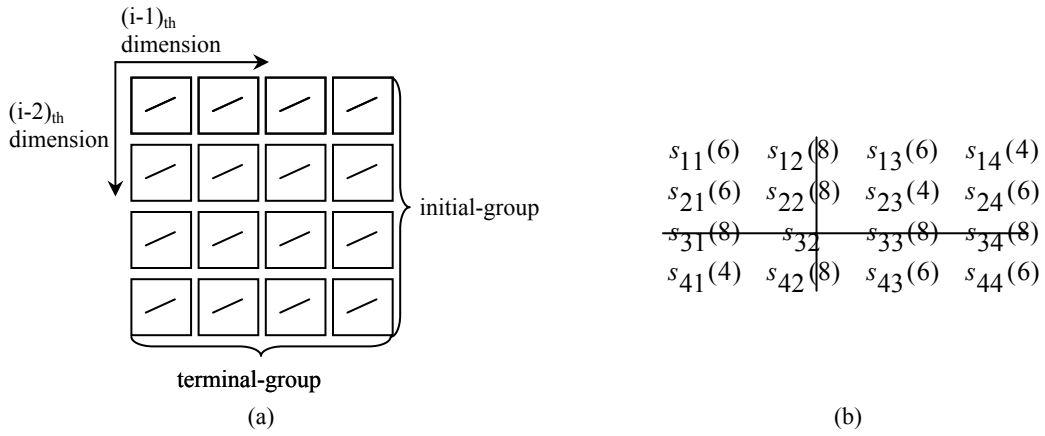


Figure 5.4 The 16 paths in STM and corresponding symbol matrix at stage i, where $3 \le i \le D$, for HDCTCM$(4, D)$ with $L = D + 2$

A simplex-transition$_{i,}$ where $2 \le i \le D$, is assigned a type B simplex. State transitions originating from different states will be assigned symbols occupying at least two different dimensions for two pulse-locations. So the symbol matrix at this stage should have the following property: the set of symbols in each row and each column is a type B simplex. And all the symbols occupy at least two different dimensions for two pulse-locations. Call this property "the property of symbol matrix".

Denote the symbol in row j and column k, $j,\ k = 1,\ 2,\ 3,\ 4$ as s_{jk}. (See Figure 5.4(b).) Then for each j, $\{s_{jk} \mid k = 1,\ 2,\ 3,\ 4\}$ is a type B simplex and for each k, $\{s_{jk} \mid j = 1,\ 2,\ 3,\ 4\}$ is a type B simplex. The "copy" rule should be satisfied in any type B simplex. We will see that we do not need to know the actual values in symbol s_{jk}; knowing the location of the reference-symbol in this symbol matrix will give all the symbols' distances.

Denote the row index or the column index of the symbol matrix as r_l or c_l, where $l = 1, 2, 3, 4$ and $r_l, c_l \in \{1, 2, 3, 4\}$. Suppose the reference-symbol is in (r_1, c_1). The distances of the other three symbols in row r_1 and column c_1 will all be 8, because each of these symbols is a member signal of a type B simplex containing the reference-symbol. These symbols are related to the reference-symbol by "copy" rule for pulse-location$_m$, $m = 1, 2, 3$ along the reference-row or the reference-column.

Cross out the reference-row and the reference-column. In the left 3×3 matrix, for each pulse-location$_m$ in the reference-symbol, apply "copy" rule twice--once along the row and once along the column. A symbol can be identified to have distance 4. That is because twice "copy" will make pulse-location$_m$ in this identified symbol have the same absolute value and the same sign as pulse-location$_m$ in the reference-symbol. Each of the other two pulse-location$_p$, $p \neq m$, occupies different dimensions in this identified symbol and the reference symbol. So the distance of the identified symbol is 4.

Let us see how to identify those three symbols. Recall that the "copy" rule is defined in matrix pair in Equation (3.2). For a given signal location, its pair locations for pulse-location$_m$, $m = 1, 2, 3$ and pulse-location$_p$, where $p \neq m$, are different. First, in column c_1, for each pulse-location$_m$ in the reference-symbol, find the pair location of the row index r_1, denoted as r_{m+1}; then $r_{m+1} \neq r_1$ and $r_2 \neq r_3 \neq r_4$. Secondly, in each row r_{m+1}, find the pair location of the column index c_1 for pulse-location$_m$, denoted as c_{m+1}; then $c_{m+1} \neq r_1$, and $c_2 \neq c_3 \neq c_4$. The distance of the symbol in (r_{m+1}, c_{m+1}) is 4. That is to say, in each of the three rows, r_{m+1}, there is one symbol having distance 4 and the column indexes of these three symbols, c_{m+1}, are different. Denote each (r_m, c_m) as a special-4-location. For the remaining 6 symbols in the 3×3 matrix, they are not related to the reference-symbol by "copy" rule either once (in the reference-row or reference-column) or twice; they occupy three different dimensions from the reference-symbol. Their distances will all be 6. (See Figure 5.4 (b).)

In summary, at this stage the distances of the 15 paths other than the reference-path have the following property: the 3 paths in the reference-initial-group and the 3 paths in the reference-terminal-group all have distance 8. For the other 9 paths P_i, $i = 1, 2, \ldots, 9$, each initial-group (terminal-group) contains one path having distance 4

and two paths having distance 6. Each of the three paths having distance 4--denoted as a special-4-path--belongs to a different terminal-group (initial-group). We name the distance property at this stage as Distance Rule 2.

Stage $D+1$**:** These 16 paths originate from 16 different states and reach 4 different next states. An initial-group constitutes a simplex-transition$_D$. They are located in the same internal locations of the set of 4 butterflies along the $(D-1)_{th}$ dimension of STM. A terminal-group constitutes a simplex-transition$_D$. They are in a butterfly. (See Figure 5.5 (a).)

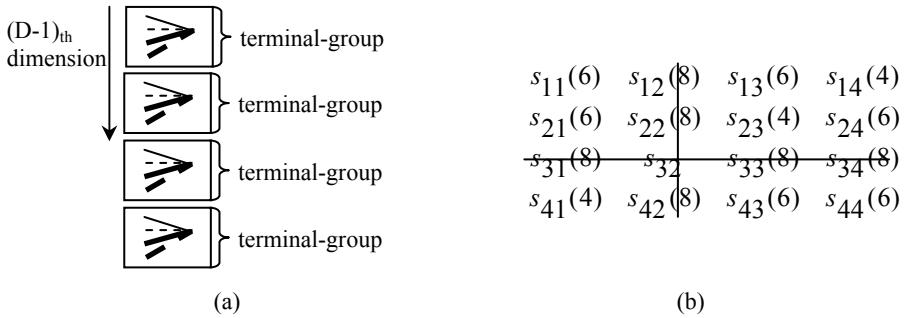
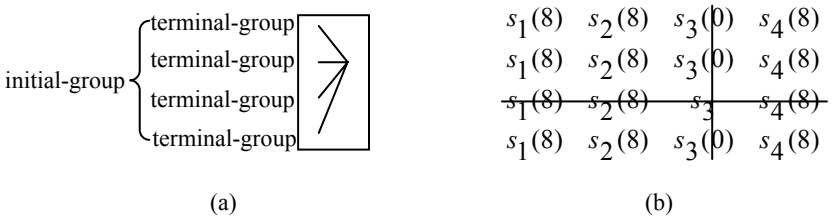


Figure 5.5 The 16 paths in STM and corresponding symbol matrix at stage $D+1$ **for HDCTCM**$(4, D)$ **with** $L = D+2$

A simplex-transition$_D$ or a simplex-transition$_{D+1}$ is assigned a type B simplex. So in the symbol matrix at this stage, the set of the 4 symbols in each row or in each column is a type B simplex, and all the symbols occupy at least two different dimensions for two pulse-locations. Therefore, the symbol matrix at this stage satisfies "the property of symbol matrix". The symbol matrix can be denoted as the same as the symbol matrix at stage 3 to D. (See Figure 5.5 (b).) Then the distance property of the 15 paths other than the reference-path will be the same as the distance property derived for stage 3 to D. That is to say, Distance Rule 2 is also satisfied at this stage.

Stage $D+2$**:** These 16 paths originate from 4 different states and reach a same next state. They are in a butterfly. An initial-group constitutes a simplex-transition$_{D+1}$ and the 4 transitions in a terminal-group are the same. (See Figure 5.6 (a).)



(a) (b)

Figure 5.6 The 16 paths in STM and corresponding symbol matrix at stage $D+2$ for HDCTCM$(4, D)$ with $L = D+2$

A simplex-transition$_{D+1}$ is assigned a type B simplex. Denote the symbols assigned to the 4 transitions in the simplex-transition$_{D+1}$ at this stage from top-to-bottom as s_1, s_2, s_3, and s_4. Then $\{s_i \mid i = 1, 2, 3, 4\}$ is a type B simplex and the distance between s_j and s_k, $j, k = 1, 2, 3, 4$ is 0 when $j = k$, and 8 when $j \neq k$. The symbol matrix is shown in Figure 5.6 (b). No matter where the reference-symbol is located, among the 15 symbols other than the reference-symbol, the three symbols in the reference-row all have distance 8 and the three symbols in the reference-column all have distance 0. The distance for the other 9 symbols is 8.

Therefore, the distances of the 15 paths other than the reference-path have the following property: the distances of the 3 paths in the reference-initial-group are all 8, the distance of the 3 paths in the reference-terminal-group are all 0, and the distances of all the other 9 paths are all 8.

5.1.4 Upper and Lower Bounds on the Minimum Distance

From the analysis of the distances of the 15 paths other than the reference-path, we can see:

1. Each of the 3 paths in the reference-initial-group and the 3 paths in the reference-terminal-group has distance 8 at $D+1$ stages and distance 0 at one stage. So the distance of each of these paths is $8 \cdot (D+1)$.

2. For each of the other 9 paths P_i, $i = 1, 2, \ldots, 9$, at stage 1 and stage $D+2$ the distance is 8. At stage 2, the distance is either 8 or 4. At stage i, $3 \le i \le D$, the distance is either 4 or 6. The distance of path P_i can be written as $8+8+H_i = 16+H_i$, where $H_i = \sum_{j=2}^{D+1} d_{ij}$, d_{ij} is the distance of path P_i at stage j. Since the smallest value d_{ij} can take is 4, the minimum H_i among all i is lower bounded by $4D$ and denoted as $H_{\min}$. The lower bound on the minimum distance of these 15 paths is

$$\min\{8\cdot(D+1),\ 16+H_{\min}\} = \begin{cases} 8\cdot(D+1), & D<3 \\ 16+4\cdot D, & D\ge 3 \end{cases} \tag{5.1}$$

It also can be seen from the Distance Rule 1 and Rule 2,

$$\sum_{i=1}^{9} H_i = 3*[(8+4+4)+(6+6+4)*(D+1-3+1)] = 48\cdot D \tag{5.2}$$

So, the minimum H_i among all i is also upper bounded by a particular value. This happens when all the H_i's take values as large as possible, but all larger than that particular value. If there is no other restriction, this happens when all the H_i's are equal. But since the distance can only be an even integer, that particular value is $\left\lfloor \frac{48D}{9} \right\rfloor_{even} = \left\lfloor \frac{16D}{3} \right\rfloor_{even}$. Denote it as $H_{\max}$.

Then the upper bound of the minimum distance of these entire 15 paths is

$$\min\{8\cdot(D+1),\ 16+H_{\max}\} = \begin{cases} 8\cdot(D+1), & D<3 \\ 16+\left\lfloor \frac{16D}{3} \right\rfloor_{even}, & D\ge 3 \end{cases} \tag{5.3}$$

The state transition pattern of the 16 paths shown above does not depend on the starting state, so the distance property will be the same for the 16 paths associated with any possible starting state. Also, the distance property derived above does not depend on which path among these 16 paths one chooses as the reference-path. Therefore, the upper and lower bounds derived are the bounds on the minimum distance among the 16 paths associated with any starting state for HDCTCM$(4, D)$ at $L = D+2$. That is to say, the bounds are the bounds on the minimum distance of HDCTCM$(4, D)$ codes with $L > 2D + (0 \sim 2)$.

5.1.5 Coding Gain

From the bounds of the minimum distance of HDCTM$(4, D)$, we can evaluate the coding gain of HDCTM$(4, D)$codes. Take uncoded 4-PSK as a reference system. Coding gain is defined in Equation (2.5). For HDCTM$(4, D)$, if using 3-out-of-N signaling scheme with unit pulse in each occupied dimension, $S_{av} = 3$. For uncoded 4-PSK, $d^2_{\min} = 2$ when $S_{av} = 1$. When HDCTM$(4, D)$ codes have the upper and lower bounds on the minimum distance, the maximum and minimum coding gain can be calculated. Table 5.1 shows the results for $D = 1,\ 2,\ 3,\ 4,\ 5$. These coding gains will be achieved at high signal-to-noise ratio.

Table 5.1 Minimum and Maximum Coding Gains of HCTCM$(4, D)$ Codes

Trellis depth D	Minimum coding gain	Maximum coding gain
1	4.26	4.26
2	6.02	6.02
3	6.69	7.27
4	7.27	7.78
5	7.78	8.45

5.2 Tightness of the Bounds on the Minimum Distance

This section proves that the upper and lower bounds on the minimum distance derived in Section 5.1.4 can be reached. First, we show that to prove that the bounds can be reached is to find a valid distance that can lead to the bounds. Then we prove a theorem on the conditions for a distance distribution to be valid. Following that, valid distance distributions that can reach the bounds are built.

5.2.1 Conditions for the Bounds to be Reached

For a given D, for HDCTCM$(4, D)$ codes at $L = D + 2$, there are 16 paths related to a certain starting state. The derivation of the distance property among these 16 paths at each stage does not depend on a particular STM. Regardless of different STMs (different codes), among those 15 paths other than the reference-path, the 3 paths in the reference-initial-group always have a same fixed distance at each stage from stage 1 to $D + 2$, as do the 3 paths in the reference-terminal-group. The other 9 paths P_i, $i = 1, 2, \ldots, 9$, always have a same fixed distance at stage 1 and $D + 2$ and comply with Distance Rule 1 at stage 2 and Distance Rule 2 at each stage from stage 3 to $D + 1$. Distance Rule 1 and Rule 2 do not give a fixed value for all these 9 paths at each stage from stage 2 to $D + 1$. A particular path P_i can have distance either 8 or 4 at stage 2 and can have distance either 4 or 6 at each stage from stage 3 to $D + 1$. This depends on the location of the symbol assigned to P_i at that stage; i.e., it depends on the path's location in STM. For different STMs, the absolute locations of these 16 paths in STM will be different. So the distance of a particular path P_i at stages from 2 to $D + 1$ can be different in different STMs. Name the distances of some paths as the distance distribution of these paths; this means different STMs can give different distance distribution of the 9 paths P_i, $i = 1, 2, \ldots, 9$. From the derivation of bounds, the minimum distance of the 9 paths, P_i, determines the minimum distance among these 16 paths. If there is a STM that gives a distance distribution of the 9 paths P_i, $i = 1, 2, \ldots, 9$ from stage 2 to stage $D + 1$, which satisfies

$$H_i \geq H_{\max}(H_{\min}), \text{ for all } i,\ i = 1, 2, \ldots, 9 \tag{5.4}$$

and there is at least one path P_i satisfying

$$H_i = H_{\max}(H_{\min}) \tag{5.5}$$

Then the minimum distance of the 9 paths P_i is $16 + H_{\max}(H_{\min})$, and the minimum distance among these 16 paths is the upper (lower) bound derived in Section 5.1.4.

Define a valid distance distribution as a distance distribution that can be obtained by at least one STM (one code). If we can build a valid distance distribution of these 16 paths that can lead to the bounds, then we can claim that the bounds can be reached.

In a valid distance distribution, among the 15 paths other than the reference-path, the distance of each of the paths in the reference-initial-group or in the reference-terminal-group must have the fixed value shown in the distance property at each stage from stage 1 to $D+2$. The distance of each of the 9 paths P_i, must have the fixed value shown in the distance property at stage 1 and $D+2$. That is to say, to build a valid distance distribution, we only need to specify the distances of the 9 paths P_i, at each stage from stage 2 to $D+1$. This is the distance distribution we will talk about later. We can see a valid distance distribution must comply with Distance Rule 1 at stage 2 and Distance Rule 2 at each stage from stage 3 to $D+1$. Later, when we say a distance distribution complies with Distance Rule 1 and Rule 2, we mean it complies with Distance Rule 1 at stage 2 and Distance Rule 2 at each stage from stage 3 to $D+1$. Before we investigate the other conditions that constrain a distance distribution to be a valid distance distribution, we discuss how to find a STM (a code) in order to obtain a distance distribution built under Distance Rule 1 and Rule 2 and prove two lemmas.

5.2.2 Find an STM to Obtain a Distance Distribution Built under Distance Rule 1 and Rule 2

A distance distribution built under Distance Rule 1 at stage 2 specifies that among the 9 paths, P_i, $i=1,\ 2,\ \ldots,\ 9$, each of the initial-groups (terminal-groups) other than the reference-initial-group (reference-terminal-group) contains one special-8-path. Each of the three special-8-paths belongs to a different terminal-group (initial-group) other than the reference-terminal-group. All the other 6 paths have distance 4.

A distance distribution built under Distance Rule 2 at any stage from stage 3 to $D+1$ specifies that among the 9 paths, P_i, $i=1,\ 2,\ \ldots,\ 9$, each of the initial-groups (terminal-groups) other than the reference-initial-group (reference-terminal-group) contains one special-4-path. Each of the three special-4-paths belongs to a different terminal-group (initial-group) other than the reference-terminal-group. All the other 6 paths have distance 6.

At each stage from stage 2 to $D+1$, these 16 paths can be depicted in STM, as in Figures 5.3 to 5.5, along with the corresponding symbol matrix. Note the symbols in the

symbol matrix and the path in STM at this stage have one to one correspondence. After we obtained the actual symbols matrix (necessary only at stage 2) and know the location of the reference-symbol in the symbol matrix, the distances of other symbols in the symbol matrix can be obtained. They comply with Distance Rule 1 at stage 2 and Distance Rule 2 at each stage from stage 3 to $D+1$. This is satisfied by all different codes (different STMs). To get the wanted distance distribution satisfying Distance Rule 1 and Rule 2, we can place these 16 paths into the symbol matrix such that the path having distance d is placed into a location where the symbol at that location has distance d. We claim the following statement:

Statement 5.1 If we place an initial-group in a row and a terminal-group in a column in the symbol matrix, then, to obtain the wanted distance distribution satisfying Distance Rule 1 and Rule 2, we only need to guarantee that among the 9 paths, P_i, $i = 1, 2, \ldots, 9$, the special-8-paths are placed in those special-8-locations at stage 2 and the special-4-paths are placed in those special-4-locations at each stage from stage 3 to $D+1$.

Proof: For each stage from stage 2 to $D+1$ in the symbol matrix, the symbols in the reference-row and the reference-column all have distance 8. If we place an initial-group in a row and a terminal-group in a column in the symbol matrix, these will be the locations where the paths in the reference-initial-group and the reference-terminal-group should be placed. In the symbol matrix, cross out the reference-row and the reference-column; in the left 3×3 matrix, the symbols in the locations other than the three special-8-locations or special-4-locations all have distance 4 at stage 2 and distance 6 at each stage from stage 3 to stage $D+1$. These will be the locations of the other 6 paths other than the special-8-paths or the special-4-paths among the 9 paths P_i, $i = 1, 2, \ldots, 9$, should be placed into. In the wanted distance distribution, the distances of the paths other than the special-8-paths or the special-4-paths have these values, respectively, because that is what the distance distribution built under Distance Rule 1 and Rule 2 means. We only need to guarantee that among the 9 paths, P_i, $i = 1, 2, \ldots, 9$, the special-8-paths are placed in those special-8-locations at stage 2 and the special-4-paths are placed in those

special-4-locations at each stage from stage 3 to $D+1$. All the other paths will have the wanted distances in the distance distribution built under Distance Rule 1 and Rule 2.

Placing these 16 paths in the symbol matrix at each stage is equivalent to placing these 16 paths in a corresponding order in STM at this stage; i.e., define some properties of STM, which are indicated by some elements in STM-FREE. If by placing these 16 paths in the symbol matrix to get one distance distribution built under Distance Rule 1 and Rule 2 can define all the elements in STM-FREE, then it shows that there is at least one STM that can give this distance distribution. In other words, this distance distribution is valid.

5.2.3 Proofs of Two Lemmas

Lemma 5.1 In HDCTCM(n, D), given a specific butterfly's location in STM--say STM$(l_1, l_2, \ldots, l_{D-1})$--then all sets of n butterflies that will be in set of n locations having $x_k = l_k,\ k = 1,\ 2,\ \ldots,\ D-1$ will be known.

Proof: We have already shown that for HDCTCM(n, D), all sets of n butterflies constitute simplex-transition$_i$ where $2 \le i \le D$ are known. Then given a butterfly in STM$(l_1, l_2, \ldots, l_{D-1})$, the simplex-transition$_2$-butterfly-set containing it will occupy STM$(:, l_2, \ldots, l_{D-1})$. Then the simplex-transition$_3$-butterfly-sets containing the butterflies in STM$(:, l_2, \ldots, l_{D-1})$ will occupy STM$(:,:, l_3, l_4, ..., l_{D-1})$; the simplex-transition$_4$-butterfly-sets containing the butterflies in STM$(:,:, l_3, l_4, ..., l_{D-1})$ will occupy $(:,:,:, l_4, l_5, ..., l_{D-1})$; The simplex-transition$_m$-butterfly-sets containing the butterflies in STM$(:,:,...,:, l_{m-1}, l_m, ..., l_{D-1})$ will occupy STM$(:,:,...,:, l_m, l_{m+1}, ..., l_{D-1})$; Until the end, the sets of butterflies occupy $(:,:,...,:, l_{D-1})$ will be recognized.

We have recognized the sets of butterflies should be located in STM$(:,:,...,:, l_k, l_{k+1}, ..., l_{D-1})$ for any k, where $1 \le k \le D-1$. The simplex-transition$_{k+2}$-butterfly-sets containing the butterflies in STM$(:,:,...,:, l_k, l_{k+1}, ..., l_{D-1})$ will occupy STM$(:,:,...,:, l_k, :, l_{k+2}, l_{k+3}, ..., l_{D-1})$. The simplex-transition$_{k+3}$-butterfly-sets containing the butterflies in STM$(:,:,...,:, l_k, :, l_{k+2}, l_{k+3}, ..., l_{D-1})$ will occupy $(:,:,...,:, l_k, :,:, l_{k+3}, l_{k+4}, ..., l_{D-1})$. Eventually, the simplex-transition$_D$-butterfly-sets containing the butterflies in STM$(:,:,...,:, l_k, :,...,:, l_{D-1})$, which are recognized in the pervious stage, will occupy

STM$(:,:,...,:,l_k,:,...,:)$, i.e., all the sets of butterflies that occupy a set of n locations having $x_k = l_k,\ k = 1,\ 2,\ \ldots,\ D-1$ are recognized.

Lemma 5.2 In a symbol matrix having "the property of symbol matrix", suppose we know the row index that one initial-group should be placed in and the column indexes of the four paths in this initial-group, i.e., the column indexes of the four terminal-groups. Call this initial-group the recognized initial-group in the proof. Then for any wanted distance distribution built under Distance Rule 2, there is one and only one way to place the other three initial-groups in the other three rows in the symbol matrix to get to obtain this distance distribution.

Proof: In the symbol matrix, denote the row index that this recognized initial-group should be placed in as r_1. The column indexes of the four terminal-groups are known. Denote the column index of the reference-terminal-group as c_1. Then, there are two situations:

Case 1: This recognized initial-group is the reference-initial-group, i.e., the reference-symbol is in (r_1, c_1). In Section 5.1.3, in the derivation of Distance Rule 2, we have shown that in the symbol matrix having " the property of symbol matrix", knowing the location of the reference-symbol in the symbol matrix, the three special-4-locations will be determined by applying the "copy" rule along the row and column. Denoted these three special-4-locations as $(r_i, c_i),\ i = 2,\ 3,\ 4$. We have shown there that $r_i \neq r_1,\ r_2 \neq r_3 \neq r_4$, $c_i \neq c_1$, and $c_2 \neq c_3 \neq c_4$. For each column $c_i,\ i = 2,\ 3,\ 4$, we know which terminal-group should be placed in this column. Then in the wanted distance distribution built under Distance Rule 2, the initial-group having a special-4-path that belongs to this terminal-group should be placed in row r_i.

Case 2: This recognized initial-group is not the reference-initial-group. Then in the wanted distance distribution, this initial-group must contain a special-4-path, which belongs to a known terminal-group other than the reference-terminal-group. The column

indexes of all the terminal-groups are known, so we know the column index that this special-4-path should be place in, and denote it as c_2 ; certainly $c_2 \neq c_1$. So, this special-4-path is placed in (r_1, c_2) in the symbol matrix. We know that the special-4-locations are the locations in the symbol matrix that relate to the reference-symbol by applying "copy" rule twice--once along the row and once column. Therefore, knowing (r_1, c_2) is such a special-4-path, we can find the location of the reference-symbol backwards. Then the other two special-4-locations can be determined afterwards.

We know the reference-terminal-group should be placed in column c_1, so the reference-symbol must also be placed in column c_1. We need to find out the row index of the reference-symbol first. From Equation (3.2), we can find out the pulse-location that has a location pair (c_1, c_2). Suppose it is pulse-location$_l$, $l \in \{1,\ 2,\ 3\}$. Then for pulse-location$_l$, in column c_1 find the pair location of row r_1 and denote it as r_2, $r_2 \neq r_1$; i.e., the reference-symbol should be placed in row r_2. Its location is (r_2, c_1). The reference-initial-group should be place in row r_2.

Then the other two special-4-locations are determined as follows. In the reference-symbol, for each of the other two pulse-location$_m$, $l \neq m$, first in column c_1 find the pair location of the row index of the reference-symbol, r_2, and denote it as r_i, where $i = 3,\ 4$ and $r_2 \neq r_3 \neq r_4$. Then, in each row r_i, find the pair location of the column index of the reference-symbol, c_1, dented as c_i, where $i = 3,\ 4$ and $c_1 \neq c_3 \neq c_4$. Then the other two special-4-locations are (r_i, c_i), $i = 3,\ 4$. In the wanted distance distribution, there are two other initial-groups other than the recognized group and the reference-initial-group. They each have a special-4-path which belongs to the terminal-groups other than the reference-terminal-group and the terminal-group in column c_2. For each of the column c_i, $i = 3,\ 4$, we know which terminal-group should be place in this column, and then the initial-group having a special-4-path which belongs to this terminal-group will be placed in row r_i.

In either case, to obtain a wanted distance distribution built under Distance Rule 2, the other three initial-groups will be uniquely placed into the other three rows of the

symbol matrix, and the column indexes of the four terminal-groups will fix the order of placing the four paths of each initial-group in its row.

5.2.4 Other Conditions for a Distance Distribution to be Valid

The only other possible constraint for a distance distribution to be valid will be the relationship of the distance distributions between consecutive stages from stage 2 to stage $D+1$. This will be answered by Theorem 5.1.

Theorem 5.1 When building a distance distribution from stage 2 to stage $D+1$ under Distance Rule 1 and Rule 2, there is no relationship of the distance distributions between consecutive stages. Any distance distribution built under Distance Rule 1 and Rule 2 is a valid distance distribution.

This theorem says, to build a valid distance distribution, among the 9 paths, P_i, $i = 1,\ 2,\ \ldots,\ 9$, one can let any three paths to be the three special-8-paths at stage 2 or the three special-4-paths at any stage from stage 3 to $D+1$ as long as the whole distance distribution at that stage satisfies Distance Rule 1 at stage 2 or Distance Rule 2 at any stage from stage 3 to $D+1$. There is no relationship between consecutive stages on which three paths one chooses to be the special-8-paths or special-4-paths. Any distance distribution built under Distance Rule 1 and Rule 2 is a valid distance; i.e., there is at least one STM that can lead to this distance distribution.

The reason is that a STM is fully determined by its STM-FREE. From Figures 5.3 to 5.5, the locations of these 16 paths in STM at each stage from stage 3 to $D+1$ are only related to two elements in STM-FREE. Applying the method described in Section 5.2.2, we will see that obtaining the wanted distance distribution under Distance Rule 2 in the previous stage only fixes one of the two elements of STM-FREE; the other is to be defined by this stage's distance distribution. Among the freedom one can choose for this element in STM-FREE, one can get any distance distribution built under Distance Rule 2 for this stage. Stage 2 is the first stage to build a distance distribution. (See Figure 5.2.) The locations of these 16 paths in STM are related to the rest of elements in STM-FREE.

Among the freedom one can choose for those elements, one can also obtain any distance distribution built under Distance Rule 1. We will prove this stage-by-stage.

Proof:

Stage 2: At this stage, consider any wanted distance distribution built under Distance Rule 1.

These 16 paths in STM and corresponding "prototype" symbol matrix are shown in Figure 5.2. The special-8-locations depend on the actual symbol matrix. So we must first place the four butterflies containing these 16 paths into some particular locations in STM to obtain the actual symbol matrix. We have not defined any part of STM yet; there are many ways to place these four butterflies in STM and place these 16 paths into particular locations in these four butterflies to obtain the wanted distance distribution. But when a particular way is selected, some elements in STM-FREE will be defined.

First, choose the butterfly whose originating state set contains the originating state of the reference-initial-group, g, as the anchor butterfly. This defines STM-FREE (first butterfly). Place the anchor butterfly in an arbitrary location in STM—say, $\mathrm{STM}(l_1, l_2, \ldots, l_{D-1})$. This defines STM-FREE (first location). Then place the originating state g in any internal location, denoted as k, in the left side of the anchor butterfly. $k \in \{1,\ 2,\ 3,\ 4\}$. This defines part of STM-FREE (left side butterfly). Then the other three butterflies containing the other three initial-groups will be placed in $\mathrm{STM}(x_1, l_2, \ldots, l_{D-1})$, where $x_1 =: / l_1$. The order of placing these three butterflies has not been determined yet. And the other three originating states of the other three initial-groups should be placed in the same internal locations in their butterflies as the originating state g in its butterfly—the anchor butterfly.

So far, we have known the four butterflies' locations in STM and the 16 state transitions' internal locations in these butterflies. From Chapter 3, the symbols assigned to these 16 paths at this stage will be identified from the signal constellation. The symbol matrix is uniquely determined. From the mapping between STM and the symbol matrix, the reference-initial-group should be placed in row $r_1 = d_1$ in the symbol matrix. But the order of placing them in this row has not been decided yet. We can place the four paths in the reference-initial-group in row d_1 in an arbitrary order. Map back to STM, the next

states of these four paths located in column 1 to column 4 in the symbol matrix will be placed in the right side of the anchor butterfly from top-to-bottom. This defines STM-FREE (right side butterfly). Since each path in an initial-group belongs to a different terminal-group, the order of placing the four paths in the reference-initial-group in row d_1 of the symbol matrix will tell which terminal-group should be place in which column. Thus, we know the column indexes of the four terminal-groups. Denote the column index of the reference-terminal-group as c_1. Then, we know the reference-symbol is in (r_1, c_1). The distance of all the other symbols can be determined. They comply with Distance Rule 1. Denote the special-8-locations as (r_i, c_i), where $i = 2,\ 3,\ 4$, $r_1 \neq r_2 \neq r_3 \neq r_4$ and $c_1 \neq c_2 \neq c_3 \neq c_4$.

In the wanted distance distribution, the three-special-8-paths should be placed into these three special-8-locations. For each column c_i, $i = 2,\ 3,\ 4$, we know the terminal-group that should be placed in this column. Then in the wanted distance distribution, the initial-group having a special-8-path which belongs to this terminal-group should be placed in row r_i and the order of placing them in this row is already fixed by the known column indexes of the four terminal-groups. Mapping this initial-group in the symbol matrix back to STM means that the butterflies containing this initial-group should be place into STM$(x_1, l_2, \ldots, l_{D-1})$, where $x_1 = r_i$. The order of the set of the four butterflies in STM$(:, l_2, \ldots, l_{D-1})$ defines STM-FREE (1_{st} dimension).

In summary, at this stage one can obtain any distance distribution built under Distance Rule 1 by defining STM-FREE (first butterfly, first location, part of left side butterfly, right side butterfly, 1_{st} dimension).

Stage 3: At this stage, consider any wanted distance distribution built under Distance Rule 2.

These 16 paths in STM are in Figure 5.3 (a). An initial-group is located along the 1_{st} dimension and a terminal-group is located along the 2_{nd} dimension. The corresponding symbol matrix is shown in Figure 5.3 (b). Regardless of different codes, the symbol matrix always has " the property of symbol matrix". From the derivation of Distance Rule 2, for this kind of symbol matrix we do not need know the actual symbols in the

symbol matrix; only the reference-symbol's location will give the distances of all the other symbols in this symbol matrix.

From Lemma 5.1, all sets of four butterflies whose location indexes satisfying $x_2 = d_2$ are known. From Figure 5.3 (a), there is one and only one set of 4 butterflies containing an initial-group that should be placed in STM with $x_2 = d_2$. So comparing the 4 sets of 4 butterflies containing these four initial-groups with all sets of four butterflies having location index $x_2 = d_2$, one initial-group will be recognized that should be placed in STM along the 1_{st} dimension and have location index $x_2 = d_2$. Also, STM-FREE (1_{st} dimension) is defined in the previous stage, so the order of placing the set of four butterflies containing the recognized initial-group along the 1_{st} dimension in STM is fixed. Each path in an initial-group belongs to a different terminal-group, so the location indexes of the four terminal-groups along the 1_{st} dimension in Figure 5.3 (a) are known.

Map the paths in STM to the symbol matrix; this recognized initial-group should be placed in row d_2 in the symbol matrix. The column indexes of the four terminal-groups are known from STM-FREE (1_{st} dimension). From Lemma 5.2, in order to get any wanted distance distribution, all the other three initial-groups will be uniquely placed in the other three rows of the symbol matrix. Map to STM; the butterflies containing these initial-groups will be placed uniquely. Then one set of the 4 butterflies along the 2_{nd} dimension of STM will define STM-FREE (2_{nd} dimension). So we can obtain any wanted distance distribution built under Distance Rule 2 at this stage by defining STM-FREE (2_{nd} dimension).

Stage 4 to D: At each stage, these 16 paths in STM and the corresponding symbol matrix are similar to those at stage 3. Is it true that for each such stage, one can obtain any wanted distance distribution built under Distance Rule 2 by defining one more element in STM-FREE? It is solved in Lemma 5.3.

Lemma 5.3 For any stage i, $3 \le i \le D$, one can obtain any wanted distance distribution built under Distance Rule 2. By obtaining that, STM-FREE ($(i-1)_{th}$ dimension) will be defined.

Proof by induction on i.

(1) For stage $i = 3$, it is proved above.

(2) Suppose it is true for stage $i = k$, $3 \le k \le D-1$. Then for stage $i = k+1$, these 16 paths in STM are shown in Figure 5.4 (a). An initial-group is located along the $(k-1)_{th}$ dimension and a terminal-group is located along the k_{th} dimension. The corresponding symbol matrix in Figure 5.4 (b) has "the property of symbol matrix". There is one and only one set of 4 butterflies containing an initial-group that should be placed in STM with location index $x_k = l_k$. From Lemma 5.1, all sets of four butterflies whose location indexes satisfying $x_k = l_k$ are known. So comparing the 4 sets of 4 butterflies containing these four initial-groups with all sets of 4 butterflies having location index $x_k = l_k$, one initial-group will be recognized that should be placed in STM along the $(k-1)_{th}$ dimension and have location index $x_k = l_k$. Also, STM-FREE ($(k-1)_{th}$ dimension) is defined in the previous stage, so the order of placing the set of four butterflies containing the recognized initial-group along the $(k-1)_{th}$ dimension in STM is fixed. That is to say, the location indexes of the four terminal-groups along the $(k-1)_{th}$ dimension in Figure 5.4 (a) are known.

Map to the symbol matrix; this recognized initial-group will be place into row l_k and the column indexes of the four terminal-groups are known. From Lemma 5.2, for any wanted distance distribution, the other three initial-groups will be uniquely placed in the other three rows of the symbol matrix. Map back to STM; the butterflies containing these initial-groups will be placed uniquely. Then one set of the 4 butterflies along the k_{th} dimension in STM will define STM-FREE (k_{th} dimension). Lemma 5.3 is proved.

Stage $D+1$**:** these 16 paths in STM are shown in Figure 5.5 (a). An initial-group is located along the $(D-1)_{th}$ dimension. A terminal-group constitutes a simplex-transition$_{D+1}$. It is in a butterfly. The corresponding symbol matrix has "the property of symbol matrix". The row indexes and the column indexes in the symbol matrix correspond to the four internal locations in the left side of a butterfly and the four location indexes along the $(D-1)_{th}$ dimension of STM.

From building STM in Chapter 3, we know that the internal order of one butterfly will fix the internal order of all the other butterflies. We have placed the originating state g of the reference-initial-group at stage 2 in the internal location k, where $k \in \{1, 2, 3, 4\}$, in the left side of the anchor butterfly. Then all the states that should be placed in the internal location k in the left side of all the other butterflies are known.

From Figure 5.5 (a), the originating states of one and only one initial-group should be placed in the internal location k of the four butterflies along the $(D-1)_{th}$ dimension of STM. Compare the originating state sets of the four initial-groups with all the states that should be placed in the internal location k in the left side of all the butterflies. One initial-group will be recognized whose originating states should be placed in the internal location k in the left side of the four butterflies. Since the previous stage has defined STM-FREE ($(D-1)_{th}$ dimension), the order of placing these four butterflies along the $(D-1)_{th}$ dimension is fixed; i.e., the location indexes of the four terminal-groups along the $(D-1)_{th}$ dimension are known. Map the paths in STM to the symbol matrix; this recognized initial-group will be placed in row k, and the column indexes of the four terminal-groups are known. From Lemma 5.2, there is one and only one way to place the other three initial-groups in the other three rows in the symbol matrix. Map back to STM; the originating states of these three initial-groups other than the recognized initial-group are placed uniquely in the left three internal locations in the left side of the four butterflies containing these 16 paths. This will fix the order of placing the other three originating state other than the state g in the left side of the anchor butterfly; i.e., the rest part of STM-FREE (left side butterfly) that has not been defined at stage 2 will be defined at this stage.

In summary, we have proved that when building a distance distribution under Distance Rule 1 and Rule 2, there is no relationship on the distance distributions between consecutive stages. At each stage from stage 2 to stage $D+1$, any distance distribution built under Distance Rule 1 and Distance Rule 2 is a valid distance distribution. We further showed that in order to obtain the wanted distance distribution, each stage will define some elements in STM-FREE. At stage 2, STM-FREE (first butterfly, first location, part of left side butterfly, right side butterfly, 1_{st} dimension) will be defined, at

stage i, $3 \le i \le D$, STM-FREE ($(i-1)_{th}$ dimension) will be defined, and at stage $D+1$, the rest of STM-FREE (left side butterfly) that has not be defined at stage 2 will be defined.

5.2.5 Build Valid Distance Distributions That Can Reach Bounds

This section will build valid distance distributions that can give the bonds of the minimum distance of HDTCM$(4, D)$. We will build a distance distribution from stage 2 to $D+1$ under Distance Rule 1 and Rule 2 specifying the distances for the 9 paths P_i, $i = 1, 2, \ldots, 9$, such that Equations (5.4) and (5.5) are satisfied.

Build a Valid Distance Distribution to Reach the Upper Bound on the Minimum Distance

We have obtained

$$H_{\max} = \left\lfloor \frac{16D}{3} \right\rfloor_{even} = \left\lfloor \frac{12D+4D}{3} \right\rfloor_{even} = 4D + \left\lfloor \frac{4D}{3} \right\rfloor_{even} = 4D + \left\lfloor D + \frac{D}{3} \right\rfloor_{even} \tag{5.6}$$

D can be written as $3m$, $3m+1$ or $3m+2$, where m is an integer and $m \ge 0$; then $H_{\max}$ can be rewritten as

$$H_{\max} = \begin{cases} 4D + \left\lfloor 3m + \frac{3m}{3} \right\rfloor_{even} = 4D + 4m, D = 3m \\ 4D + \left\lfloor 3m + 1 + \frac{3m+1}{3} \right\rfloor_{even} = 4D + 4m, D = 3m+1 \\ 4D + \left\lfloor 3m + 2 + \frac{3m+2}{3} \right\rfloor_{even} = 4D + 4m + 2, D = 3m+2 \end{cases} \tag{5.7}$$

In any case, $m = \lfloor D/3 \rfloor$.

For $m \ge 1$, a valid distance distribution is built using Procedure 5.1. For $m = 0$, it is trivial, we will deal with it after describing Procedure 5.1.

Procedure 5.1 Build a valid distance distribution to reach the upper bound on the minimum distance

1. Build one path--say path P_1--such that $H_1 = H_{\max}$. First, place distance 4 in each stage from stage 2 to $D+1$; this will contribute $4D$ to H_1. Then add distance 4 to stage 2,

and add distance 2 (if $D = 3m + 2$) in any one stage between stage 3 and stage $D+1$. The largest distance for a path at stage 2 is 8, at the other stages is 6. So stage 2 is now completed. There left $D-1$ or $D-2$ (if $D = 3m + 2$) stages having distance 4; these stages can only be added distance 2 in each stage. In order to obtain $H_1 = H_{\max}$, the left distance to be distributed is $4D + 4m - 4D - 4$ for $D = 3m$ or $D = 3m + 1$ and is $4D + 4m + 2 - 4D - 4 - 2$ for $D = 3m + 2$. In both cases, it is $2(2(m-1)) \geq 0$ when $m \geq 1$. They need to be distributed to $2(m-1)$ different stages with distance 2 at each stage. For any D, it can be verified that $2(m-1) \leq D-2$; i.e., there are enough stages having distance 4 so far that can be added distance 2 at each of those stage. So, secondly, we add distance 2 to any $2(m-1)$ stages among the stages having distance 4 so far. Path P_1 is now completed and has $H_1 = H_{\max}$. (See Figures 5.7 (a) and (b).)

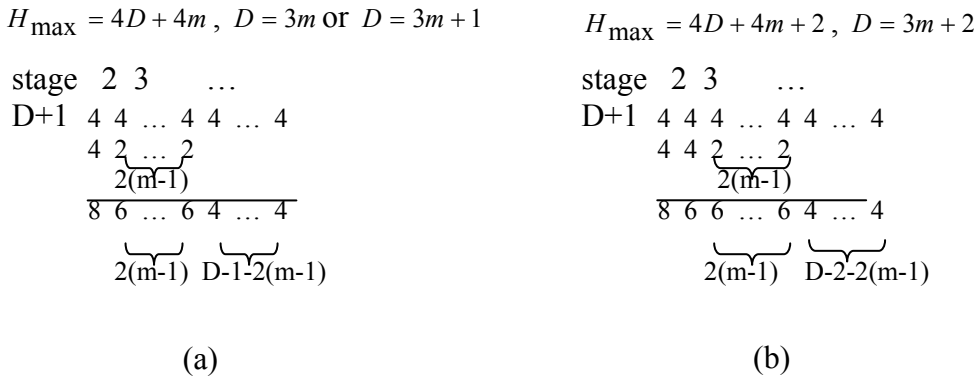


Figure 5.7 Build the distance distribution to reach the upper bound on the minimum distance for HDCTCM$(4, D)$ (a) the first path for $D = 3m$ or $D = 3m + 1$ (b) the first path for $D = 3m + 2$

2. Under Distance Rule 1 and Rule 2, build two other paths--say path P_2 and P_3--that in the same initial-group as path P_1. At stage 2, path P_1 has distance 8; the distance should be 4 in both paths P_2 and P_3. For the stages that P_1 has 4, both P_2 and P_3 should have distance 6. For the stages at which path P_1 has distance 6, one of the path among P_2 and P_3 should have distance 4 and the other one should have distance 6. For any D, there are at least $2(m-1)$ such stages; let each of the paths P_2 and P_3 has distance 4 at $m-1$stages and distance 6 at the other $m-1$stages. For $D = 3m + 2$,

there is one more stage at which path P_1 has distance 6. For this stage, let any one of paths P_2 and P_3 have distance 4 and the other one have distance 6. These three paths are shown in Figures 5.8 (a) and (b).

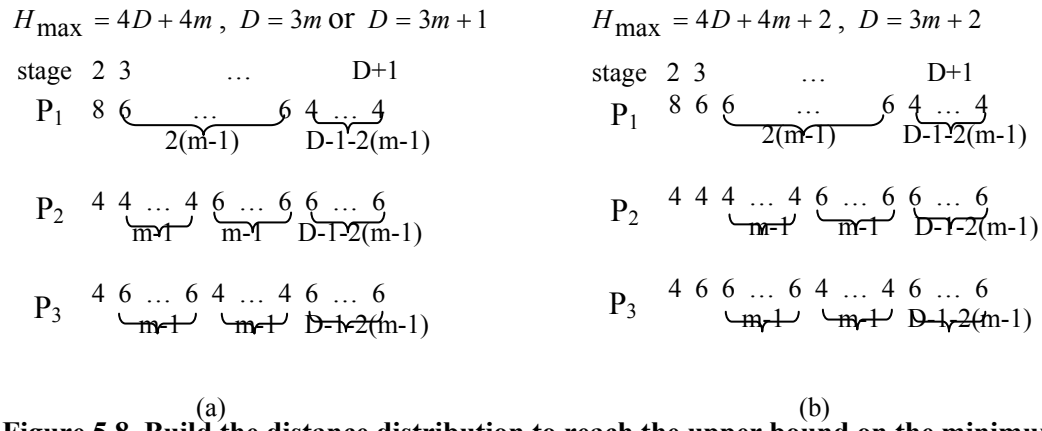


Figure 5.8 Build the distance distribution to reach the upper bound on the minimum distance for HDCTCM$(4, D)$ (a) the three paths in an initial-group for $D = 3m$ or $D = 3m + 1$ (b) the three paths in an initial-group for $D = 3m + 2$

For $D = 3m$ or $D = 3m + 1$,

$$H_2 = H_3 = 4 + 10(m-1) + 6(D - 1 - 2(m-1)) = 4D + 4m + 2D - 6m = \begin{cases} H_{\max}, & D = 3m \\ H_{\max} + 2, & D = 3m + 1 \end{cases} \tag{5.8}$$

For $D = 3m + 2$,

$$H_2 = 4 + 4 + 10(m-1) + 6(D - 2 - 2(m-1)) = 4D + 4m + 2 = H_{\max} \tag{5.9}$$

$$H_3 = 4 + 6 + 10(m-1) + 6(D - 2 - 2(m-1)) = 4D + 4m + 2 = H_{\max} + 2. \tag{5.10}$$

3. Build the other two sets of three paths; each set belongs to one of the other two initial-groups. Interchange the distance distribution of the three paths P_1, P_2, and P_3 and assign them to the three paths in the other two initial-groups. The interchange is done

in such a way that each initial-group (terminal-group) has distance distributions of P_1, P_2, and P_3, and the path that has distance distribution of P_i, $i = 1, 2, 3$ in each initial-group (terminal-group) belongs to a different terminal-group (initial-group). Suppose the other initial-groups are $\{P_4, P_5$ and $P_6\}$ and $\{P_7, P_8$ and $P_9\}$. $\{P_1, P_4, P_7\}$, $\{P_2, P_5, P_8\}$ and $\{P_3, P_6, P_9\}$ are the three terminal-groups. Then let P_4, P_5, and P_6 have the same distance distributions as that of P_2, P_3, and P_1, respectively and let P_7, P_8, and P_9 have the same distance distributions as that of P_3, P_1, and P_2, respectively. Then it can be verified that each initial-group (terminal-group) has one special-8-path at stage 2 or one special-4-path at each stage from stage 3 to $D+1$, and the special-8-path or the special-4-path from each initial-group (terminal-group) belongs to a different terminal-group (initial-group), i.e., Distance Rule 1 and Rule 2 are satisfied. Also, for the 9 paths, P_i, $i = 1, 2, \ldots, 9$, Equations (5.4) and (5.5) are satisfied.

When $m = 0$, $D = 1$ or $D = 2$. For $D = 1$, $H_{\max} = 4D + 4m = 4$. There is only one stage--stage 2--which needs to build distance distribution. The first three paths P_1, P_2 and P_3, can have distance 4, 8 and 4 at this stage, respectively.

For $D = 2$, $H_{\max} = 4D + 4m + 2 = 10$. There are two stages--stage 2 and 3--which needs to build distance distribution. The distance distributions can be built as follows. At stage 2 and 3, P_1 has distance 4 and 6, P_2 has distance 8 and 4, and P_3 has distance 4 and 6.

Following step 3 in Procedure 5.1, the distance distribution of all the other paths can be obtained.

It can be verified that Distance Rule 1 and Rule 2 are satisfied. Also, for the 9 paths, P_i, $i = 1, 2, \ldots, 9$, Equation (5.4) and (5.5) are satisfied.

So, for all D, we have built a valid distance distribution that can achieve the upper bound on the minimum distance of HDCTCM$(4, D)$.

Build a Valid Distance Distribution to Reach the Lower Bound on the Minimum Distance

For lower bound, $H_{\min} = 4D$. First build a path P_1 that satisfies $H_1 = H_{\min}$. This path has distance 4 at each stage from stage 2 to $D+1$. Then the other two paths in the same initial-group as path P_1 is easy to built under Distance Rule 1 and Rule 2. The three paths are shown in Figure 5.9. Following the step 3 in Procedure 5.1, the distance distributions of all the other paths can be obtained. It can be verified that Distance Rule 1, Rule 2, and Equations (5.4) and (5.5) are satisfied.

$H_{\min} = 4D$				
stage	2	3	…	D+1
P_1	4	4	…	4
P_2	8	6	…	6
P_3	4	6	…	6

Figure 5.9 Build the distance distribution for the three paths in an initial-group to reach the lower bound on the minimum distance for HDCTCM$(4, D)$

So far, valid distance distributions have been built that can reach the bounds. In other words, we have proved that the bounds on the minimum distance can be reached.

5.3 Optimum Distance Codes

When a code achieves upper bound of the MD of its kind, this code is called an optimum distance code. This section will give the method to built an optimum distance code and show the possible distance distributions for an optimum distance code.

5.3.1 Build Optimum Distance Codes

In the proof of Theorem 5.1, we actually describe the procedure to obtain a STM that will give any wanted distance distribution built under Distance Rule 1 and Rule 2. Then the way to build an optimum code or a code that has the lower bound on the minimum distance is described as follows.

Procedure 5.2 Build an optimum distance code or a code that has the lower bound of the minimum distance

1. Using the method in described in Section 5.2.5, build a valid distance distribution that can give the upper or lower bounds on the minimum distance.
2. Following the proof of Theorem 5.1, obtain the distance distribution built in step 1 from stage 2 to stage $D+1$ by defining some elements in STM-FREE at each stage. This STM-FREE will fix a STM. Then map the signal constellation to this STM; the symbol assignment can be obtained. The code using this symbol assignment will reach the bounds on the minimum distance.

5.3.2 Possible Distance Distributions for Optimum Distance Codes

The method described in Section 5.2.5 is not the only way to build a valid distance distribution that can reach the bounds. But all the valid distance distribution that can reach the bounds must satisfy Equations (5.4) and (5.5). So there are limited numbers of distance distribution patterns which exist for the optimum distance codes.

Among the 9 paths, P_i, $i = 1,\ 2,\ \ldots,\ 9$, Suppose P_j, $j = 1,\ 2,\ 3$ are the three paths in any one initial-group or any one terminal-group. From Distance Rule 1 and Rule 2, it can be verified that, $\sum_{j=1}^{3} H_j = 16D$, for any D.

There are three cases for D.

(1) For $D = 3m$,

$$H_{\max} = 4D + 4m = 4 \cdot 3m + 4m = 16m \tag{5.11}$$

$$\sum_{j=1}^{3} H_j = 16D = 3 \cdot 16m = 3 \cdot H_{\max} \tag{5.12}$$

So in order to satisfy Equations (5.4) and (5.5), the set of distances H_j of the three paths P_j should be $\{H_{\max}, H_{\max}, H_{\max}\}$. Then the distances H_i of the 9 paths, P_i, $i = 1,\ 2,\ \ldots,\ 9$, are shown in Figure 5.10.

$$\begin{matrix} H_{\max} & H_{\max} & H_{\max} \\ H_{\max} & H_{\max} & H_{\max} \\ H_{\max} & H_{\max} & H_{\max} \end{matrix}$$

Figure 5.10 The distances H_i of the 9 paths, P_i, $i = 1,\ 2,\ \ldots,\ 9$, for optimum HCTCM$(4, D)$ codes with $D = 3m$

In Figure 5.10, each row represents the distances H_i of the three paths in an initial-group and each column represents the distances H_i of the three paths in a terminal-group. It can be seen that in the optimum distance codes, for the 9 paths P_i, $i = 1,\ 2,\ \ldots,\ 9$, their distances are all equal to $16 + H_{\max}$.

(2) For $D = 3m + 1$

$$H_{\max} = 4D + 4m = 4 \cdot 3m + 4 + 4m = 16m + 4 \tag{5.13}$$

$$\sum_{j=1}^{3} H_j = 16D = 16 \cdot (3m + 1) = 3 \cdot H_{\max} + 4 \tag{5.14}$$

So in order to satisfy Equations (5.4) and (5.5), the set of distances H_j, of the three paths P_j should be $\{H_{\max}, H_{\max} + 2, H_{\max} + 2\}$ or $\{H_{\max}, H_{\max}, H_{\max} + 4\}$. Then the possible distances H_i of the 9 paths, P_i, $i = 1,\ 2,\ \ldots,\ 9$, are shown in Figure 5.11.

$$\begin{array}{ccc} H_{\max} & H_{\max}+2 & H_{\max}+2 \\ H_{\max}+2 & H_{\max} & H_{\max}+2 \\ H_{\max}+2 & H_{\max}+2 & H_{\max} \end{array}$$

(a)

$$\begin{array}{ccc} H_{\max} & H_{\max} & H_{\max}+4 \\ H_{\max} & H_{\max}+4 & H_{\max} \\ H_{\max}+4 & H_{\max} & H_{\max} \end{array}$$

(b)

$$\begin{array}{ccc} H_{\max} & H_{\max}+4 & H_{\max} \\ H_{\max}+2 & H_{\max} & H_{\max}+2 \\ H_{\max}+2 & H_{\max} & H_{\max}+2 \end{array}$$

(c)

Figure 5.11 The distances H_i of the 9 paths, P_i, $i = 1,\ 2,\ \ldots,\ 9$, for optimum HCTCM$(4, D)$ codes with $D = 3m + 1$

In the optimum distance codes, for the 9 paths P_i, $i = 1,\ 2,\ \ldots,\ 9$, in Figure 5.11 (a), there are 6 paths having distance $16 + H_{\max} + 2$ and 3 paths having distance

$16 + H_{\max}$. In Figure 5.11 (b), there are 3 paths having distance $16 + H_{\max} + 4$ and 3 paths having distance $16 + H_{\max}$. In Figure 5.11 (c), there are 4 paths having distance $16 + H_{\max} + 2$, one path having distance $16 + H_{\max} + 4$, and 4 paths having distance $16 + H_{\max}$.

The distance distribution built in Section 5.2.5 gives the distance distribution in Figure 5.11 (a).

(3) For $D = 3m + 2$,

$$H_{\max} = 4D + 4m + 2 = 4 \cdot (3m + 2) + 4m + 2 = 16m + 10 \tag{5.15}$$

$$\sum_{j=1}^{3} H_j = 16D = 16 \cdot (3m + 2) = 3 \cdot H_{\max} + 2 \tag{5.16}$$

So in order to satisfy Equations (5.4) and (5.5), the set of distances H_j of the three paths P_j should be $\{H_{\max}, H_{\max}, H_{\max} + 2\}$. Then the distances H_i of the 9 paths, P_i, $i = 1,\ 2,\ \ldots,\ 9$, are shown in Figure 5.12.

$H_{\max}$	$H_{\max}$	$H_{\max} + 2$
$H_{\max}$	$H_{\max} + 2$	$H_{\max}$
$H_{\max} + 2$	$H_{\max}$	$H_{\max}$

Figure 5.12 The distances H_i of the 9 paths, P_i, $i = 1,\ 2,\ \ldots,\ 9$, for optimum HCTCM$(4, D)$ codes with $D = 3m + 2$

It can be seen that in the optimum distance codes, for the 9 paths P_i, $i = 1,\ 2,\ \ldots,\ 9$, there are 3 paths having distance $16 + H_{\max} + 2$ and 6 paths having distance $16 + H_{\max}$.

CHAPTER 6 DECODING AND ERROR PERFORMANCE OF HDCTCM

This chapter will address the decoding algorithms for HDCTCM. Circular BCJR algorithms are investigated for decoding HDCTCM. In Section 6.1, the problems in practical implementation of circular BCJR and solutions to them are discussed; then in section 6.2, an iterative circular shift BCJR is developed for decoding HDCTCM. In Section 6.3, the properties of this decoding algorithm are demonstrated through simulations and compared with other decoding algorithms. Finally, the error performance of HDCTCM using this decoding algorithm is presented in Section 6.4.

6.1 Problems in Practical Implementation of Circular BCJR

BCJR and circular BCJR are introduced in Section 2.4; we will employ all the denotations and terms introduced there. Derivation of circular BCJR for tail biting codes is based on Equation (2.27). All circular trellis codes satisfy this, as does HDCTCM. In this section, we will explore the circular BCJR for decoding of HDCTCM. First, implementation problems need to be solved to actually implement these algorithms. In HDCTCM(n, D), the total number of states is $S = n^D$ and the information sequence length is denoted as L. In circular BCJR with eigenvectors, eigenvectors of the multiplication of a sequence of matrixes $\gamma_1 ... \gamma_L$ need to be calculated, where γ_t is an $S \times S$ matrix, $t = 1,...,\mathrm{L}$. Each element in γ_t is a probability. The elements in $\gamma_1 ... \gamma_L$ can be extremely small for a reasonably large L, and due to this, the eigenvectors of this matrix will not converge. When the values of elements in $\gamma_1 ... \gamma_L$ become so small that the computer does not have enough precision to represent them, they will simply be treated as zero. This will cause $\gamma_1 ... \gamma_L$ to be a matrix with all zeros and the eigenvectors obtained will not be correct.

Our solution to this problem is as follows: when calculating $\gamma_1 ... \gamma_L$, scale $\gamma_1 ... \gamma_t$ whenever it is getting too small and multiply this scaled value with γ_{t+1} to get $\gamma_1 ... \gamma_{t+1}$.

Because what we want is the normalized eigenvectors of $\gamma_1 ... \gamma_L$, scaled $\gamma_1 ... \gamma_L$ will have scaled eigenvectors, and normalizing the scaled eigenvectors will give the same results as normalizing the unscaled ones. With this solution, the normalized eigenvectors of $\gamma_1 ... \gamma_L$ can be calculated for any large L and S.

Large delay in decoding will be incurred by the normalization of α_t and β_t at each trellis stage and for each iteration (in iterative circular BCJR) in circular BCJR, when L is reasonably large. The normalization is introduced previously to control the calculation precision. But in order to control computation precision, the normalization of α_t and β_t is only necessary at stages when they are getting too small. We call this "selective normalization". In iterative circular BCJR, the iterations stop when the difference on α between consecutive r_{th} and $(r-1)_{th}$ iterations, $\left\| \alpha^o_{rL+t} - \alpha^o_{(r-1)L+t} \right\|$, is sufficiently small, by a suitable measure, for all t, $t = 1,...,\mathrm{L}$. The superscript "o" indicates the normalized vectors. This requires α to be normalized at each trellis stage t in every iteration. When "selective normalization" is employed, α is only normalized at stages when it is getting too small; this stop condition should be modified accordingly. Since the speed for $\alpha^o_{L+t}, \alpha^o_{2L+t}, \ldots, \alpha^o_{rL+t}$ to converge to α^o_t is the same for all t, $t = 1,...,\mathrm{L}$, we can choose one stage from the total L stages, normalize α at this stage no matter it is too small or not, and compare the difference of α^o at this stage between consecutive iterations to decide if more iterations are needed. Without loss of generality, stage L is chosen. Define $\Delta = \max(abs(\alpha^0_{rL} - \alpha^0_{(r-1)L}))$, where "$abs$" means the absolute value; then when Δ is sufficiently small, stop the iteration and the set of α vectors is obtained at the r_{th} iteration. A similar argument holds for iterations on β.

Although the eigenvectors of $\gamma_1 ... \gamma_L$ can be obtained for any large L and S. The computation load to obtain it is much larger than that involved in iterative circular BCJR with several iterations--especially for large L and S. The circular BCJR with eigenvectors is served more as a theoretical basis to derive the iterative circular BCJR than to be implemented practically. For practical decoding of HDCTCM, iterative circular BCJR will be investigated.

6.2 Iterative Circular Shift BCJR for HDCTCM

The initial α_0 and β_L needs to be set in order to run the iterative circular BCJR. Some choices will lead to faster convergence than others will. In this section, we introduce a scheme based on the statistical estimate of the starting state on the observation of the received sequence. In this scheme, the reliability of the estimation is defined and used to select the most reliable symbol in the received sequence. Also, the circular property of HDCTCM will be incorporated to make this most reliable symbol as the starting point in the decoding.

6.2.1 Statistic Estimation of Starting State in HDCTCM

First let the receiver tentatively hard decode each symbol in the received sequence $Y_1^L = Y_1, Y_2, ..., Y_L$. On the observation of Y_t, the decoder hard decodes it to $\hat{X}_t$, where $\hat{X}_t$ belongs to the channel signal constellation. For the situation that all channel symbols are equally likely, which is true for HDCTCM, we have shown in Section 2.4 that in order to minimize the probability of making an erroneous decision, the receiver should choose $\hat{X}_t$ such that

$$P(Y_t \mid \hat{X}_t) = \max_{\text{over all X}} P(Y_t \mid X) \tag{6.1}$$

In an N dimensional space, where N is the space dimension of signal constellation for HDCTCM(n, D), channel symbol X and Y_t are represented as $X = (x_1, x_2, ..., x_N)$ and $Y_t = (y_{t1}, y_{t2}, ..., y_{tN})$, then for a discrete memoryless AWGN channel with one-sided noise spectrum density N_0,

$$P(Y_t \mid X) = \frac{1}{\sqrt{\pi N_0}} e^{-\frac{\|Y_t - X\|^2}{N_0}} = \prod_{k=1}^{N} \frac{1}{\sqrt{\pi N_0}} e^{-\frac{(y_{tk} - x_k)^2}{N_0}} \tag{6.2}$$

$\|Y_t - X\|^2$ is the squared Euclidean distance between Y_t and X.

The larger $P(Y_t \mid \hat{X}_t)$ is, the smaller is the likelihood of making an error when decoding Y_t to $\hat{X}_t$, i.e., the more reliable is this decision. We define $P(Y_t \mid \hat{X}_t)$ as the reliability of decoding Y_t to $\hat{X}_t$ or simply the reliability of symbol Y_t. In this way, the receiver decodes Y_1^L to $\hat{X}_1^L = \hat{X}_1, \hat{X}_2, ..., \hat{X}_L$ with reliability information $P(Y_t \mid \hat{X}_t)$ for each

t. Compare $P(Y_t \mid \hat{X}_t)$ for all t, find the largest one—say, $P(Y_i \mid \hat{X}_i)$ --then Y_i is the most reliable symbol. Denote the state transition to be assigned $\hat{X}_i$ is $m'' \to m'''$, m'' and m''' are two trellis states. If we circular shift Y_1^L to let Y_i be the first symbol in the shifted sequence, denoted as $Y_1^{L\,SHIFT(i)} = Y_i, Y_{i+1}, ..., Y_L, Y_1, Y_2, ..., Y_{i-1}$, then when running iterative circular BCJR on $Y_1^{L\,SHIFT(i)}$, initial α_0 can be set as

$$\alpha_0(m) = \begin{cases} 1, & \text{m} = m'' \\ 0, & \text{otherwise} \end{cases} \tag{6.3}$$

β_L is set as same as α_0 initially.

6.2.2 Circular Property of HDCTCM and Iterative Circular Shift BCJR

The circular property of trellis path in HDCTCM will be used to select the most reliable symbol in Y_1^L as the starting point for decoding. Denote the information sequence as $U = (U_1, U_2, ..., U_t, ..., U_L)$, where U_i belongs to the finite information symbol alphabet. The corresponding starting state is S_0, and the state transition sequence is $S_0 \to S_1 \to S_2 \to ... \to S_{t-1} \to S_t \to ...S_{L-1} \to S_0$. The corresponding code sequence is denoted as $V = (V_1, V_2, ..., V_t, ..., V_L)$ where V_i belongs to the channel signal constellation. The permuted state transition table employed by HDCTCM guarantees that a circular shift version of U, denoted as $U^{SHIFT(t)} = (U_t, U_{t+1}, ..., U_L, U_1, U_2, ..., U_{t-1})$, will have a state transition sequence and a code sequence which are the circular shift versions of those for U. The state transition sequence will be $S_{t-1} \to S_t \to ... \to S_{L-1} \to S_0 \to S_1 \to ... \to S_{t-2} \to S_{t-1}$, and the code sequence will be $V^{SHIFT(t)} = (V_t, V_{t+1}, ..., V_L, V_1, V_2, ..., V_{t-1})$.

Therefore, if we can decode the circular shift version of receive sequence, $Y_1^{L\,SHIFT(i)}$, and get information sequence $\hat{U}$, then decoding of Y_1^L will give a circular shift-back version of $\hat{U}$, i.e., $\hat{U}^{SHIFT(L-i)}$. As discussed in the previous section, the initialization problem for decoding $Y_1^{L\,SHIFT(i)}$ using iterative circular BCJR is solved. We name this scheme of running iterative circular BCJR on the circular shift version of received sequence and the corresponding way to set initial α_0 and β_L as iterative circular shift BCJR algorithm for decoding HDCTCM.

For decoding HDCTCM(n, D), the symbols in information sequence are directly related to state transitions other than related to the states in shift register based trellis coding. Therefore, we only need to calculate σ's in the iterative circular shift BCJR. Let $C^{(j)}$ be the set of transitions $S_{t-1} = m' \to S_t = m$ caused by input symbol j, where j belongs to the information symbol alphabet. Then after $\sigma_t(m', m)$ is obtained, the probability of the input symbol at stage t is

$$P(U_t = j \mid Y_1^L) = \frac{1}{P(Y_1^L)} \sum_{(m', m) \in C^{(j)}} \sigma_t(m', m) = \frac{1}{\sum_{\text{all } (m', m)} \sigma_t(m', m)} \sum_{(m', m) \in C^{(j)}} \sigma_t(m', m) \tag{6.4}$$

We decode $U_t = j'$ such that

$$P(U_t = j' \mid Y_1^L) = \max_{\text{all j}} P(U_t = j \mid Y_1^L) \tag{6.5}$$

Some other parameters in iterative circular shift BCJR for decoding HDCTCM(n, D) are specified as follows.

For the calculation of γ_t in (2.23), we have

$$P(S_t = m \mid S_t = m') = \begin{cases} 1/n, & \text{if transition m}' \to m \text{ is allowed at stage t} \\ 0, & \text{otherwise} \end{cases} \tag{6.6}$$

Denote the channel symbol assigned to transition $m' \to m$ as $X^{(m', m)}$, we have

$$P(X_t = X, S_t = m \mid S_{t-1} = m') = \begin{cases} 1, & \text{X} = \text{X}^{(m', m)} \\ 0, & \text{otherwise} \end{cases} \tag{6.7}$$

In implementation, $\gamma_t(m', m)$ is stored as a sparse matrix only for m' and m where transition $m' \to m$ is allowed in order to reduce the computation load and memory usage.

6.2.3 Basic Steps of Iterative Circular Shift BCJR Decoding

Now, we can summarize the basic steps for the decoding of HDCTCM using iterative circular shift BCJR. "Selective normalization " and modified stop condition for iterations are included here as well.

Given $Y_1^L = Y_1, Y_2, ..., Y_L$:

1. Hard decode Y_1^L to $\hat{X}_1^L = \hat{X}_1, \hat{X}_2, ..., \hat{X}_L$ and find the most reliable symbol Y_i. Denote the state transition to which $\hat{X}_i$ is assigned as $m'' \to m'''$. Circular shift Y_1^L to $Y_1^{L\,SHIFT(i)} = Y_i, Y_{i+1}, ..., Y_L, Y_1, Y_2, ..., Y_{i-1}$. Decode $Y_1^{L\,SHIFT(i)}$ in the following steps.
2. Set initial α_0 as in Equation (6.3).
3. Set iteration count $r = 1$. Calculate a set of $\alpha_{rL+1}, \ldots, \alpha_{(r+1)L}$ by

   $$\alpha_{rL+t} = \alpha_{rL+t-1}\gamma_{rL+t}, \qquad t = 1,...,L \tag{6.8}$$

   using $\gamma_{rL+t} = \gamma_t$. (6.9)

   Normalize α_{rL+t} only when it is too small or $t = L$ using

   $$\alpha_{rL+t} = \alpha_{rL+t} / \sum_i \alpha_{rL+t}(i) \tag{6.10}$$

   Calculate $\Delta = \max(abs(\alpha_{rL}^0 - \alpha_{(r-1)L}^0))$, when Δ is sufficiently small, stop and the set of α vectors is obtained as $\alpha_t = \alpha_{rL+t}$, $t = 1,...,\mathrm{L}$, Otherwise, $r = r + 1$, repeat the iteration.
4. Execute a similar procedure backward along the trellis circle to find the set $\beta_1^o \ldots \beta_L^o$. Set $\beta_L = \alpha_0$ initially.
5. Calculate $\sigma_t(m', m)$ using Equation (2.20)
6. Decode the information sequence using Equations (6.4) and (6.5), denoted as $\hat{U}$, then the original information sequence corresponding to Y_1^L is $\hat{U}^{SHIFT(L-i)}$.

6.2.4 Calculate Bit Error Probability for HDCTCM(n, D)

For simulations of decoding using iterative circular shift BCJR on HDCTCM(n, D) codes, bit error probability needs to be calculated. This can be done in two ways.

In the first method, decode the symbol in information sequence at each stage, and rewrite each symbol as $\log_2 n$ bits.

The second method is to decode each of the $\log_2 n$ bits in each symbol at each stage. This can be done by establishing the relationship between the bits in the symbol and the state transitions at that stage. For example, for HDCTCM$(4, D)$, the information symbol alphabet is $\{0,1,2,3\}$. Symbols 0 and 1 have bit 0 at the first place and symbol 0

and 2 have bit 0 at the second place. Rewrite each symbol in the information sequence U_i as two bits $U_i^{(1)}U_i^{(2)}$. Similar to Equation (6.4), we have

$$P(U_t^{(1)} = 0 \mid Y_1^L) = \frac{1}{\sum_{\text{all } (m',m)} \sigma_t(m',m)} \sum_{(m',m)\in(C^{(0)} \cup C^{(1)})} \sigma_t(m',m) \tag{6.11}$$

$$P(U_t^{(2)} = 0 \mid Y_1^L) = \frac{1}{\sum_{\text{all } (m',m)} \sigma_t(m',m)} \sum_{(m',m)\in(C^{(0)} \cup C^{(2)})} \sigma_t(m',m) \tag{6.12}$$

We decode $U_t^{(k)} = 0$, k=1,2, if $P(U_t^{(k)} = 0 \mid Y_1^L) \geq 0.5$, otherwise $U_t^{(k)} = 1$.

The bit error probability in simulation is the number of different bits between the decoded information sequence and the information sequence inputted to the encoder over total number of bits in the information sequence. Since these two methods both relate their decoding events to the APP of state transitions at each stage, there should be no significant difference between the resultant bit error probabilities. Figure 6.1 shows the bit error probability for decoding a HDCTCM$(4,2)$ code with information length L=16 using iterative circular shift BCJR. E_b/N_0 is the average bit energy per noise power spectrum density. The "decode by symbol" and "decode by bit" refer to the first and second method, respectively.

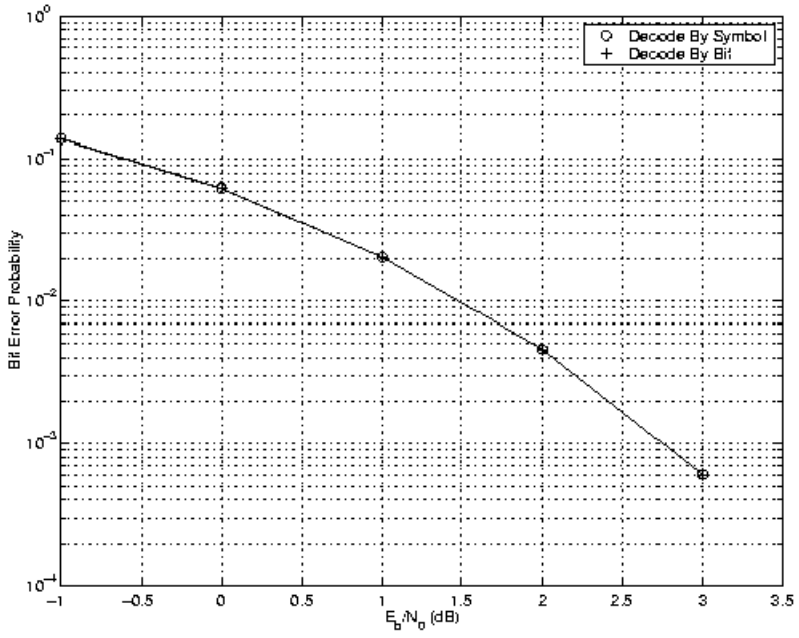


Figure 6.1 Bit error probability of decoding a HDCTCM (4,2) code with L=16 using iterative circular shift BCJR, for two different error bit probability calculation methods

6.3 Properties of Iterative Circular Shift BCJR Decoding

In this section, simulation results are given to explore the properties of iterative circular BCJR decoding algorithm. Also, comparisons with other decoding algorithms are presented.

6.3.1 Comparison with Circular BCJR with Eigenvectors

We have argued that the computation load for circular BCJR with eigenvectors is much larger than the iterative circular BCJR for large trellis. Tables 6.1 and 6.2 show the execution time spent by CPU for decoding HDCTCM$(4, D)$codes with $L = 16$ at $\frac{E_b}{N_0} = -1$ *d*B and $\frac{E_b}{N_0} = -2$ *d*Busing these two algorithms. The stop condition used by iterative circular shift BCJR is $\Delta < 10^{(-3)}$. Execution time is accumulated over 100 information sequences. Time unit is seconds. Simulations are written in Matlab running on a DEC-alpha-600MHz workstation. It can be seen that the difference between the execution times is getting larger when trellis depth D is increased.

Table 6.1 Execution Time of Decoding HDCTCM$(4, D)$ Codes Using Different Circular BCJR Algorithms at E_b/N_0=-1 dB

Trellis depth D	Circular BCJR with eigenvectors	Iterative circular shift BCJR ($\Delta < 10^{(-3)}$)
2	2.48	2.87
3	18.51	11.97
4	763.60	164.10
5	148294.00	4711.00

Table 6.2 Execution Time of Decoding HDCTCM$(4, D)$ Codes Using Different Circular BCJR Algorithms at E_b/N_0=2 dB

Trellis depth D	Circular BCJR with eigenvectors	Iterative circular shift BCJR ($\Delta < 10^{(-3)}$)
2	2.34	2.5100
3	18.14	9.8600
4	735.49	111.4900
5	147336.00	3615.00

Figures 6.2 and 6.3 show the bit error probability of decoding a HDCTCM$(4,2)$ code and a HDCTCM$(4,3)$ code using these two schemes. These figures show that at stop condition $\Delta < 10^{(-3)}$, the set of α's and β's in iterative circular shift BCJR converge close enough to those obtained by circular BCJR with eigenvectors to make almost the same decode decision.

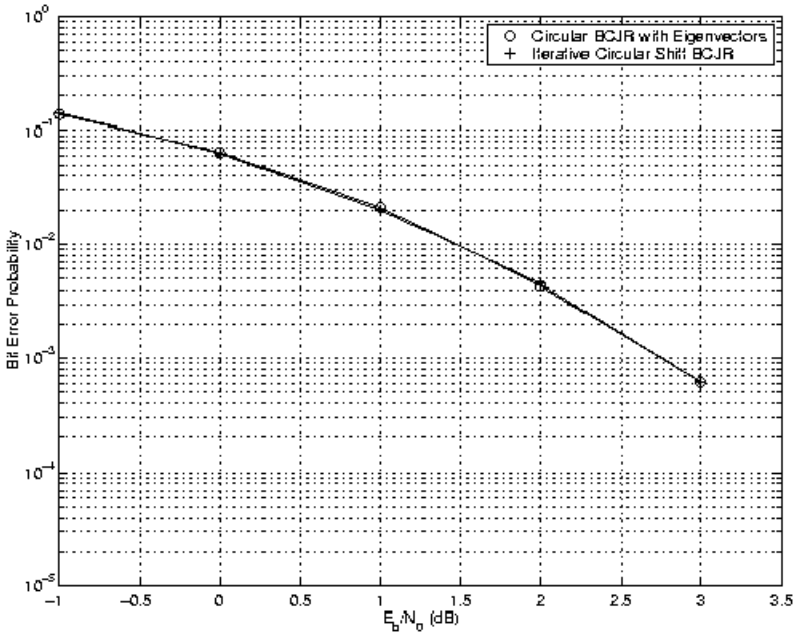


Figure 6.2 Bit error probability of decoding a HDCTCM (4,2) code with L=16 for different decoding schemes

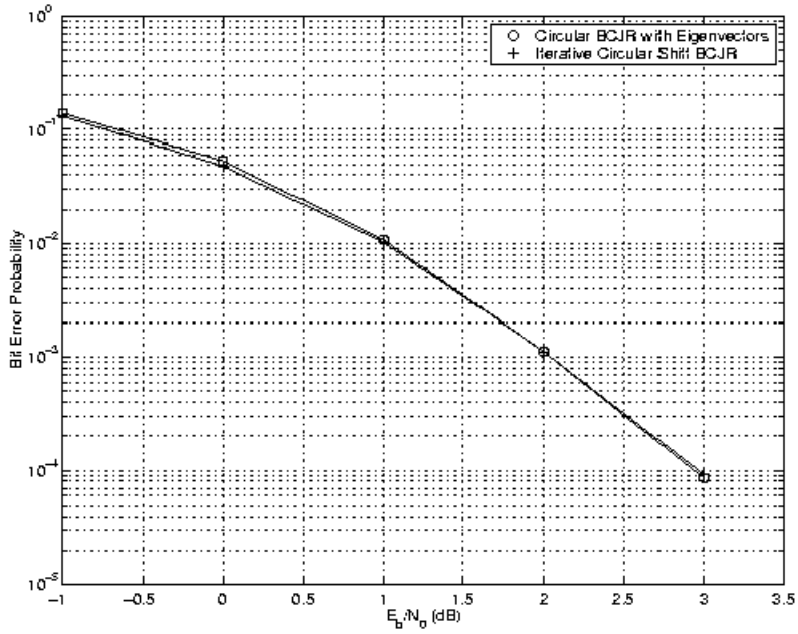


Figure 6.3 Bit error probability of decoding a HDCTCM (4,3) code with L=16 for different decoding schemes

6.3.2 Convergence Property of Iterative Circular Shift BCJR Algorithm

The speed of convergence for iterative circular BCJR is affected by the initialization of α_0 and β_L and the stop condition for the iterations. The convergence property of iterative circular shift BCJR is investigated here. For comparison, an iterative circular BCJR using a different initialization is chosen as a reference decoding scheme. In this reference scheme, since a legal path can start and end at one of all the possible states with equal possibility in HDCTCM(n, D), in the absence of better knowledge, simply set initial α_0 and β_L as follows:

$$\alpha_0(m) = \beta_L(m) = \frac{1}{n^D} \quad \text{, for all state } m \tag{6.13}$$

Then decode the received sequence (not shift version) using the iterative circular BCJR. The number of iterations required for these two schemes is compared.

Simulations are done on a HDCTCM$(4,3)$ code with L=16. Figures 6.4 and 6.5 show the number of iterations required by α and β in these two decoding schemes both at stop condition $\Delta < 10^{(-3)}$. "Equally likely initialization" refers to the reference scheme. The number of iterations is averaged over 10,000 information sequences. For low E_b / N_0, the estimation of starting state in iterative circular shift BCJR is no better than the simple initialization in the reference scheme, but for medium and high E_b / N_0, the estimation of starting state leads to faster convergence than the reference scheme. For $\frac{E_b}{N_0} \geq 2$ dB, no more than two iterations are needed to converge at $\Delta < 10^{(-3)}$ for both schemes.

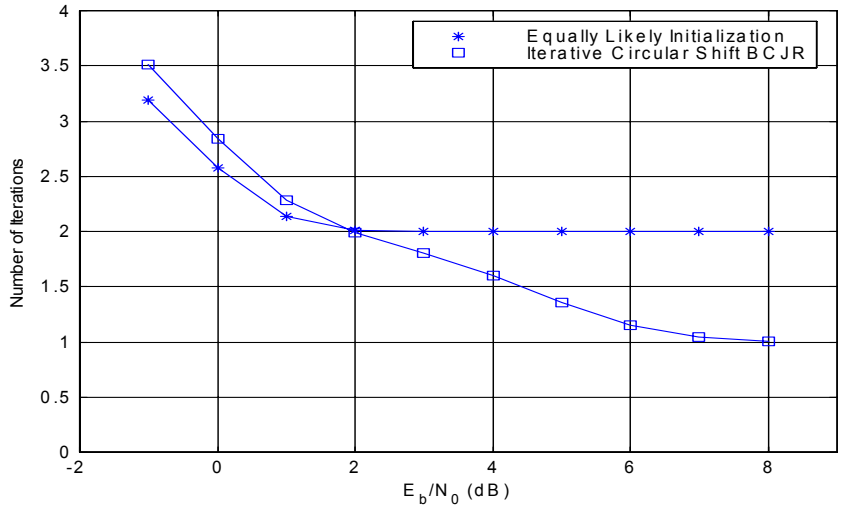


Figure 6.4 Number of iterations required by α for decoding a HDCTCM$(4,3)$ code using different decoding schemes at $\Delta < 10^{(-3)}$

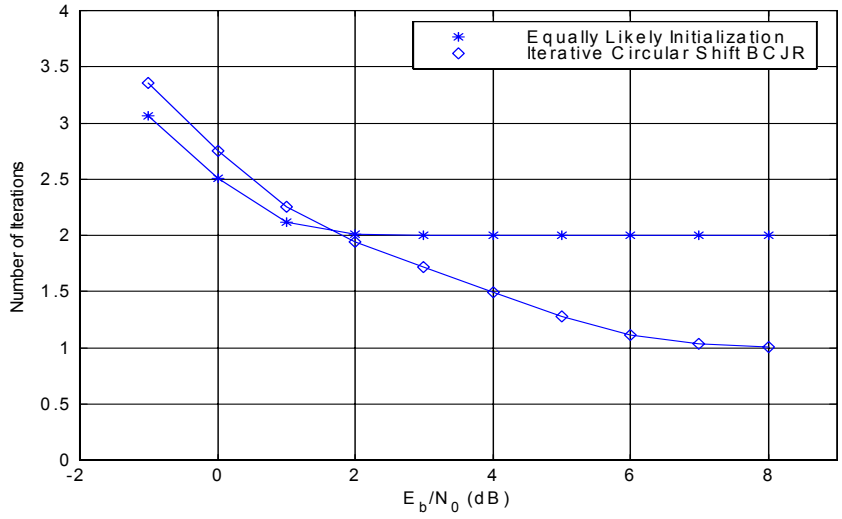


Figure 6.5 Number of iterations required by β for decoding a HDCTCM$(4,3)$ code using different decoding schemes at $\Delta < 10^{(-3)}$

Figure 6.6 shows the bit error probability of decoding this code using these two decoding schemes. All these figures show iterative circular BCJR forgets an erroneous

starting α_0 and β_L very rapidly and converge to the objective steady value after just several iterations.

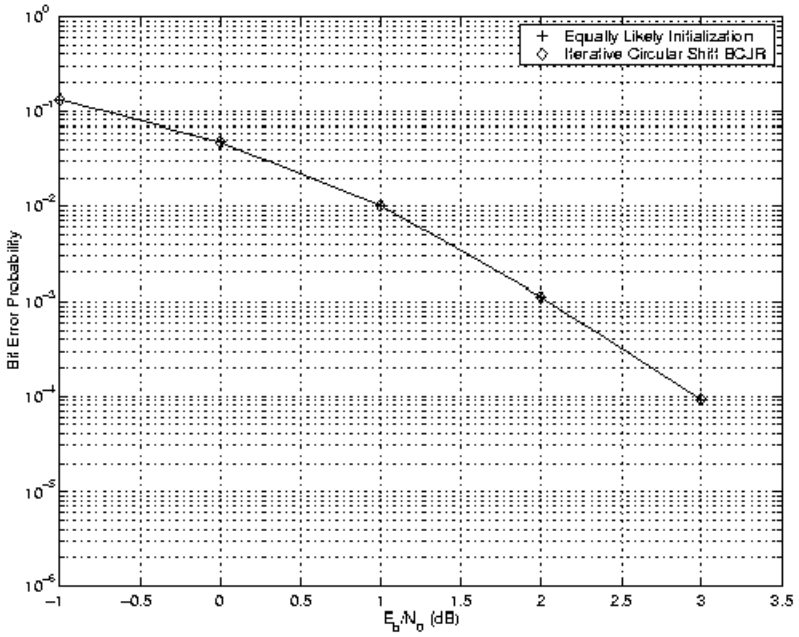


Figure 6.6 Bit error probability of decoding a HDCTCM (4,3) code with L=16 for different decoding schemes at $\Delta < 10^{(-3)}$

Also, for iterative circular shift BCJR decoding, the number of iterations needed at two different stop conditions, $\Delta < 10^{(-10)}$ and $\Delta < 10^{(-3)}$, are compared. Decoding is on the same HDCTCM (4,3) code with L=16. Figures 6.7 and 6.8 show the number of iterations required by α and β averaged over 10,000 information sequences. They show that at very low E_b / N_0, under stop condition $\Delta < 10^{(-10)}$, α and β need about twice as many iterations as under stop condition $\Delta < 10^{(-3)}$. The difference becomes smaller as E_b / N_0 increases. Figure 6.9 is the bit error probability of decoding this code under these two stop conditions. It shows that α and β obtained at $\Delta < 10^{(-3)}$ lead to almost the same decode decision as those obtained at $\Delta < 10^{(-10)}$. $\Delta < 10^{(-3)}$ is small enough to control the convergence.

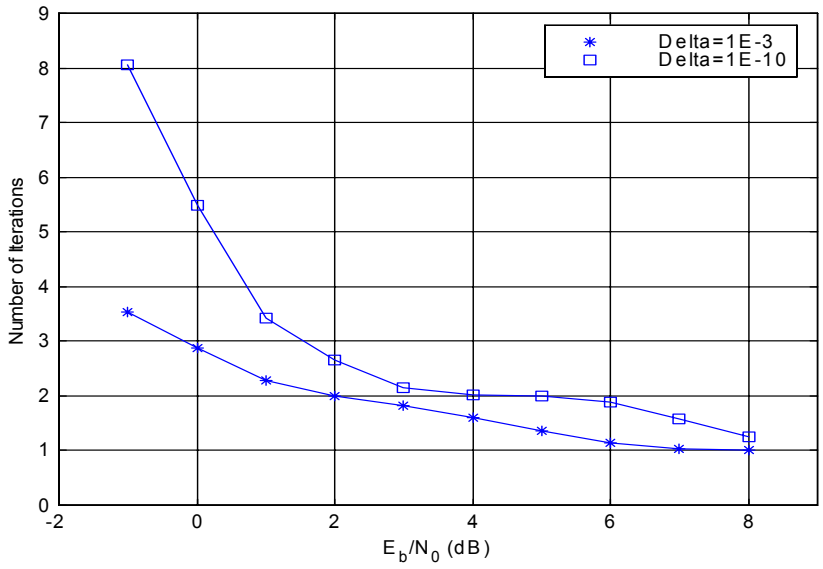


Figure 6.7 Number of iterations required by α of decoding a HDCTCM (4,3) with L=16 using iterative circular shift BCJR for two stop conditions $\Delta < 10^{(-3)}$ and $\Delta < 10^{(-10)}$

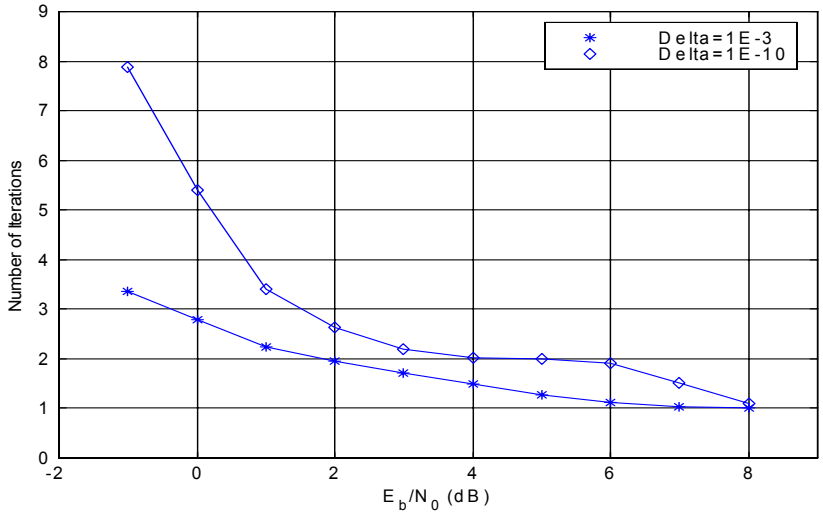


Figure 6.8 Number of iterations required by β of decoding a HDCTCM (4,3) with L=16 using iterative circular shift BCJR for two stop conditions $\Delta < 10^{(-3)}$ and $\Delta < 10^{(-10)}$

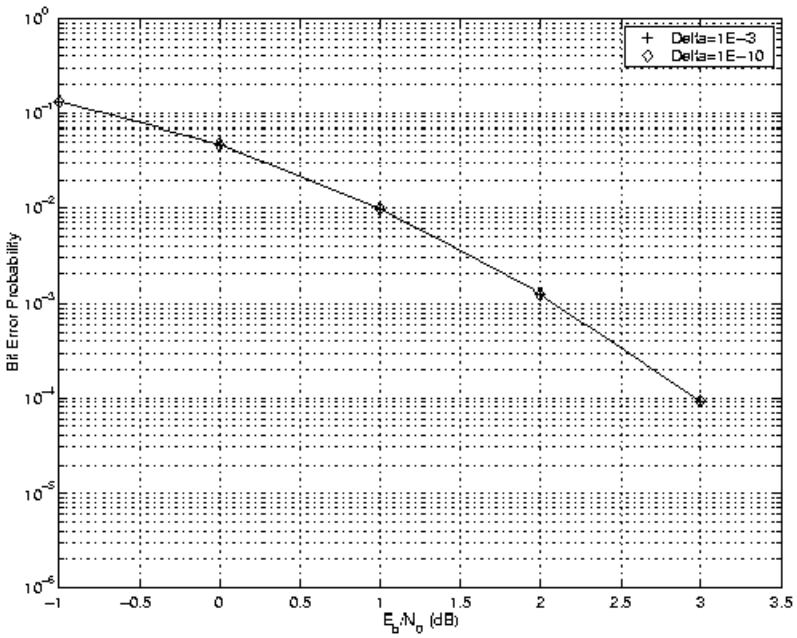


Figure 6.9 Bit error probability of decoding a HDCTCM (4,3) with L=16 using iterative circular shift BCJR for two stop conditions $\Delta < 10^{(-3)}$ and $\Delta < 10^{(-10)}$

6.3.3 Comparison with Viterbi Decoding

Error performance of iterative circular shift BCJR is compared with Viterbi decoding. Here Viterbi decoding means that for decoding HDCTCM(n, D), run Viterbi algorithm a total number of S times--one time for each possible starting state. Figures 6.10 and 6.11 show the error performance of decoding a HDCTCM$(4,2)$ code and a HDCTCM$(4,3)$ code, both with L=16 using these two decoding algorithms. These figures show iterative circular shift BCJR decoding achieves at least as good a bit error probability as ML decoding—Viterbi decoding. But in simulation, the execution time of iterative circular shift BCJR is less than the Viterbi decoding and the difference gets bigger when the trellis becomes larger. This can be explained from the analysis of the computation complexity of these two decoding schemes as follows.

For decoding HDCTCM(n, D) with information sequence length L, in iterative circular shift BCJR decoding, at each trellis stage t, the calculation of the set of α_t or β_t requires $S \cdot n$ multiplications and S additions of n numbers each, and the calculation of σ_t requires $S \cdot n$ multiplications of three numbers each. The computation of γ_t at each

stage is quite simple and implemented as table lookup. Therefore, if R is the number of iterations needed, then roughly the total computation load is $R \cdot L \cdot (3S \cdot n(\otimes) + S \cdot (n-1)(\oplus))$. $\otimes$ and $\oplus$ denote a multiplication and addition operation on two numbers. Due to the good convergence property of this algorithm, R is always in the range of integer from 1 to 4. In Viterbi decoding--referring to Equation (4.5), at each stage t--it requires a calculation quantity essentially similar to γ_t and the

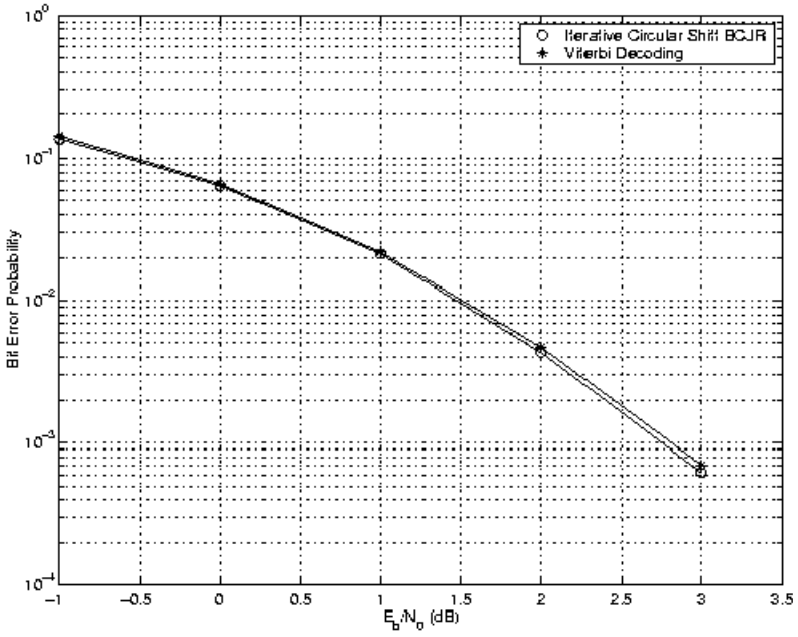


Figure 6.10 Error performance of Viterbi and iterative circular shift BCJR decoding algorithms on a HDCTCM (4,2) code with L=16

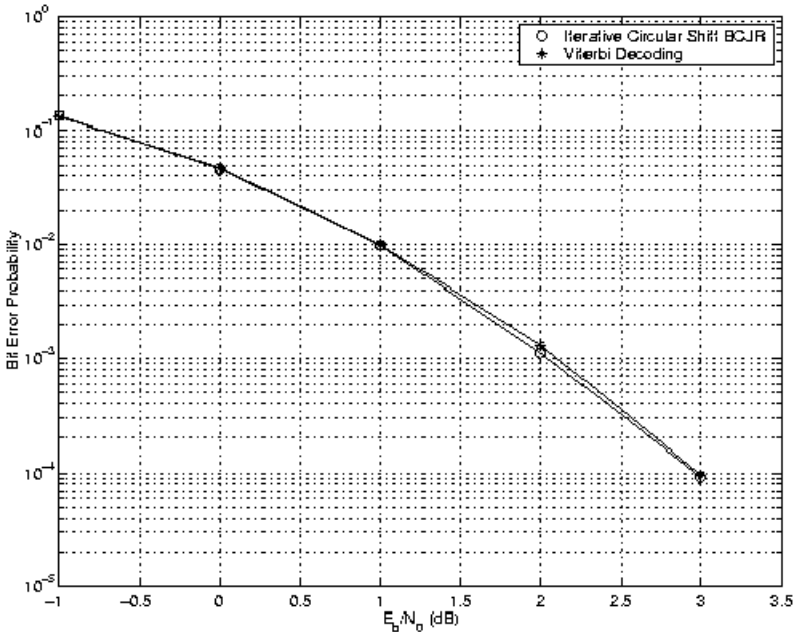


Figure 6.11 Error performance of Viterbi and iterative circular shift BCJR decoding algorithms on a HDCTCM (4,3) code with L=16

add-compare-select operation on all the states. The add-compare-select operations require $S \cdot n$ additions and S comparisons of $S \cdot n$ numbers each. For S Viterbi runs, the computation load is about $S \cdot L \cdot (S \cdot n(\oplus) + S(\text{comparision on (S} \cdot \text{n) numbers}))$. We can see the computation load is linear to the second order of total number of states S, whereas in iterative circular shift BCJR, it is linear to the first order of S with a small slope R--a small fraction of S. This explains the difference of the execution times and why the difference becomes bigger for larger S.

6.4 Bit Error Performance of HDCTCM

In this section, simulation results are given to obtain an overall bit error performance of HDCTCM$(4, D)$. The decoding uses iterative circular shift BCJR algorithm.

We have discussed the importance of the minimum distance of a code in determining its error performance. From Figures 4.2 to 4.5, we see the minimum distance of HDCTCM$(4, D)$ is increased with big steps when the information length L is increased starting from $D+1$, and then will reach a steady value at some point.

Accordingly, the bit error probability will decrease with big steps when L is increased starting from $D+1$ and then will not decrease as much after L reaches some value. This trend is shown in Figures 6.12 and 6.13. In these figures, we plot the bit error probability versus the information length for decoding a HDCTCM$(4,2)$ code and a HDCTCM$(4,3)$ code.

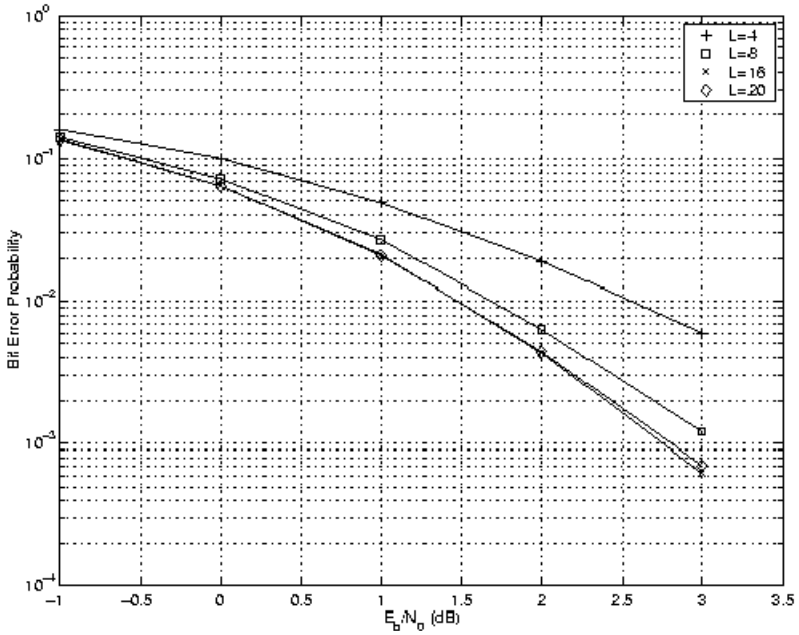


Figure 6.12 Bit error probability of a HDCTCM (4,2) code with different information sequence length L decoded using iterative circular shift BCJR

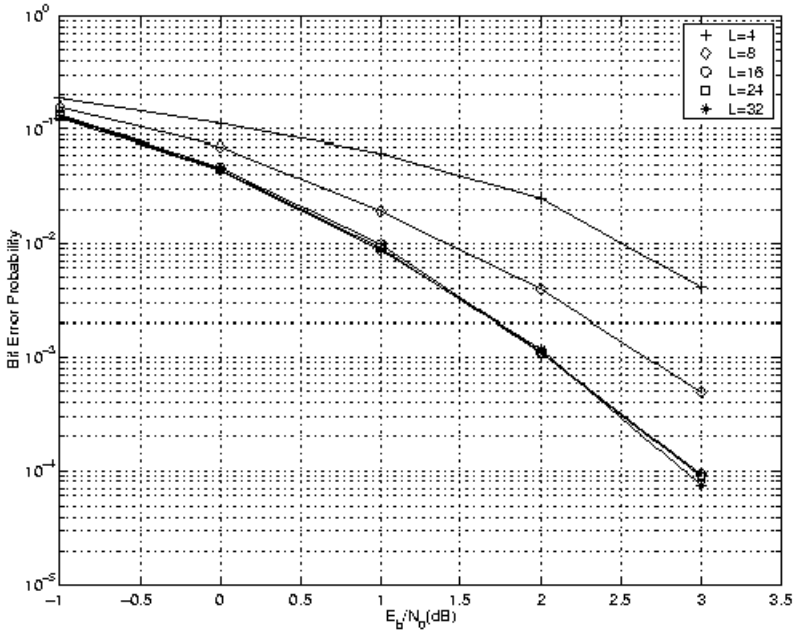


Figure 6.13 Bit error probability of a HDCTCM (4,3) code with different information sequence length L decoded using iterative circular shift BCJR

For any HDCTCM$(4, D)$ code with $L > 2D + (0 \sim 2)$, optimal distance codes are built in Chapter 5. Figure 6.14 shows the bit error probability of the optimal distance codes for HDCTCM$(4, D)$, $D = 2,3,4,5$. Uncoded 4-PSK is regarded as a reference system. Its bit error probability is also plotted here. S is the total states in trellis. $DIS_{\min}$ is the minimum distance of this code. It can be seen that, at bit error probability $10^{(-5)}$, the coding gain for 16, 64, 256, and 1024 states trellis are approximately 5.0 dB, 6.0 dB, 6.6 dB, and 7.2 dB, respectively.

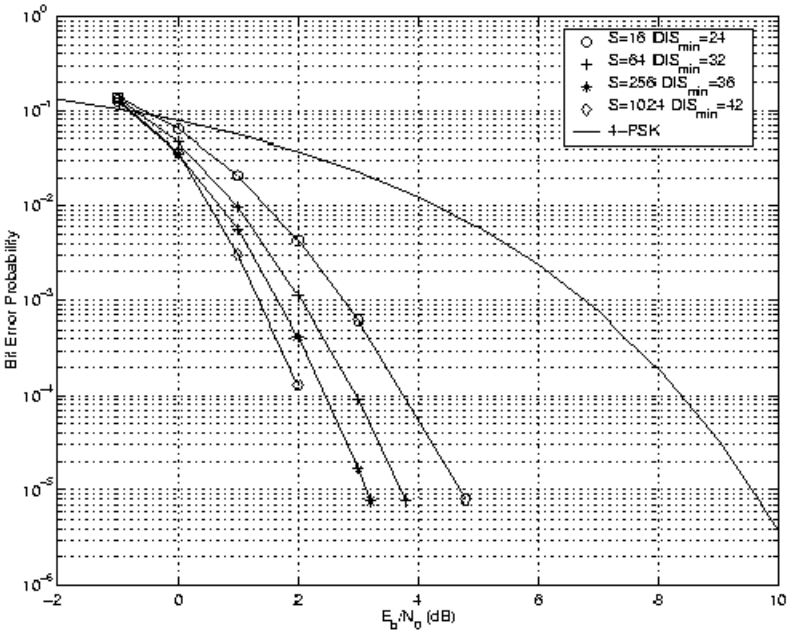


Figure 6.14 Error performance of optimal distance HDCTCM (4,D) codes with L=16 for D=2,3,4,5, compared with uncoded 4-PSK signaling

CHAPTER 7 CONCLUSIONS

7.1 Summary

Prior to this dissertation, a circular trellis coding with permuted state structure was invented to satisfy the state constraint without code rate loss or initialization of the shift register. Trellis coded modulation has achieved successful applications in band-limited channels.

This dissertation develops a systematic high dimensional circular trellis-coded modulation with permuted state structure (HDCTCM) for power-limited spread spectrum channels. High dimensional simplex signal constellation is systematically developed to achieve the optimal energy efficiency and maximize the minimum distance among error events for any size trellis. This is done by analyzing the error events and identifying sets of state transitions that should be assigned simplex to achieve maximum coding gain. Butterfly structure of trellis coding is successfully related to those state transitions and perfectly aligned into a multidimensional matrix--State Transition Matrix (STM). This matrix is designed in such a way that sets of state transitions that should be assigned simplex are located in the same internal location of the butterflies along a particular matrix dimension. An algebraic representation of a simplex in a high dimensional space is built. And a similar multidimensional matrix--Initial Input Simplex (IIS)--having simplex as unit element is built in such a way that the signals in the same internal location of the simplexes along a particular matrix dimension is also a simplex. A one to one correspondence is built between the states in STM and signals in IIS, and between the state transitions in STM and signals in the signal constellation. The construction of STM, IIS, and the symbol assignment are applicable to any size trellis.

Minimum distance of a code is a primary parameter in determining its error correcting capacity. In circular trellis coding, the starting state of a legal path can be any of the total states in this trellis; conventional algorithm for calculating the minimum distance of a trellis code always assumes that the starting state is 0 state. This dissertation

develops a practical computational algorithm to calculate minimum distance of circular trellis coding. The minimum distance of HDCTCM is obtained using this algorithm. The coding gain of HDCTCM is evaluated after the minimum distance is obtained.

Due to the systematic construction of the signal constellation and symbol assignment, upper and lower bonds on the minimum distance of HDCTCM codes are derived. Also, this dissertation proves that these bounds can be reached. Furthermore, a method to build codes that have the bounds of the minimum distance is developed. Currently, in most coding schemes, the optimal distance codes can only be obtained through exhaustive search.

With an unknown starting state, the maximum likelihood decoder (Viterbi decoder) is not suitable for HDCTCM. This dissertation explores the circular BCJR algorithm for decoding HDCTCM. Iterative circular shift BCJR is developed. In this decoding scheme, the starting state is statistically estimated using tentative hard decision with reliability information, and the circular character of HDCTCM is incorporated to make the most reliable symbol in the received sequence as the starting point for decoding. Simulations show very good convergence property of iterative circular shift BCJR by comparison with other circular BCJR algorithms and for different stop conditions. Error performance of this decoding algorithm is compared with Viterbi decoding. Finally, satisfactory bit error performance achieved by HDCTCM using iterative circular shift BCJR decoding is given and compared with corresponding uncoded system.

7.2 Further Research

Further research may explore the concatenated HDCTCM codes. Two types of systems are possible:

1. Concatenate with other kind of error correcting codes. Using HDCTCM as the inner code and other error correcting codes, such as BCH codes [30], parity check codes etc. as the outer code.
2. Parallel concatenation of two or more HDCTCM codes separated by an interleaver. This structure is similar to turbo codes.

Concatenated codes are showed to have increased error-correcting capacity compared to their constituent codes. The decoder for concatenated codes should comprise decoders for each individual constituent code. Soft-output information (reliability of a hard decision) other than hard decision itself should be generated and exchanged between the constituent decoders [38] [39] [40].

Because of the good error correcting capacity of HDCTCM, we expect the concatenated HDCTCM to have better performance--possibly as good as turbo codes. Since the iterative circular shift BCJR already calculates the soft-output information and has iterative nature, it is very possible to be extended for decoding the concatenated HDCTCM codes.

Further research can also explore the possibility to increase the speed of iterative circular shift BCJR algorithm. By incorporating more characteristics of HDCTCM, knowledge on partial path can determine the whole path, and it is possible to reduce the computation in current algorithm in this direction.

REFERENCES

[1] Bernard Sklar, Digital Communications: Fundamentals and Applications, Prentice Hall, 1988.

[2] H. H. Ma and J. K. Wolf, "On Tail Biting Convolutional Codes," *IEEE Trans. Communication*, Vol. COM-34, pp. 104-111, Feb. 1986.

[3] G. Solomon and H. C. A. Van Tilborg, "A Connection Between Block and Convolutional Codes," *SIAM J. Appl. Math.*, Vol. 37, pp. 358-369. Oct. 1979.

[4] C.Berrou, A. Glavieux, and P. Thitimajashima, "Near Shannon Limit Error Correcting Coding and Decoding: Turbo codes," Proceeding of ICC'93, Geneva, Switzerland, May 1993, pp. 1064-1070.

[5] P. Jung and M. Nabhan, "Performance Evaluation of Turbo Codes for Short Frame Transmission Systems," *Electron. Lett.*, 1994. 30. (2), pp. 111-113.

[6] P. Jung and M. Nabhan, "Dependence of the Error Performance of Turbo Codes on the Interleaver Structure in Short Frame Transmission Systems," *Electron. Lett.*, 1994. 30 (4), pp. 287-288.

[7] O. Joerssen and H. Meyr, "Terminating the Trellis of Turbo Codes," *Electron. Lett.*, 1994. 30. (16), pp. 1285-1286.

[8] W. J. Blackert, E. K. Hall and S. G. Wilson, "Turbo Code Termination and Interleaver Cconditions," *Electron. Lett.*, 1995. 31. (24), pp. 2082-2084.

[9] Y. C. Lo, J. C. Dill, and A. R. Lindsey, "Circular Trellis-Coded Modulation With Permuted State Structure," *Defense Applications of Signal Proceeding*, Adelaide, Australia, Aug. 1997.

[10] Y. C. Lo, J. C. Dill, C. Chen, and S. R. Lopez-Permouth, "The Encoding of a New Trellis-Coded Modulation with State Permutation Structure," *4th Int. Symposium on Communication Theory and Applications*, Ambleside, UK, July 1997.

[11] J. C. Dill, A. R. Lindsey, Y. C. Lo, and C. Chen, "A Novel Error Correcting Codec for M-ary Orthogonal Modulations," *Asilomar Conf. On Signals, Systems and Computers, Monterey*, CA. Nov. 1996.

[12] G. Ungerboeck, "Channel Coding With Multil-Level/Phase Signals," *IEEE Trans. Inf. Theory*, Jan. 1982, pp. 55—67.

[13] M. Eyuboglu, G. D. Forney, P. Doug, G. Long, "Advanced Modulation Techniques for V. Fast," *Eur. Trans. on Telecom*, May,1993, pp. 243-256.

[14] Frank A. Alder, "Signal Assignment and Performance of Simplex Signaling in High Dimensional Trellis-Coded Modulation," *Thesis*, August 1998.

[15] A. J. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," *IEEE Trans. Inf. Theory*, IT-13, April 1967, pp. 260-269.

[16] R. M. Fano, "A Heuristic Discussion of Probabilistic Decoding," *IEEE Trans. Inf. Theory*, IT-9, April 1963, pp. 64-74.

[17] J. L. Massey, Threshold Decoding, MIT Press, Cambridge, Mass., 1963.

[18] J. M. Wozencraft and B. Reiffen, Sequential Decoding, MIT Press, Cambridge, Mass., 1961.

[19] G. D. Forney, Jr., "The Viterbi Algorithm," *Proc. IEEE*, 62, March 1973, pp. 268-278.

[20] G. D. Forney, Jr., "Convolutional Codes II: Maximum Likelihood Decoding," *Inf. Control*, 25, July 1974, pp. 222-266.

[21] L. R. Bahl, J. Cocke, F. Jelinek and Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Trans. Inf. Theory*, vol. IT-20, pp. 284-287. Mar. 1974.

[22] John B. Anderson and Stephen M. Hladik, "Tailbiting MAP Decoders," *IEEE Journal on Selected Areas in Communications*, vol. 16, No. 2, Feb., 1998, pp297-302.

[23] X. Y. Song and J. C. Dill, "A High Dimensional Simplex Signaling Scheme in Circular Trellis Coded Modulation with Permuted State Structure," to be submitted.

[24] Stephen G. Wilson, Digital Coding and Modulation, Prentice Hall, Upper Saddle River, New Jersey 1996.

[25] M. M. Mulligan and S. G. Wilson, "An Improved Algorithm for Evaluating Trellis Phase Codes," *IEEE Trans. Inf. Theory*, Vol. IT-30, No. 6, pp. 846-851, Nov. 1984.

[26] S. Benedetto, E. Biglieri and V. Castellani, Digital Transmission Theory, Prentice-Hall, Englewood Cliffs, N. J., 1987,

[27] X. Y. Song and J. C. Dill, "Practical Computational Algorithm to Calculate Minimum Distance in Circular Trellis Coding," to be submitted

[28] X. Y. Song and J. C. Dill, "Upper and Lower Bounds on the Minimum Distance of High Dimensional Circular Trellis-Coded Modulation," to be submitted.

[29] X. Y. Song and J. C. Dill, "Optimum Distance Codes for High Dimensional Circular Trellis-Coded Modulation," to be submitted

[30] Shu Lin and Daniel J. Costello, Jr. Error control coding: Fundamentals and Applications, Prentice Hall, 1983.

[31] P. Elias, "Coding for Noisy Channels," *IRE Conv. Rec.*, Part 4, 1955, pp.37-47.

[32] Thapar, H. K., "Real-Time Application of Trellis Coding to High-Speed Voiceband Data Transmission," *IEEE J. Sel. Areas Commun.*, Vol. SAC2, no.5, Sept. 1984, pp.648-658.

[33] John M. Wozencraft and Irwin Mark Jacobs, Principles of Communication Engineering, Waveland Press, Inc., 1990, c1965.

[34] George R. Cooper and Clare D. McGillem, Modern Communications and Spread Spectrum, McGraw-Hill, New York, c1986.

[35] Ezio Biglieri, D. Divsalar, P. J. Mclane and M. K. Simon, Introduction to Trellis-Coded Modulation with Applications, Macmillan Publishing Co., NY., 1991.

[36] Richard E. Blahut, Theory and Practice of Error Control Codes, Addison-Wesley Publishing Company, May. 1984.

[37] R. C. P. Saxena, "Optimum Encoding in Finite State Coded Modulation," *Report TR 83-2*, Department of Electrical, Computer and System Engineering, Rensselaer Polytechnic Institute, Troy, N.Y., 1983.

[38] Joachim Hagenauer and Peter Hoeher, "A Viterbi Algorithm with Soft-Decision Outputs and its Applications," *IEEE Global Telecommunications Conference & Exhibition*, Dallas, TX, 1989, pp 1680-1686.

[39] Xiao-an Wang and Stephen B. Wicker, "A Soft-Output Decoding Algorithm for Concatenated Systems," *IEEE Trans. Inf. Theory*, Vol. 42, No. 2, March 1996, pp 543-553.

[40] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Soft-Output Decoding Algorithm in Iterative Decoding of Turbo Codes," *TDA Progress Report 42-124*, Feburary 15, 1996.

Appendix A List of all paths for HDCTCM (4,3) with information sequence length L=D+1

1	1	1	1	1
1	2	3	4	1
1	5	6	64	1
1	34	35	33	1
2	3	4	1	2
2	8	9	4	2
2	15	16	64	2
2	37	38	33	2
3	4	1	2	3
3	14	3	14	3
3	10	6	7	3
3	50	35	36	3
4	5	26	13	4
4	34	18	49	4
4	1	2	3	4
4	2	8	9	4
5	6	64	1	5
5	26	13	4	5
5	11	63	64	5
5	47	32	33	5
6	7	3	10	6
6	64	1	5	6

6	17	35	46	6
6	25	6	25	6
7	8	29	16	7
7	3	10	6	7
7	37	21	24	7
7	15	30	63	7
8	9	4	2	8
8	28	9	14	8
8	29	16	7	8
8	20	38	36	8
9	10	26	27	9
9	50	18	19	9
9	4	2	8	9
9	14	8	28	9
10	11	12	13	10
10	47	48	49	10
10	6	7	3	10
10	26	27	9	10
11	12	13	10	11
11	63	64	5	11
11	31	32	46	11
11	62	63	25	11

12	13	10	11	12
12	27	29	30	12
12	53	30	62	12
12	55	21	61	12
13	14	15	54	13
13	4	5	26	13
13	50	51	52	13
13	10	11	12	13
14	15	54	13	14
14	37	56	49	14
14	3	14	3	14
14	8	28	9	14
15	16	64	2	15
15	54	13	14	15
15	30	63	7	15
15	43	32	36	15
16	17	18	42	16
16	25	26	53	16
16	7	8	29	16
16	64	2	15	16
17	18	42	16	17
17	35	46	6	17
17	40	60	24	17
17	51	61	63	17
18	19	9	50	18
18	49	4	34	18
18	57	38	39	18
18	42	16	17	18
19	20	58	41	19
19	29	43	48	19

19	28	20	56	19
19	9	50	18	19
20	21	52	27	20
20	58	41	19	20
20	38	36	8	20
20	56	19	28	20
21	22	48	57	21
21	61	12	55	21
21	52	27	20	21
21	24	7	37	21
22	23	22	23	22
22	44	59	60	22
22	32	39	51	22
22	48	57	21	22
23	24	25	47	23
23	52	53	43	23
23	61	62	31	23
23	22	23	22	23
24	25	47	23	24
24	17	40	60	24
24	64	34	51	24
24	7	37	21	24
25	26	53	16	25
25	6	25	6	25
25	47	23	24	25
25	11	62	63	25
26	27	9	10	26
26	13	4	5	26
26	55	38	46	26
26	53	16	25	26

27	28	29	54	27
27	9	10	26	27
27	20	21	52	27
27	29	30	12	27
28	29	54	27	28
28	20	56	19	28
28	9	14	8	28
28	28	28	28	28
29	30	12	27	29
29	43	48	19	29
29	16	7	8	29
29	54	27	28	29
30	31	48	42	30
30	62	12	53	30
30	12	27	29	30
30	63	7	15	30
31	32	46	11	31
31	48	42	30	31
31	23	61	62	31
31	44	60	61	31
32	33	5	47	32
32	36	15	43	32
32	46	11	31	32
32	39	51	22	32
33	34	40	45	33
33	5	47	32	33
33	2	37	38	33
33	1	34	35	33
34	35	33	1	34
34	18	49	4	34

34	51	24	64	34
34	40	45	33	34
35	36	3	50	35
35	33	1	34	35
35	39	35	39	35
35	46	6	17	35
36	37	58	45	36
36	15	43	32	36
36	8	20	38	36
36	3	50	35	36
37	38	33	2	37
37	56	49	14	37
37	21	24	7	37
37	58	45	36	37
38	39	18	57	38
38	46	26	55	38
38	36	8	20	38
38	33	2	37	38
39	40	59	45	39
39	51	22	32	39
39	18	57	38	39
39	35	39	35	39
40	41	49	50	40
40	45	33	34	40
40	59	45	39	40
40	60	24	17	40
41	42	43	44	41
41	57	58	59	41
41	49	50	40	41
41	19	20	58	41

42	43	44	41	42
42	30	31	48	42
42	54	55	56	42
42	16	17	18	42
43	44	41	42	43
43	23	52	53	43
43	48	19	29	43
43	32	36	15	43
44	45	46	47	44
44	41	42	43	44
44	60	61	31	44
44	59	60	22	44
45	46	47	44	45
45	39	40	59	45
45	33	34	40	45
45	36	37	58	45
46	47	44	45	46
46	11	31	32	46
46	26	55	38	46
46	6	17	35	46
47	48	49	10	47
47	32	33	5	47
47	44	45	46	47
47	23	24	25	47
48	49	10	47	48
48	19	29	43	48
48	42	30	31	48
48	57	21	22	48
49	50	40	41	49
49	10	47	48	49

49	14	37	56	49
49	4	34	18	49
50	51	52	13	50
50	40	41	49	50
50	35	36	3	50
50	18	19	9	50
51	52	13	50	51
51	24	64	34	51
51	22	32	39	51
51	61	63	17	51
52	53	43	23	52
52	55	58	60	52
52	13	50	51	52
52	27	20	21	52
53	54	53	54	53
53	16	25	26	53
53	43	23	52	53
53	30	62	12	53
54	55	56	42	54
54	53	54	53	54
54	27	28	29	54
54	13	14	15	54
55	56	42	54	55
55	38	46	26	55
55	58	60	52	55
55	21	61	12	55
56	57	56	57	56
56	42	54	55	56
56	19	28	20	56
56	49	14	37	56

57	58	59	41	57
57	21	22	48	57
57	56	57	56	57
57	38	39	18	57
58	59	41	57	58
58	60	52	55	58
58	41	19	20	58
58	45	36	37	58
59	60	22	44	59
59	59	59	59	59
59	45	39	40	59
59	41	57	58	59
60	61	31	44	60
60	22	44	59	60
60	24	17	40	60
60	52	55	58	60
61	62	31	23	61
61	31	44	60	61
61	63	17	51	61
61	12	55	21	61
62	63	25	11	62
62	12	53	30	62

62	62	62	62	62
62	31	23	61	62
63	64	5	11	63
63	7	15	30	63
63	25	11	62	63
63	17	51	61	63
64	2	15	16	64
64	1	5	6	64
64	34	51	24	64
64	5	11	63	64

APPENDIX B SYMBOL ASSIGNMENT CODE

```
function sym_tab=sym_asgn(n,D,DIM,start_st,permu,siv_free)

% Purpose: This function is the main function to assign channel symbols to state transitions in the state transtion table.
%Functions to generate State Transition Matrix (STM)--siv_gen.m and to generate the Initial Input Simplex(IIS)-inp_gen.m are called by this function.

%Input Parameters
%n: n-ary information symbol
%D: Trellis Depth
%DIM: Dimensions of the signal constellation
%Permu: Record the order of assign the four signals in the related source simplex to start_st to the four transitions generating from start_st.
%siv_free: a vector containing D+1 elements. They are the D+1 element in STM_FREE to build STM. start_st equavalent to STM_FREE(first %location), permu is equivalent to STM_FREE(right side butterfly), the D+1 elements in siv_free are the other D+1 elements in STM_FREE.

%In Function
%TOT_ST: Total number of states in this trellis.
%ST_TAB: State transition table for this trellis, a TOT_ST x n matrix. ST_TAB(i,:) are the next states of the transitions generating from state i; ST_TAB(i,j),i=1,2,...,TOT_ST, j=1,2,...,n, is the next state of transition generating from state i when the input is j-1.
%INV_TAB: Inverse state transition table for this trellis, a TOT_ST x n matrix. INV_TAB(i,j) is the state that can transit to state i when the input is j-1.
%ALL_PATHS: All the legal paths in this trellis when information sequence length is equal to D+1.
%SYM_INP: Symbols in the IIS.
%ALL_SYMS: All the symbols in the signal constellation.
%SIV: the left side of butterflies in STM
%INDX_TAB: A TOT_ST x TOT_ST matrix. INDX_TAB(i,j), i,j=1,2,...,TOT_ST is the symbol index assigned to the transition i->j. When i->j is not allowed, INDX_TAB(i,j)=0.

%Output Parameters
```

%sym_tab: Channel symbols assigned to each state transition in ST_TAB. It is a TOT_ST x (3 x n) matrix. sym_tab(i,[(j-1)*3+1:j*3]), j=1,2,...,n, is the channel symbol assigned to state transition i->ST_TAB(i,j-1).

%Date: Mar 31, April 6, May 25, 1999
%Author: Xiangyu Song

```
global TOT_ST ALL_PATHS ARY SIV RESHP_SYM_INP SYM_INP INV_STAB INDX_TAB ALL_SYMS L_USE;

ARY=n;
TOT_ST=n^D;
ST_TAB=st_gen(n,D);%Generate state transition table
INV_STAB=inv_tab(n,D);%Generate inverse state transition table
ALL_PATHS=all_paths(n,D);% Generate all legal path

SYM_INP=inp_gen(n,D);%Generate the IIS
if D<=2
RESHP_SYM_INP=SYM_INP;
else
RESHP_SYM_INP=[];
unit=n^2*(n*3);
for j=0:TOT_ST/n^3-1
    sel_sym_inp=SYM_INP(j*unit+1:(j+1)*unit);
    sel_sym_inp=reshape(sel_sym_inp,n^2,n*3);
    for i=0:n-1
        RESHP_SYM_INP=[RESHP_SYM_INP;sel_sym_inp(:,i*3+1:(i+1)*3)];
    end
end
end

SIV=siv_gen(n,D,siv_free);%Generate the left part of STM

ALL_SYMS=get_all_syms;
INDX_TAB=zeros(TOT_ST);
L_USE=zeros(TOT_ST,n);%THis is a loop variable.L_USE(i,:) is to recorde that among the four symbols belonging to the source simplex related to state i ,which one has been used(1 if already been used, 0 otherwise). The purpose is to reduce the searching effort.

%==============assign the first simplex-transition1=============
stg=1;
if start_st==0 start_st=SIV(1);end   %==start from siv(1)===
```

```
ini_fr=start_st;
ini_to=ST_TAB(start_st,:)';
ft_trans=[ones(n,1)*ini_fr ini_to(:)];
ft_siv_loc=find(SIV(:)==start_st);
old_trans=ft_trans;
old_siv_loc=ft_siv_loc;

%=======define permu==================================
permutation=[1 2 3 4;
    1 3 2 4;
    3 4 1 2;
    2 3 1 4;
    3 1 2 4;
    1 2 4 3;
    4 3 2 1;
    4 1 2 3;
    3 4 2 1;
    2 4 3 1;
    3 2 4 1;
    3 1 4 2;
    2 3 4 1;
    3 2 1 4;
    2 1 4 3;
    1 4 3 2;
    1 3 4 2;
    4 1 3 2;
    1 4 2 3;
    4 3 1 2;
    2 4 1 3;
    4 2 1 3;
    2 1 3 4;
    4 2 3 1];
INDX_TAB(ini_fr,ini_to)=n*(start_st-1)+permutation(permu,:);


%=====================assingn simplex-transition_2 till simplex-transition_D=====
while stg<D
      stg=stg+1;
      new_trans=get_new_trans(stg,old_trans);
      new_siv_loc=get_new_siv_loc(stg,old_siv_loc);
      mid_simp=get_mid_simp(new_siv_loc);
      assign(old_trans,new_trans,mid_simp);
```

```
    old_trans=new_trans;
    old_siv_loc=new_siv_loc;

end

%=====================assign simplex-transition_{D+1} ================
if D==1 new_trans=old_trans; end
final_simp=get_final_simp;
final_assign(new_trans,final_simp);

%==========reform INDX_TAB into sym_tab=========================
sym_tab=[];
pos=(ST_TAB-1)*TOT_ST;
add=repmat([1:TOT_ST]',1,n);
loc=pos+add;
indx=INDX_TAB(loc(:));
sym=ALL_SYMS(indx(:),:);
for i=0:n-1
    sym_tab=[sym_tab sym(i*TOT_ST+1:(i+1)*TOT_ST,:)];
end
%=========end of function==============================
%======SUBFUNCTIONS==============================

%============================================================
%obtain simplex-transition_{i+1}'s (new_trans), each containing a simplex-transition_i in old_trans
%============================================================
function new_trans=get_new_trans(stg,old_trans)
global TOT_ST ALL_PATHS ARY ;
n=ARY;
new_trans=[];
for i=1:size(old_trans,1)
  r=find(ALL_PATHS(:,stg)==old_trans(i,1));
  rr=find(ALL_PATHS(r,stg+1)==old_trans(i,2));
  int=floor((r(rr)-0.5)/n);
  temp=[ALL_PATHS([int*n+1:(int+1)*n],stg) ALL_PATHS([int*n+1:(int+1)*n],stg+1)];
  new_trans=[new_trans;temp];
end


%=========================================================
%obtain the locations of new_trans in SIV
%=========================================================
```

```
function new_siv_loc=get_new_siv_loc(stg,old_siv_loc)
global ARY;
n=ARY;
aa=floor((old_siv_loc-0.5)/n^(stg-1));
bb=mod(aa,n);
pos=bb(1);

exp_old_siv_loc=repmat(old_siv_loc(:),1,n);
%add=[0:n^(stg-1):(n-1)*n^(stg-1)];
add=[[-pos:0][1:n-1-pos]].*n^(stg-1);
exp_add=repmat(add,size(old_siv_loc(:),1),1);
new_siv_loc=exp_old_siv_loc+exp_add;

%=========================================================
%obtain type B simplexes by mapping the locations of simplex-transtions in SIV to symobls in IIS
%=========================================================
function mid_simp=get_mid_simp(siv_loc)
global ARY TOT_ST SYM_INP SIV RESHP_SYM_INP;
n=ARY;
mid_simp=[];
for i=1:size(siv_loc,1)
  base=RESHP_SYM_INP(siv_loc(i,:),:);
  mid=simplex1(base);
  mid_simp=[mid_simp;mid];
end

%=========================================================
%assing symbols to simplex-transtion2 till simplex-transtionD
%=========================================================
function assign(old_trans, new_trans, simp);
global ARY ALL_SYMS INDX_TAB L_USE;
n=ARY;
j_use=zeros(1,size(simp,1));
for i=1:size(old_trans,1)
  sel_new_trans=new_trans((i-1)*n+1:i*n,:);
  %r1=find(sel_new_trans(:,2)==old_trans(i,2));
  r2=find(sel_new_trans(:,2)~=old_trans(i,2));
  sym_indx=INDX_TAB(old_trans(i,1), old_trans(i,2));
  sym_comp=ALL_SYMS(sym_indx,:);
  j_no_use=find(j_use==0);
  sel_simp=simp(j_no_use,:);
  j_flag=0;
```

```
  for j=1:size(sel_simp,1)

     if sel_simp(j,:)==sym_comp
        j_flag=1;
        loc=floor((j-0.5)/n)*n+1:(floor((j-0.5)/n)+1)*n;
        j_use(j_no_use(loc))=1;
        loc_simp=sel_simp(loc,:);
         m_use=zeros(1,n);
        m_use(j-floor((j-0.5)/n)*n)=1;

       for k=1:size(r2,1)
         fr=sel_new_trans(r2(k),1);
        l_no_use=find(L_USE(fr,:)==0);
        for l=1:size(l_no_use,2)
           m_flag=0;
            m_no_use=find(m_use==0);
           left_simp=loc_simp(m_no_use,:);

           for m=1:size(left_simp,1)
             if ALL_SYMS((fr-1)*n+l_no_use(l),:)==left_simp(m,:)
                 INDX_TAB(fr,sel_new_trans(r2(k),2))=(fr-1)*n+l_no_use(l);
               m_flag=1;
               m_use(m_no_use(m))=1;
               L_USE(fr,l_no_use(l))=1;
              end
             if m_flag==1 break;end
            end %==end of m=======
           if m_flag==1 break;end
         end     %==end of l=======
       end       %==end of k=======
     end         %==end of if simp(j,:)==
     if j_flag==1 break;end
  end            %==end of j========
end              %==end of i===========

%=========================================================
%Generate the signal constellation
%=========================================================
function all_syms=get_all_syms();
global TOT_ST SIV RESHP_SYM_INP ARY;
n=ARY;
%RESHP_SYM_INP=reshape(SYM_INP,TOT_ST, 3);
```

```
for i=1:TOT_ST
  sr_simp=simplex2(RESHP_SYM_INP(i,:));%simplex2.m generate a source simplex realted to the input symbol
  all_syms((SIV(i)-1)*n+1:SIV(i)*n,:)=sr_simp;
end

%=========================================================
%assign symbols to simplex-transtionD+1
%=========================================================
function final_assign(st_trans,final_simp)
global TOT_ST ARY INV_STAB INDX_TAB ALL_SYMS L_USE;
n=ARY;
j_use=zeros(1,size(final_simp,1));
for i=1:TOT_ST
    sym_indx=INDX_TAB(st_trans(i,1), st_trans(i,2));
    sym_comp=ALL_SYMS(sym_indx,:);
    r=find(INV_STAB(st_trans(i,2),:)~=st_trans(i,1));
    j_no_use=find(j_use==0);
    sel_simp=final_simp(j_no_use,:);
    j_flag=0;
    for j=1:size(sel_simp,1)
        if sym_comp==sel_simp(j,:)
            j_flag=1;
            loc=floor((j-0.5)/n)*n+1:(floor((j-0.5)/n)+1)*n;
            j_use(j_no_use(loc))=1;
            loc_simp=sel_simp(loc,:);
            m_use=zeros(1,n);
            m_use(j-floor((j-0.5)/n)*n)=1;
            for k=1:size(r,2)
                fr=INV_STAB(st_trans(i,2),r(k));
                l_no_use=find(L_USE(fr,:)==0);
                for l=1:size(l_no_use,2)
                    m_flag=0;
                    m_no_use=find(m_use==0);
                    left_simp=loc_simp(m_no_use,:);
                    for m=1:size(left_simp,1)

                        if ALL_SYMS((fr-1)*n+l_no_use(l),:)==left_simp(m,:);
                        INDX_TAB(fr,st_trans(i,2))=(fr-1)*n+l_no_use(l);
                        m_flag=1;
                        m_use(m_no_use(m))=1;
                        L_USE(fr,l_no_use(l))=1;
```

```
                                end
                                if m_flag==1 break;end
                        end %==end of m
                        if m_flag==1 break;end
                    end     %==end of l
                end         %==end of k
            end             %==end of if sym_comp
            if j_flag==1 break; end
        end                 %==end of j
end                         %==end of i



%=========================================================
%Obtain all the type B simplexes that should be assinged to all the simplex-transition_{D+1}'s
%=========================================================
function final_simp=get_final_simp
global TOT_ST RESHP_SYM_INP  ARY;
n=ARY;
final_simp=[];
for i=1:TOT_ST/n
    temp=simplex1(RESHP_SYM_INP((i-1)*n+1:i*n,:));%simplex1.m generate four type B simplexes formed by the four source simpelxes related to the four symbols of a type B simplex
    final_simp=[final_simp;temp];
end
%=================================================
```

APPENDIX C STATE TRANSITION MATRIX GENERATING CODE

```
function siv=siv_gen(n,D,siv_free)

%Purpose: Generate State Transition Matrix(STM). This function only generate the left
side (originating states) of butterflies in STM, the right side of butterflies in STM is
considered when doing the symbol assignment--sym_asgn.m.

%Input Parameters
%n: n-ary information symbol
%D: Trellis Depth
%siv_free: a vector containing D+1 elements. They are STM_FREE(first butterly, left side butterfly, 1st
dimension, 2nd dimenstion, ...., (D-1)th dimenstion).

%In Function
%FLY: All butterflies in this trellis. FLY is a FLY_NO*4 x 2 matrix.
%L_FLY: Left side of all the butterflies.
%R_FLY: Right side of all the butterflies.
%MID_SIMP: The originating state sets of simplex-transtion3 to simplex-transtionD+1.It is a n
x(FLY_NO*(D-2)) matrix. Columns MID_SIMP(:,(i-1)*FLY_NO+1:i*FLY_NO), i=1,2,...D-1, are the originating
state sets of all the simplex-transitioni+2's. The originating state sets of simplex-transition2's are the right
sides of all the butterflies, recorded in R_FLY.
%L_USE, R_USE, MID_USE: Loop variables used to record which L_FLY, R_FLY, MID_SIMP has been
placed into SIV already. The purpose is to reduce searching efforts.
%FREE: The particular ways to place the butterflies along each of the D-1 dimensions of SIV. FREE(i,:),
i=1,2,...,D-1, defines STM_FREE(ith dimension).
%FREE_FT_L_FLY: Defines STM_FREE(left side butterfly).
%free_ft_sel_l_fly:Defines STM_FREE(first butterfly).

%Output Parameters
%siv: a D-1 dimensional matrix, unit element is the left side of a butterfly.
```

```
%Date: May 13,24,1999
%Author: Xiangyu Song


global L_USE R_USE;
global MID_USE;
global FREE;
global FREE_FT_L_FLY;
global L_FLY R_FLY;
global ARY;
global MID_SIM;
global FLY_NO;%=MID_NO

ARY=n;
%=====define freedom for first R-FLY and MID_SIMP=====
free_mat=[2 3 4;
2 4 3;
3 2 4;
3 4 2;
4 2 3;
4 3 2];
free_ft_l_fly_mat=[1 2 3 4;
    1 2 4 3;
    1 3 2 4;
    1 3 4 2;
    1 4 2 3;
    1 4 3 2;
    2 1 3 4;
    2 1 4 3;
    2 3 1 4;
    2 3 4 1;
    2 4 1 3;
    2 4 3 1;
    3 1 2 4;
    3 1 4 2;
    3 2 1 4;
    3 2 4 1;
    3 4 1 2;
    3 4 2 1;
    4 1 2 3;
    4 1 3 2;
    4 2 1 3;
```

```
    4 2 3 1;
    4 3 1 2;
    4 3 2 1;];
if (isempty(siv_free)==1)
      FREE=repmat([2 3 4],D-1 ,1);
      free_ft_sel_l_fly=1;
      FREE_FT_L_FLY=free_ft_l_fly_mat(1,:);
else
      FREE=free_mat(siv_free(3:size(siv_free,2)),:);
      free_ft_sel_l_fly=siv_free(1);
      FREE_FT_L_FLY=free_ft_l_fly_mat(siv_free(2),:);
end

TOT_ST=n^D;
FLY_NO=TOT_ST/n;%Total number of butterflies
L_USE=zeros(1,FLY_NO);
R_USE=zeros(1,FLY_NO);
FLY=butterfly(n,D);%Generate all butterflies.
FLY=reshape(FLY,n,FLY_NO*2);%First FLY_NO columns are the left side of all butterflies, second FLY_NO
columns are the right side of all the butterflies.
L_FLY=FLY(:,1:FLY_NO);
R_FLY=FLY(:,FLY_NO+1:2*FLY_NO);
MID_SIM=mid_simp(n,D);Generate MID_SIM.
MID_USE=zeros(D-2,FLY_NO);


anchor_ini=L_FLY(1,free_ft_sel_l_fly);%==This MayNOt be the siv(1),since the First L_FLY has
FREE_FT_L_FLY can put it to any place among siv(1:4).

%=================D=1 4 STATES=======================
if D==1
      tab(FREE_FT_L_FLY)=L_FLY(:,free_ft_sel_l_fly);
      siv=tab';break;
end

%===============D=2 16 STATES=======================
tab=build_ft_dim(anchor_ini,0,[]);
if D==2 siv=tab;break;end


anchor=tab(1);
```

```
%===============D=3 64 STATES=========================
tot_ck_mat=build_tot_ck_mat(tab,2);
tab(:,1)=tab;
index=find(tot_ck_mat(1,:)~=anchor);
tab(1,FREE(2,:))=tot_ck_mat(1,index);%DIM 2 using FREE(2,:)

for sd_dim=2:ARY
      tab(:,sd_dim)=build_ft_dim(tab(1,sd_dim),1,tot_ck_mat);
end
if D==3  siv=tab;break;end


%================D=4 256 STATES=======================
tot_ck_mat=build_tot_ck_mat(tab,3);
tab(:,:,1)=tab;
index=find(tot_ck_mat(1,1,:)~=anchor);
tab(1,1,FREE(3,:))=tot_ck_mat(1,1,index); %===FREE(3,:)=====
for td_dim=2:ARY
      for sd_dim=1:ARY
            ck_mat=tot_ck_mat(:,sd_dim,:);
            ck_mat=squeeze(ck_mat);
            tab(:,sd_dim,td_dim)=build_ft_dim(tab(1,sd_dim,td_dim),1,ck_mat);
            if sd_dim==1
        ext_ck_mat=tot_ck_mat(1,[2:ARY],:);
        ext_ck_mat=squeeze(ext_ck_mat);
                  tab(1,[2:ARY],td_dim)=extend(tab(1,1,td_dim),2,ext_ck_mat);%===[2 3 4] is not flexible
            end
      end
end
if D==4 siv=tab;break;end

%================D=5 1024 STATES======================
tot_ck_mat=build_tot_ck_mat(tab,4);
tab(:,:,:,1)=tab;
index=find(tot_ck_mat(1,1,1,:)~=anchor);
tab(1,1,1,FREE(4,:))=tot_ck_mat(1,1,1,index);%===FREE(4,:)=======
for four_dim=2:ARY
      for td_dim=1:ARY
            for sd_dim=1:ARY
                  ck_mat=tot_ck_mat(:,sd_dim,td_dim,:);
                  ck_mat=squeeze(ck_mat);
                  tab(:,sd_dim,td_dim,four_dim)=build_ft_dim(tab(1,sd_dim,td_dim,four_dim),1,ck_mat);
```

```
                if sd_dim==1
                    ext_ck_mat=tot_ck_mat(1,[2:ARY],td_dim,:);
                    ext_ck_mat=squeeze(ext_ck_mat);% TO (3,4)
                    tab(1,[2:ARY],td_dim,four_dim)=extend(tab(1,1,td_dim,four_dim),2,ext_ck_mat);%===[2  3
4] is not flexible
                end
            end
            if td_dim==1
                ext_ck_mat=tot_ck_mat(1,1,[2:ARY],:);
                ext_ck_mat=squeeze(ext_ck_mat);
                tab(1,1,[2:ARY],four_dim)=extend(tab(1,1,1,four_dim),3,ext_ck_mat);
            end
        end
    end
if D==5 siv=tab;break;end


%================D=6 4096 STATES======================
tot_ck_mat=build_tot_ck_mat(tab,5);
tab(:,:,:,:,1)=tab;
index=find(tot_ck_mat(1,1,1,1,:)~=anchor);
tab(1,1,1,1,FREE(5,:))=tot_ck_mat(1,1,1,1,index);%==FREE(5,:)======
for five_dim=2:ARY
    for four_dim=1:ARY
        for td_dim=1:ARY
            for sd_dim=1:ARY
                ck_mat=tot_ck_mat(:,sd_dim,td_dim,four_dim,:);
                ck_mat=squeeze(ck_mat);

    tab(:,sd_dim,td_dim,four_dim,five_dim)=build_ft_dim(tab(1,sd_dim,td_dim,four_dim,five_dim),1,ck_mat)
;
                if sd_dim==1
                    ext_ck_mat=tot_ck_mat(1,[2:ARY],td_dim,four_dim,:);
                    ext_ck_mat=squeeze(ext_ck_mat);

    tab(1,[2:ARY],td_dim,four_dim,five_dim)=extend(tab(1,1,td_dim,four_dim,five_dim),2,ext_ck_mat);%==
=[2 3 4] is not flexible
                end
            end
            if td_dim==1
                ext_ck_mat=tot_ck_mat(1,1,[2:ARY],four_dim,:);
                ext_ck_mat=squeeze(ext_ck_mat);
```

```
                    tab(1,1,[2:ARY],four_dim,five_dim)=extend(tab(1,1,1,four_dim,five_dim),3,ext_ck_mat);

                end
            end
            if four_dim==1
                ext_ck_mat=tot_ck_mat(1,1,1,[2:ARY],:);
                ext_ck_mat=squeeze(ext_ck_mat);
                tab(1,1,1,[2:ARY],five_dim)=extend(tab(1,1,1,1,five_dim),4,ext_ck_mat);
            end
        end
    end
    if D==6 siv=tab;break;end

    %=========================================================


    %=================SUBFUNCTIONS=============================
    %=========================================================
    %Place the proper left side of butterfies along the first dimension
    %=========================================================
    function temp=build_ft_dim(start,ck_flag,ck_mat)
    %ck_mat is in a 16X4 matrix

    global ARY;
    global L_USE R_USE ;
    global FREE;
    global L_FLY R_FLY;
    global FREE_FT_L_FLY;
    temp=zeros(ARY*ARY,1);

    %===================PUT THE FIRST L_FLY==
    no_use=find(L_USE==0);
    no_use_l_fly=L_FLY(:,no_use);
    [row col]=find(no_use_l_fly==start);
    sel_l_fly=no_use_l_fly(:,col);
    L_USE(no_use(col))=1;
    index=find(sel_l_fly~=start);
    if ck_flag==0%===The very First L_FLY in the whole siv    temp(FREE_FT_L_FLY)=sel_l_fly;
    else
        temp(1)=start;
        sel_ck_mat=ck_mat(2:ARY,:);
        for i=1:size(index,1)
```

```
            [row col]=find(sel_ck_mat==sel_l_fly(index(i)));
            temp(row+1)=sel_l_fly(index(i));
        end
    end
    %==================PUT THE FIRST R_FLY========
    no_use=find(R_USE==0);
    no_use_r_fly=R_FLY(:,no_use);
    [row col]=find(no_use_r_fly==temp(1));
    sel_r_fly=no_use_r_fly(:,col);
    R_USE(no_use(col))=1;
    index=find(sel_r_fly~=temp(1));
    if ck_flag==0
        temp((FREE(1,:)-1)*ARY+1)=sel_r_fly(index);%===R_FLY using FREE(1,:)==
    else
        sel_ck_mat=ck_mat([1:ARY-1]*ARY+1,:);
        for i=1:size(index,1)
            [row col]=find(sel_ck_mat==sel_r_fly(index(i)));
            temp(row*ARY+1)=sel_r_fly(index(i));
        end
    end
    %==================PUT THE OTHER THREE L_FLY==
    if ck_flag==0 %===USING R_FLY CHECK========
    r_fly_ck=[];
    for i=2:ARY
        no_use=find(R_USE==0);
        no_use_r_fly=R_FLY(:,no_use);
        [row col]=find(no_use_r_fly==temp(i));
        R_USE(no_use(col))=1;
        r_fly_ck=[r_fly_ck no_use_r_fly(:,col)];
    end
    for l_num=1:ARY-1
        no_use=find(L_USE==0);
        no_use_l_fly=L_FLY(:,no_use);
        [row col]=find(no_use_l_fly==temp(l_num*ARY+1));
        sel_l_fly=no_use_l_fly(:,col);
        L_USE(no_use(col))=1;
        index=find(sel_l_fly~=temp(l_num*ARY+1));
        for ele_no=1:size(index,1)
            [row col]=find(r_fly_ck==sel_l_fly(index(ele_no)));
            temp(l_num*ARY+1+col)=sel_l_fly(index(ele_no));
        end
    end
```

```
else %===============USING CK_MAT=========================
for l_num=1:ARY-1
    no_use=find(L_USE==0);
    no_use_l_fly=L_FLY(:,no_use);
    [row col]=find(no_use_l_fly==temp(l_num*ARY+1));
    sel_l_fly=no_use_l_fly(:,col);
    L_USE(no_use(col))=1;
    index=find(sel_l_fly~=temp(l_num*ARY+1));
    sel_ck_mat=ck_mat(l_num*ARY+[2:ARY],:);
    for ele_no=1:size(index,1)
        [row col]=find(sel_ck_mat==sel_l_fly(index(ele_no)));
        temp(l_num*ARY+1+row)=sel_l_fly(index(ele_no));
    end
end
end %============END of IF ELSE============================


%=========================================================
%Place the first three originating states in the three butterflies along dim dimension of SIV
%=========================================================
function temp=extend(from, dim,ext_ck_mat)%ext_ck_mat in (3,4)
global ARY;
global MID_SIM;
global MID_USE;
global FLY_NO;

    sel_mid_sim=MID_SIM(:,(dim-2)*FLY_NO+1:(dim-1)*FLY_NO);
    temp=zeros(1,ARY-1);
    no_use=find(MID_USE(dim-1,:)==0);
    no_use_mid=sel_mid_sim(:,no_use);
    [row col]=find(no_use_mid==from);
    ext_mid=no_use_mid(:,col);
    MID_USE(dim-1,no_use(col))=1;
    index=find(no_use_mid(:,col)~=from);
    for ele_no=1:size(index,1)
        [row col]=find(ext_ck_mat==ext_mid(index(ele_no)));
        temp(row)=ext_mid(index(ele_no));
    end


%=========================================================
```

```
%Find all the simplex-transitions containg transitions in tab_asgned along the dim dimension of SIV.
%============================================================
function temp=build_tot_ck_mat(tab_asgned , dim)
global MID_SIM;
global MID_USE;
global FLY_NO;
global ARY;
sel_mid_sim=MID_SIM(:,(dim-2)*FLY_NO+1:(dim-1)*FLY_NO);

temp=[];
for i=1:ARY^dim
      no_use=find(MID_USE(dim-1,:)==0);
      no_use_mid=sel_mid_sim(:,no_use);
      [row col]=find(no_use_mid==tab_asgned(i));
      temp=[temp ;no_use_mid(:,col)'];
      MID_USE(dim-1,no_use(col))=1;
end
shape=[size(tab_asgned) ARY];
temp=reshape(temp,shape);
temp=squeeze(temp);
%============================================================
```

APPENDIX D INITIAL INPUT SIMPLEX GENERATING CODE

```
function inp_sym=inp_gen(n,D);

%Purpose: This function generate the Initial Input Simplex(IIS).Using 6*2^(D-1) dimensions.

%Input Parameters
%n: n-ary information symbol
%D: Trellis Depth

%In Function
%pair: “Copy” rule complied by a type B simplex
%fil_pat: “fill” pattern (new dimension needed) for the first simplexes built along the first dimension
%fil: “fill” pattern for later simplexes built along the first dimension.

%Output Parameters
inp_sym: Regard a 3-out-of-N symbol as a unit element. inp_sym is a TOT_STX(4 symbls) matrix.

%When D increase one, just create a new dimension, copy the previous program and add one more loop.
%n=4.


ini(:,1)=[1 -1 2 -2]';
ini(:,2)=[3 4  -3 -4]';
ini(:,3)=[5 6  -6 -5]';
dim=6;
tab=ini;
pair(:,1)=[1 2 3 4]';
pair(:,2)=[1 3 2 4]';
pair(:,3)=[1 4 2 3]';

%=======D=1 4 STATES=======================================
if D==1 inp_sym=tab;break;end
```

```
%=======D=2 16 STATES=======================================
%=======fill patern=======================
fil_pat(:,1)=[1 1 2 2]';
fil_pat(:,2)=[1 2 1 2]';
fil_pat(:,3)=[1 2 2 1]';
%======match to pair(2,i)==================
for i=1:3
      tab((pair(2,i)-1)*n+1:pair(2,i)*n,i)=(-1).*ini(:,i);
end;
%==========fill pair(3,i) and match to pair(4,i)=======
for i=1:3
tab((pair(3,i)-1)*n+1:pair(3,i)*n,i)=(fil_pat(:,i)+dim).*sign(ini(:,i));

tab((pair(4,i)-1)*n+1:pair(4,i)*n,i)=(-1).*tab((pair(3,i)-1)*n+1:pair(3,i)*n,i);
dim=dim+2;
end;

if D==2 inp_sym=tab;break;end

%====later fill one clolum by one colum=============================
fil_sign=sign(tab);
fil(:,1)=[1 1 2 2 1 1 2 2 3 3 4 4 3 3 4 4]';
fil(:,2)=[1 2 1 2 3 4 3 4 1 2 1 2 3 4 3 4]';
fil(:,3)=[1 2 2 1 3 4 4 3 3 4 4 3 1 2 2 1]';
step=3;

%===========D=3 64 state=====================================
%======match to pair(2,i)==================
for i=1:3
  tab(:,(pair(2,i)-1)*step+i)=(-1).*tab(:,i);
end;
%==========fill pair(3,i) and match to pair(4,i)====
for i=1:3
tab(:,(pair(3,i)-1)*step+i)=(fil(:,i)+dim).*fil_sign(:,i);
tab(:,(pair(4,i)-1)*step+i)=(-1).*tab(:,(pair(3,i)-1)*step+i);
dim=dim+4;
end;

if D==3 inp_sym=tab;break;end
```

```
%=======D=4 256 STATES=====================================
%==========match to pair(2,i)====================
for i=1:3
tab(:,[0:step:(n-1)*step]+i,pair(2,i))=(-1).*tab(:,[0:step:(n-1)*step]+i);
end

%=======fill pair(3,i) and match to pair(4,i)============
for i=1:3 % 3 col
%===COMPLETELY FILL A TWO-DIM PLANE OF COL I THIRD DIM=pair(3,i)
for j=1:2:3
tab(:,(pair(j,i)-1)*step+i,pair(3,i))=(fil(:,i)+dim).*fil_sign(:,i);
tab(:,(pair(j+1,i)-1)*step+i,pair(3,i))=(-1).*tab(:,(pair(j,i)-1)*step+i,pair(3,i));
dim=dim+4;
end ;
%======match to third dim=pair(4,i)=============
tab(:,[0:step:(n-1)*step]+i,pair(4,i))=(-1).*tab(:,[0:step:(n-1)*step]+i,pair(3,i));
end
if D==4 inp_sym=tab;break;end



%================D=5 1024================================
%============MATCH TO A NEW DIM(FOURTH)=PAIR(2,I)
tab(:,:,:,1)=tab;
for i=1:3
tab(:,[0:step:(n-1)*step]+i,:,pair(2,i))=(-1).*tab(:,[0:step:(n-1)*step]+i,:,1);
end
%========fill and match==============
for i=1:3 %COL OF I IN 3-OUT-OF-DIM========

%==COMPLETE FILL THE ITH COL IN FOURTH DIM=PAIR(3,I),IT IS  A THREE-DIM CUBE,ALL
OPERATION DONE IN THE SAME FOURTH DIM=PAIR(3,I)=========
    for td_dim=1:2:3

%==COMPLETE FILL THE ITH COL IN THIRD DIM=PAIR(TD_DIM,I),IT IS A TWO-DIM PLANE OF THE
FORMAT OF THE INPUT FOR 64 STATES=========================
        for sd_dim=1:2:3
            tab(:,(pair(sd_dim,i)-1)*step+i,pair(td_dim,i),pair(3,i))=(fil(:,i)+dim).*fil_sign(:,i);
            tab(:,(pair(sd_dim+1,i)-1)*step+i,pair(td_dim,i),pair(3,i))=(-1).*tab(:,(pair(sd_dim,i)-
1)*step+i,pair(td_dim,i),pair(3,i));
dim=dim+4;
        end
```

```
%==MATCH IN THE THIRD DIM===============
        tab(:,[0:step:(n-1)*step]+i,pair(td_dim+1,i),pair(3,i))=(-1).*tab(:,[0:step:(n-
1)*step]+i,pair(td_dim,i),pair(3,i));
    end
%===match to fouth dim=pair(3,i)=================
    tab(:,[0:step:(n-1)*step]+i,:,pair(4,i))=(-1).*tab(:,[0:step:(n-1)*step]+i,:,pair(3,i));
end
if D==5 inp_sym=tab;break;end



%============D=6 4096 STATES================================
%===========MATCH TO A NEW DIM(FIFTH)=========
tab(:,:,:,:,1)=tab;
for i=1:3
tab(:,[0:step:(n-1)*step]+i,:,:,pair(2,i))=(-1).*tab(:,[0:step:(n-1)*step]+i,:,:,1);
end
%===FILL & MATCH IN FIFTH DIM=PAIR(3,I)==
for i=1:3
    for four_dim=1:2:3
        for td_dim=1:2:3
            for sd_dim=1:2:3
                tab(:,(pair(sd_dim,i)-
1)*step+i,pair(td_dim,i),pair(four_dim,i),pair(3,i))=(fil(:,i)+dim).*fil_sign(:,i);
                tab(:,(pair(sd_dim+1,i)-1)*step+i,pair(td_dim,i),pair(four_dim,i),pair(3,i))=(-
1).*tab(:,(pair(sd_dim,i)-1)*step+i,pair(td_dim,i),pair(four_dim,i),pair(3,i));
dim=dim+4;
            end
            tab(:,[0:step:(n-1)*step]+i,pair(td_dim+1,i),pair(four_dim,i),pair(3,i))=(-1).*tab(:,[0:step:(n-
1)*step]+i,pair(td_dim,i),pair(four_dim,i),pair(3,i));
        end
        tab(:,[0:step:(n-1)*step]+i,:,pair(four_dim+1,i),pair(3,i))=(-1).*tab(:,[0:step:(n-
1)*step]+i,:,pair(four_dim,i),pair(3,i));
    end
%===MATCH TO FIFTH DIM=PAIR(4,I)========
    tab(:,[0:step:(n-1)*step]+i,:,:,pair(4,i))=(-1).*tab(:,[0:step:(n-1)*step]+i,:,:,pair(3,i));
end
if D==6 inp_sym=tab;break;end
```

APPENDIX E MINIMUM DISTANCE CALCULATION CODE (1)

```
function [min_dis, min_path]=min_dis_samest(n,D,DIM,B,start_st)

%Purpose: This function seek the minimum distance among paths with the same start_st.

%Input Parameters
%n: n-ary information symbol
%D: Trellis Depth
%DIM: Dimensions of the signal constellation
%start_st: Starting states for all the paths
%B: Information sequence length
%stab: State transition table for this trellis
%sym_tab: symbol assignment table, generated by sym_asgn.m

%In Function
%TOT_ST: Total number of states in this trellis.
%full_txs: Turn sym_tab in TOT_ST x (n*3) matrix into (S*n) x DIM matrix.The TOT_ST rows--full_txs((i-1)*TOT_ST+1:i*TOT_ST, :) are the symbols assinged to the TOT_ST transissons when input is i-1. i=1,2,...,n. Symbols are represented as 0 or (+)(-)1 in all the DIM dimensions.
%dis_old(new): TOT_ST x TOT_ST matrix. dis_old(new)(i,j) is the minimum distance among all the paths reaching state i and j at previous(current) stage. i,j=1,2,...TOT_ST.
%path_old(new):TOT_ST x TOT_ST x (2*B) matrix. path_old(new)(i,j) is the path history of the pair of paths reaching state i and j and having dis_old(new)(i,j).

%Output Parameters
%min_dis: The minimum distance among paths with the same start_st.
%min_path: Paths history of the pair of paths having distance min_dis between them.

%Date: May 16, 1999   Xiangyu Song


global TOT_ST ARY full_txs INV_ST_TAB TXS;
ARY=n;
TOT_ST=n^D;
```

```
ST_TAB=stab;
TXS=sym_tab;
INV_STAB=inv_tab(n,D);%Generate inverse state transition table
full_txs=fu_sym(n,D,DIM);%Generate full_txs

%==============Initialization=========================================
dis_old=ones(TOT_ST)*inf;
dis_old(start_st,start_st)=0;

path_old=(-1)*ones(TOT_ST, TOT_ST, 2*(B+1));
path_new=path_old;

reach=start_st;
inv_reach=zeros(D-1,n^(D-1));
fr_reach=zeros(D-1, n^(D-1));

temp=start_st;
for i=1:D-1
    temp=INV_ST_TAB(temp(:),:);
    temp=temp(:)';
    inv_reach(i,[1:size(temp,2)])=temp;
end

temp=start_st;
for i=1:D-1
    temp=ST_TAB(temp(:), :);
    temp=temp(:)';
    fr_reach(i,[1:size(temp,2)])=temp;
end
%====get the common part for updating all state pair with p~q;==========
st_p_all=[1:TOT_ST];st_q_all=[1:TOT_ST];
[pair_p_all_dif,pair_q_all_dif,reach_p_all_dif,reach_q_all_dif]=pair_st_dif(st_p_all,st_q_all);
add_dis_all_dif=get_add_dis(reach_p_all_dif,reach_q_all_dif,16);
[pair_p_all_same,pair_q_all_same,reach_p_all_same,reach_q_all_same]=pair_st_same(st_p_all,st_q_all);
add_dis_all_same=get_add_dis(reach_p_all_same,reach_q_all_same,16);
end

%==========travel along the trellis============================
for stg=1:B-D
%deal with pair-p-q with p~=q;=================
    if stg>=B-D+1
        st_p=inv_reach(B-stg,[1:n^(B-stg)]);
```

```
        st_q=inv_reach(B-stg,[1:n^(B-stg)]);
        [pair_p,pair_q,reach_p,reach_q]=pair_st_dif(st_p,st_q);
        add_dis=get_add_dis(reach_p,reach_q,16);

    elseif stg<D %consider reach states, others are INF========
        reach=ST_TAB(reach,:);
        st_p=reach(:)';st_q=reach(:)';
        %==not_reach states INF , used by later stage=====
        all_st=[1:TOT_ST];
        all_st(reach)=0;
        [not_reach_pos]=find(all_st~=0);
        dis_new(not_reach_pos,not_reach_pos)=inf;%========
        dis_new(reach,not_reach_pos)=inf;
        dis_new(not_reach_pos,reach)=inf;
        %====state pair with p~=q
        [pair_p,pair_q,reach_p,reach_q]=pair_st_dif(st_p,st_q);
        add_dis=get_add_dis(reach_p,reach_q,16);

    else %consider all states================
        st_p=[1:TOT_ST];st_q=[1:TOT_ST];
        add_dis=add_dis_all_dif;
        pair_p=pair_p_all_dif;pair_q=pair_q_all_dif;
        reach_p=reach_p_all_dif;reach_q=reach_q_all_dif;
    end
    dis_mat=dis_old(reach_p+(reach_q-1)*TOT_ST)'+add_dis;
    dis_mat=reshape(dis_mat,n*n,size(reach_p,2)/(n*n));
    [min_pair_dis, min_pos]=min(dis_mat,[],1);
    dis_new(pair_p+(pair_q-1)*TOT_ST)=min_pair_dis;
    dis_new(pair_q+(pair_p-1)*TOT_ST)=dis_new(pair_p+(pair_q-1)*TOT_ST);%symmetry
    %========Update symbols in the paths reaching state p and q
    path_update1(stg,pair_p, pair_q, reach_p, reach_q, min_pos,16);

  %deal with pair-p-q with p=q;==============
    if stg<B-D
        dis_new(st_p+(st_q-1)*TOT_ST)=0;
        %=======Update paths reaching state p and q
         path_update2(stg,st_p, st_q,fr_reach, start_st);

    else %B-D, min(12)
        if stg>=B-D+1
            [pair_p,pair_q,reach_p,reach_q]=pair_st_same(st_p,st_q);
        elseif stg<D
```

```
            [pair_p,pair_q,reach_p,reach_q]=pair_st_same(st_p,st_q);
            %add_dis=add_dis(reach_p,reach_q,12);
        else %all state pair with p==q
            pair_p=pair_p_all_same;pair_q=pair_q_all_same;
            reach_p=reach_p_all_same;reach_q=reach_q_all_same;

        end
        [non_same_pos]=find((reach_p-reach_q)~=0);
        reach_p=reach_p(non_same_pos);
        reach_q=reach_q(non_same_pos);
        add_dis=get_add_dis(reach_p,reach_q,12);  %12===
        dis_mat=dis_old(reach_p+(reach_q-1)*TOT_ST)'+add_dis;
        dis_mat=reshape(dis_mat,n*(n-1),size(reach_p,2)/(n*(n-1)));%12=
        [min_pair_dis,min_pos]=min(dis_mat,[],1);
        dis_new(pair_p+(pair_q-1)*TOT_ST)=min_pair_dis;
        dis_new(pair_q+(pair_p-1)*TOT_ST)=dis_new(pair_p+(pair_q-1)*TOT_ST);%symmetry
        %========Update symbols in the paths reaching state p and q
        path_update1(stg,pair_p, pair_q, reach_p, reach_q, min_pos,12);

    end
    dis_old=dis_new;
    path_old=path_new;
end;

for stg=B-D+1:B
    if stg<B
        st_p=inv_reach(B-stg,[1:n^(B-stg)]);
        st_q=inv_reach(B-stg,[1:n^(B-stg)]);
    else
        st_p=start_st;
        st_q=start_st;
    end
    %====p~=q==================
    if stg~=B %===stg==B,only p==q=====
    [pair_p,pair_q,reach_p,reach_q]=pair_st_dif(st_p,st_q);
    add_dis=get_add_dis(reach_p,reach_q,16);
    dis_mat=dis_old(reach_p+(reach_q-1)*TOT_ST)'+add_dis;
    dis_mat=reshape(dis_mat,n*n,size(reach_p,2)/(n*n));%===16
    [min_pair_dis,min_pos]=min(dis_mat,[],1);
    dis_new(pair_p+(pair_q-1)*TOT_ST)=min_pair_dis;
    dis_new(pair_q+(pair_p-1)*TOT_ST)=dis_new(pair_p+(pair_q-1)*TOT_ST);%symetrity
    path_update1(stg,pair_p, pair_q, reach_p, reach_q, min_pos,16);
```

```
    end
    %=====p==q==================
    [pair_p,pair_q,reach_p,reach_q]=pair_st_same(st_p,st_q);
    add_dis=get_add_dis(reach_p,reach_q,16);
    dis_mat=dis_old(reach_p+(reach_q-1)*TOT_ST)'+add_dis;
    dis_mat=reshape(dis_mat,n*n,size(reach_p,2)/(n*n));%===16
    [min_pair_dis,min_pos]=min(dis_mat,[],1);
    dis_new(pair_p+(pair_q-1)*TOT_ST)=min_pair_dis;
    dis_new(pair_q+(pair_p-1)*TOT_ST)=dis_new(pair_p+(pair_q-1)*TOT_ST);%symetrity
    path_update1(stg,pair_p, pair_q, reach_p, reach_q, min_pos,16);
    dis_old=dis_new;
    path_old=path_new;
end

min_dis=dis_old(start_st,start_st);
sel_path=squeeze(path_old(start_st, start_st,:))';
min_path=[sel_path(1:2:2*B-1);sel_path(2:2:2*B)];


%======SUBFUNCTIONS==================================
%=====================================================
%Update path history
%=====================================================
function path_update1(stg,pair_p, pair_q, reach_p, reach_q, min_pos,item_no)
global ARY ST_TAB path_old path_new;
n=ARY;
for i=1:size(pair_p,2)
    pre_p=reach_p(item_no*(i-1)+min_pos(i));
    pre_q=reach_q(item_no*(i-1)+min_pos(i));
    pos_p=find(ST_TAB(pre_p,:)==pair_p(i));
    sym_p=pos_p-1;
    pos_q=find(ST_TAB(pre_q,:)==pair_q(i));
    sym_q=pos_q-1;
    if stg==1
        a=[ sym_p sym_q];
    else
        a=[squeeze(path_old(pre_p, pre_q,[1:(stg-1)*2]))' sym_p  sym_q];
    end
    path_new(pair_p(i), pair_q(i), [1:2*stg])=a;
end;
```

```
%=======================================================
%Update path history for stg<B-D and p==q
%=======================================================
function path_update2(stg,st_p, st_q,fr_reach, start_st)
global ARY TOT_ST Depth INV_ST_TAB ST_TAB path_old path_new;
n=ARY;
D=Depth;

if stg==1
    pre_p=start_st;
elseif stg>D
    pre_p=[1:TOT_ST];
else
    pre_p=fr_reach(stg-1, [1:n^(stg-1)]);
end

for i=1:size(st_p,2)
    from=intersect(pre_p,INV_ST_TAB(st_p(i),:));
    pos=find(ST_TAB(from(1),:)==st_p(i));
    sym_p=pos-1;
    if stg==1
        a=[sym_p sym_p];
    else
        a=[squeeze_old(path(from(1),from(1),[1:2*(stg-1)]))' sym_p sym_p];
    end
    path_new(st_p(i),st_q(i),[1:2*stg])=a;
end

%=======================================================
%Obtain the path metric of  from previous stage to current stage,for the paths reaching state pair(p,q)
%=======================================================
function add_dis=get_add_dis(reach_p,reach_q,itm_no)
global TOT_ST ARY full_txs;
n=ARY;

add_dis=[];
adder=[0:TOT_ST:TOT_ST*(n-1)];
adder_p=repmat(adder,itm_no/n,1);
adder_p=repmat(adder_p(:)',1,size(reach_p,2)/(itm_no));
pos_p=adder_p+reach_p;

adder_q=repmat(adder,1,n);
```

```
if itm_no==12
lost_pos=[1:n]+[0:n:(n-1)*n];
adder_q(lost_pos)=-1;
keep_pos=find(adder_q~=-1);
adder_q=adder_q(keep_pos);
end
adder_q=repmat(adder_q,1,size(reach_q,2)/(itm_no));
pos_q=adder_q+reach_q;

%==== This is deal with "out of Memory " when 256 states====
size_64=64*63/2*itm_no;
no_64=floor(size(pos_p,2)/size_64);
for count=1:no_64
reach_p_sym=full_txs(pos_p((count-1)*size_64+1:count*size_64),:);
reach_q_sym=full_txs(pos_q((count-1)*size_64+1:count*size_64),:);

add_dis_part=sum((reach_p_sym-reach_q_sym).^2,2);
add_dis=[add_dis; add_dis_part];
end;

reach_p_sym=full_txs(pos_p(no_64*size_64+1:size(pos_p,2)),:);
reach_q_sym=full_txs(pos_q(no_64*size_64+1:size(pos_q,2)),:);
add_dis_part=sum((reach_p_sym-reach_q_sym).^2,2);
add_dis=[add_dis; add_dis_part];


%=========================================================
%Generate state pair(p,q) and their precessors with p~=q, p blongs to st_p, q belongs to st_q
%=========================================================
function [pair_p, pair_q ,reach_p, reach_q]=pair_st_dif(st_p,st_q)
        global ARY INV_ST_TAB;
        n=ARY;

        pair_p=[];pair_q=[];reach_p=[];reach_q=[];
        for i=1:size(st_p,2)-1
            pair_p=[pair_p st_p(i)*ones(1,size(st_q,2)-i)];
            pair_q=[pair_q st_q((i+1):size(st_q,2))];
        end
        reach_p=INV_ST_TAB(pair_p,:);
        reach_p=reach_p';
        reach_p=repmat(reach_p(:),1,n);%n=4
        reach_p=reach_p';
```

```
        reach_p=reach_p(:)';%row vector
        reach_q=INV_ST_TAB(pair_q,:);
        reach_q=repmat(reach_q,1,n);
        reach_q=reach_q';
        reach_q=reach_q(:)';%row vector,16*size(pair_q,2)

%=========================================================
%Generate state pair(p,q) and their precessors with p==q, p blongs to st_p, q belongs to st_q
%=========================================================
function [pair_p, pair_q ,reach_p, reach_q]=pair_st_same(st_p,st_q)
        global ARY INV_ST_TAB;
        n=ARY;
        pair_p=st_p;pair_q=st_q;reach_p=[];reach_q=[];
        reach_p=INV_ST_TAB(pair_p,:);
        reach_p=reach_p';
        reach_p=repmat(reach_p(:),1,n);%n=4
        reach_p=reach_p';
        reach_p=reach_p(:)';%row vector
        reach_q=INV_ST_TAB(pair_q,:);
        reach_q=repmat(reach_q,1,n);
        reach_q=reach_q';
        reach_q=reach_q(:)';%row vector,16*size(pair_q,2)
%=========================================================
```

APPENDIX F MINIMUM DISTANCE CALCULATION CODE (2)

```
function [min_dis, ml_sym]=vit(n,D,DIM,B_Size,ref_path,stab,sym_tab);

%Purpose: This function seek the minimum distance from a reference path to all the
paths that have different starting states as the starting state of the reference path.The
method is to run viterbi algorithm on the trellis having different starting states as the
starting state of the reference path.

%n: n-ary information symbol
%D: Trellis Depth
%DIM: Dimensions of the signal constellation
%ref_path: The reference path
%B_Size: Information sequence length
%stab: State transition table for this trellis
%sym_tab: Symbol assignment table, generated by sym_asgn.m

%In Function
%TOT_ST: Total number of states in this trellis.
%st_metric: A vector containing TOT_ST element, st_metric(i) is the path metric of state i. At a certain
stage, it is the minmum distance between the paths reaching state i and the reference path.
%st_sym: Path history. A B_SizeXTOT_ST matrix. st_sym(:,i), i=1,2,...,TOT_ST,is the input symbles on the
path reaching state i and having distance st_metric(i) from the reference path.

%Output Parameters
%min_dis: The minimum distance from a reference path to all the paths that have different starting states as
the starting state of the reference path.
%ml_sym: The path having distance min_dis from the reference path.

%Date: April 14,1999 Xiangyu Song


global TOT_ST ST_TAB TXS ;
```

```
TOT_ST=n^D;
TOT_ST=n^D;
ST_TAB=stab;
TXS=sym_tab;
INV_STAB=inv_tab(n,D);%Generate inverse state transition table
TRS_INV_ST_TAB=INV_ST_TAB';
full_txs=fu_sym(n,D,DIM);%Generate full_txs


st_metric=zeros(TOT_ST,1);
adder=[0:TOT_ST:(n-1)*TOT_ST]';
adder=repmat(adder,1,TOT_ST);
pos_merge_st=TRS_INV_ST_TAB+adder;
B=B_Size;
full_sym=repmat(full_txs,[1 1 B]);

[in_sym, ref_out, st_seq]=enco_fr_deco(ref_path,n,D,DIM,B,1);%Get the state transition sequence and channel symbol sequence for ref_path
ref_st=st_seq(1);
ref_out=full(ref_out);

rev_mat=repmat(ref_out,[TOT_ST*n 1]);
rev_mat=reshape(rev_mat,n*TOT_ST,DIM,B);

dif=((full_sym-rev_mat).^2);
dif=-sum(dif,2);
dif=squeeze(dif);

sel_metric=[];sel_sym=[];
for STAR_ST=[[1:ref_st-1][ref_st+1:TOT_ST]] %===run for each state

%========Get path metrics and survivors for states at each stage=======
    st_sym=[];
    METRIC_MAT=reshape(dif,TOT_ST,n,B);
    %==travel trellis for stg_no=2:D=====
    metric=METRIC_MAT(STAR_ST,:,1);
    reach=ST_TAB(STAR_ST,:);
    sym=[0:n-1];

    for stg_no=2:D
        add_metric=METRIC_MAT(reach(:),:,stg_no);
        exp_metric=repmat(metric(:),1, 4);
```

```
        metric=exp_metric+add_metric;
        reach=ST_TAB(reach(:),:);

        add_sym=repmat([0:n-1],n^(stg_no-1),1);
        exp_sym=repmat(sym,1,n);
        sym=[exp_sym; add_sym(:)'];
    end %end of stg_no=2:D

    st_metric(reach(:))=metric;
    st_sym(:,reach(:))=sym;


    %========travel trellis for stage_no=D+1:B===
    for stage_no=D+1:B

        add_metric=dif(pos_merge_st(:),stage_no);
        merg_metric=st_metric(TRS_INV_ST_TAB(:))+add_metric;
        merg_metric=reshape(merg_metric,n,TOT_ST);
        [max_merg_metric max_pos]=max(merg_metric,[],1);
        st_metric=max_merg_metric';

        fr_st_pos=max_pos+n*[0:TOT_ST-1];
        fr_st=TRS_INV_ST_TAB(fr_st_pos(:));
        st_sym=[st_sym(:,fr_st(:)); max_pos-1];%update

    end;

    sel_metric=[sel_metric st_metric(STAR_ST)];
    sel_sym=[sel_sym st_sym(:,STAR_ST)];

end;%====end loop for each starting state============================

[ml_metric ml_pos]=max(sel_metric);
ml_sym=sel_sym(:,ml_pos)';
min_dis=-ml_metric;
```

APPENDIX G ITERATIVE CIRCULAR SHIFT BCJR DECODING ALGORITHM CODE

```
function err_pb=bcjr(n,D,DIM,B_Size,B_Num,ns,stab,sym_tab)

%Purpose: Iterative Circular Shift BCJR using the Most Reliable State Transition as the
starting point for decoding.

%Input Parameters
%n: n-ary information symbol
%D: Trellis Depth
%DIM: Dimensions of the signal constellation
%B_Size: Information sequence length. Can be a vector
%B_Num: Number of information sequences tested
%ns:Eb/N0(in dB)
%stab: State transition table for this trellis
%sym_tab: symbol assignment table, generated by sym_asgn.m

%In Function
%TOT_ST: Total number of states in this trellis.
%full_txs: Turn sym_tab in TOT_STx (n*3) matrix into (S*n) x DIM matrix. The TOT_ST rows--full_txs((i-
1)*TOT_ST+1:i*TOT_ST, :) are the symbols assigned to the TOT_ST transmissions when input is i-1.
i=1,2,...,n. Symbols are represented as 0 or (+)(-)1 in all the DIM dimensions.

%Output Parameters
%err_pb: Is the bit error probability averaged over total number of B_Num information sequences.

%Date:Jan 24, 1999   Xiangyu Song


global  TOT_ST ST_TAB  TXS ;

TOT_ST=n^D;
ST_TAB=stab;
TXS=sym_tab;
```

```
EB=3/2;
full_txs=fu_sym(n,D,DIM);

err_pb=[];
for b_size=1:size(B_Size,2) %*****************
B=B_Size(b_size);
full_sym=repmat(full_txs,[1 1 B]);

pb_bk=[];
[in_bit,in_sym,out]=enco(n,D,DIM,B,B_Num);%==Encoding==
for ns_no=1:size(ns,2)%*****
    N0=EB/(10^(ns(ns_no)/10));%==N0==
    var=sqrt(N0/2);
    noise=randn(1,DIM*B*B_Num)*var;
    rev=out+noise;

    err=0;
    for b_no=1:B_Num %**

        rev_mat=repmat(rev((b_no-1)*B*DIM+[1:B*DIM]),[TOT_ST*n 1]);
        rev_mat=reshape(rev_mat,n*TOT_ST,DIM,B);

        dif=((full_sym-rev_mat).^2)/N0;
        dif=-sum(dif,2);
        dif=squeeze(dif);
        prob=exp(dif);

        %=====Use Most Reliable state transition as starting point======
        %===Circular shift the received sequence===================
        %===Initilize Alpha & Beta as the Most Reliable State=========
        [maxr, row]=max(prob,[],1);
        [maxc,col]=max(maxr);
        m_row=row(col);
        st_fr=m_row-floor((m_row-0.5)/TOT_ST)*TOT_ST;
        st_to=ST_TAB(m_row);
        shift=col-1;
        alpha=zeros(B,TOT_ST);beta=zeros(TOT_ST,B);
        alpha(1,st_fr)=1;
        beta(st_fr,B)=1;
        prob_shift=[prob(:,[shift+1:B]) prob(:,[1:shift])];%shift=0.o.k
        prob=prob_shift;
```

```
%====Calculate gama=================================

row=repmat([1:TOT_ST],1,n);

gama=zeros(TOT_ST,TOT_ST,B);%saving time than gama=[].
gama_seq=eye(TOT_ST,TOT_ST);
for i=1:B
      s=sparse(row,ST_TAB,prob(:,i),TOT_ST,TOT_ST);
      s_full=full(s);
      gama(:,:,i)=s_full;

end;

delta=10;
%=====Forward and Backward recursion=============
while delta>10^(-3)%===Stop Condition==============
for t=2:B
      alpha(t,:)=alpha(t-1,:)*gama(:,:,t-1);
      if alpha(t,1)<10^(-250)
            alpha(t,:)=alpha(t,:)/sum(alpha(t,:));
      end;

       beta(:,B-t+1)=gama(:,:,B-t+2)*beta(:,B-t+2);
      if beta(1,B-t+1)<10^(-250)
            beta(:,B-t+1)=beta(:,B-t+1)/sum(beta(:,B-t+1));
      end;
end;
alphabp1=alpha(B,:)*gama(:,:,B);
alphabp1=alphabp1/sum(alphabp1);

betazr=  gama(:,:,1)*beta(:,1);
betazr=betazr/sum(betazr);

delta=max(abs(alpha(1,:)-alphabp1));

if delta>10^(-3)
      alpha(1,:)=alphabp1;
      beta(:,B)=betazr;
end;

end;%===End of while
```

```
            %====Calculate sigama============================
            mat=0:TOT_ST:(B-1)*TOT_ST;
            addmat=repmat(mat,TOT_ST*n,1);
            pos=repmat(ST_TAB(:),1,B)+addmat;
            exp_alpha=repmat(alpha,1,n);
            exp_alpha=exp_alpha';

            siga=exp_alpha(:).*prob(:).*beta(pos(:));
            siga_mat=reshape(siga,TOT_ST,n,B);
            sum_mat=sum(siga_mat);
            [i,j]=max(sum_mat);

            %====Decode and shift back===========================
            deco_sym_shift=squeeze(j)'-1;
            deco_sym=[deco_sym_shift(B-shift+1:B) deco_sym_shift(1:B-shift)];


            ft_bt=bitshift(deco_sym,-1);
            sd_bt=deco_sym-ft_bt*2;

            err_b=size(find((ft_bt-in_bit((b_no-1)*2*B+[1:2:2*B-1]))~=0),2)+size(find((sd_bt-in_bit((b_no-
1)*2*B+[2:2:2*B]))~=0),2);

            err=err+err_b;
        end;%**

        pb_ns=err/(log2(n)*B*B_Num);
        pb_bk=[pb_bk,pb_ns];
    end;%*****
    err_pb=[err_pb;pb_bk];

end;%**********************
```