View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by DigitalCommons@USU

Utah State University

DigitalCommons@USU

All Graduate Theses and Dissertations

Graduate Studies

8-2019

# Generalization of Signal Point Target Code

Md Munibun Billah

*Utah State University*

Follow this and additional works at: https://digitalcommons.usu.edu/etd

Part of the Electrical and Computer Engineering Commons

## Recommended Citation

Billah, Md Munibun, "Generalization of Signal Point Target Code" (2019). *All Graduate Theses and Dissertations*. 7586.
https://digitalcommons.usu.edu/etd/7586

This Thesis is brought to you for free and open access by the Graduate Studies at DigitalCommons@USU. It has been accepted for inclusion in All Graduate Theses and Dissertations by an authorized administrator of DigitalCommons@USU. For more information, please contact digitalcommons@usu.edu.

UtahStateUniversity
MERRILL-CAZIER LIBRARY

GENERALIZATION OF SIGNAL POINT TARGET CODE

by

Md Munibun Billah

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Electrical Engineering

Approved:

| | |
|---|---|
| Todd Moon, Ph.D.<br>Major Professor | Jacob Gunther, Ph.D.<br>Committee Member |
| Janak Sodha, Ph.D.<br>Committee Member | Richard S. Inouye, Ph.D.<br>Vice Provost for Graduate Studies |

UTAH STATE UNIVERSITY
Logan, Utah

2019

Copyright © Md Munibun Billah 2019

All Rights Reserved

ABSTRACT

Generalization of Signal Point Target Code

by

Md Munibun Billah, MASTER OF SCIENCE
Utah State University, 2019

Major Professor: Todd Moon, Ph.D.
Department: Electrical and Computer Engineering

Signal Point Target Codes have been proposed as an alternative to Trellis Coded Modulation. In this thesis, the basic definition of Signal Point Target Code including larger signal constellations, larger state size, different rate, and different shapes are extended in several ways. A notation is introduced to describe the operations, constellations of different sizes and codes of different rates are presented. Performance is evaluated by simulation and by distance bounds on the decoding trellis. Simulations are performed using different Programming languages to determine the suitable language for studying error correction coding.

(160 pages)

# PUBLIC ABSTRACT


Generalization of Signal Point Target Code

Md Munibun Billah

Detecting and correcting errors occurring in the transmitted data through a channel is a task of great importance in digital communication. In Error Correction Coding (ECC), some redundant data is added with the original data while transmitting. By exploiting the properties of the redundant data, the errors occurring in the data from the transmission can be detected and corrected. In this thesis, a new coding algorithm named Signal Point Target Code has been studied and various properties of the proposed code have been extended.

Signal Point Target Code (SPTC) uses a predefined shape within a given signal constellation to generate a parity symbol. In this thesis, the relation between the employed shape and the performance of the proposed code have been studied and an extension of the SPTC are presented.

This research presents simulation results to compare the performances of the proposed codes. The results have been simulated using different programming languages, and a comparison between those programming languages is provided. The performance of the codes are analyzed and possible future research areas have been indicated.

To my parents who shaped my knowledge about life and to Dr. Moon who shaped my knowledge about mathematics.

## ACKNOWLEDGMENTS

There are a lot of people in my life without their influence it wasn't possible for me to be here where I am today.

First and foremost, I would like to thank Dr. Todd K. Moon for supervising my master's thesis and giving me valuable direction throughout my research. Dr. Moon, without any doubt, is the best teacher and guide I have ever seen in my life. He made me like and hate math at the same time and made me realize how much I still need to learn.

Next, I would like to thank my committee members, Dr. Gunther, and Dr. Sodha. I would especially like to thank Dr. Sodha for introducing the topic on which my entire thesis is based on and giving me valuable insights during my research. I also want to mention Dr. Phillips's name here. I learned a lot while working as a grader for his courses.

I would like to thank Dr. Mehedi Hasan and Waled Al-Dulaimi for all their support. I also want to mention the name of Abrar Zahin, Rakin Muhammad Shadab, Abdullah Al Sarfin, Nazmus Sakib Rafi, Nazmus Sakib, Dr. Abul Bashar Mohammad Giasuddin, Ferdous Parvez, Saju Saha for being my good friends.

I want to thank Tricia Brandenburg for being such a wonderful person and Diane Buist for all the chocolates in front of her table.

Last but not least, I want to thank my parents and siblings for all their guidance, support and for being patient with me throughout the journey of my life.

Md Munibun Billah

CONTENTS

# LIST OF TABLES

LIST OF FIGURES

# ACRONYMS

| | |
|---|---|
| ECC | Error Correction Coding |
| AWGN | Additive White Gaussian Noise |
| BER | Bit Error Rate |
| SER | Symbol Error Rate |
| SNR | Signal to Noise Ratio |
| LDPC | Low Density Parity Check |
| QAM | Quadrature Amplitude Modulation |
| QPSK | Quadrature Phase Shift Keying |
| PSK | Phase Shift Keying |
| SPTC | Signal Point Target Code |
| SC | Shape Code |
| CAC | Constellation Arithmetic Code |
| SIE | Sequential Input Encoding |
| PPE | Partial Parallel Encoding |
| TCM | Trellis Coded Modulation |
| IDE | Integrated Development Environment |

CHAPTER 1

# INTRODUCTION

## 1.1 Overview

Error correction coding is the means of detecting and correcting errors induced by communication channels on received data by utilizing the properties of redundant data added with the message data for transmission. Since Shannon's channel coding theorem [1] in 1948, studies to find good codes which achieve near channel capacity have been conducted. Finding a code of lower complexity that achieves Shannon channel capacity is of primary interest in the field of error correction coding. Those studies led to the discovery of some good codes, such as Low Density Parity Check (LDPC) [2], Turbo Codes [3] and Polar Codes [4]. The redundant bits of error correcting codes increase the ability to detect and correct errors but reduce the rate of transmission. Ungerboeck proposed a coding scheme Trellis Coded Modulation (TCM) [5] which incorporates error correction coding and modulation as a single unit to improve the information transmission rate. In [5], Ungerboek used a convolutional encoder followed by mapping into the signal constellation to encode and a sequential decoder to decode TCM. A new coding algorithm named Shape Code (SC) was proposed in [6] and extended in [7], which takes into account a shape or predefined path in the signal constellation. That shape was used to encode input symbols to generate parity symbols. Like TCM, SC employs error correction coding within the signal space. The work of this thesis is to study, extend and evaluate multiple attributes of the encoding algorithm [7] in simulation.

## 1.2 Literature Review

The concept of shapes formed within the decoder and how these shapes affect the decoding capability was introduced in [8]. Using the concept of shape, Sodha devised a rate 1/2 systematic code (Shape Code) on QPSK constellation [6]. A square shape within the signal space was formed by selecting appropriate points from the constellations. This collection of points was used to encode input symbols by calculating the numbers of clockwise 90 degree rotations to go from the input symbol to a point from that collection. This collection of points was later named as target points in [7] where the idea of Shape Code was extended for a larger constellation (16QAM). In [7], alongside with the extension of Shape Code for a larger constellation, a state variable was introduced which enabled the encoder to form a trellis by utilizing input symbols and target points. The trellis structure allowed for trellis based decoding. From [7], it can be seen that for shape code, a standard Viterbi decoder increases the coding gain compared to the performance in [6]. It will be also shown that the coding gain depends on the free Euclidean distance on the trellis and by increasing the distance is possible to achieve more coding gain.

### 1.3 Thesis Objectives

The primary goal of this thesis is to study and generalize Signal Point Target Code (SPTC). The performance of the codes will be measured in Bit Error Rate (BER) and Symbol Error Rate (SER) and will be compared with the results of uncoded modulations. The objectives of this thesis are:

1. Generalize the encoder which is already proposed for QPSK and 16QAM, to other QAM signal constellation.
2. Study the performance of the proposed code as shapes are modified.
3. Study encoding under different coding rates.
4. Study the performance of the encoder for the different number of states.
5. Study the effect on the performance of the code due to the free Euclidean distance in trellis to gain some understanding of the theoretical coding gain.

A secondary objective of this thesis is to compare the simulation speed using between MATLAB, Python, and Julia to determine which tool is more suitable for error correction studies.

### 1.4 Chapter Outlines

The chapter organization for this thesis is as follows:

Chapter 2 presents the background of SPTC. At the beginning of chapter 2, the encoding operation of SPTC is by introducing appropriate notations. Using trellis structure, the free Euclidean distance related to the shape is calculated to show that the performance of SPTC depends on the employed shape. How the codes can be decoded using the standard Viterbi algorithm is also described in this chapter. Following the motivation to increase free Euclidean distance for codes by searching for the best shape, an alternative of SPTC called Constellation Arithmetic Code (CAC) which depends on the bit assignment rather than the shape is introduced. A solution for bit assignment for CAC in the 16QAM and 64QAM signal constellation is presented that provides better coding gain compared to the performances of the shapes for encoding SPTC explored in this thesis. At the end of this chapter, two methods for increasing the rate of CAC are proposed.

Chapter 3 starts with the required calculations for simulating the performance of the proposed codes using an AWGN channel. How the noise is generated using the variance obtained for a given constellation is described here. The equations of the probability of symbol error for both theory and simulation are provided. Later in this chapter, simulation results for SPTC and CAC are presented, comparison between the performances of SPTC and CAC are made and the performances of rate 2/3 and rate 3/4 codes for CAC are presented. Finally, this chapter ends with the comparison between the runtime of MATLAB, Python, and Julia to determine a suitable language for error correction coding study.

In chapter 4, the idea of how increasing the number of states can increase the free Euclidean distance and therefore increase the coding gain has been explored. An approach for increasing the number of states and the free Euclidean distances is proposed. For rate 1/2 CAC using QPSK and 16QAM constellation, a list for the different number of states and the associated free Euclidean distances is provided. However, the proposed approach failed to provide any coding gain for the increment of the number of states in simulations. So this chapter is presented in this thesis in order to provide an insight for future researches.

Chapter 5 concludes the thesis by discussing the findings for SPTC and CAC and by pointing out areas for possible future research. The appendix contains all of the codes written for the simulations in this thesis.

# CHAPTER 2

# SIGNAL POINT TARGET CODE

The focus of this chapter is the Signal Point Target Code (SPTC). In this chapter, the encoding operations of SPTC are explained first. Then, a modified version of SPTC is presented and using that modified version two methods for changing the rate of codes are proposed.

Let $M$ denote the number of points in a digital constellation and $n = \log_2 M$ be the number of bits per symbol. Let $\{b_0, b_1, b_2, ....\}$ denote a sequence of randomly generated bits where $b \in \{0, 1\}$. At the $i$th symbol interval, $n$ bits are stacked into a binary vector $\mathbf{b}_i = [b_{ni}\ b_{ni+1}.....\ b_{ni+n-1}]^T$.

Let $\mathbb{S}$ be the set of points in the signal constellation and $\phi : \{0, 1\}^n \rightarrow \mathbb{S}$ be a bijective mapping from $n$ dimensional binary vector $\mathbf{b}$ to a point in the signal space $S \in \mathbb{S}$,

$$S = \phi(\mathbf{b}). \tag{2.1}$$

$\phi$ is the bit assignment operation in the signal constellation. In this thesis, for SPTC gray code indexing has been used in all the signal constellations. However, for a modified version of SPTC called Constellation Arithmetic Code, bit assignment operations beside gray code indexing are used to evaluate coding gain.

## 2.1 Encode

Let $\psi : \mathbb{S} \rightarrow \mathbb{Z}_{\geq 0}$ be a bijective mapping between points in the signal constellation $S$ and an integer $m$ where $\psi$ assigns each point in the signal constellation to a unique integer index $m \in \mathbb{Z}_{\geq 0}$. In this thesis, the integer index $m$ is equal to the decimal representation of the binary vector that has been assigned to that point by $\phi$. The set of such integers will be denoted as the set of signal space index $\mathcal{I}$. For example, in the signal constellation shown in figure 2.1, $\phi$ maps binary vectors [0 0 1 0] and [1 1 0 1] to points $(-3, 3)$ and $(1, -1)$

while $\psi$ maps the point $(-3,3) \in \mathbb{S}$ to the integer index 2 and the point $(1,-1) \in \mathbb{S}$ to the integer index 13.

Under this encoding scheme, let an "addition" operation $\oplus : \mathcal{I} \times \mathbb{Z}_{\geq 0} \to \mathcal{I}$ in the signal constellation, such that $s_{\text{new}} = s_i \oplus j$ moves from the point $s_i$ in the signal space to a new point $s_{\text{new}}$ by the amount $j$ following a predefined path, where $s_{\text{new}}, s_i \in \mathcal{I}$ and $j \in \mathbb{Z}_{\geq 0}$. For example, for the path defined by blue arrows in the signal constellation shown in Fig 2.1, $7 \oplus 6 = 8$ because from point 7 moving 6 steps along the path takes to point 8. In this thesis, the predefined path which is followed by the $\oplus$ operator is defined as *shape*. Any shape can be used for encoding as long as the shape goes through all of the points in the signal constellation and the shape goes through each point only once. The total number points which are used to form the shape is $M$. The addition defined by $\oplus$ associated with the shape shown in Fig. 2.1 is listed in Table 2.1.

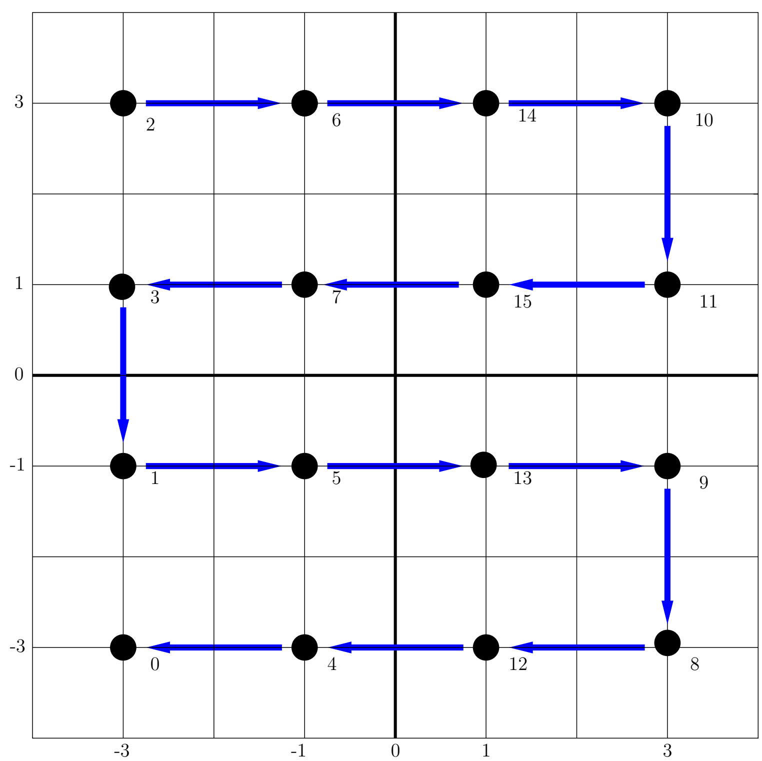


Fig. 2.1: 16-QAM signal constellation with a shape indicated by blue arrows.

| | | $j$ | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\oplus$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| | 0 | 0 | 2 | 6 | 14 | 10 | 11 | 15 | 7 | 3 | 1 | 5 | 13 | 9 | 8 | 12 | 4 |
| | 1 | 1 | 5 | 13 | 9 | 8 | 12 | 4 | 0 | 2 | 6 | 14 | 10 | 11 | 15 | 7 | 3 |
| | 2 | 2 | 6 | 14 | 10 | 11 | 15 | 7 | 3 | 1 | 5 | 13 | 9 | 8 | 12 | 4 | 0 |
| | 3 | 3 | 1 | 5 | 13 | 9 | 8 | 12 | 4 | 0 | 2 | 6 | 14 | 10 | 1 | 15 | 7 |
| | 4 | 4 | 0 | 2 | 6 | 14 | 10 | 11 | 15 | 7 | 3 | 1 | 5 | 13 | 9 | 8 | 12 |
| | 5 | 5 | 13 | 9 | 8 | 12 | 4 | 0 | 2 | 6 | 14 | 10 | 11 | 15 | 7 | 3 | 1 |
| | 6 | 6 | 14 | 10 | 11 | 15 | 7 | 3 | 1 | 5 | 13 | 9 | 8 | 12 | 4 | 0 | 2 |
| | 7 | 7 | 3 | 1 | 5 | 13 | 9 | 8 | 12 | 4 | 0 | 2 | 6 | 14 | 10 | 11 | 15 |
| $s$ | 8 | 8 | 12 | 4 | 0 | 2 | 6 | 14 | 10 | 11 | 15 | 7 | 3 | 1 | 5 | 13 | 9 |
| | 9 | 9 | 8 | 12 | 4 | 0 | 2 | 6 | 14 | 10 | 11 | 15 | 7 | 3 | 1 | 5 | 13 |
| | 10 | 10 | 11 | 15 | 7 | 3 | 1 | 5 | 13 | 9 | 8 | 12 | 4 | 0 | 2 | 6 | 14 |
| | 11 | 11 | 15 | 7 | 3 | 1 | 5 | 13 | 9 | 8 | 12 | 4 | 0 | 2 | 6 | 14 | 10 |
| | 12 | 12 | 4 | 0 | 2 | 6 | 14 | 10 | 11 | 15 | 7 | 3 | 1 | 5 | 13 | 9 | 8 |
| | 13 | 13 | 9 | 8 | 12 | 4 | 0 | 2 | 6 | 14 | 10 | 11 | 15 | 7 | 3 | 1 | 5 |
| | 14 | 14 | 10 | 11 | 15 | 7 | 3 | 1 | 5 | 13 | 9 | 8 | 12 | 4 | 0 | 2 | 6 |
| | 15 | 15 | 7 | 3 | 1 | 5 | 13 | 9 | 8 | 12 | 4 | 0 | 2 | 6 | 14 | 10 | 11 |

Table 2.1: Input output relation table for $\oplus$ where column $s$ is input symbol and row $j$ is the number of steps along the predefined path.

The SPTC uses a target point $t_k$ selected from a sequence of target points within the signal space to calculate the required number of steps to go from any point $s_v$ to that target point. The sequence of target points may consist of a single point (a sequence of length 1) or multiple points. If the sequence of target points consists of multiple points, then at each $i$ the target point $t_k$ is time varying with $t_k = t_{i \ (mod \ u)}$, where $u$ is the length of the sequence of target points. In this thesis, fixed target point $t = 0$ has been used in all encoding even though the target point can follow a shape by selecting appropriate points from the signal constellation as the sequence of target points. For example, points $\{2, 6, 14, 10, 11, 9, 8, 12, 4, 0, 1, 3, 2\}$ are selected as sequence of target points so that the target point traces a square shape shown in Fig. 2.2.

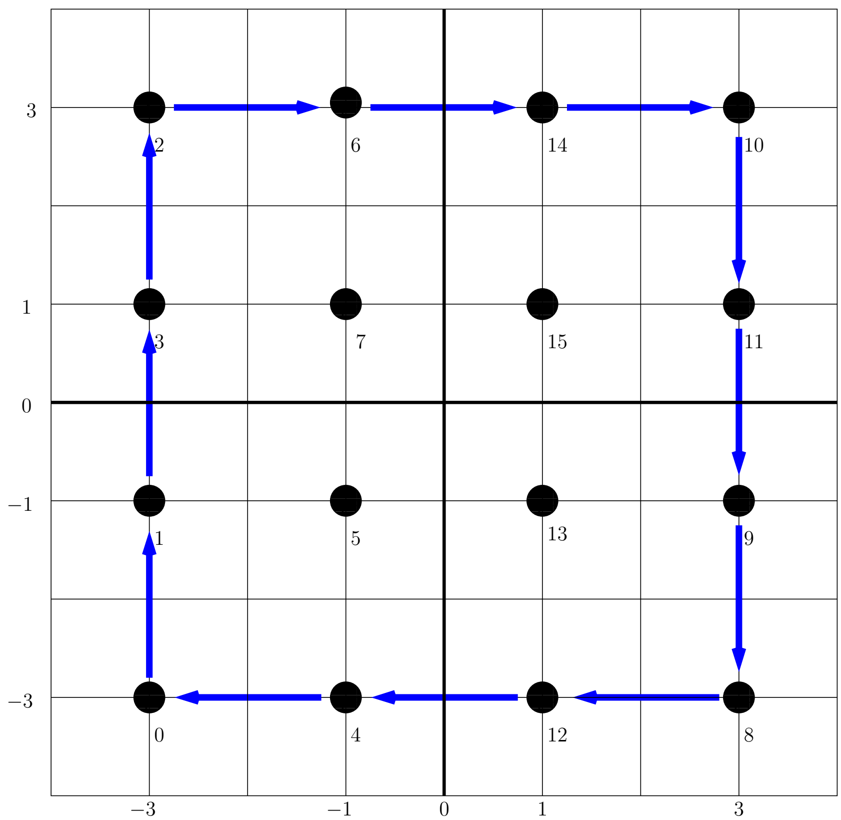


Fig. 2.2: 16-QAM signal constellation with a square shape indicated by blue arrows which can be traced by target point.

Let $\ominus : \mathcal{I} \times \mathcal{I} \rightarrow \mathbb{Z}_{\geq 0}$ be defined as the operation $r = t_k \ominus s_v$ that calculates the number of steps $r$ along the path necessary to move from point $s_v$ to a point $t_k$, where, $t_k, s_v \in \mathcal{I}$ and $r \in \mathbb{Z}_{\geq 0}$. For example, for the shape shown in Fig. 2.1, $4 \ominus 5 = 5$ because from point 5 in the signal space, it takes 5 steps along the path indicated by the blue arrows to reach point 4. For fixed target points $t = 0$, $t = 1$, ....., $t = 15$, the required number of movements from each point in the signal constellation along the path shown in Fig. 2.1 is listed in the

Table 2.2.

| | | | | | | | | | $a$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\ominus$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| | 0 | 0 | 7 | 15 | 8 | 1 | 6 | 14 | 9 | 3 | 4 | 12 | 11 | 2 | 5 | 13 | 10 |
| | 1 | 9 | 0 | 8 | 1 | 10 | 15 | 7 | 2 | 12 | 13 | 5 | 4 | 11 | 14 | 6 | 3 |
| | 2 | 1 | 8 | 0 | 9 | 2 | 7 | 15 | 10 | 4 | 5 | 13 | 12 | 3 | 6 | 14 | 11 |
| | 3 | 8 | 15 | 7 | 0 | 9 | 14 | 6 | 1 | 11 | 12 | 4 | 3 | 10 | 13 | 5 | 2 |
| | 4 | 15 | 6 | 14 | 7 | 0 | 5 | 13 | 8 | 2 | 3 | 11 | 10 | 1 | 4 | 12 | 9 |
| | 5 | 10 | 1 | 9 | 2 | 11 | 0 | 8 | 3 | 13 | 14 | 6 | 5 | 12 | 15 | 7 | 4 |
| | 6 | 2 | 9 | 1 | 10 | 3 | 8 | 0 | 11 | 5 | 6 | 14 | 13 | 4 | 7 | 15 | 12 |
| | 7 | 7 | 14 | 6 | 15 | 8 | 13 | 5 | 0 | 10 | 11 | 3 | 2 | 9 | 12 | 4 | 1 |
| $t$ | 8 | 13 | 4 | 12 | 5 | 14 | 3 | 11 | 6 | 0 | 1 | 9 | 8 | 15 | 2 | 10 | 7 |
| | 9 | 12 | 3 | 11 | 4 | 13 | 2 | 10 | 5 | 15 | 0 | 8 | 7 | 14 | 1 | 9 | 6 |
| | 10 | 4 | 11 | 3 | 12 | 5 | 10 | 2 | 13 | 7 | 8 | 0 | 15 | 6 | 9 | 1 | 14 |
| | 11 | 5 | 12 | 4 | 13 | 6 | 11 | 3 | 14 | 8 | 9 | 1 | 0 | 7 | 10 | 2 | 15 |
| | 12 | 14 | 5 | 13 | 6 | 15 | 4 | 12 | 7 | 1 | 2 | 10 | 9 | 0 | 3 | 11 | 8 |
| | 13 | 11 | 2 | 10 | 4 | 12 | 1 | 9 | 4 | 14 | 15 | 7 | 6 | 13 | 0 | 8 | 5 |
| | 14 | 3 | 10 | 2 | 11 | 4 | 9 | 1 | 12 | 6 | 7 | 15 | 14 | 5 | 8 | 0 | 13 |
| | 15 | 6 | 13 | 5 | 14 | 7 | 12 | 4 | 15 | 9 | 10 | 2 | 1 | 8 | 11 | 3 | 0 |

Table 2.2: Input output relation table for $\ominus$ where column $t$ is the input target point and row $a$ is the input current point in the signal space.

For 64 QAM, a shape similar to Fig. 2.1 is shown by the blue arrows in Fig. 2.3. For this shape, Tables 2.3, 2.4, 2.5, 2.6 contain the outputs of the $\oplus$ operator and the required number of movement to reach fixed target point $t = 0$ from every point in the signal constellation is listed in Tables 2.7 and 2.8.

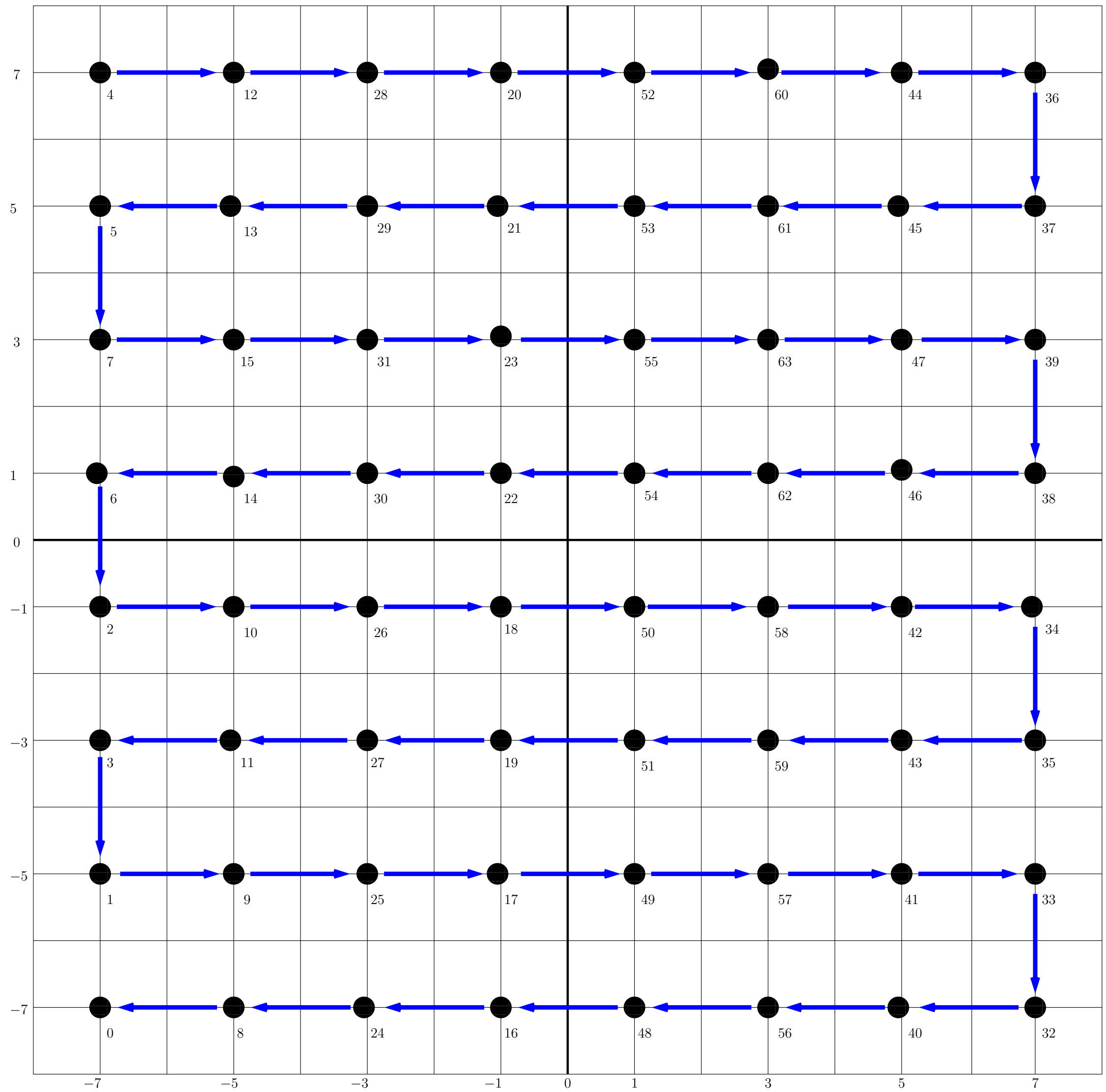


Fig. 2.3: 64-QAM signal constellation with a shape indicated by blue arrows.

| | $j$ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\oplus$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 |
| 1 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 |
| 2 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 |
| 3 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 |
| 4 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 |
| 5 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 |
| 6 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 |
| 7 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 |
| 8 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 |
| 9 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 |
| 10 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 |
| 11 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 |
| 12 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 |
| 13 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 |
| 14 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 |
| $s$ 15 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 |
| 16 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 |
| 17 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 |
| 18 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 |
| 19 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 |
| 20 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 |
| 21 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 |
| 22 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 |
| 23 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 |
| 24 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 |
| 25 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 |
| 26 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 |
| 27 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 |
| 28 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 |
| 29 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 |
| 30 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 |
| 31 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 |

Table 2.3: Input output relation table for $\oplus$ where column $s$ is input symbol and row $j$ is the number of steps along the predefined path.

| | | $j$ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\oplus$ | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
| | 0 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 |
| | 1 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 |
| | 2 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 |
| | 3 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 |
| | 4 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 |
| | 5 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 |
| | 6 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 |
| | 7 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 |
| | 8 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 |
| | 9 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 |
| | 10 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 |
| | 11 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 |
| | 12 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 |
| | 13 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 |
| | 14 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 |
| $s$ | 15 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 |
| | 16 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 |
| | 17 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 |
| | 18 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 |
| | 19 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 |
| | 20 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 |
| | 21 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 |
| | 22 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 |
| | 23 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 |
| | 24 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 |
| | 25 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 |
| | 26 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 |
| | 27 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 |
| | 28 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 |
| | 29 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 |
| | 30 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 |
| | 31 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 |

Table 2.4: Input output relation table for $\oplus$ where column $s$ is input symbol and row $j$ is the number of steps along the predefined path.

| | | | | | | | | | | | | | | | | | $j$ | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\oplus$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| | 32 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 |
| | 33 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 |
| | 34 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 |
| | 35 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 |
| | 36 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 |
| | 37 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 |
| | 38 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 |
| | 39 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 |
| | 40 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 |
| | 41 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 |
| | 42 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 |
| | 43 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 |
| | 44 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 |
| | 45 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 |
| | 46 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 |
| $s$ | 47 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 |
| | 48 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 |
| | 49 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 |
| | 50 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 |
| | 51 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 |
| | 52 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 |
| | 53 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 |
| | 54 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 |
| | 55 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 |
| | 56 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 |
| | 57 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 |
| | 58 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 |
| | 59 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 |
| | 60 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 |
| | 61 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 |
| | 62 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 |
| | 63 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 |

Table 2.5: Input output relation table for $\oplus$ where column $s$ is input symbol and row $j$ is the number of steps along the predefined path.

| | | $j$ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\oplus$ | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
| | 32 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 |
| | 33 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 |
| | 34 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 |
| | 35 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 |
| | 36 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 |
| | 37 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 |
| | 38 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 |
| | 39 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 |
| | 40 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 |
| | 41 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 |
| | 42 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 |
| | 43 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 |
| | 44 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 |
| | 45 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 |
| | 46 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 |
| $s$ | 47 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 |
| | 48 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 |
| | 49 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 |
| | 50 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 |
| | 51 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 |
| | 52 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 |
| | 53 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 |
| | 54 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 |
| | 55 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 |
| | 56 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 |
| | 57 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 |
| | 58 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 |
| | 59 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 | 62 | 54 | 22 | 30 | 14 | 6 | 2 | 10 | 26 | 18 | 50 | 58 | 42 | 34 | 35 | 43 |
| | 60 | 58 | 42 | 34 | 35 | 43 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 |
| | 61 | 59 | 51 | 19 | 27 | 11 | 3 | 1 | 9 | 25 | 17 | 49 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 |
| | 62 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 | 63 | 47 | 39 | 38 | 46 |
| | 63 | 57 | 41 | 33 | 32 | 40 | 56 | 48 | 16 | 24 | 8 | 0 | 4 | 12 | 28 | 20 | 52 | 60 | 44 | 36 | 37 | 45 | 61 | 53 | 21 | 29 | 13 | 5 | 7 | 15 | 31 | 23 | 55 |

Table 2.6: Input output relation table for $\oplus$ where column $s$ is input symbol and row $j$ is the number of steps along the predefined path.

| | | $a$ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\ominus$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| $t$ | 0 | 0 | 15 | 31 | 16 | 63 | 48 | 32 | 47 | 1 | 14 | 30 | 17 | 62 | 49 | 33 | 46 | 3 | 12 | 28 | 19 | 60 | 51 | 35 | 44 | 2 | 13 | 29 | 18 | 61 | 50 | 34 | 45 |

Table 2.7: Input output relation table for $\ominus$ where column $t$ is the input target point and row $a$ is the input current point in the signal space.

| | | $a$ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\ominus$ | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
| $t$ | 0 | 7 | 8 | 24 | 23 | 56 | 55 | 39 | 40 | 6 | 9 | 25 | 22 | 57 | 54 | 38 | 41 | 4 | 11 | 27 | 20 | 59 | 52 | 36 | 43 | 5 | 10 | 26 | 21 | 58 | 53 | 37 | 42 |

Table 2.8: Input output relation table for $\ominus$ where column $t$ is the input target point and row $a$ is the input current point in the signal space.

The $\oplus$ and $\ominus$ operations can be viewed on an $M$ sided regular polygon where the vertices are labeled in clockwise order as the points of the path defined in the signal constellation. For example, the path from Fig 2.1 can be mapped to the vertices of hexadecagon shown in Fig. 2.4. Now $7 \oplus 6 = 8$ means moving 6 edges from vertex labeled as 7 in the clockwise direction will lead to vertex 8 and $4 \ominus 5 = 5$ means there are 5 edges between vertex 5 and 4 in the clockwise direction.

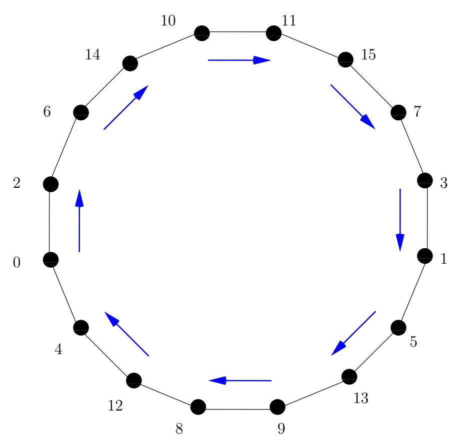


Fig. 2.4: Representation $\oplus$ and $\ominus$ operation on a M sided regular polygon.

The encoder has a state variable $j_i \in \{0, 1, 2, ..., M-1\}$ which determines the location along the path in the trellis after encoding for each time $i$. At each time $i$, the path starts at state variable $j_i$ and ends at $j_{i+1}$. The state variable $j_{i+1}$ is calculated by

$$j_{i+1} = j_i + r_i \pmod{M}. \tag{2.2}$$

The state variable accumulates parity symbol index $r_i$ generated at time step $i$ and the previous state $j_i$ to determine where the path is going to end at that time step. The endpoint for a branch depends on the input symbol $S_i$, the state variable $j_i$ and the target point $t$. For a fixed $j_i$, a different combination of $S_i$ and $t$ will lead to a different endpoint $j_{i+1}$. So at each $i$, $j_i$ and $j_{i+1}$ determine which branch of the trellis is taken during the encoding thus decides the direction of the path in the trellis. For this encoder, the number of states is equal to the number of points in the signal constellation.

For a given shape in the signal constellation and $\{t_0, t_1, t_2, t_3, .....\}$ as the sequence of target points, the encoding operation for SPTC at every symbol time $i$ is given in Algorithm 1.

**Algorithm 1** Signal Point Target Code Encoding

1: input: symbol $S_i$ , previous state variable $j_i$

2: $s_i \leftarrow \psi(S_i)$ compute the integer index of $S_i$

3: $a_i \leftarrow s_i \oplus j_i$

4: determine the target point $t_k$ where $k = i \pmod u$

5: $r_i \leftarrow t_k \ominus a_i$

6: $j_{i+1} \leftarrow j_i + r_i \pmod M$ update the state after encoding

7: $R_i \leftarrow \psi^{-1}(r_i)$ compute the parity symbol $R_i$ from the integer index $r_i$

8: return parity symbol $R_i$ and state variable $j_{i+1}$

Associated with each branch at the time $i$, there is an input symbol $S_i$ and parity symbol $R_i$. An example of trellis using QPSK constellation is shown in Fig. 2.5. In this trellis all of the possible branches at the time $i$ that can go from every initial state $j_i$ to every final state $j_{i+1}$ is represented by lines. Starting from the initial state 1, the branch which goes to the final state 2 is presented by a black solid line. The input symbol $S_i$ and parity symbol $R_i$ associated with this branch is presented on top of the branch labeled as $S_i/R_i$ in the figure. Since the endpoint of each branch in the trellis depends on the state variable and the state variable depends on the target point, the trellis is time varying if the sequence of target points contains multiple points. Initially, the encoder starts at state $j_0 = 0$ and with each increment of time $i$ it creates a path which allows a trellis based decoder to determine the sequence of transmitted symbols. At each $i$, the encoder output is the input symbol $S_i$ and parity symbol $R_i$ which makes the encoder of rate 1/2.

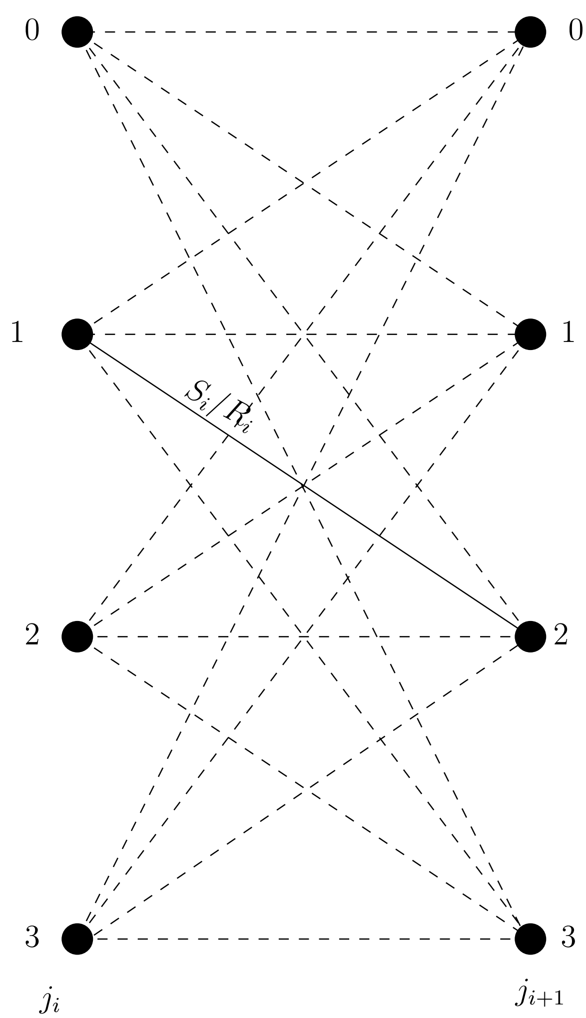


Fig. 2.5: Example of trellis structure using a QPSK constellation.

## 2.2 Decode

To decode SPTC, a trellis is formed in which for every symbol time $i$ all of the branches that can go from every possible initial state $j_i$ to every possible final state $j_{i+1}$ are calculated. Those branches store their corresponding input symbols and parity symbols. For QPSK signal constellation with the shape shown in Fig. 2.6 and a fixed target point $t = \{0\}$, the trellis is shown in Fig. 2.7, where each branch from state $j_i$ to state $j_{i+1}$ is labeled as $S_i/R_i$ which corresponds to the branch input symbol and parity symbol.

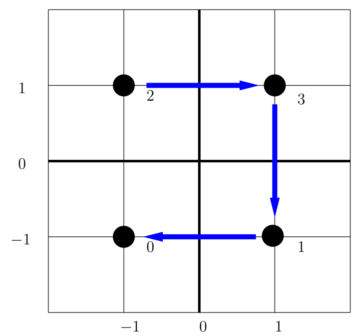


Fig. 2.6: QPSK signal constellation with a shape indicated by blue arrows.

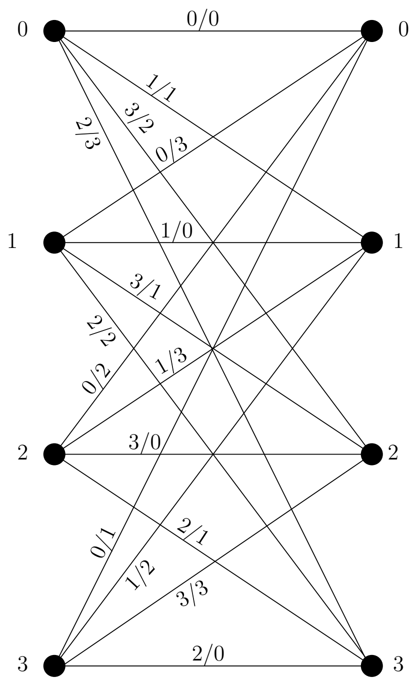


Fig. 2.7: Trellis structure for SPTC using QPSK constellation for fixed target point $t = 0$ and shape shown in Fig. 2.6.

In order to decode using the formed trellis, a standard Viterbi algorithm [9] has been implemented which uses squared Euclidean distance between branch input symbol $S_i$ and received symbol $\hat{S}_i$ plus squared Euclidean distance between branch parity symbol $R_i$ and received parity symbol $\hat{R}_i$ as branch metric

$$\mu_t = d_e^2(S_i, \hat{S}_i) + d_e^2(R_i, \hat{R}_i). \tag{2.3}$$

Here squared Euclidean distance is used to calculate branch metric which is also a measure of the energy difference between the symbols. The Viterbi decoder starts at all zero-state and the trellis depth of the Viterbi algorithm is 10. Simulation results for the proposed code using different trellis depth in the Viterbi decoder are presented in Fig. 3.16. From Fig. 3.16, it can be seen that the performance of the decoder remains almost the same after a certain trellis depth. So trellis depth of 10 for the Viterbi algorithm has been selected to decode. For each state $j_{i+1}$ at time $i$ branch metric $\mu_t$ is calculated for all the possible branches originated from each state $j_i$ at time $i-1$ and the branch metric added with the path metric $M(j_i)$ for each surviving path at time $i-1$ to get path metric $M(j_{i+1})$ and the path to state $j_{i+1}$ with the minimum path metric is selected as the surviving path at time $i$. The optimum output sequence is the path which has the minimum path metric through the trellis.

### 2.3 Free Euclidean Distance

Similar to convolutional codes and TCM, the performance of SPTC in terms of coding gain depends on the smallest distance between any two paths which diverge and then merge into the same state in the trellis. This smallest distance is also called the free Euclidean distance [9] and is denoted by $d^2_{\text{free}}$. The distance between these two paths is the sum of squared Euclidean distances in the signal constellation between the input symbols associated with these paths and the parity symbols associated with these paths. In Fig. 2.8, the Euclidean distances between two pair of points are shown by blue lines where the average energy per coded symbol is $E_{c,s}$. For this constellation, $d_e(0,1) = d_e(3,1) = d_e(2,0) = d_e(2,3) = 2\sqrt{E_{c,s}}$ and $d_e(2,1) = d_e(0,3) = \sqrt{8E_{c,s}}$. For example, for the trellis shown in Fig. 2.7, among all of these diverging and then merging paths, the pair of paths which has the smallest distance is indicated by black solid lines in Fig. 2.9. In Fig. 2.9, starting from state 0 one path goes through state 0 which has input symbol 0 and parity symbol 0 while the other path diverges to state 1 with input symbol 1 and parity symbol 1. Then the second path merges back to state 0 with input symbol 0 and parity symbol 3. So using the Euclidean distances from the signal constellation shown in Fig. 2.8, the free Euclidean distance

$$\begin{aligned} d^2_{\text{free}} &= d^2_e(0,1) + d^2_e(0,1) + d^2_e(0,0) + d^2_e(0,3) \\ &= 4E_{c,s} + 4E_{c,s} + 0 + 8E_{c,s} \\ &= 16E_{c,s}. \end{aligned}$$

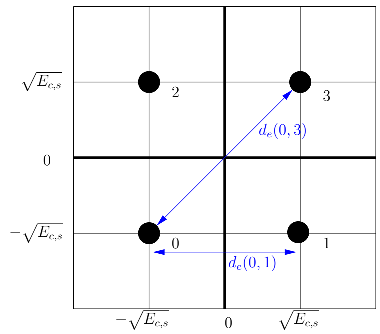


Fig. 2.8: QPSK constellation.

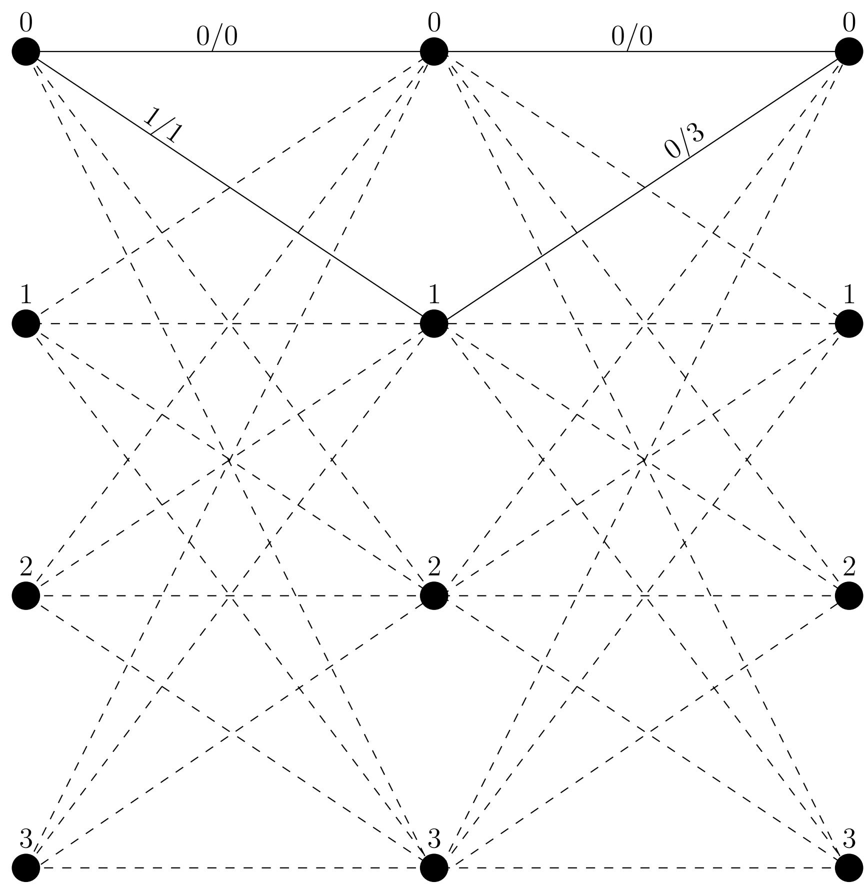


Fig. 2.9: Branches (black solid line) on the trellis that results in $d^2_{\text{free}}$ for SPTC on QPSK for fixed target point $t = 0$ and shape shown in Fig. 2.6.

For a given bit assignment in the signal constellation, different shapes result in different trellis structure. So each shape corresponds with it's own $d^2_{\text{free}}$ and from the simulation results presented in chapter 3, it can be seen that the shape with larger $d^2_{\text{free}}$ has better coding gain than other shapes. For example, the trellis and the pair of paths that produces free Euclidean distance for the shape shown in Fig. 2.10 are presented in Figs. 2.11 and 2.12. For this shape, the free Euclidean distance on the trellis using the signal constellation from Fig. 2.8 is

$$\begin{aligned} d^2_{\text{free}} &= d_e^2(2,1) + d_e^2(2,3) + d_e^2(1,2) + d_e^2(2,1) \\ &= 8E_{c,s} + 4E_{c,s} + 8E_{c,s} + 8E_{c,s} \\ &= 28E_{c,s} \end{aligned}$$

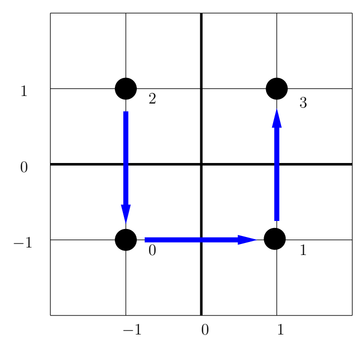


Fig. 2.10: QPSK signal constellation with a shape indicated by blue arrows.

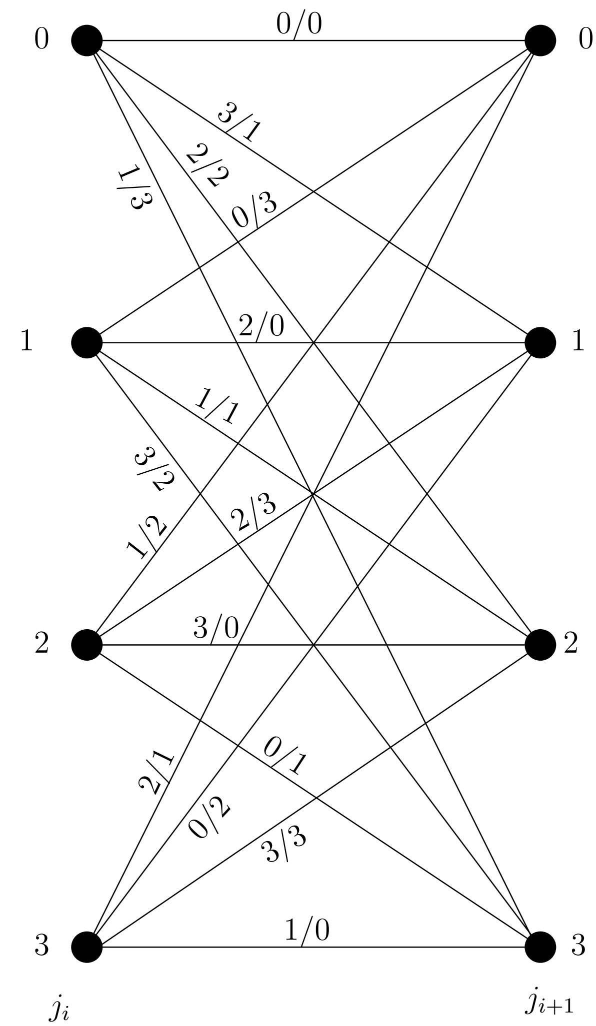


Fig. 2.11: Trellis structure for SPTC using QPSK constellation for fixed target point $t = 0$ and shape shown in Fig. 2.10.

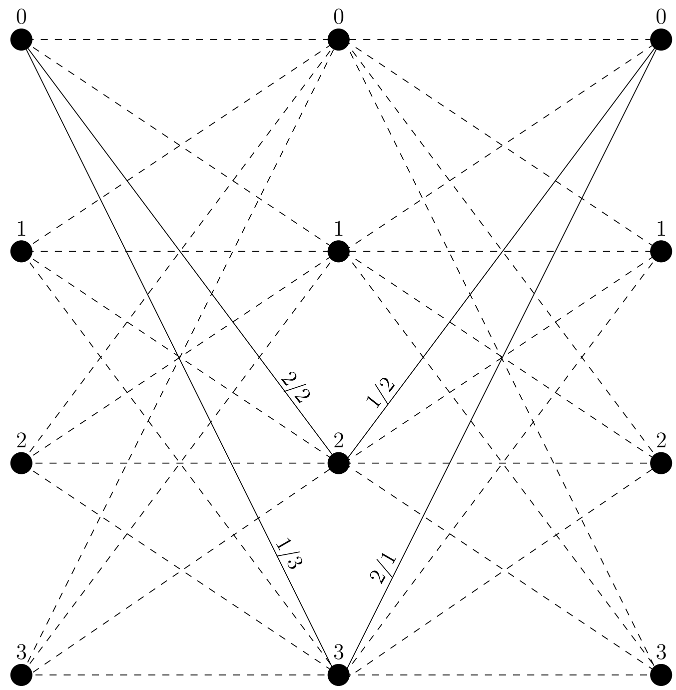


Fig. 2.12: Branches (black solid line) on the trellis that results in $d^2_{\text{free}}$ for SPTC on QPSK for fixed target point $t = 0$ and shape shown in Fig. 2.10.

By rotating the target point after each input signal point does not yield an improved coding gain because the relative distance between branches within a given code trellis is the same. So a constant target point $t = 0$ has been used in this thesis to encode the input symbols. For 16QAM constellation, the shape and bit assignment in the signal constellation shown in Fig. 2.1 and fixed target point $t = 0$, the pair of paths which constitutes the free Euclidean distance is shown in Fig. 2.13 by black solid lines. For this shape, the free Euclidean distance is

$$\begin{aligned} d^2_{\text{free}} &= d^2_e(4, 13) + d^2_e(1, 5) + d^2_e(0, 0) + d^2_e(15, 11) \\ &= 8E_{c,s} + 4E_{c,s} + 0 + 4E_{c,s} \\ &= 16E_{c,s} \end{aligned}$$

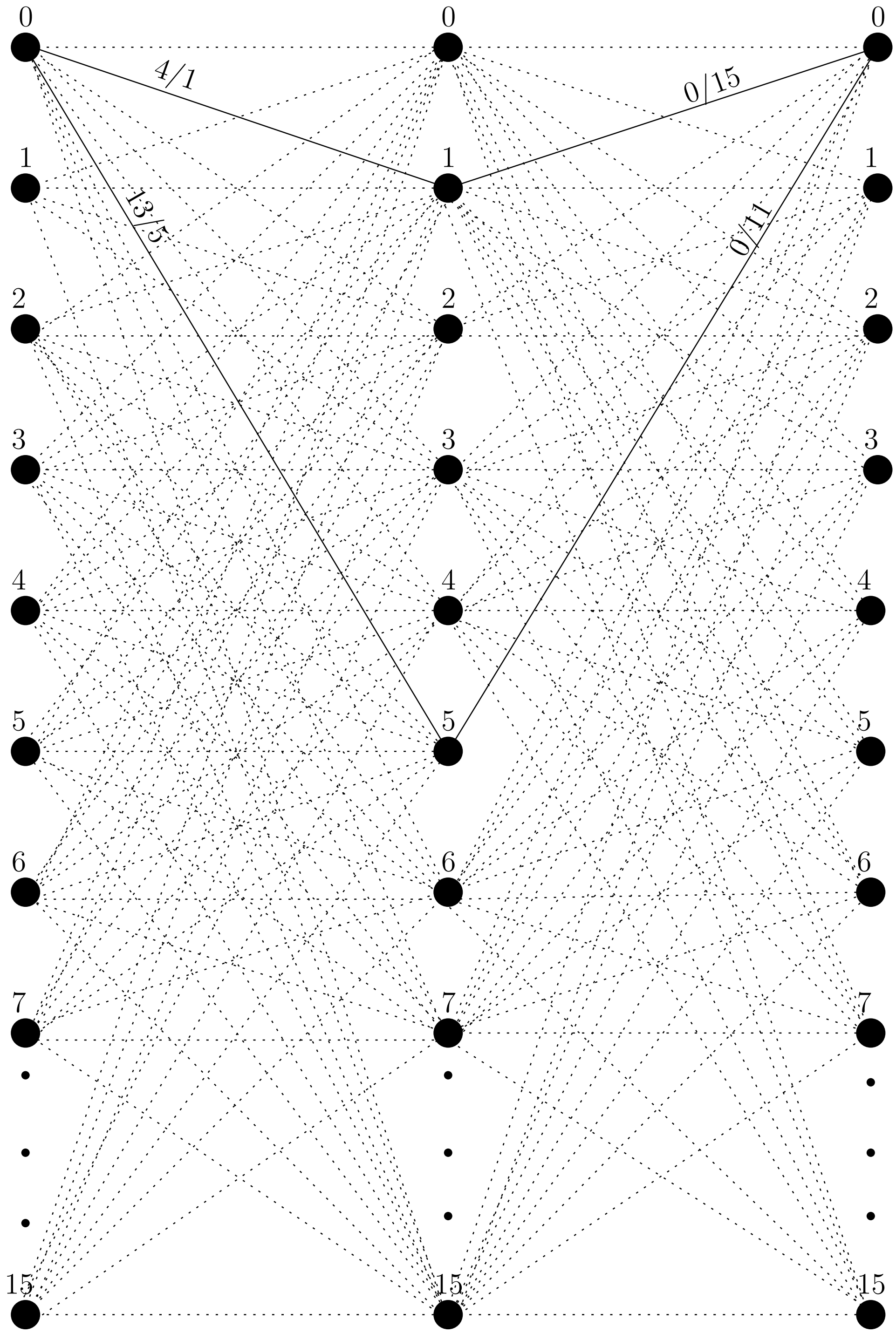


Fig. 2.13: Branches (black solid line) on the trellis that results in $d^2_{\text{free}}$ for SPTC on 16QAM for fixed target point $t = 0$ and shape shown in Fig. 2.1.

## 2.4 Generalization to any M-ary QAM

As the performance of SPTC depends on $d^2_{\text{free}}$, finding the shape and the symbol assignment in the signal constellation corresponding to that shape which has the largest free Euclidean distance will result in maximum possible coding gain. However, for a signal constellation of $M$ points, there are $M!$ possible ways the bits can be assigned to signal constellation. As the shapes go through all of the points in the signal constellation as well as they can not go through each point more than once, there are $M!$ shapes which can be used to encode. Finding the best shape that maximizes the $d^2_{\text{free}}$ for a given bit assignment in the signal constellation is equivalent to finding the arrangement of symbols in the signal constellation that also maximizes $d^2_{\text{free}}$ for a given shape in term of search space. As the $\oplus$ and $\ominus$ operations depend on the employed shape, reduction of search space for finding the larger $d^2_{\text{free}}$ is possible by employing an encoding scheme that depends only on bit assignment rather than shape and then by finding the bit assignment which maximizes the free Euclidean distance for that encoding operation. From Tables 2.1 and 2.2, the output of $\oplus$ and $\ominus$ operations are always an integer ranging from 0 to $M-1$ and all of the integers from 0 to $M-1$ appears exactly once in each row. So it can be concluded that the encoding works as long as the outputs of $\oplus$ and $\ominus$ satisfy:

- For a fixed state variable $j$, the output of the $\oplus$ operation is unique for each unique input symbol index $s$.
- For a fixed target point $t$, the output of the $\ominus$ operation is unique for each unique signal space index $a$.

Using the above-mentioned properties, the encoding was simplified by replacing $\oplus$ and $\ominus$ with following equations:

$$s_i \oplus j = s_i + j \pmod{M} \tag{2.4}$$

$$t_k \ominus a_i = t_k + a_i \pmod{M} \tag{2.5}$$

This modified encoding depends on how the bits are assigned in the signal constellation and doesn't require any shape to encode. To distinguish between this modification and SPTC, this new encoding will be mentioned as Constellation Arithmetic Code (CAC) as all the encoding is computed using arithmetic on the signal constellation. The encoding is given below:

**Algorithm 2** Constellation Arithmetic Code Encoding

```
input to the encoder: symbol S_i , previous state variable j_i
s_i ← ψ(S_i)            compute the integer index of S_i
a_i ← s_i + j_i (mod M)
determine the target point t_k where k = i  (mod u)
r_i ← t_k + a_i (mod M)
j_{i+1} ← j_i + r_i (mod M)            update the state after encoding
R_i ← ψ^{-1}(r_i)            compute the parity symbol R_i from the integer index r_i
return parity symbol R_i and state variable j_{i+1}
```

In (2.5), adding $t = 0$ yields to the same result from (2.4) and using constant target points other than $t = 0$ results in different trellis structure with no significant coding gain. However, as the motivation of CAC is to find the best bit assignment for a fixed trellis, further simplification of Constellation Arithmetic Code has been done for calculating the parity symbol index by excluding (2.5) from the encoding operation of CAC.

**Algorithm 3** Simplified Constellation Arithmetic Code Encoding

```
input to the encoder: symbol S_i , previous state variable j_i
s_i ← ψ(S_i)            compute the integer index of S_i
r_i ← s_i + j_i (mod M)
j_{i+1} ← j_i + r_i (mod M)            update the state after encoding
R_i ← ψ^{-1}(r_i)            compute the parity symbol R_i from the integer index r_i
return parity symbol R_i and state variable j_{i+1}
```

The trellis for CAC using QPSK constellation from Fig. 2.6 is presented in Fig. 2.14.

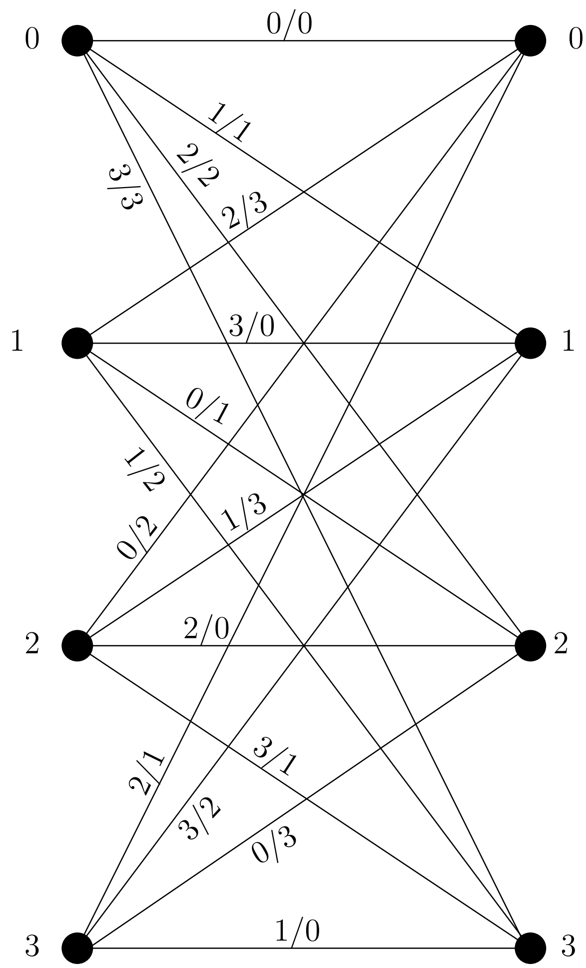


Fig. 2.14: Trellis structure for CAC using QPSK constellation shown in Fig. 2.6.

Now for this trellis and signal constellation the pair of paths on the trellis that has the minimum distance between them is shown in Fig. 2.15 by solid black lines. For this constellation

$$\begin{aligned} d^2_{\text{free}} &= d^2_e(0,2) + d^2_e(0,2) + d^2_e(0,0) + d^2_e(0,2) \\ &= 4E_{c,s} + 4E_{c,s} + 0 + 4E_{c,s} \\ &= 12E_{c,s}. \end{aligned} \tag{2.6}$$

From (2.6), $d^2_{\text{free}}$ results from the distance between symbol 0 and 2 in the signal constellation. Symbol 0 is assigned to point $(-1,-1)$ and symbol 2 is assigned to point $(-1,1)$ and the distance between them is $2\sqrt{E_{c,s}}$ while the distance between the point $(-1,-1)$ and $(1,1)$ is $2\sqrt{2E_{c,s}}$ for average energy per coded symbol $\sqrt{E_{c,s}}$. So it is possible to increase $d^2_{\text{free}}$ by swapping symbol 2 with symbol 3 in the constellation as shown in Fig. 2.16.

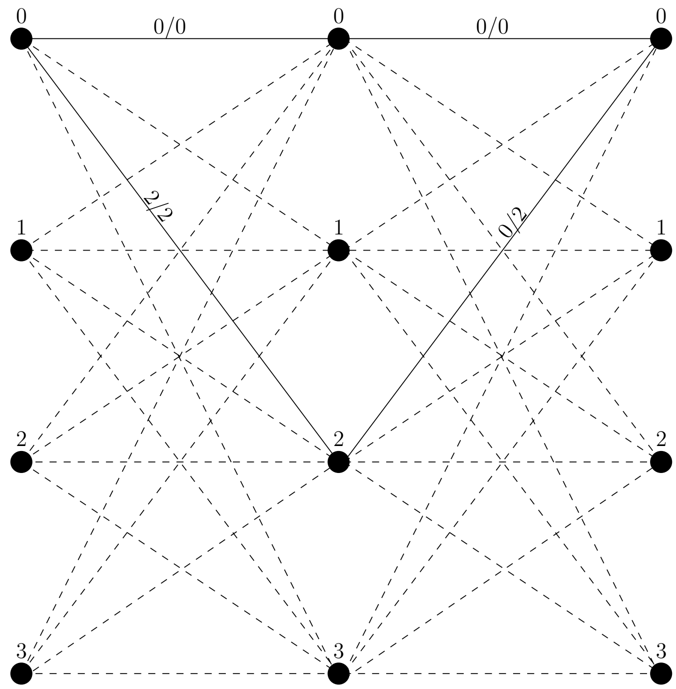


Fig. 2.15: Branches (black solid line) on the trellis that results in $d^2_{\text{free}}$ for CAC on QPSK constellation shown in Fig. 2.6.

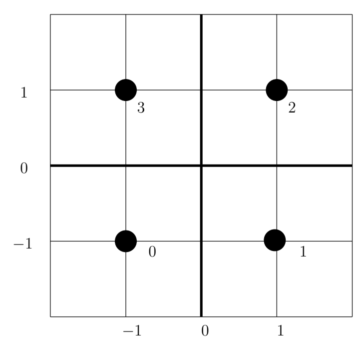


Fig. 2.16: QPSK signal constellation with bit assignment that produces $d^2_{\text{free}} = 20E_{c,s}$.

For the signal constellation shown in Fig. 2.16 the minimum distance between sequences on the trellis is shown in Fig. 2.17 and the free Euclidean distance

$$\begin{aligned} d^2_{\text{free}} &= d^2_e(0,1) + d^2_e(0,1) + d^2_e(0,2) + d^2_e(0,3) \\ &= 4E_{c,s} + 4E_{c,s} + 8E_{c,s} + 4E_{c,s} \\ &= 20E_{c,s} \end{aligned}$$

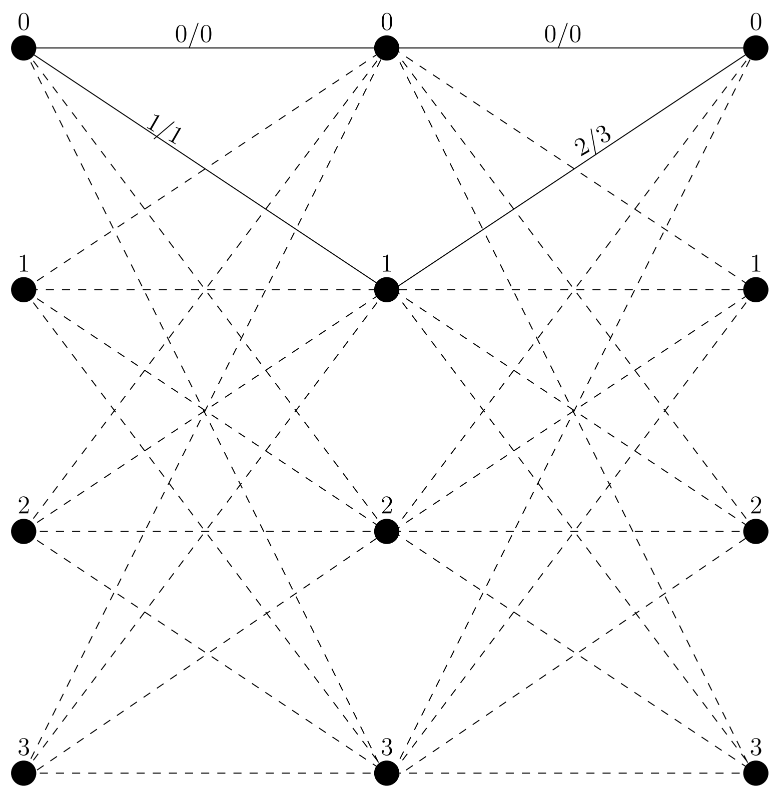


Fig. 2.17: Branches (black solid line) on the trellis that results in $d^2_{\text{free}}$ for CAC on QPSK constellation shown in Fig. 2.16.

Free Euclidean distance corresponding to CAC on 16QAM using the bit assignment shown in Fig. 2.1 is $d^2_{\text{free}} = 12E_{c,s}$. For this bit assignment, the pair of paths that produces the free Euclidean distance on the trellis is shown in Fig. 2.18 by black solid lines.

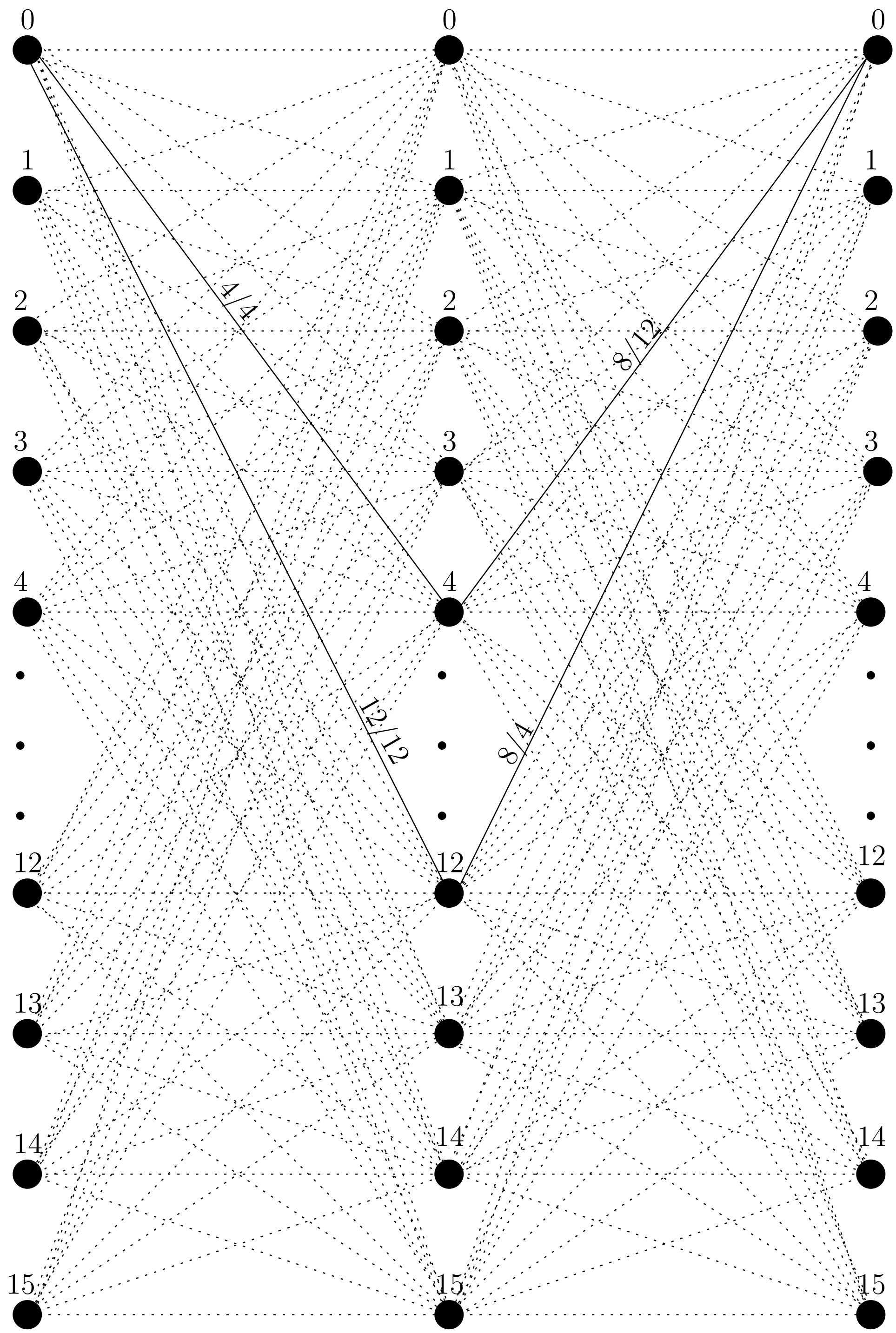


Fig. 2.18: Branches (black solid line) on the trellis that results in $d^2_{\text{free}}$ for CAC on 16QAM using constellation shown in Fig. 2.1.

In order to show the relation between free Euclidean distance and coding gain, a bit assignment is shown in Fig. 2.19. This bit assignment produces $d^2_{\text{free}} = 20E_{c,s}$ which is larger than the free Euclidean distance results from using gray code indexing as shown in Fig. 2.1. There are 16! possible bit assignments in the 16QAM constellation. So finding the bit assignment which has the largest $d^2_{\text{free}}$ among those 16! bit assignments are computationally expensive. In order to find bit assignments in the 16QAM constellation with larger $d^2_{\text{free}}$, 4 symbols are selected arbitrarily and they have been fixed at points $(-3,-3)$, $(3,-3)$, $(-3,3)$ and $(3,3)$ respectively in order to reduce the search space from 16! to 12! and find a bit assignment with larger $d^2_{\text{free}}$ rather than finding the largest $d^2_{\text{free}}$. Then an exhaustive search has been made using the other 12 symbols and points in the signal constellation to find possible bit assignment with largest $d^2_{\text{free}}$ from all possible 12! symbol assignments. By randomly selecting those 4 symbols, it has been found that by selecting symbols 0, 3, 12 and 15 and fixing those symbols to points $(-3,-3)$, $(3,-3)$, $(-3,3)$ and $(3,3)$, it is possible to produce two bit assignments with $d^2_{\text{free}} = 28E_{c,s}$. Those two bit assignments are shown in Fig. 2.20 and 2.21. For 2.20, the pair of paths that produces the free Euclidean distance on the trellis is shown in Fig. 2.22 by black solid lines. The simulated results for these bit assignments with $d^2_{\text{free}} = 12E_{c,s}$, $d^2_{\text{free}} = 20E_{c,s}$ and $d^2_{\text{free}} = 28E_{c,s}$ are presented in Figs. 3.7, 3.8 and 3.9 respectively. From these results, it can be seen that it is possible to increase coding gain for CAC by increasing $d^2_{\text{free}}$.

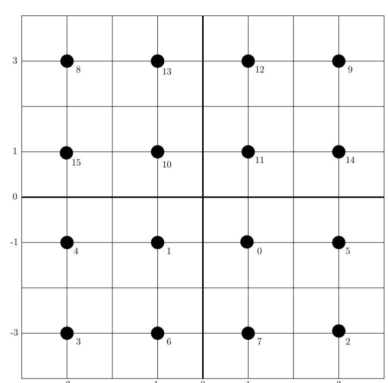


Fig. 2.19: 16-QAM signal constellation with bit assignment that produces $d^2_{\text{free}} = 20E_{c,s}$.

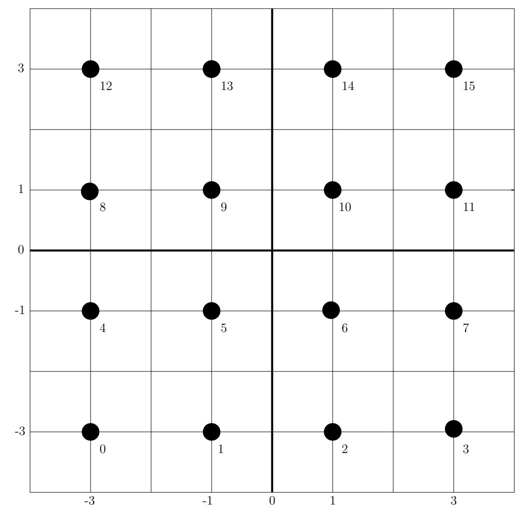


Fig. 2.20: 16-QAM signal constellation with bit assignment that produces $d_{\text{free}}^2 = 28E_{c,s}$.

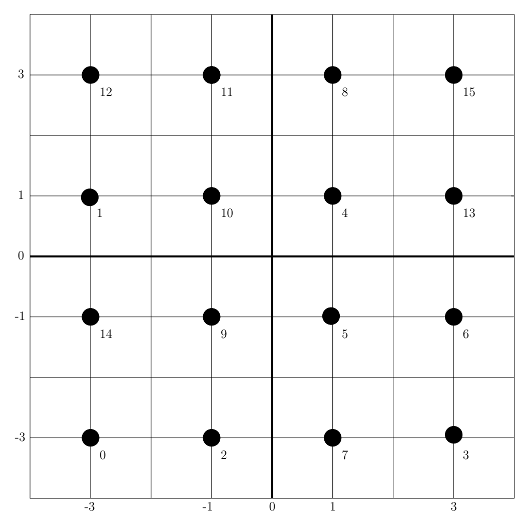


Fig. 2.21: 16-QAM signal constellation with bit assignment that produces $d_{\text{free}}^2 = 28E_{c,s}$.

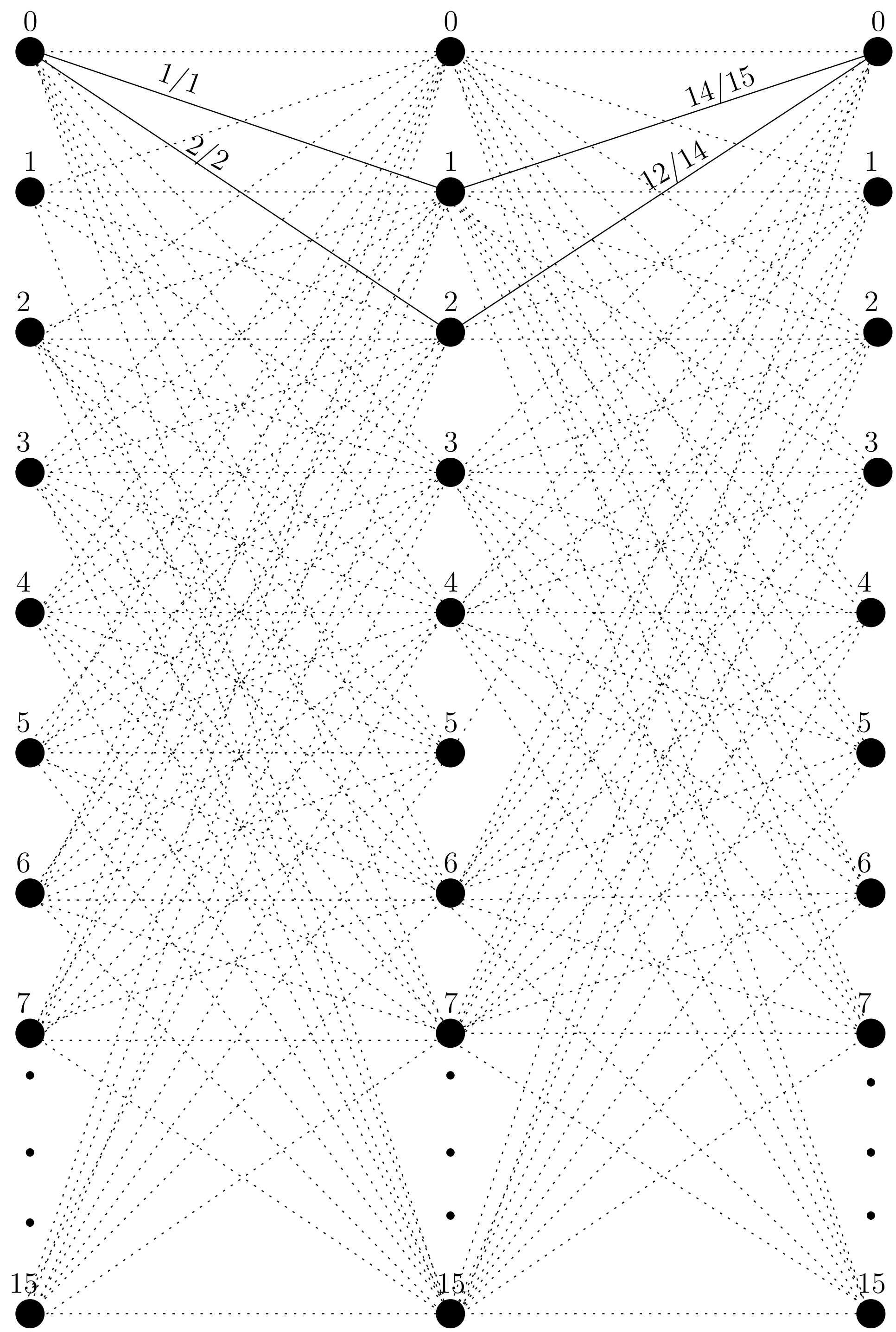


Fig. 2.22: Branches (black solid line) on the trellis that results in $d^2_{\text{free}}$ for CAC on 16QAM using bit assignment shown in Fig. 2.20.

In Fig. 2.20, the bits are assigned in the signal constellation as follows: first, the combination of bits that are to be assigned are represented in decimal integer values. Then starting from the bottom left point in the signal constellation and moving towards right those integer representation of bits are assigned to the points in ascending order. When all the points of that row have their assigned bits, the immediate upper row in the signal constellation is selected for bit assignment. This operation is performed until all the points in the signal constellation have their assigned bits. This method of bits assignment using CAC provides the maximum coding gain compared to all of the simulated performance in this thesis for the 16QAM constellation and a coding gain for CAC on 64QAM compared to SPTC. It is denoted as “Sequential Bits Assignment”. Sequential Bit Assignment on a 64QAM constellation shown in Fig. 2.23, produces $d_{\text{free}}^2 = 28E_{c,s}$ for CAC.

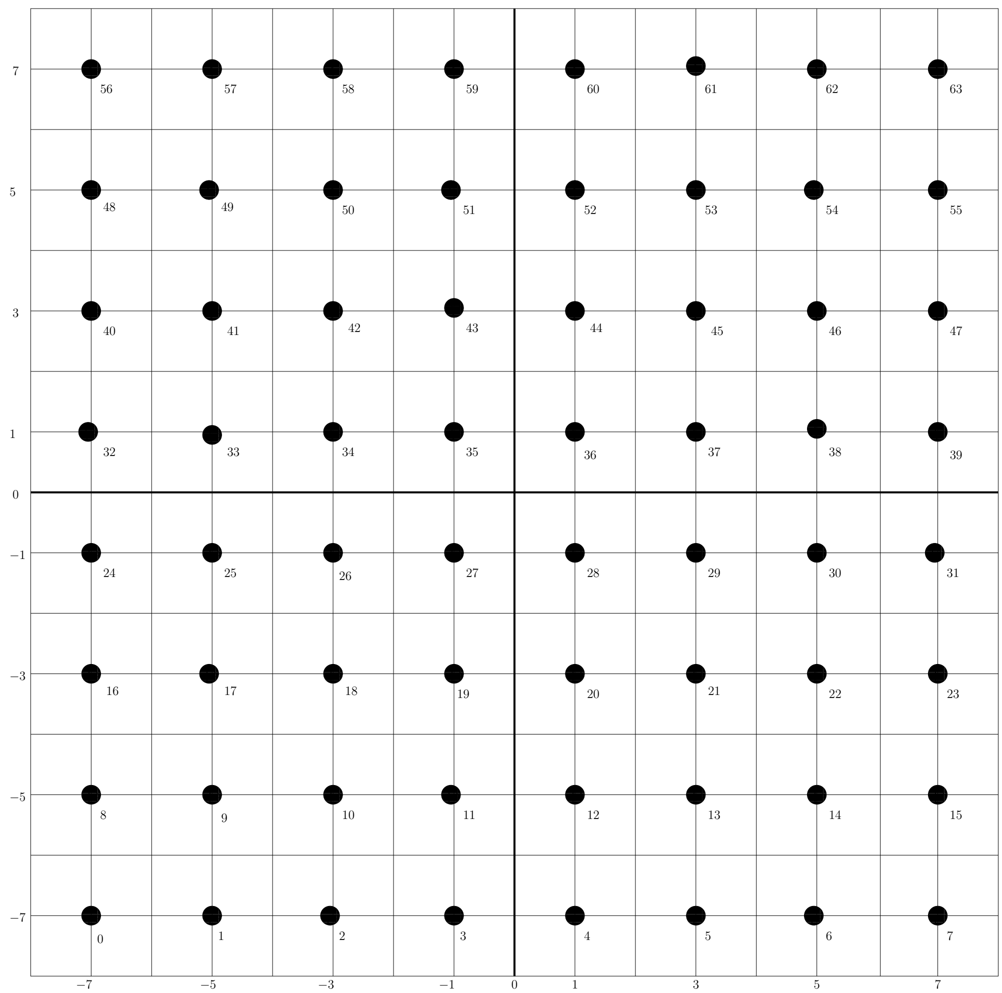


Fig. 2.23: 64-QAM signal constellation with Sequential Bits Assignment that produces $d^2_{\text{free}} = 28E_{c,s}$.

Now, as the coding gain depends on $d^2_{\text{free}}$, the maximum coding gain for a given free Euclidean distance is bounded by the asymptotic coding gain [9] which is calculated as follows:

$$\gamma = \left(\frac{E_{s,m}}{E_{c,s}}\right)\left(\frac{d^2_{\text{free}}}{d^2_{\text{min,uncoded}}}\right). \tag{2.7}$$

Here $d^2_{\text{free}}$ is the free Euclidean distance and $d^2_{\text{min,uncoded}}$ is the smallest distance between any two points in the constellation. $E_{s,m}$ is the energy per uncoded symbol and $E_{c,s}$ is the total energy required to send coded symbols. For code with rate $R$ and a given constellation, $1/R$ coded symbols are transmitted for every message symbol which requires $1/R$ times more energy than for an uncoded symbol transmitted using that constellation. Therefore, $\frac{E_{s,m}}{E_{c,s}} = R$. For the constellations used in this thesis, $d^2_{\text{min,uncoded}}$ is always $4E_{c,s}$. Using the free Euclidean distances obtained for different shapes and bit assignments, the asymptotic coding gains for different codes along with the asymptotic coding gains for larger states (which will be introduced in Chapter 4) have been listed in Table 4.1. This list provides a theoretical upper bounds for coding gains.

## 2.5 Encoding of Rate $\frac{k}{k+1}$ Code

Initially, the proposed encoders of both SPTC and CAC are of rate $\dfrac{1}{2}$ because for each input symbol the encoder generates one parity symbol. Though lower rate increases the error correction capability of codes, it decreases the information transmission rate per coded symbol. Therefore increasing the rate of codes while maintaining a good coding gain is of significant interest in error correction coding. In this section, two methods for changing rate from $\dfrac{1}{2}$ to $\frac{k}{k+1}$ using Constellation Arithmetic Code have been proposed. In the encoding algorithms shown in Figs. 2.24 and 2.25, all the inputs $s$ and outputs $r$ are integer indexes of the input symbol $S$ and the output parity symbol $R$ and the $\sum$ represents $a + b$ $(mod$ $M)$. For this section, at every time $i$, $k$ symbols are to encoded to generate a parity symbol $R_i$ which makes the proposed encodes of rate $\frac{k}{k+1}$. The encoding methods are described below.

### 2.5.1 Sequential Input Encoding

In this encoding scheme, only one target point is used to calculate the parity symbol index. The motivation for this encoding is to first encode all of the input symbols sequentially and then generate a parity symbol using only one target point and the output obtained from the sequential encoding of the input symbols. As this method encodes the input symbols sequentially to generate a parity symbol, it is called Sequential Input Encoding (SIE). To encode, each of $k$ input symbol is mapped to their corresponding signal space index. Then among the $k$ signal space indexes, the signal space index corresponding to the first input symbol is added with the state variable $j_i$ using (2.4) to move to a new signal space index $a$ corresponding to a point of the signal constellation. Then for the rest of $k-1$ symbols, this $a$ is updated by adding previous $a$ with the next symbol's signal space index using (2.4). Finally this updated signal space index $a$ is added with the target symbol's signal space index using (2.5) to get the signal space index $r$ of the parity symbol $R$. This encoding process is shown in Fig. 2.24. This encoding is also presented in algorithm 4.

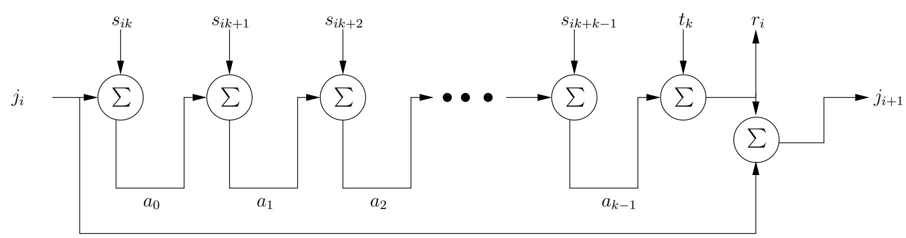


Fig. 2.24: Schematic diagram for rate $\frac{k}{k+1}$ using Sequential Input Encoding (SIE).

---

**Algorithm 4** Rate k/k+1: Sequential Input Encoding (SIE)

---

input to the encoder: symbol $S_{ik}, S_{ik+1}, ..., S_{ik+k-1}$ , previous state variable $j_i$

$s_{ik} \leftarrow \psi(S_{ik})$ compute the integer index of $S_{ik}$

$a_0 \leftarrow s_{ik} + j_i \pmod M$

**for** $l = 1 : k-1$ **do**

    $s_{ik+l} \leftarrow \psi(S_{ik+l})$ compute the integer index of $S_{ik+l}$

    $a_l \leftarrow s_{ik+l} + a_{l-1} \pmod M$

**end for**

determine the target point $t_k$ where $k = i \pmod u$

$r_i \leftarrow t_k + a_{k-1} \pmod M$

$R_i \leftarrow \psi^{-1}(r_i)$ compute the parity symbol $R_i$ from the integer index $r_i$

$j_{i+1} \leftarrow j_i + r_i \pmod M$ update the state after encoding

return parity symbol $R_i$ and state variable $j_{i+1}$

---

### 2.5.2 Partial Parallel Encoding

Presented here is another method for changing the rate of CAC. In this method, $k$ input symbols are encoded individually using $k$ target points to generate the parity symbol. All of those $k$ target points might have signal space index of same values or might have different values. For this thesis, while simulating results using this method, target point with 0 signal space index has been selected for all $k$ of those target points. In this encoding, there is a signal space index defined as an intermediate state variable $q$ that acts as a state variable during each encoding of symbol and is used to update the state variable $j$ after generating the parity symbol $R$. Using target points and $q$, all of the $k$ input symbols are encoded individually to generate total $k$ different signal space index $g$. Initially, the $q$ is equal to the input state variable $j_i$ which gets updated after encoding of each symbol and acts as an input state variable for the next symbol. Those $k$ signal space indexes $g$, generated from each individual encoding of input symbols can be viewed as temporary parity symbol indexes for $k$ input symbols which are added together using $a + b \pmod M$ to generate the parity symbol index $r$. This encoding algorithm is called Partial Parallel Encoding (PPE) because all of the input symbols are encoded in a partially parallel method. The schematic diagram for this encoding is shown in Fig. 2.25 and also presented in algorithm 5.

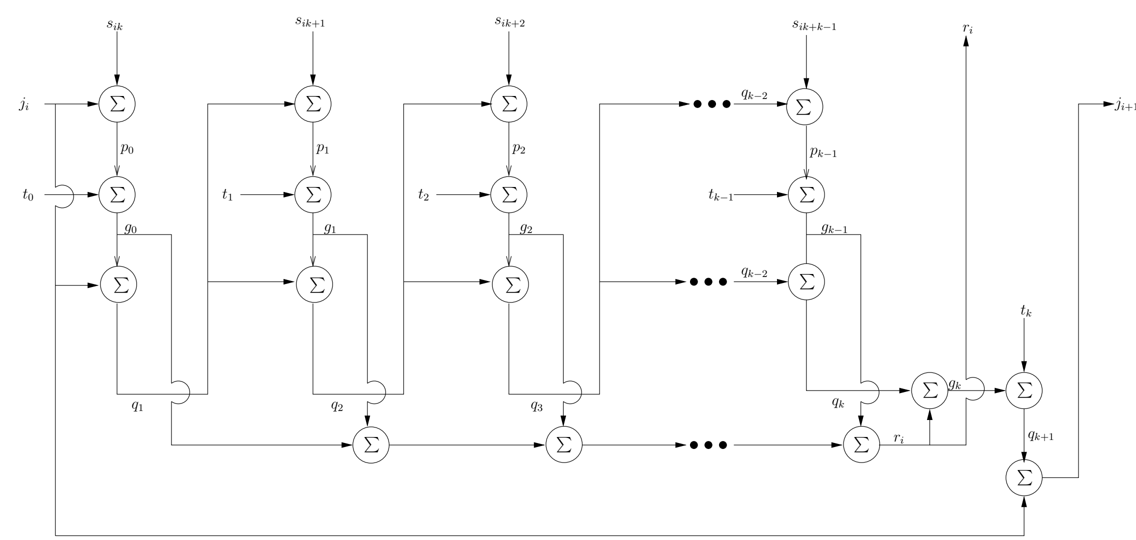


Fig. 2.25: Schematic diagram for rate $\frac{k}{k+1}$ using Partial Parallel Encoding (PPE).

**Algorithm 5** Rate k/k+1: Partial Parallel Encoding (PPE)

input to the encoder: symbol $S_{ik}, S_{ik+1}, ..., S_{ik+k-1}$ , previous state variable $j_i$

$q_0 \leftarrow j_i$

**for** $l = 0 : k-1$ **do**

  $s_{ik+l} \leftarrow \psi(S_{ik+l})$

  $p_l \leftarrow s_{ik+l} + q_l \pmod{M}$

  determine the target point $t_c$ where $c = l \pmod{u}$

  $g_l \leftarrow p_l + t_c \pmod{M}$

  $q_{l+1} = g_l + q_l \pmod{M}$

**end for**

$r_i \leftarrow \sum_{l=0}^{k-1} g_l \pmod{M}$

$R_i \leftarrow \psi^{-1}(r_i)$

$g_k \leftarrow r_i + q_k \pmod{M}$

$q_{k+1} \leftarrow g_k + t_k \pmod{M}$

$j_{i+1} \leftarrow q_{k+1} + j_i \pmod{M}$

return parity symbol $R_i$ and state variable $j_{i+1}$

### 2.5.3 Decoding a k/(k+1) Rate Code

For codes with a rate higher than 1/2, the trellis contains parallel branches from state $j_i$ to state $j_{i+1}$ at each $i$. At every $i$, $k$ symbols are encoded to generate a parity symbol $R_i$. Each of the branches of the trellis has $k$ input symbols and one parity symbol. A trellis for SIE rate 2/3 CAC using QPSK constellation is shown in Fig. 2.26. In Fig. 2.26, all of the branches are indicated by black dashed lines and from each of the initial state to each of the final state, there are 4 parallel branches. As this trellis is for rate 2/3 code each branch has two input symbols and one parity symbols and the symbols associated with each branch are labeled as $S_0, S_1/R$, located in the left of each state where $S_0$ is the first input symbol, $S_1$ is the second input symbol and $R$ is the parity symbol. In Fig. 2.26, at the left of each initial state there are four horizontal lines which contain all the input and parity symbols associated with the branches that originated from that state. For each of the collection of such four lines, by using zero-based indexing, the order of the lines represents the branch's final state. For example, among the four lines located on the left of state 0, line 0 represents all of the parallel branches that end at state 0 starting from state 0. In each of those lines, there are four triplets of symbols $(S_0, S_1/R)$, each associated with a branch. The first symbol triplet represents the top branch among all of the parallel branches that started from the same state and ended at the same state, similarly the second symbol triplet represents the second topmost branch and so on. For example from line 0 located on the left of initial state 0, the third symbol triplet $2, 0/0$ (surrounded by an oval box) is associated with the third topmost branch (indicated by solid black arrow) that starts from start 0 and ends at state 0. Now in order to decode, the Viterbi algorithm searches through all of the branches including the parallel branches at each time $i$ and calculates the branch metric $\mu_t$ using

$$\mu_t = d_e^2(S_{ik}, \hat{S}_{ik}) + d_e^2(S_{ik+1}, \hat{S}_{ik+1}) + ..... + d_e^2(S_{ik+k-1}, \hat{S}_{ik+k-1}) + d_e^2(R_i, \hat{R}_i) \tag{2.8}$$

and selects the branch symbols associated with the path with minimum path metric.

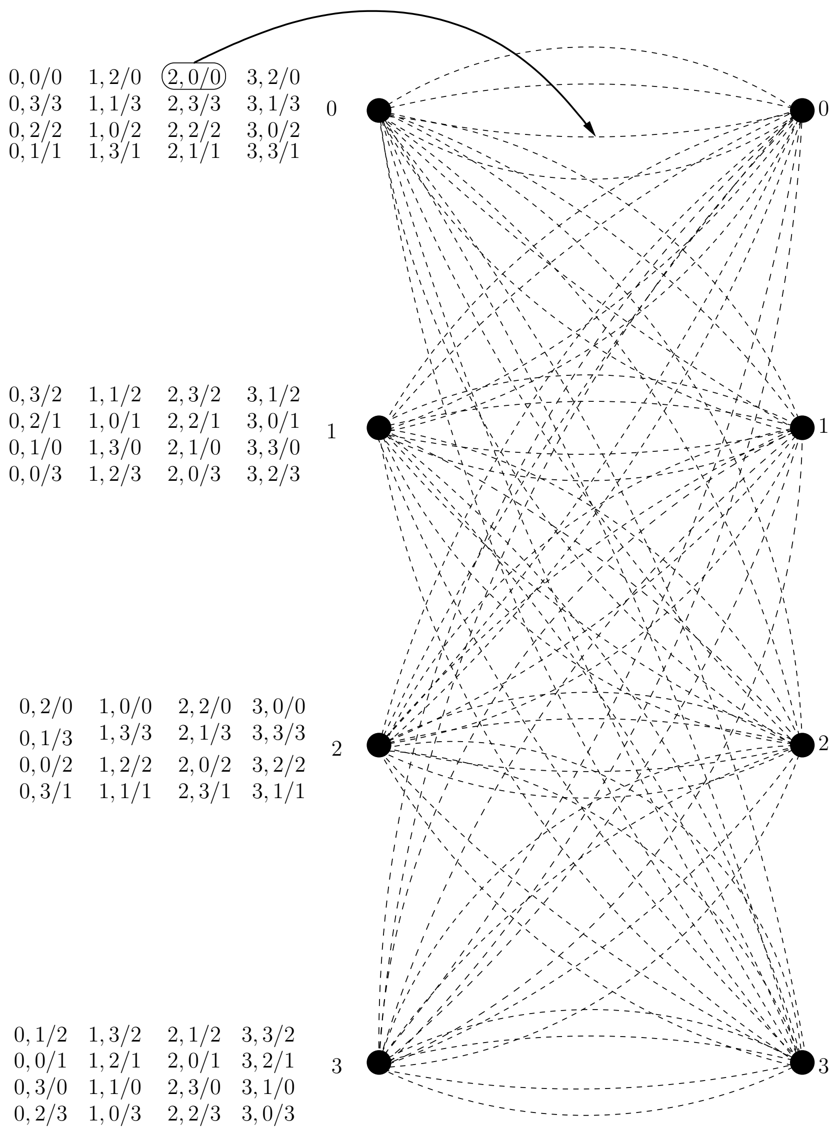


Fig. 2.26: Trellis structure for rate 2/3 CAC (SIE) using QPSK constellation.

# CHAPTER 3

# SIMULATION AND RESULTS

In this chapter, the required calculations for simulating the Signal Point Target Code and Constellation Arithmetic Code as well as the simulation results for both of the proposed codes are provided.

## 3.1 Calculation of Energy Per Bit for Simulation

In the simulated performance of an error correction coding system, the horizontal axis of the performance plot is usually the signal-to-noise ratio, $E_b/N_0$ expressed in dB, where $E_b$ is the energy per transmitted message bit (that is, a bit that actually conveys information, as opposed to merely a bit in a codeword), and $N_0$ is the noise power spectral density. Since $E_b/N_0$ is expressed as a ratio, either $E_b$ or $N_0$ may be fixed, and other quantity determined from the ratio. In simulations presented here, the amplitude $A$ of the signal constellation is fixed. For that signal constellation, $E_b$ is calculated and $N_0$ is determined from that $E_b$. Simulated additive white Gaussian noise is produced with the variance

$$\sigma^2 = \frac{N_0}{2}. \tag{3.1}$$

This section describes the relationship between the various quantities.
First, the BPSK modulation, with transmitted symbol amplitude fixed at $\pm A$ is considered. These amplitudes are used to transmit all the bits, that is, the coded bits. Let the energy associated with this transmission be denoted as $E_c$, the energy per coded bit. Then

$$E_c = A^2 \quad \text{Joules/coded bit.}$$

For a given constellation, $E_c$ is fixed in the simulation. In a coded system, the rate $R$ is the ratio

$$R = \frac{\text{number of message bits}}{\text{number coded bits}}.$$

$R$ is always less than 1.

The energy per message bit, $E_b$ is thus

$$E_b \, \frac{\text{Joules}}{\text{message bits}} = E_c \, \frac{\text{Joules}}{\text{coded bits}} \, \frac{1}{R} \, \frac{\text{number of coded bits}}{\text{number of message bits}}.$$

$$E_b = E_c/R.$$

By looking at from another point of view, to transmit coded bits using the same energy that has been allocated for uncoded message bits, the energy per message bit $E_b$ must be distributed among the coded bits, so

$$E_c = RE_b.$$

In a simulation, given an SNR $\gamma = (E_b/N_0)_{dB}$, solve for $N_0$ as

$$N_0 = \frac{E_b}{10^{\gamma/10}}. \tag{3.2}$$

from which the noise variance is computed using (3.1). For higher-order modulation with M symbols, let $n = log_2 M$ be the number of bits per symbol. Let $E_{c,s}$ denote the fixed energy per symbol in the constellation (carrying coded symbols). For QAM with the square constellations, where the points in the in-phase axis and quadrature axis are at $\pm A, \pm 3A, ...., \pm(M-1)A$, the energy per coded symbol is

$$E_{c,s} = \frac{2}{3}\,(M-1)A^2 \, \frac{\text{Joules}}{\text{coded symbol}}.$$

This energy is fixed by the constellation employed. So the energy per uncoded message symbol $E_{s,m}$ is

$$E_{s,m} = E_{c,s}/R.$$

Since each symbol carries $n$ bits of information, the energy per bit is

$$\begin{aligned} E_b &= E_{s,m} \frac{\text{Joules}}{\text{uncoded message symbol}} \frac{\text{1 symbol}}{\text{n bits}} \\ &= \frac{E_{s,m}}{n} \frac{\text{Joules}}{\text{message bits}} \\ &= \frac{E_{c,s}}{nR}. \end{aligned}$$

Using the energy per bit for coded symbol $E_b$, $\sigma^2$ and $N_0$ are calculated from (3.1) and (3.2) for a given SNR $\gamma$. For $M = 16$, $n = 4$ , $R = \frac{1}{2}$ , let $A = 1$ . So $E_b = \frac{2\times 15}{3\times 4\times .5} = 5$. Using $E_b$, $\sigma$ is calculated for SNR $\gamma$ and then $N \sim \mathcal{N}(0, \sigma^2)$ is generated where $N$ is the noise added with the transmitted signal at that SNR $\gamma$.

The received signal is

$$\hat{S}_i = S_i + N$$

$$\hat{R}_i = R_i + N,$$

where $R_i$ =transmitted parity symbol, $S_i$ =transmitted message symbol. Received symbols $\hat{S}_i$ and $\hat{R}_i$ are used as input of Viterbi decoder to get decoded message symbol $\tilde{S}_i$. If $\tilde{S}_i \neq S_i$ then an error has occurred. Let $z_{\gamma,i}$ is a Bernoulli random variable with probability $p_\gamma$, where $p_\gamma$ is the probability of symbol error at SNR $\gamma$. So

$$z_{\gamma,i} = \begin{cases} 1 & \text{if an error occurs in the decoder} \\ 0 & \text{if there is no error.} \end{cases}$$

Let $e_\gamma$ be the total number of errors occurred for SNR $\gamma$ and $N_{s,\gamma}$ be the total number of symbols sent through the channel at that SNR $\gamma$. As so $e_\gamma = \sum_{i=1}^{N_{s,\gamma}} z_{\gamma,i}$, $e_\gamma$ is binomially distributed $\sim B(N_{s,\gamma}, p_\gamma)$ with $\mathrm{E}[e_\gamma] = N_{s,\gamma} p_\gamma$ and $\mathrm{var}(e_\gamma) = N_{s,\gamma} p_\gamma (1 - p_\gamma)$. The likelihood function for $e_\gamma$

$$f_{e_\gamma}(e_\gamma|p_\gamma) = \binom{N_{s,\gamma}}{e_\gamma} p_\gamma^{e_\gamma}(1-p_\gamma)^{(N_{s,\gamma}-e_\gamma)}.$$

The log likelihood function is

$$\begin{aligned}\Lambda(e_\gamma, p_\gamma) &= \log f_{e_\gamma}(e_\gamma|p_\gamma)\\ &= \log \binom{N_{s,\gamma}}{e_\gamma} + e_\gamma \log p_\gamma + (N_{s,\gamma}-e_\gamma)\log(1-p_\gamma).\end{aligned}$$

Now by taking gradient with respect to $p_\gamma$ and equating to zero, maximum likelihood estimation of the probability of symbol error $\hat{p}_\gamma$ at SNR $\gamma$ is obtained.

$$\frac{\partial \Lambda(e_\gamma, p_\gamma)}{\partial p_\gamma} = 0$$

$$\frac{e_\gamma}{p_\gamma} - \frac{(N_{s,\gamma}-e_\gamma)}{1-p_\gamma} = 0$$

$$\hat{p}_\gamma = \frac{e_\gamma}{N_{s,\gamma}} \tag{3.3}$$

The variance of $\hat{p}_\gamma$ is

$$\begin{aligned}\mathrm{var}(\hat{p}_\gamma) &= \mathrm{var}(\frac{e_\gamma}{N_{s,\gamma}})\\ &= \frac{1}{N_{s,\gamma}^2}\ N_{s,\gamma} p_\gamma(1-p_\gamma)\\ &= \frac{p_\gamma(1-p_\gamma)}{N_{s,\gamma}}\\ &= \frac{\hat{p}_\gamma(1-\hat{p}_\gamma)}{N_{s,\gamma}} \qquad \text{using the estimated } \hat{p}_\gamma\\ &= \frac{\frac{e_\gamma}{N_{s,\gamma}}(1-\frac{e_\gamma}{N_{s,\gamma}})}{N_{s,\gamma}}\\ &= \frac{e_\gamma N_{s,\gamma} - e_\gamma^2}{N_{s,\gamma}^3}.\end{aligned}$$

So the standard deviation is

$$\begin{aligned}\sigma_{\hat{p}_\gamma} &= \sqrt{\text{var}(\hat{p}_\gamma)} \\ &= \sqrt{\frac{e_\gamma N_{s,\gamma} - e_\gamma^2}{N_{s,\gamma}^3}}.\end{aligned} \tag{3.4}$$

For simulation, $N_{s,\gamma}$ is counted until $e_\gamma = 100$. Using $N_{s,\gamma}$ and $e_\gamma = 100$ probability of symbol error is calculated from (3.3). The bit error rate of coded symbols is calculated by taking the ratio of the total number of erroneous bits and the total number of transmitted bits. The total number of transmitted bits is calculated from $N_{s,\gamma}$ and the total number of erroneous bits is calculated by counting how many bits vary between $\tilde{S}_i$ and $S_i$ when $\tilde{S}_i \neq S_i$. In this thesis, the level of confidence for calculating $\hat{p}_\gamma$ is considered as inversely proportional to standard deviation of $\hat{p}_\gamma$. The smaller the $\hat{p}_\gamma$ is, the higher the level of confidence is. For $e_\gamma = 100$, at each SNR $\gamma$, the standard deviation of $\hat{p}_\gamma$ is calculated from (3.4). In Figs. 3.1 and 3.3, the error bars represent one standard deviation above and below from the estimated probability of symbol errors. As the error bars for each $\hat{p}_\gamma$ is very small, the confidence level is high. So the error bars are not presented in the rest of the BER and SER plots for this thesis.

In this thesis, the performances of codes are evaluated as BER and SER for constellations of different sizes. Gray code indexing, as well as other bit assignments in the signal constellation, have been used to calculate $d^2_{\text{free}}$ and simulate their coding gain. As there is no theoretical bound for bit error rate if the bits are not assigned using gray code indexing so to evaluate BER of proposed encoding schemes, first theoretical SER for uncoded symbols are calculated using (3.5) and (3.6). The plots of those theoretical SERs have been labeled by "th" with their corresponding signal constellation name. For example, in Fig. 3.1 the blue dotted line labeled by "QPSK(th) SER" represents the theoretical symbol error rate of QPSK constellation. The simulated SER and SER are calculated by sending uncoded symbols over an AWGN channel with SNR $\gamma$ and by counting the total number of erroneous bits and symbols after decoding the received symbols. The plots of such SERs and BERs, obtained from simulations are labeled by "ex" with their corresponding signal constellation name. For example, in Fig. 3.1, the red solid line labeled by "QPSK(ex) SER" represents the SER of uncoded symbols using QPSK constellation and the red dotted line labeled by "QPSK(ex) BER" represents BER of uncoded symbols using QPSK constellation. In these labels, "(th)" represents theoretical result and "(ex)" represents experimental results. After calculating SER for uncoded symbols from simulations, the results have been compared with theoretical SER for uncoded symbols of the same constellation size to make sure the simulations are working properly. If the simulated SER overlays with theoretical SER then the BER obtained from that simulation has been used to compare with the BER of coded symbols. The theoretical BER for BPSK [9] is calculated using

$$P_e = Q\left(\sqrt{\frac{2E_b}{N_0}}\right), \tag{3.5}$$

where $Q(.)$ is standard Q function defined in A. The theoretical SER [10] for constellation larger than BPSK is calculated using

$$P_e = 1 - \left(1 - 2\left(1 - \frac{1}{\sqrt{M}}\right)Q\left(\sqrt{\frac{3nE_b}{(M-1)N_0}}\right)\right)^2. \tag{3.6}$$

### 3.2 Performance of SPTC and CAC for Rate 1/2 Codes

In this section, the simulation results for both rate 1/2 SPTC and rate 1/2 CAC are presented. For SPTC different shapes and constellations are used to generate BER and SER while for CAC different constellations and bit assignments are used. The performances of both codes are evaluated using BER and SER obtained from simulations.

#### 3.2.1 Performance of SPTC

From Fig. 3.1, using the shape shown in Fig. 2.6 and QPSK constellation with gray code indexing provides approximately 2.25 dB coding gain for both SER and BER compared to uncoded QPSK at $P_s = 10^{-5}$. The shape shown in Fig. 2.6, has $d^2_{\text{free}} = 16E_{c,s}$.

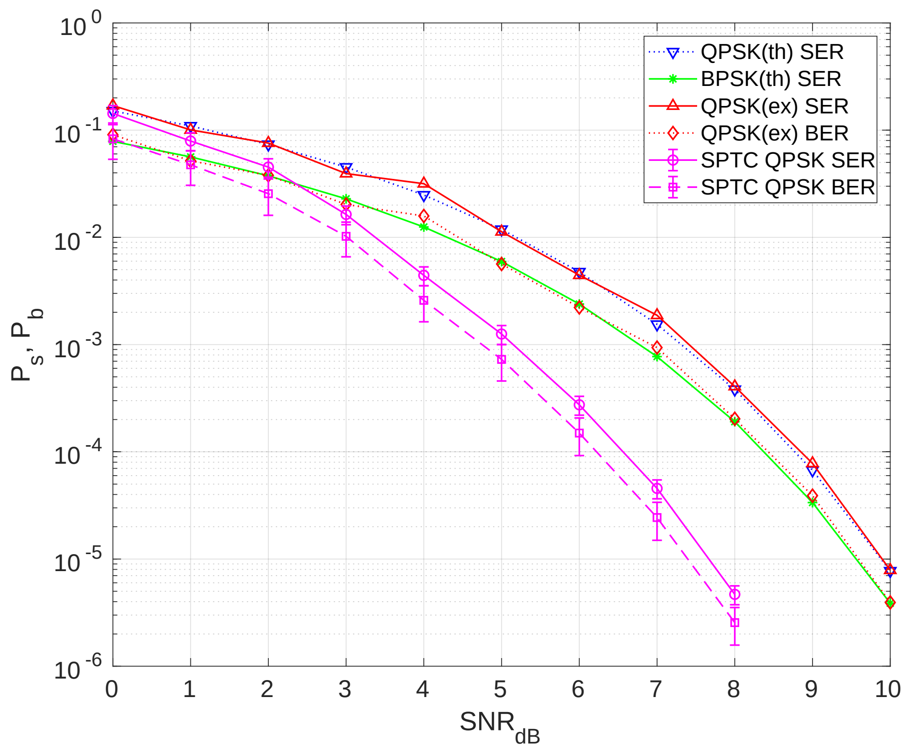


Fig. 3.1: BER and SER Curve for rate $\frac{1}{2}$ SPTC with target point $t = 0$ on a QPSK constellation for the shape shown in Fig. 2.6.

The performance of SPTC using the shape from Fig. 2.10 which has $d^2_{\text{free}} = 28E_{c,s}$ and QPSK constellation with gray code indexing, is presented in Fig. 3.2. As this shape has a larger $d^2_{\text{free}}$ than the shape shown in Fig. 2.6, at $P_s = 10^{-5}$ it provides a coding gain of 3 dB compared to uncoded QPSK and a coding gain of 0.75 dB compared to the performance of the shape shown in Fig. 2.6.

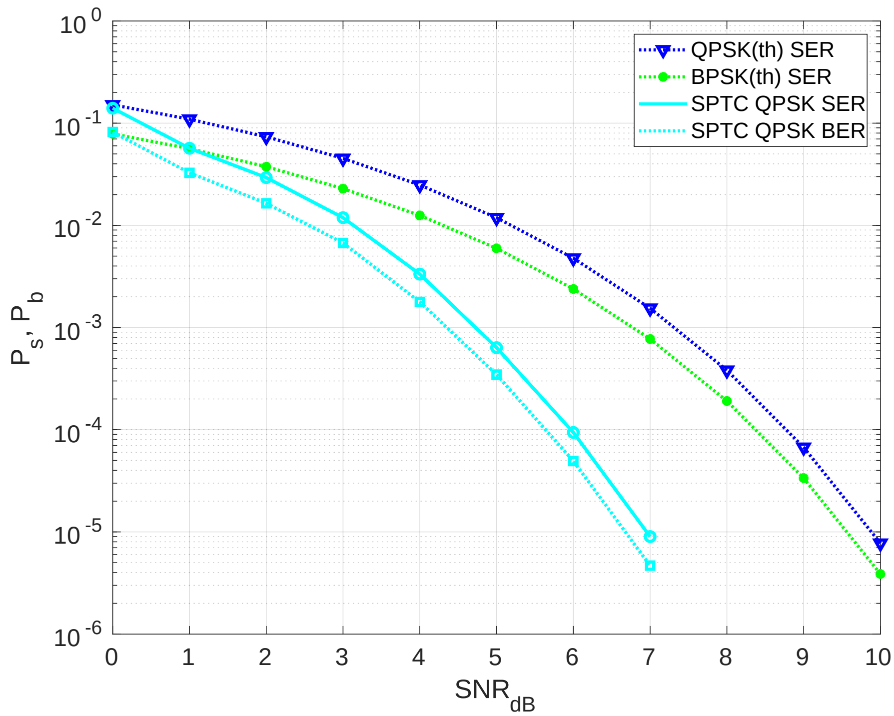


Fig. 3.2: BER and SER Curve for rate $\frac{1}{2}$ SPTC with target point $t = 0$ on a QPSK constellation for shape shown in Fig. 2.10.

To simulate the performance of SPTC on 16QAM constellation with gray code indexing, the shape shown in Fig. 2.1 has been used and the performance is presented in Fig. 3.3. The shape used for this simulation has $d_{\text{free}}^2 = 16E_{c,s}$. At $P_s = 10^{-4}$, this code has a coding gain of 2.75 dB compared to the performance of uncoded 16QAM constellation.

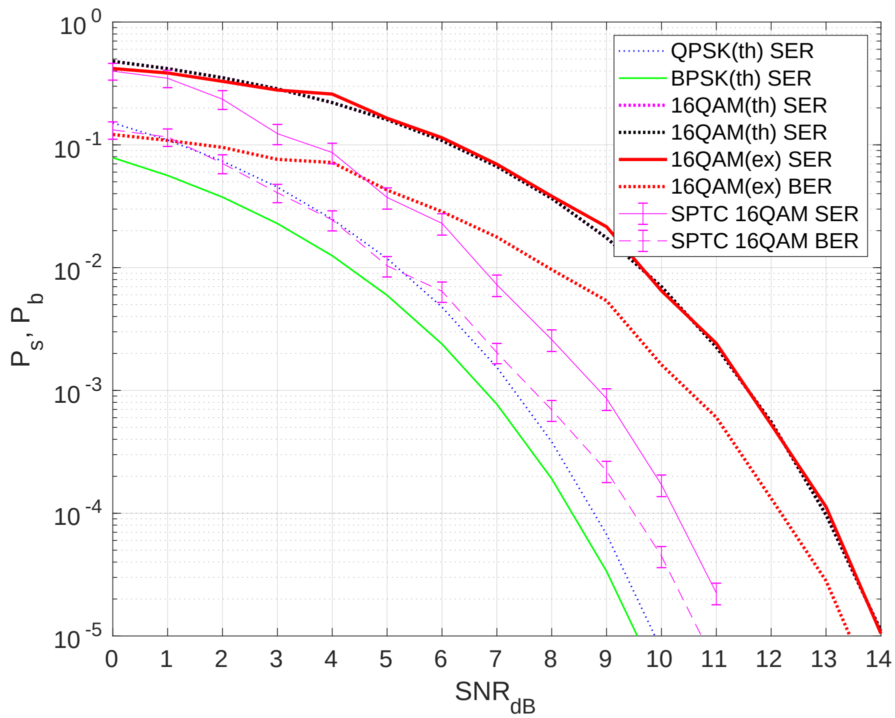


Fig. 3.3: BER and SER Curve for rate $\frac{1}{2}$ SPTC with target point $t = 0$ on a 16QAM constellation for shape shown in Fig. 2.1.

Fig. 3.4 shows the performance of SPTC using 64QAM constellation with gray code indexing, constant target point $t = 0$ and the shape shown in Fig. 2.3. From the BER and SER curves, it can be seen that using this shape a coding gain of 2.5 dB compared to the performance of uncoded 64QAM can be achieved.

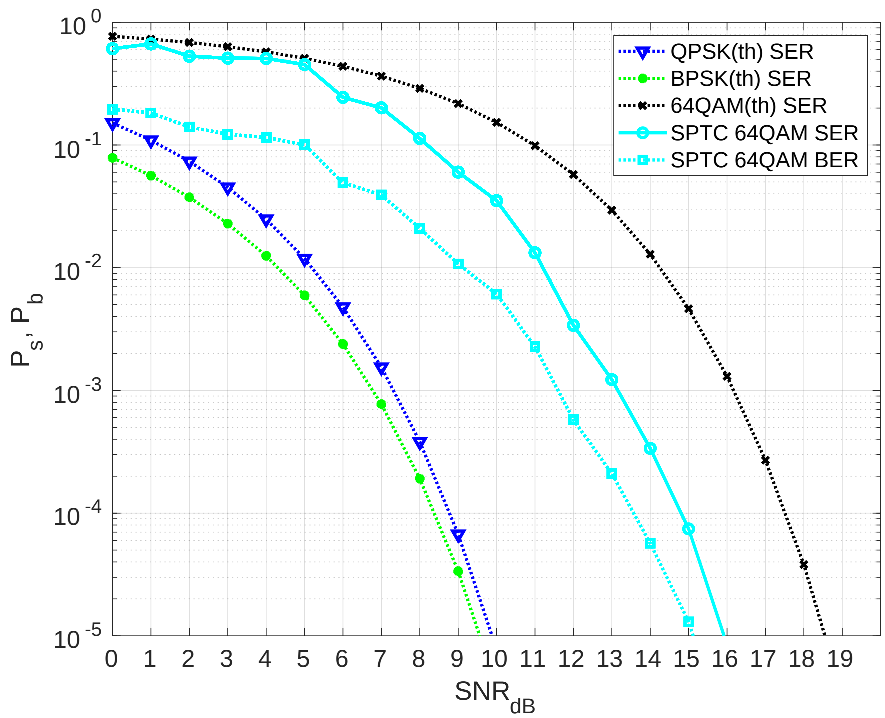


Fig. 3.4: BER and SER Curve for rate $\frac{1}{2}$ SPTC with target point $t = 0$ on a 64QAM constellation for shape shown in Fig. 2.3.

### 3.2.2 Performance of CAC

The performance of CAC using QPSK constellation and bit assignment shown in Fig. 2.6. This bit assignment results in similar performance with SPTC using QPSK constellation and shape shown in Fig. 2.6. For this bit assignment, the free Euclidean distance is $12E_{c,s}$.

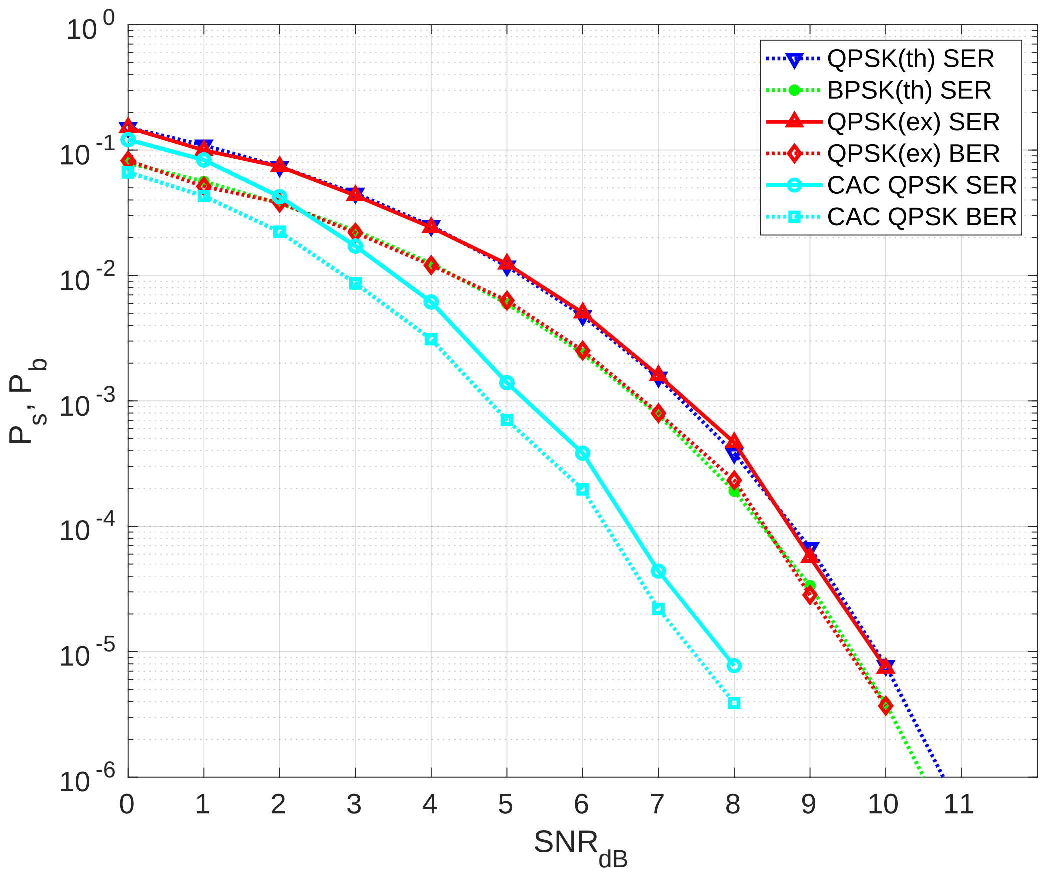


Fig. 3.5: BER and SER of rate $\frac{1}{2}$ CAC using QPSK signal constellation and bit assignment shown in Fig. 2.6.

Fig. 3.6 shows that it is possible to increase the coding gain for CAC in QPSK constellation by increasing the $d^2_{\text{free}}$ to $20E_{c,s}$ results from the bit assignment shown in Fig. 2.16. The bit assignment from Fig. 2.16 provides 3.5 dB coding gain compared to the performance of uncoded QPSK.

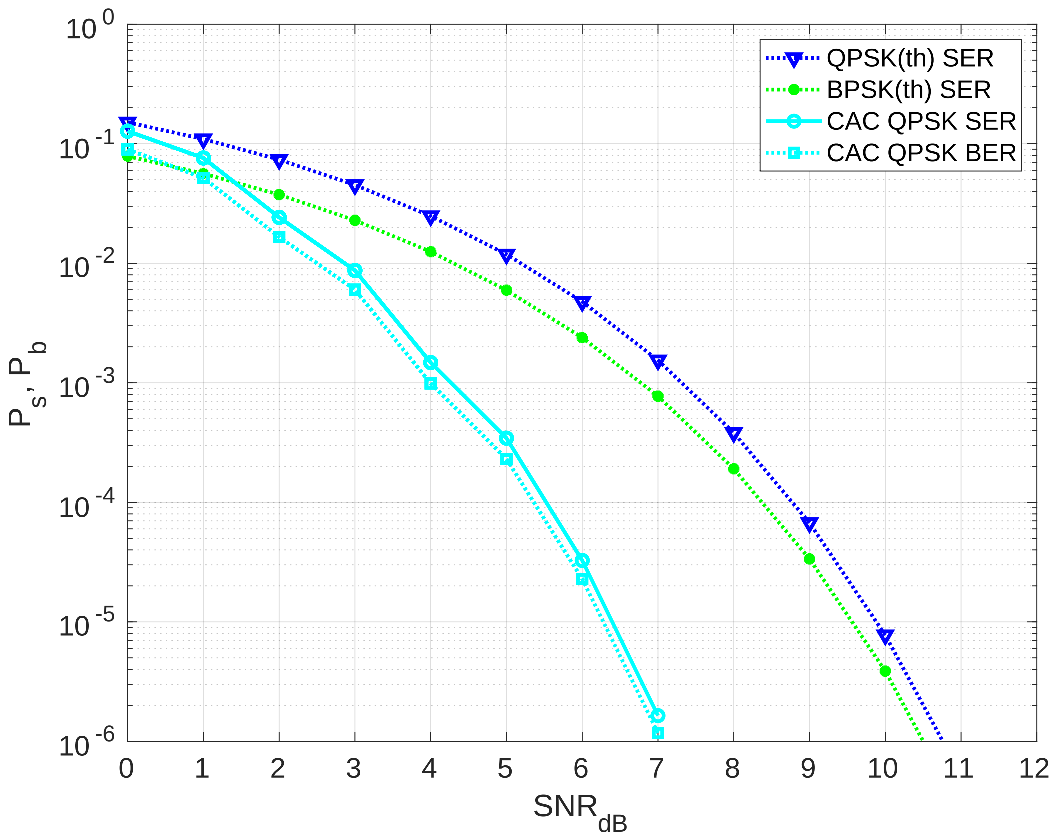


Fig. 3.6: BER and SER of rate $\frac{1}{2}$ CAC using QPSK signal constellation and bit assignment shown in Fig. 2.16.

To evaluate the performance of CAC on 16QAM constellation for $d^2_{\text{free}}$, different bit assignments have been used. Using gray code indexing shown in Fig. 2.1 the simulation results for rate 1/2 CAC on 16QAM constellation is presented in Fig. 3.7. Gray code indexing results in $d^2_{\text{free}} = 16E_{c,s}$ and a coding gain of 3 dB compared to the performance of uncoded 16QAM at $P_s = 10^{-5}$.

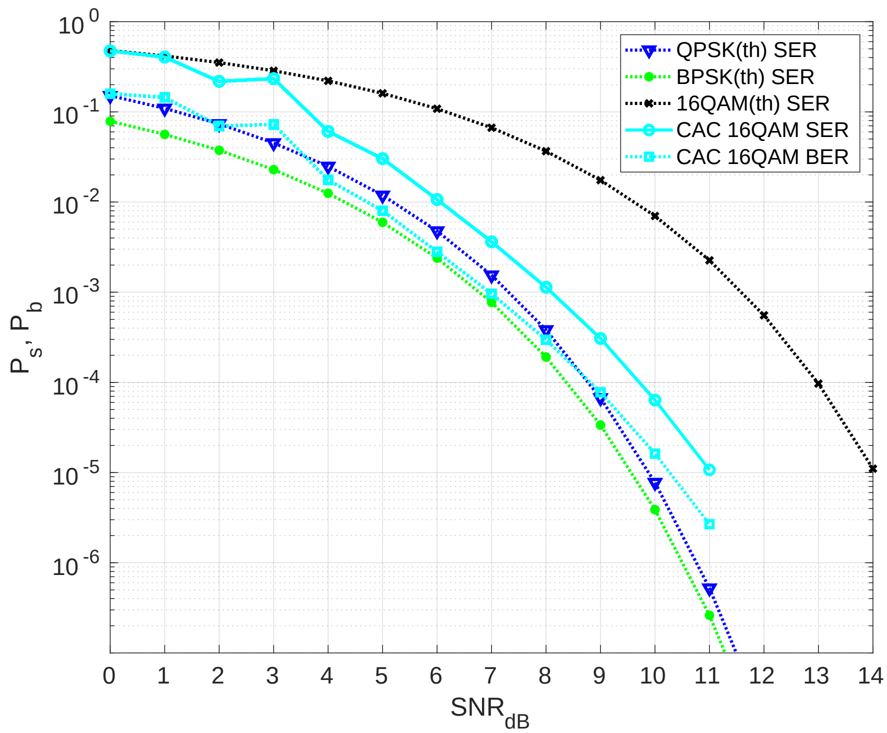


Fig. 3.7: BER and SER for rate 1/2 CAC using 16QAM constellation and bit assignment shown in Fig. 2.1.

The bit assignment shown in Fig. 2.19, has $d^2_{\text{free}} = 20E_{c,s}$ and the performance of CAC using this bit assignment is provided in Fig. 3.8. This bit assignment results in a coding gain of 4 dB at $P_s = 10^{-5}$ compared to uncoded 16QAM BER and SER.

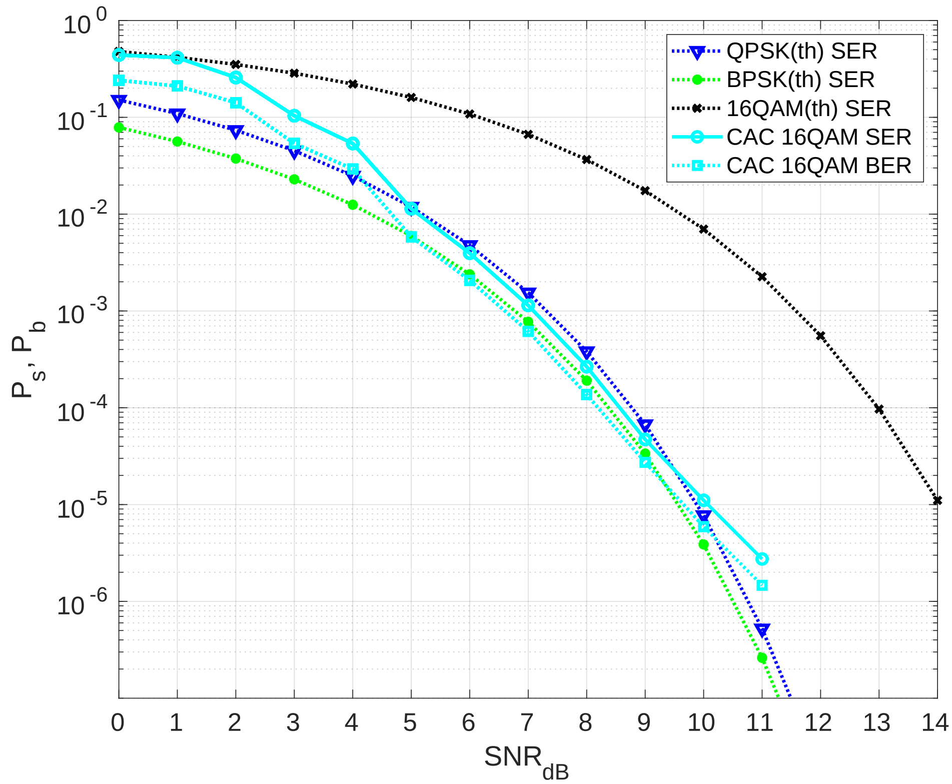


Fig. 3.8: Ber and SER of rate $\frac{1}{2}$ CAC using 16QAM constellation and bit assignment shown in Fig. 2.19.

Using the Sequential Bit Assignment shown in Fig. 2.20, the performance of the rate 1/2 CAC on 16QAM constellation is presented in Fig. 3.9. For this bit assignment with $d^2_{\text{free}} = 28E_{c,s}$, at $P_s = 10^{-5}$, 5 dB coding gain is possible compared to uncoded performance of 16QAM signal constellation. This simulation also shows that rate 1/2 CAC on 16QAM constellation using this bit assignment provides 1 dB coding gain for SER at $Ps = 10^{-5}$ compared to the SER of uncoded QPSK. The uncoded QPSK has the same information per symbol as the rate 1/2 code using 16QAM constellation.

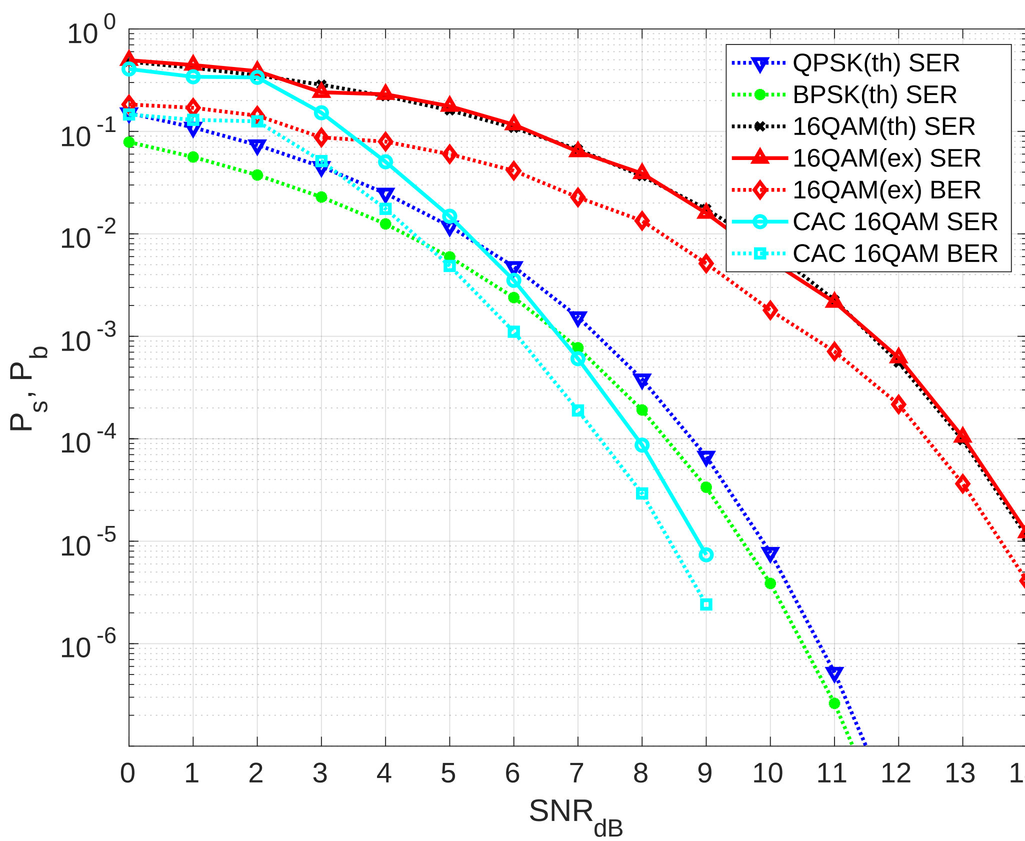


Fig. 3.9: Ber and SER of rate $\frac{1}{2}$ CAC using 16QAM constellation and bit assignment shown in Fig. 2.20.

For 64QAM, the BER and SER for rate 1/2 CAC using Sequential Bit Assignment shown in Fig. 2.23 are provided in Fig. 3.10. From Fig. 3.10 it can be seen that the Sequential Bit Assignment provides a 5.5 dB coding gain at $P_s = 10^{-5}$ compared to the BER and SER of uncoded 64QAM.

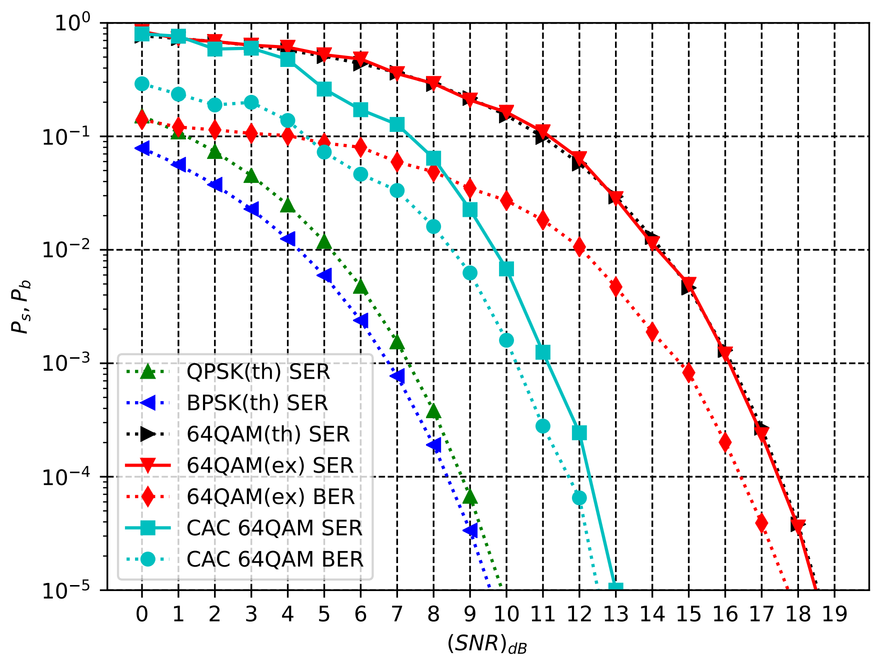


Fig. 3.10: Ber and SER of rate $\frac{1}{2}$ CAC using 64QAM constellation and bit assignment shown in Fig. 2.23.

Simulation results for rate 1/2 CAC using 256QAM signal constellation are shown in Fig. 3.11. For this simulation, Sequential Bit Assignment has been used in the signal constellation. The rate 1/2 CAC provides 5 dB coding gain for both BER and SER compared to uncoded BER and SER for 256QAM.

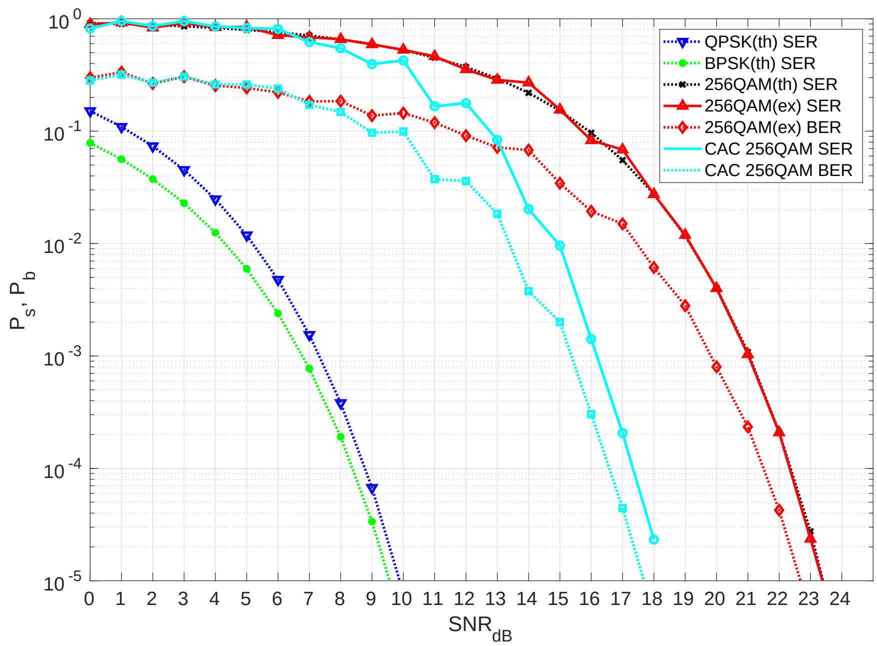


Fig. 3.11: Ber and SER of rate $\frac{1}{2}$ CAC using 256QAM constellation and Sequential Bit Assignment.

### 3.2.3 Comparison of Performance Between CAC and SPTC

Though the performance of SPTC depends on the employed shape and the performance of CAC depends on bit assignment in the signal constellation, in this part of the thesis the performance of SPTC and CAC is compared using free Euclidean distance. In Figs. 3.12 and 3.13, comparison between SPTC and CAC on QPSK constellation is shown. In Fig. 3.12, the performance of SPTC is calculated using the shape shown in Fig. 2.6. For SPTC and CAC, gray code indexing has been used. While for this shape the SPTC has $d^2_{\text{free}} = 16E_{c,s}$ and for this bit assignment the CAC has $d^2_{\text{free}} = 12E_{c,s}$, they provided approximately similar performance. In Fig. 3.13, the shape shown in Fig. 2.10 has been used for SPTC with gray code indexing and for CAC, the bit assignment shown in Fig. 2.16 has been used. Though the shape provides $d^2_{\text{free}} = 28E_{c,s}$ and the bit assignment provides $d^2_{\text{free}} = 20E_{c,s}$, CAC has a 0.5 dB coding gain compared to SPTC at $P_s = 10^{-5}$.

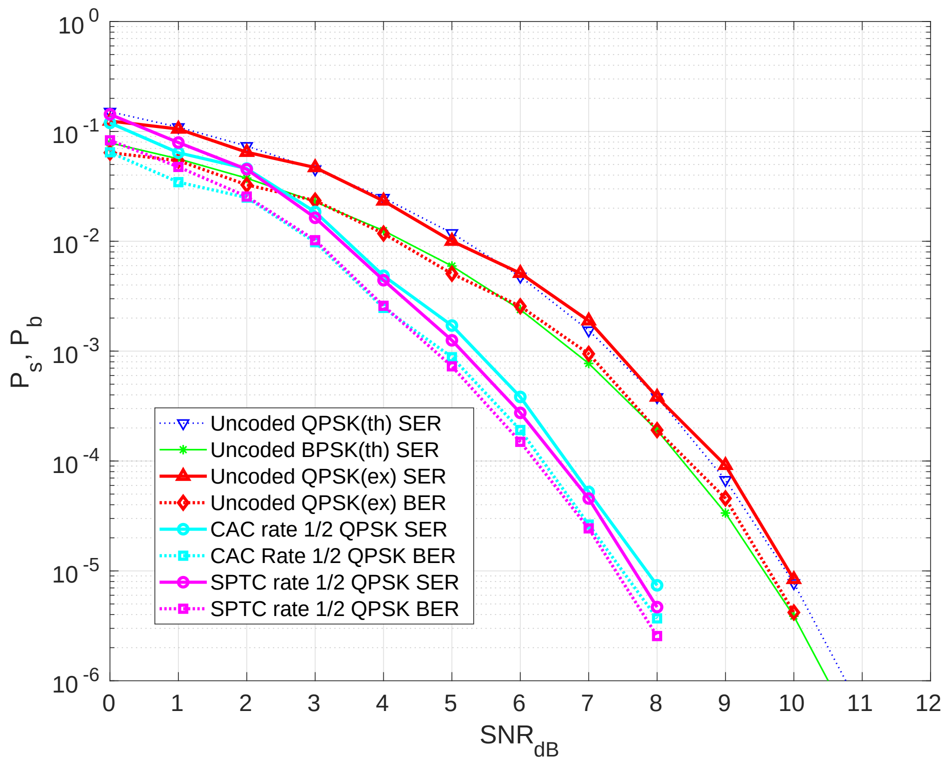


Fig. 3.12: Comparison between rate $\frac{1}{2}$ Constellation Arithmetic Code and SPTC on a QPSK constellation.

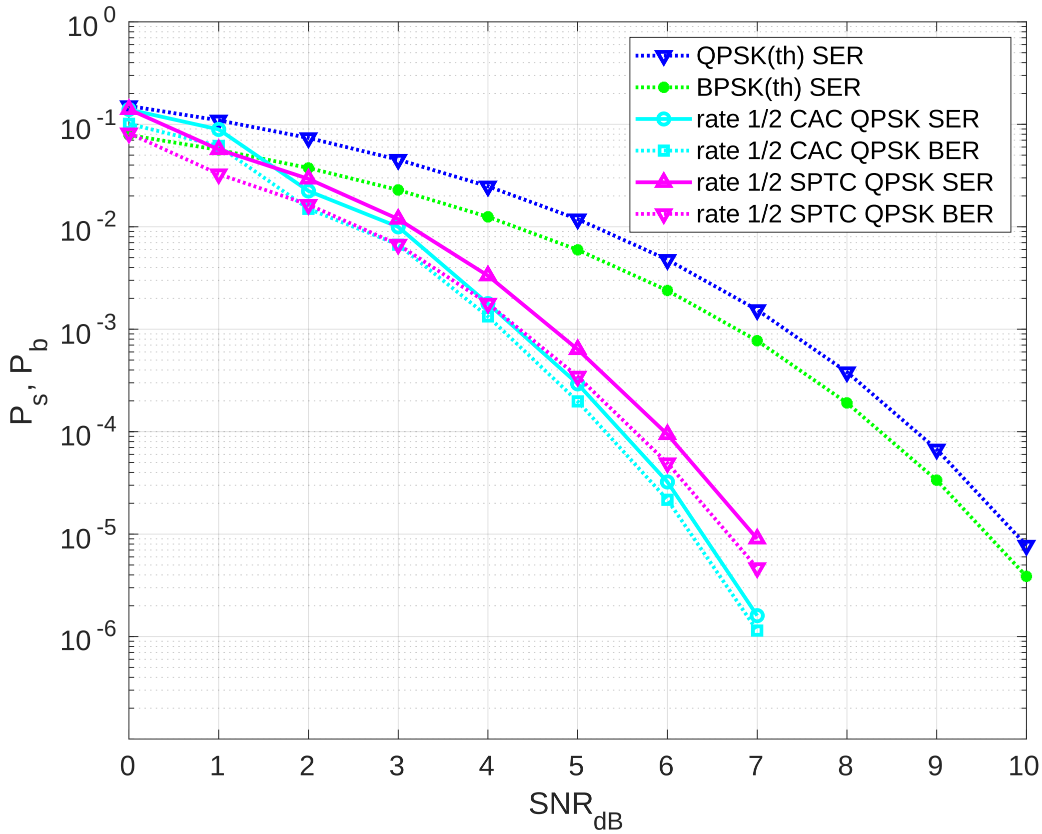


Fig. 3.13: Comparison of performance between rate $\frac{1}{2}$ Constellation Arithmetic Code and SPTC on a QPSK constellation.

Fig. 3.14 shows the comparison of performance between SPTC and CAC on 16QAM constellation where CAC has 2 dB coding gain at $Ps = 10^{-4}$. The shape from Fig. 2.1 used for encoding SPTC has $d^2_{\text{free}} = 16E_{c,s}$ while the Sequential Bit Assignment used in CAC has $d^2_{\text{free}} = 28E_{c,s}$. The comparison between SPTC and CAC using 64QAM constellation is shown in Fig. 3.15. The SPTC has been simulated using the shape shown in Fig. 2.3 using gray code indexing and the CAC has been simulated using the bit assignment shown in Fig. 2.23. From Fig. 3.15, CAC has 2.5 dB coding gain compared to SPTC at $P_s = 10^{-4}$.

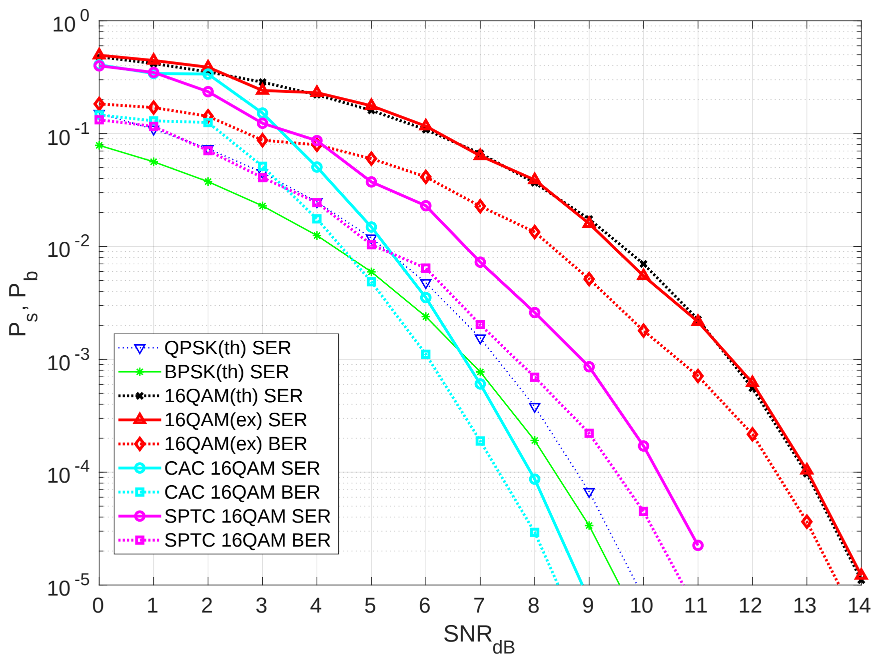


Fig. 3.14: Comparison of performance between rate $\frac{1}{2}$ Constellation Arithmetic Code and SPTC on a 16QAM constellation.

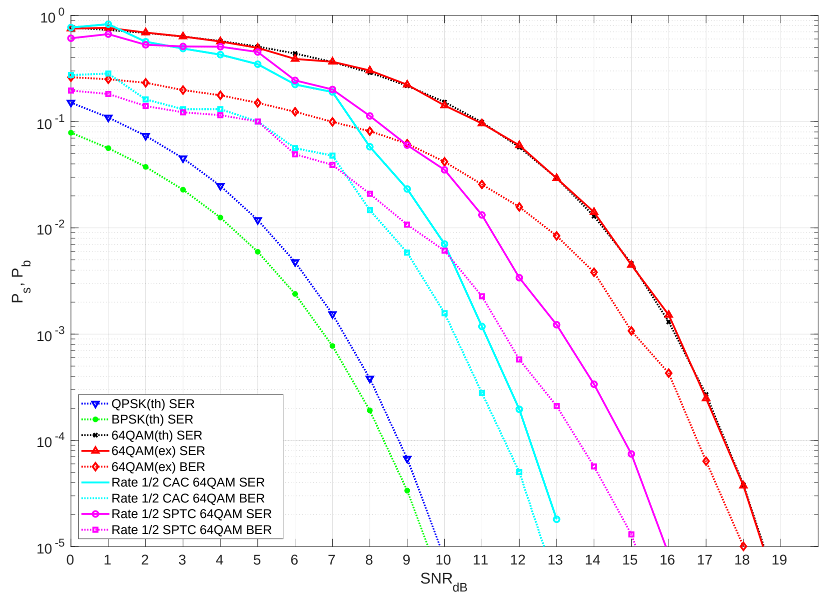


Fig. 3.15: Comparison of performance between rate $\frac{1}{2}$ Constellation Arithmetic Code and SPTC on a 64QAM constellation.

## 3.3 Performance of Codes for Different Trellis Depth

Fig. 3.16 presents the performance of the Viterbi decoder for different trellis depth using QPSK signal constellation with rate 1/2 CAC and bit assignment shown in Fig. 2.16. From this figure, it can be seen that by increasing the trellis depth the performance of the decoder can be increased. However, for larger trellis depth no noticeable coding gain is achieved. The performance of the decoder remains same for trellis depth of 5, 10 and 20. Therefore trellis depth of 10 has been used to decode the received symbols using the Viterbi algorithm.

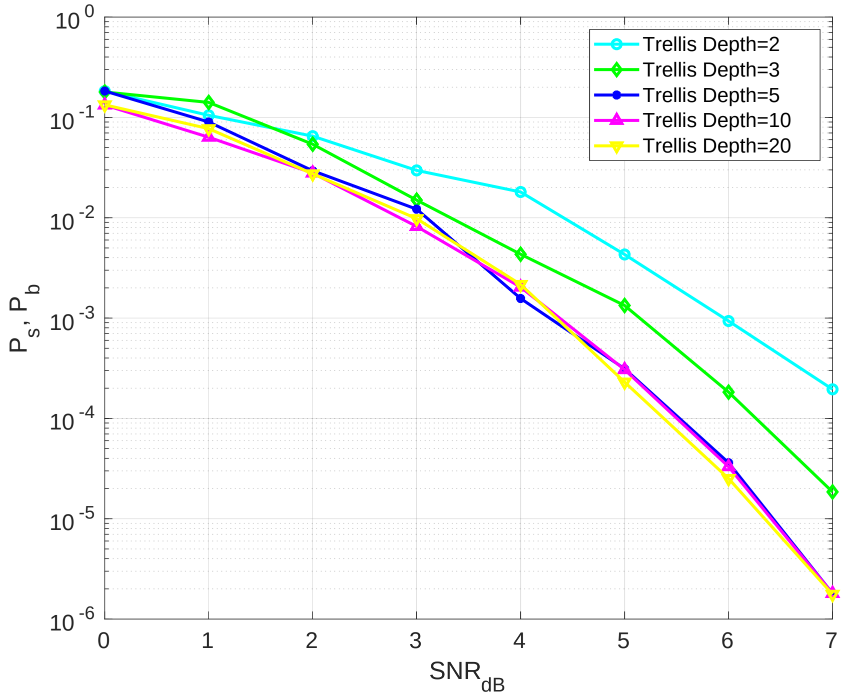


Fig. 3.16: Comparison of performance for rate $\frac{1}{2}$ CAC using QPSK signal constellation and bit assignment shown in Fig. 2.16 with various trellis depths.

## 3.4 Performance of CAC for Rate 2/3 and Rate 3/4 Codes

In this section, the simulated performances of CAC for rate 2/3 and rate 3/4 have been provided. In QPSK constellation, gray code indexing has been used and for 16QAM and 64QAM constellation bit assignments shown in Figs. 2.20 and 2.23 have been used. Results for both SIE and PPE are provided below. For evaluating the performance of the proposed methods, comparisons have been made using BER and SER between the codes for a given constellation and uncoded modulation for that constellation.

### 3.4.1 Simulation Results for Rate 2/3 and Rate 3/4 Codes Using SIE

Simulation results for CAC using QPSK constellation for rate 2/3 and rate 3/4 are provided in Figs. 3.17 and 3.18. From these results it can be seen that this method provides 1.5 dB coding gain $P_s = 10^{-4}$ compared to uncoded QPSK for both 2/3 and 3/4 rate codes on QPSK constellation. The BER and SER curves for rate 2/3 and rate 3/4 codes using 16QAM constellation are shown in Figs. 3.19 and 3.20. In Figs. 3.19 and 3.20, SIE has a 1.5 dB coding gain at $P_s = 10^{-5}$ for both rate 2/3 and rate 3/4 codes from uncoded 16QAM. For SIE, the BER and SER curves for rate 2/3 CAC using 64QAM is shown in Fig. 3.21 and from this plot it can be seen that there is a 1 dB coding gain at $P_s = 10^{-4}$.

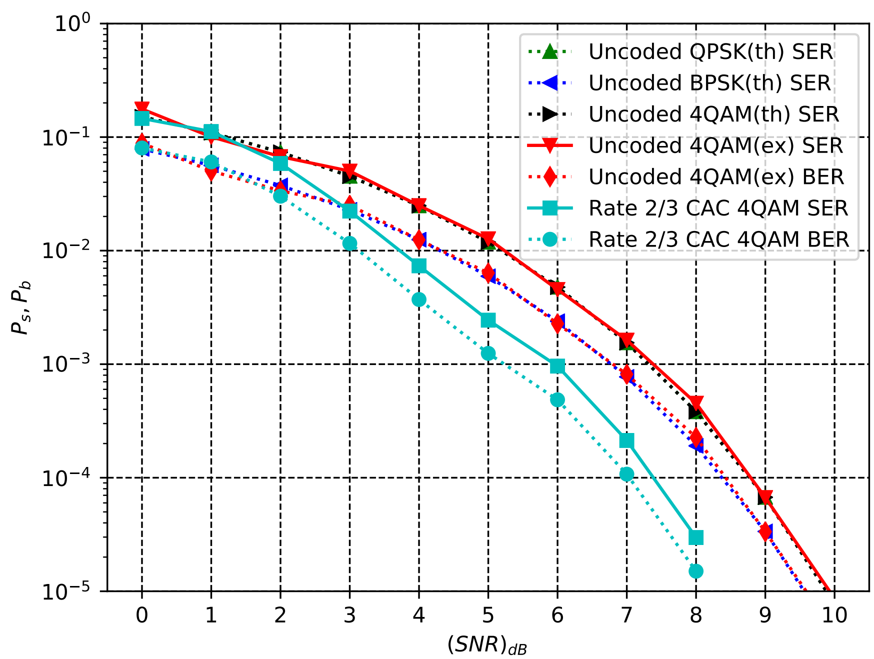


Fig. 3.17: BER and SER curve of rate $\frac{2}{3}$ CAC (SIE) on a QPSK constellation.

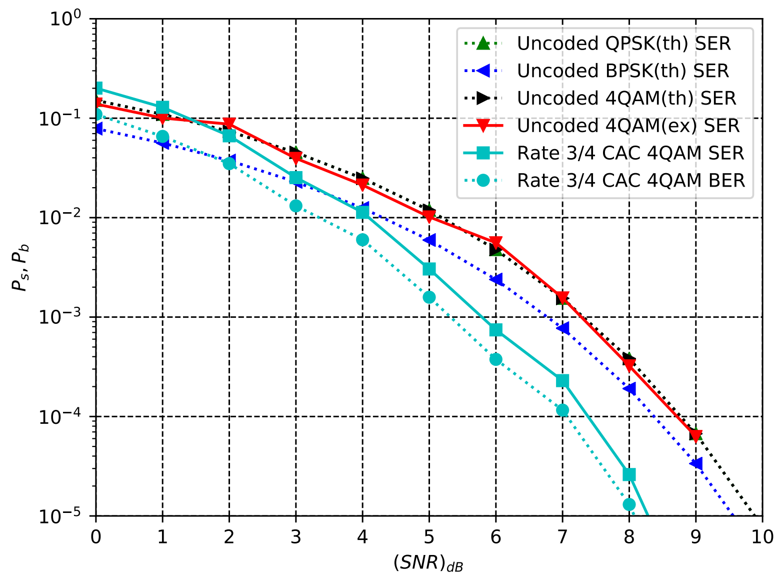


Fig. 3.18: BER and SER curve of rate $\frac{3}{4}$ CAC (SIE) on a QPSK constellation.

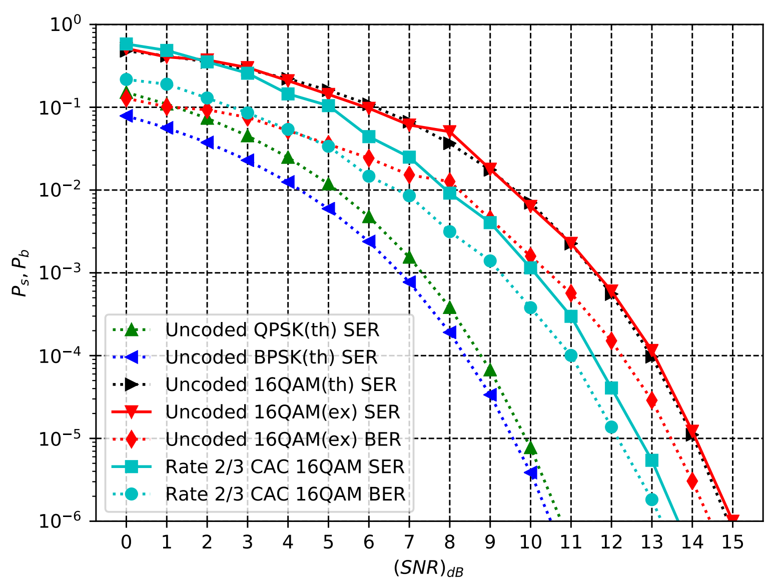


Fig. 3.19: BER and SER curve of rate $\frac{2}{3}$ CAC (SIE) on a 16QAM constellation.

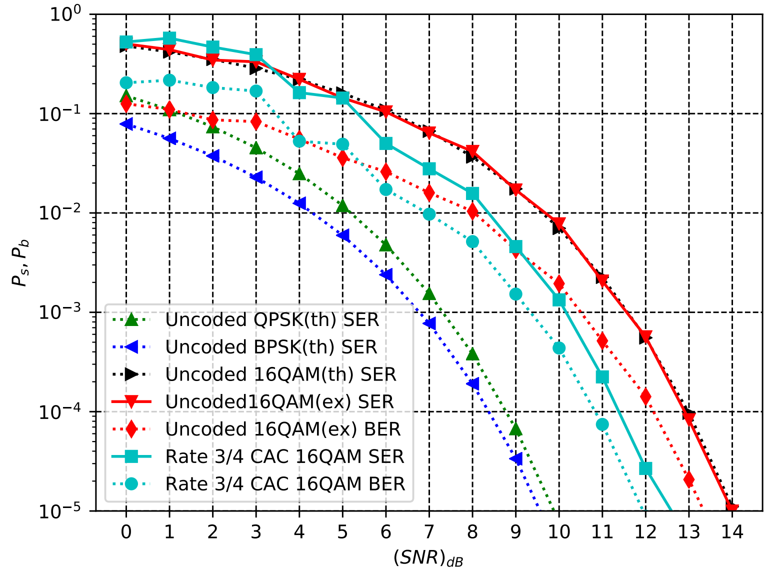


Fig. 3.20: BER and SER curve of rate $\frac{3}{4}$ CAC (SIE) on a 16QAM constellation.

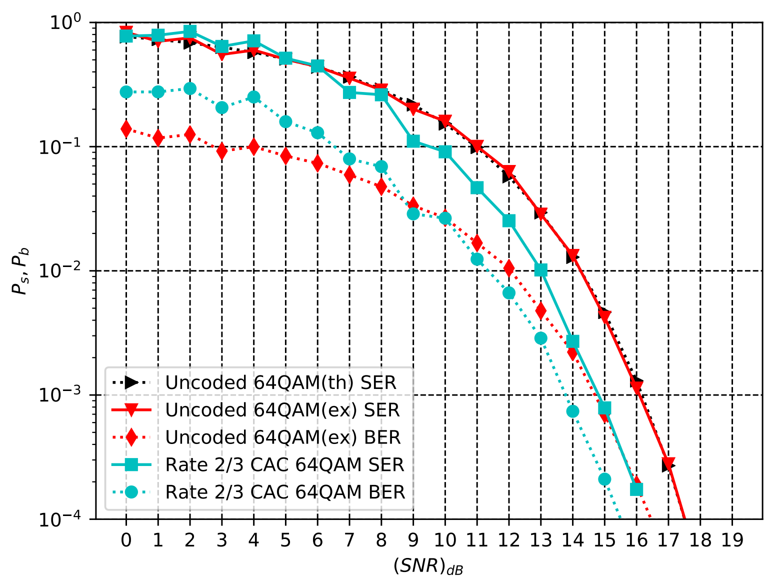


Fig. 3.21: BER and SER curve of rate $\frac{2}{3}$ CAC (SIE) on a 64QAM constellation.

### 3.4.2 Simulation Results for Rate 2/3 and Rate 3/4 Codes Using PPE

The BER and SER curves of rate 2/3 and rate 3/4 codes encoded using PPE and QPSK, 16QAM, and 64QAM constellations are provided in Figs. 3.22, 3.23, 3.24, 3.25 and 3.26. From Figs. 3.22, 3.23 and 3.25 it can be seen that this method fails to provide any coding gain for QPSK constellation and rate 3/4 codes for 16QAM and 64QAM constellation. However, from the results of Figs. 3.24 and 3.26, PPE has a good coding gain for rate 2/3 code on 16QAM and 64QAM constellation compared to the coding gain of SIE rate 2/3 codes for these constellations. In Fig. 3.24, PPE has a 4 dB coding gain at $P_s = 10^{-5}$ compared to the BER and SER of the uncoded 16QAM constellation and in 3.26, PPE has 5.5 dB coding gain at $P_s = 10^{-5}$ compared to the performance of the uncoded 64QAM constellation.

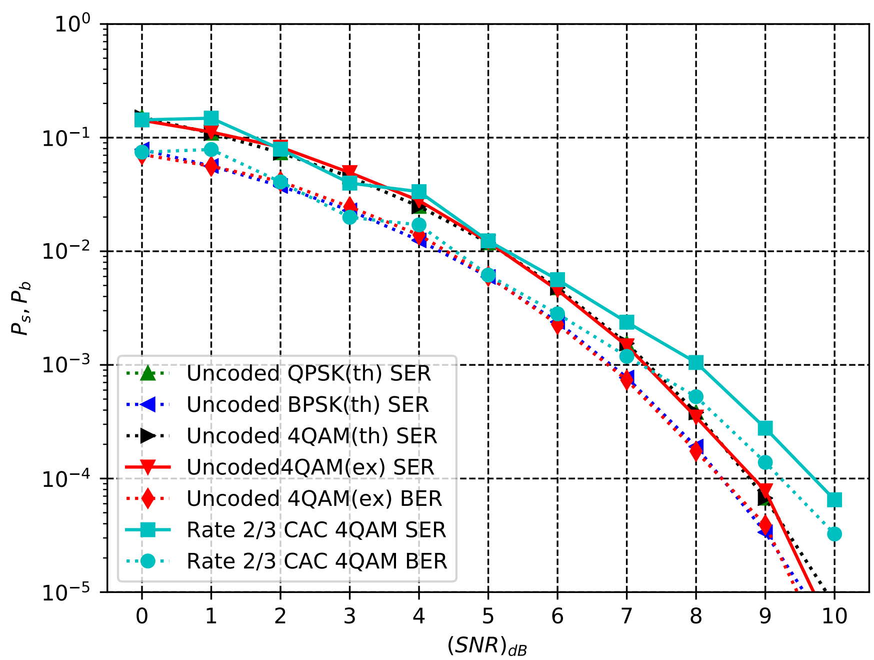


Fig. 3.22: BER and SER curve of rate $\frac{2}{3}$ CAC (PPE) on a QPSK constellation.

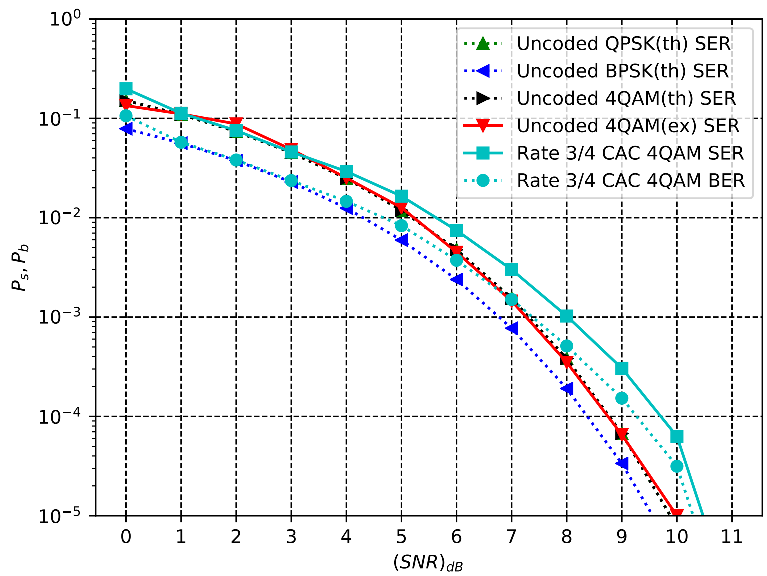


Fig. 3.23: BER and SER curve of rate $\frac{3}{4}$ CAC (PPE) on a QPSK constellation.

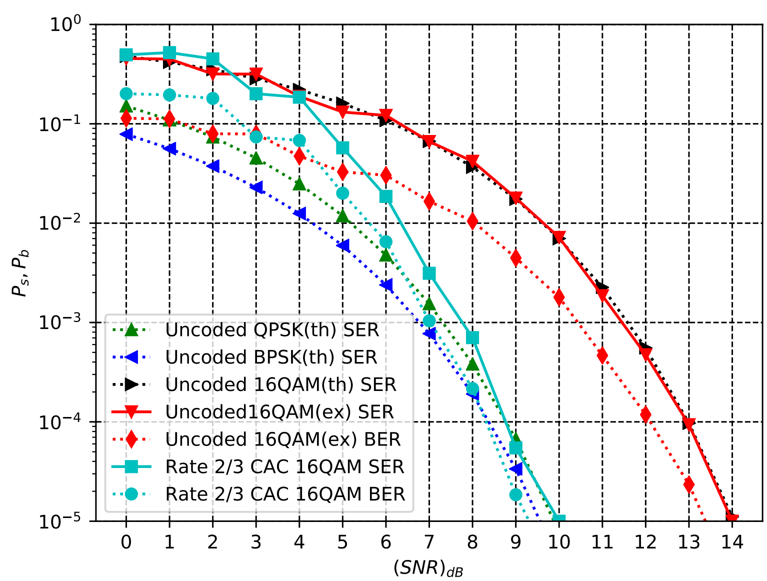


Fig. 3.24: BER and SER curve of rate $\frac{2}{3}$ CAC (PPE) on a 16QAM constellation.

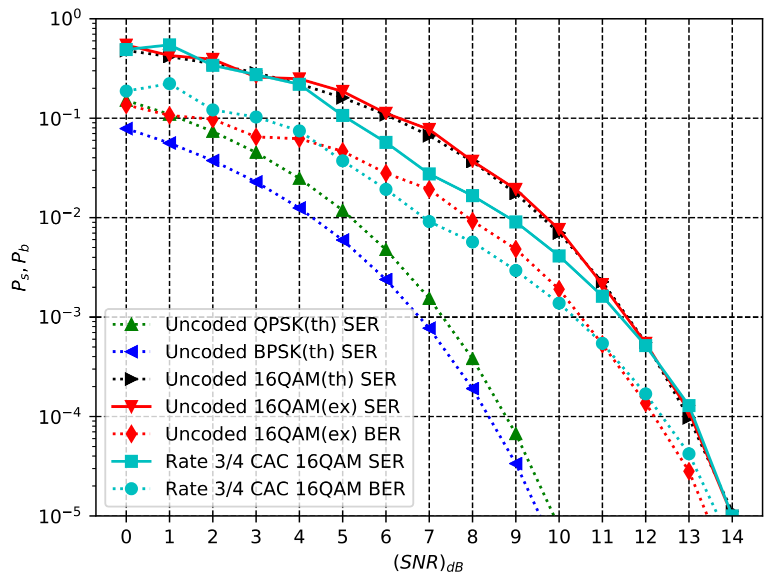


Fig. 3.25: BER and SER curve of rate $\frac{3}{4}$ CAC (PPE) on a 16QAM constellation.

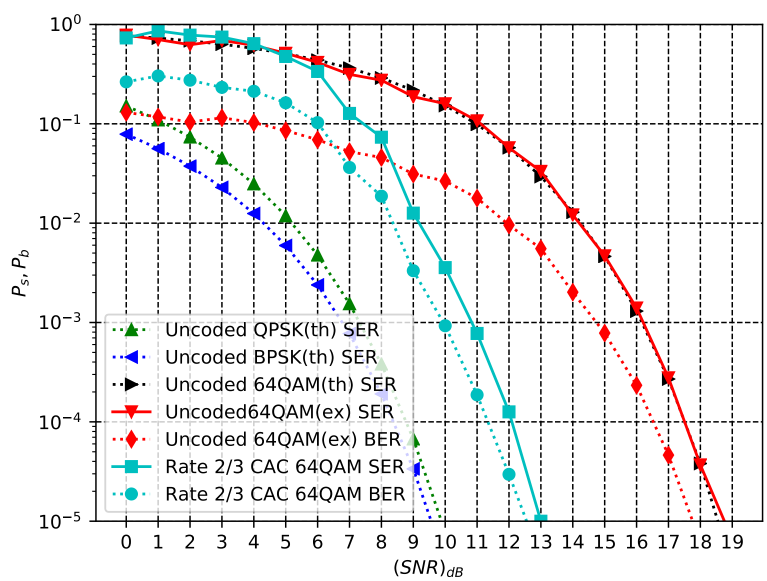


Fig. 3.26: BER and SER curve of rate $\dfrac{2}{3}$ CAC (PPE) on a 64QAM constellation.

### 3.4.3 Summarization of Results

Table 3.1 summarizes the performance of SPTC and CAC for different constellations and rates. In this table, for a given constellation coding gain is compared with the SER of the uncoded modulation using that constellation. For rate 2/3 and 3/4 codes and rate 1/2 SPTC using 64QAM, the free Euclidean distance $d^2_{\text{free}}$ is not provided in this table.

| Constellation | Encoding Method | Rate | $d^2_{\text{free}}$ | Coding Gain |
|---|---|---|---|---|
| QPSK | SPTC | 1/2 | $16E_{c,s}$ | 2.25 dB at $P_s = 10^{-5}$ |
| QPSK | SPTC | 1/2 | $28E_{c,s}$ | 3.0 dB at $P_s = 10^{-5}$ |
| QPSK | CAC | 1/2 | $12E_{c,s}$ | 2.0 dB at $P_s = 10^{-5}$ |
| QPSK | CAC | 1/2 | $20E_{c,s}$ | 3.5 dB at $P_s = 10^{-5}$ |
| QPSK | CAC (SIE) | 2/3 | — | 1.5 dB at $P_s = 10^{-4}$ |
| QPSK | CAC (PPE) | 2/3 | — | No Coding Gain |
| QPSK | CAC (SIE) | 3/4 | — | 1.5 dB at $P_s = 10^{-5}$ |
| QPSK | CAC (PPE) | 3/4 | — | No Coding Gain |
| 16QAM | SPTC | 1/2 | $16E_{c,s}$ | 2.75 dB at $P_s = 10^{-4}$ |
| 16QAM | CAC | 1/2 | $12E_{c,s}$ | 3.0 dB at $P_s = 10^{-5}$ |
| 16QAM | CAC | 1/2 | $20E_{c,s}$ | 4.0 dB at $P_s = 10^{-5}$ |
| 16QAM | CAC | 1/2 | $28E_{c,s}$ | 5.0 dB at $P_s = 10^{-5}$ |
| 16QAM | CAC (SIE) | 2/3 | — | 1.5 dB at $P_s = 10^{-6}$ |
| 16QAM | CAC (PPE) | 2/3 | — | 4.0 dB at $P_s = 10^{-5}$ |
| 16QAM | CAC (SIE) | 3/4 | — | 1.5 dB at $P_s = 10^{-5}$ |
| 16QAM | CAC (PPE) | 3/4 | — | No Coding Gain |
| 64QAM | SPTC | 1/2 | — | 2.5 dB at $P_s = 10^{-5}$ |
| 64QAM | CAC | 1/2 | $28E_{c,s}$ | 5.5 dB at $P_s = 10^{-5}$ |
| 64QAM | CAC (SIE) | 2/3 | — | 1.0 dB at $P_s = 10^{-4}$ |
| 64QAM | CAC (PPE) | 2/3 | — | 5.5 dB at $P_s = 10^{-5}$ |
| 256QAM | CAC | 1/2 | $28E_{c,s}$ | 5.0 dB at $P_s = 10^{-4}$ |

Table 3.1: Summarization of results obtained for SPTC and CAC.

### 3.5 Comparison Between Different Programming Languages Used for Simulations

In this thesis, three programming languages, Matlab version 9.4, Python version 3.6.4 and Julia version 0.6 have been used to perform various simulations. One of the objectives of this thesis is to compare among those three languages and determine which language is more suitable to study error correction coding theory. In order to compare, all the programs written in those languages has been made as similar to each other as possible in terms of syntax. For simulation, an object containing methods for encoding and decoding has been created and vectorized operations are used. While Matlab and Python are fully object-oriented language, Julia is not fully object-oriented in the conventional structure. In Julia, an object is called a composite type and is defined as Struct. But unlike Python and Matlab, in Julia, the methods are declared outside of Struct and then are linked to that Struct. To compare simulation time between Matlab, Python, and Julia, SPTC using the shape and QPSK constellation shown in Fig. 2.10 has been used as a reference and the simulations are performed using the same hardware. To take into account of randomness while generating bits, the average of 5 simulation time for each of the language has been considered. For SNR 0 to 7, the average simulation time required to generate BER and SER curves while counting 100 errors at each SNR is presented in Table 3.2.

| Language | Version | Average Simulation Time (second) |
|---|---|---|
| Matlab | 9.4 | 380.55 |
| Python | 3.6.4 | 487.95 |
| Julia | 0.6 | 155.40 |

Table 3.2: Average Simulation Times for Different programming Languages.

From Table 3.2, it can be seen that Julia is twice faster than Matlab and Python. This high speed of Julia can be useful for calculating the BER by counting a large number of errors at high SNR. Despite Julia being faster than the other two languages it has some issues. As a fairly new language, the documentation for Julia is not as rich as Matlab or Python. Recent release Julia's stable version 1.1.1 has made some of the functions from version 0.6 deprecated. Another notable issue is while using Julia in Atom IDE, random crashes have been faced during simulations.

Both Python and Matlab have stable IDE, well-defined documentation and similar simulation speed while Julia is a lot faster than the other two. However, Python and Julia are open-source while Matlab can be expensive for students. Julia can be used in the study of error correction coding when simulation speed is an important requirement and Python can be used when stability during simulation is needed. Examining the different aspects of these programming languages, Julia is recommended among Matlab, Python, and Julia for the study of error correction coding because of its high simulation speed.

# CHAPTER 4

# PERFORMANCE OF CODES WITH LARGER STATES

In this chapter, an approach is presented for increasing the free Euclidean distance for Constellation Arithmetic Code by increasing the number of states during encoding. By exploiting the property of state variable, a modification in the encoding operation has been proposed to increase the number of states. Though this modification increases the $d^2_{\text{free}}$, it fails to provide any coding gain. Therefore this chapter provides a possible foundation for one of the future area for conducting research in Constellation Arithmetic Code.

## 4.1 Study the Performance of Codes for Different number of States:

For Constellation Arithmetic Code, the number of states is equal to the number of points in the signal space. From (2.2), the modulus operation enforces the state variable to stay between 0 to $M-1$. By increasing the number of states, it is possible to create more possible paths in the trellis which can. The number of states can be changed by modifying (2.2) to :

$$j_{i+1} = j_i + r_i \pmod{N_s} \quad \text{where } N_s \text{ is number of state} \tag{4.1}$$

This modification allows for the increment of the possible number of paths in the trellis. But just only by increasing the number of states doesn't increase the free Euclidean distance. Because, if the branches tend to reach the state variables which are in the close vicinity then the distance between the branches will not increase. So a multiplicative factor $\alpha \in \mathbb{Z}_{>0}$ is introduced to calculate the state variable which will enforce the branches to disperse and therefore increasing the free Euclidean distance.

$$j_{i+1} = \alpha(j_i + r_i) \pmod{N_s} \quad \text{where } N_s \text{ is number of state} \tag{4.2}$$

Using these proposed modifications it is possible to increase the $d^2_{\text{free}}$ for codes in a given constellation by increasing $N_s$ and finding $\alpha$ for that $N_s$. Table 4.1 contains the $d^2_{\text{free}}$ for the different number of $N_s$ and $\alpha$. From this table, it can be seen that the asymptotic coding gain can be increased by increasing $N_s$ and using an appropriate $\alpha$ for a code in a given constellation. For Constellation Arithmetic Code on QPSK constellation using bit assignment shown in Fig. 2.16, a trellis of consisting of 13 number of states and $\alpha = 3$ is shown in Fig. 4.1. In this Fig. the input and parity symbols associated with each branch are placed in the left of the initial state of that branch. Starting from left, moving to right each input and parity symbol pair $S/R$ corresponds to the branches selected from top to bottom. For example, located at the left of initial state 0, 2/2 corresponds to the third branch which has started at initial state 0 and ended at state 6. Now for this trellis, the pair of paths that corresponded to $d^2_{\text{free}}$ is shown in Fig. 4.2 by black solid lines. From those paths, the free Euclidean distance

$$\begin{aligned} d^2_{\text{free}} &= d^2_e(0,1) + d^2_e(0,1) + d^2_e(0,2) + d^2_e(0,1) + d^2_e(0,1) + d^2_e(0,1) \\ &= 4E_{c,s} + 4E_{c,s} + 8E_{c,s} + 4E_{c,s} + 4E_{c,s} + 4E_{c,s} \\ &= 28E_{c,s}. \end{aligned}$$

Though increasing the $N_s$ increases $d^2_{\text{free}}$, in this thesis no coding gain has been found for larger $d^2_{\text{free}}$ when $N_s > M$ where $M$ is the number of points in the signal constellation. In Fig. 4.3 performances of CAC using 16QAM constellation with bit assignment from Fig. 2.20 for $N_s = 16$ with $d^2_{\text{free}} = 28E_{c,s}$, $N_s = 64$ with $d^2_{\text{free}} = 68E_{c,s}$ and $N_s = 128$ with $d^2_{\text{free}} = 120E_{c,s}$ is shown. From the simulation results, it can be seen that no coding gain has been achieved.

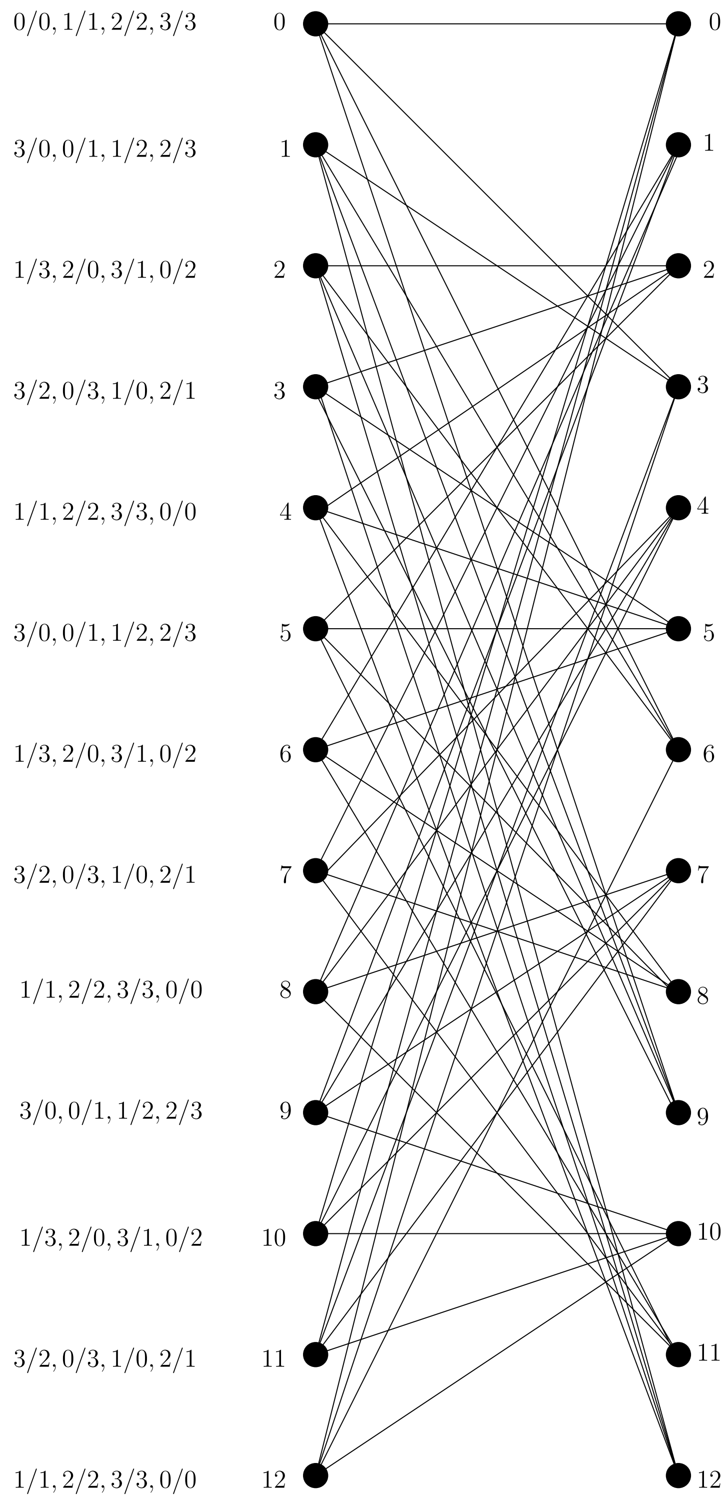


Fig. 4.1: Trellis structure for Constellation Arithmetic Code using QPSK constellation for $\alpha = 3$ and $N_s = 13$.

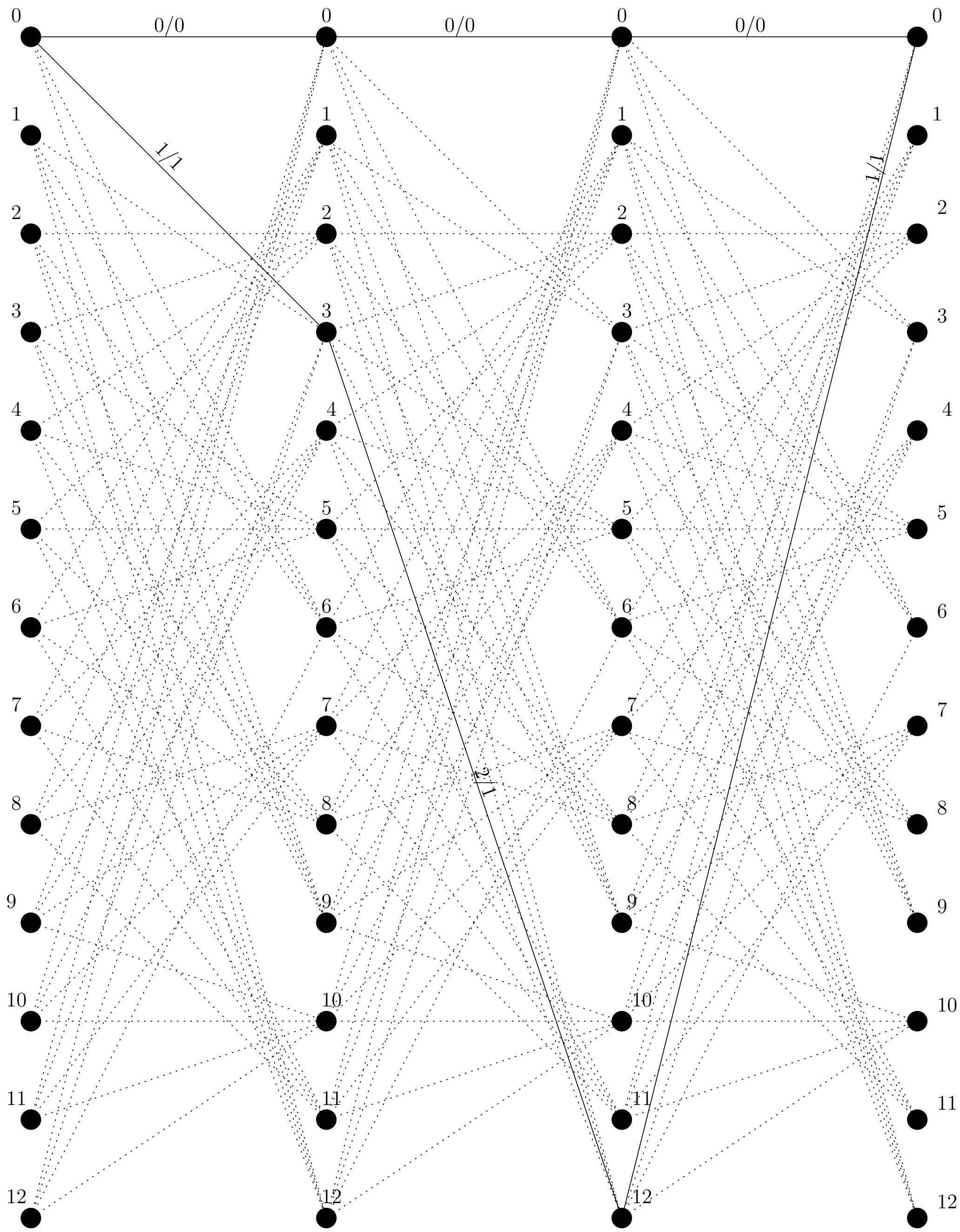


Fig. 4.2: Branches (black solid line) on the trellis that results in $d^2_{\text{free}}$ for CAC on QPSK constellation shown in Fig. 3.6.

| constellation | Number of States, $N_s$ | $\alpha$ | $d^2_{\text{free}}$ | Asymptotic Coding Gain (dB) |
|---|---|---|---|---|
| QPSK | 4 | 1 | $12E_{c,s}$ | 1.7609 |
| QPSK | 4 | 1 | $16E_{c,s}$ | 3.0103 |
| QPSK | 4 | 1 | $20E_{c,s}$ | 3.9794 |
| QPSK | 11 | 6 | $24E_{c,s}$ | 4.7712 |
| QPSK | 12 | 1 | $32E_{c,s}$ | 6.0206 |
| QPSK | 13 | 3 | $28E_{c,s}$ | 5.4407 |
| QPSK | 15 | 7 | $24E_{c,s}$ | 4.7712 |
| QPSK | 16 | 7 | $24E_{c,s}$ | 4.7712 |
| QPSK | 17 | 3 | $28E_{c,s}$ | 5.4407 |
| QPSK | 17 | 6 | $24E_{c,s}$ | 4.7712 |
| QPSK | 32 | 7 | $32E_{c,s}$ | 6.0206 |
| QPSK | 32 | 9 | $36E_{c,s}$ | 6.5321 |
| QPSK | 32 | 11 | $36E_{c,s}$ | 6.5321 |
| 16QAM | 16 | 1 | $28E_{c,s}$ | 5.4407 |
| 16QAM | 32 | 3 | $36E_{c,s}$ | 6.5321 |
| 16QAM | 32 | 5 | $48E_{c,s}$ | 7.7815 |
| 16QAM | 32 | 11 | $36E_{c,s}$ | 6.5321 |
| 16QAM | 32 | 13 | $36E_{c,s}$ | 6.5321 |
| 16QAM | 32 | 19 | $36E_{c,s}$ | 6.5321 |
| 16QAM | 32 | 27 | $36E_{c,s}$ | 6.5321 |
| 16QAM | 64 | 5 | $68E_{c,s}$ | 9.2942 |
| 16QAM | 64 | 11 | $52E_{c,s}$ | 8.1291 |
| 16QAM | 64 | 19 | $52E_{c,s}$ | 8.1291 |
| 16QAM | 64 | 29 | $60E_{c,s}$ | 8.7506 |
| 16QAM | 64 | 35 | $64E_{c,s}$ | 9.0309 |
| 16QAM | 64 | 37 | $52E_{c,s}$ | 8.1291 |
| 16QAM | 64 | 45 | $60E_{c,s}$ | 8.7506 |
| 16QAM | 128 | 19 | $100E_{c,s}$ | 10.9691 |
| 16QAM | 128 | 23 | $88E_{c,s}$ | 10.4139 |
| 16QAM | 128 | 39 | $120E_{c,s}$ | 11.7609 |
| 16QAM | 128 | 53 | $92E_{c,s}$ | 10.6070 |
| 64QAM | 64 | 1 | $28\ E_{c,s}$ | 5.4407 |
| 256QAM | 256 | 1 | $28\ E_{c,s}$ | 5.4407 |

Table 4.1: Free Euclidean Distance and Asymptotic Coding Gain for Different Number of States.

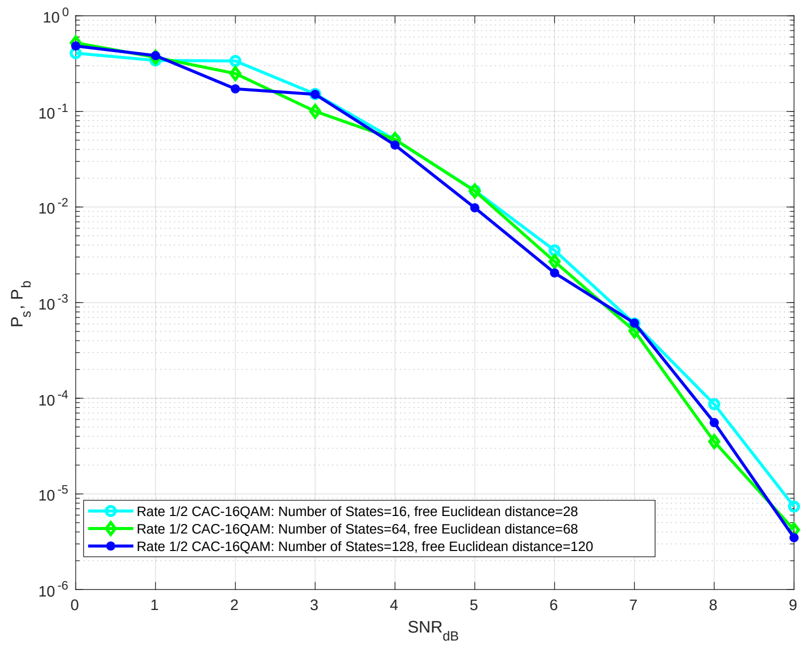


Fig. 4.3: Simulation results for rate 1/2 CAC-16QAM using different number of states.

The proposed method in this section fails to provide coding gain but it can be used as an insight for increasing the coding gain by increasing the number of states. Therefore further research is required in topic to find a viable solution that provides coding gain by increasing the number of states for CAC on any constellation.

## 4.2 Comparison Between The Performances of Some Well Known TCM and CAC

In TCM a convolutional encoder is used to encode the input bits and then the coded bits are mapped to a larger signal constellation. However, in CAC, each input symbol is encoded to generate a parity symbol. Therefore in this section, the comparison between TCM and CAC is done for the codes with the same *spectral efficiency*. The spectral efficiency is the number of information bits transmitted by each symbol [9]. The Trellis Codes for 8-PSK uses a rate 2/3 convolutional encoder followed by mapping to an 8-PSK signal constellation. So each symbol carries 2 information bits. For the rate 1/2 CAC on 16QAM constellation, each symbol carries 2 information bit. Therefore the rate 1/2 CAC on 16QAM constellation and the Trellis Codes for 8-PSK have spectral efficiency of 2. Table 4.2 contains the comparison between the rate 1/2 CAC on 16QAM constellation and the Trellis Codes for 8-PSK [9] using asymptotic coding gain for different number of states.

| Number of States | Asymptotic Coding Gain (dB) (coded/uncoded) | |
|---|---|---|
| | 8-PSK/4-PSK | 16QAM/QPSK |
| 16 | 4.1 | 5.44 |
| 32 | 4.6 | 7.7 |
| 64 | 5.0 | 8.12 |
| 128 | 5.2 | 10.4 |

Table 4.2: Comparison Between Rate 1/2 CAC on 16QAM Constellation and Trellis Codes for 8-PSK using Asymptotic Coding Gain.

# CHAPTER 5

# CONCLUSION AND FUTURE WORK

The research presented in this thesis provides a detailed study of Signal Point Target Code. By investigating various aspects of SPTC, the proposed code has been extended for larger signal constellations, different rates, and larger states. The findings from this work provide directions for future research in SPTC.

## 5.1 Contribution

The contributions of this thesis can be summarized as given below:

- The operation of the SPTC has been described by introducing a notation.
- The relation between the employed shape and the performance of SPTC has been examined using free Euclidean distance. For different shapes, the performance of SPTC has been evaluated on different QAM signal constellations.
- An alternative of SPTC, called Constellation Arithmetic Code has been presented.
- The relation between the bit assignment in the signal constellation and the performance of CAC has been examined using free Euclidean distance. Using different bit assignments and M-ary QAM signal constellations, the performance of CAC has been evaluated.
- A method for bit assignment in the signal constellation has been proposed that provides coding gains for CAC compared to performances of SPTC examined in this thesis.
- Using the CAC, two methods of increasing the rate of codes have been proposed and their performances for different rates have been simulated.

- The free Euclidean distances and asymptotic coding gains of CAC for larger number of states has been calculated. A method for increasing $d^2_{\text{free}}$ using larger number of states has been proposed. Form proposed method, one possible future research avenue for the code has been outlined.

- A suitable language among Matlab, Python, and Julia has been determined for the study of error correction coding by considering simulation speed. Different aspects of these programming languages have been discussed. By calculating average simulation speed, it has been found that Julia is the fastest and most suitable language among those three.

## 5.2 Conclusion

Both SPTC and CAC provides coding gain compared to the performance of uncoded modulation. The coding gain can be increased for SPTC by employing shapes that provide larger free Euclidean distance and for CAC by using bit assignments which results in larger free Euclidean distance. In this thesis, for QPSK constellation the maximum coding gain found for SPTC is 3 dB and for CAC is 3.5 dB. The shape used for SPTC 16QAM provides a coding gain of 2.75 while for CAC up to of 5 dB coding gain has been achieved. The proposed method for bit assignment in CAC for 64QAM and 256QAM provides good coding gain. From the proposed methods of increasing rates, SIE provides coding gain for all codes. Among Matlab, Python, and Julia, Julia has been found twice faster than the other two languages in terms of simulation speed. Therefore Julia has been selected as a suitable language for studying error correction coding between those three. Though the proposed method for increasing the number of states increases the free Euclidean distance and asymptotic coding gain, it fails to provide any coding gain compared to the performance of CAC using $M$ number of states.

## 5.3 Future Work

The future work for this thesis includes finding the best shape for SPTC and bit assignment for CAC on any given constellation. Further research is needed to develop a method similar to set partitioning used in TCM in order to maximize the free Euclidean distance on the trellis. As the PPE provides good coding gain for some selected codes, this method should be investigated further. The proposed method for increasing number of states needs to be studied more to find out why it failed to provide coding gain.

Overall, by providing a detailed study of the Signal Point Target code and by exploring and extending various attributes of the proposed codes, the objectives of this thesis have been investigated.

# REFERENCES


[1] C. E. Shannon, “A mathematical theory of communication,” *Bell Sys. Tech. Journal*, vol. 27, no. 3, pp. 379–423, July 1948.

[2] R. Gallager, “Low-density parity-check codes,” *IRE Tran. Inf. Theory*, vol. 8, no. 1, pp. 21–28, January 1962.

[3] C. Berrou and A. Glavieux, “Near optimum error correcting coding and decoding: turbo-codes,” *IEEE Trans. Commun.*, vol. 44, no. 10, pp. 1261–1271, Oct 1996.

[4] E. Arikan, “Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels,” *IEEE Trans. Inf. Theory*, vol. 55, no. 7, pp. 3051–3073, July 2009.

[5] G. Ungerboeck, “Channel coding with multilevel/phase signals,” *IEEE Trans. Inf. Theory*, vol. 28, no. 1, pp. 55–67, January 1982.

[6] J. Sodha, “Rotating signal point shape code,” in *Intelligent Computing.* Springer International Publishing, 2019, pp. 876–880.

[7] J. Sodha and T. Moon, “Signal point target code,” in *Proc. IEEE 62nd IMWSCAS (Accepted)*, Aug 2019.

[8] J. Sodha and A. Als, “Shape nature of error-control codes,” *Elsevier Sig. Process. Journal*, vol. 83, no. 7, pp. 1457 – 1465, 2003.

[9] T. Moon, *Error correction coding : mathematical methods and algorithms.* Hoboken, N.J: Wiley-Interscience, 2005.

[10] J. Proakis and M. Salehi, *Digital Communications.* McGraw-Hill, 2008.

# APPENDICES

# APPENDIX A

## Q function

$Q(x)$ is the area under the tail of a standard normal distribution. If $X$ is a standard Gaussian random variable then $Q(x)$ is the probability that $X$ is greater than $x$, so

$$
\begin{aligned}
Q(x) &= P(X > x) \\
&= 1 - P(X \leq x) \\
&= 1 - \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{u^2}{2}} du \\
&= \frac{1}{\sqrt{2\pi}} \int_{x}^{\infty} e^{-\frac{u^2}{2}} du \\
&= \frac{1}{2} \left( \frac{2}{\sqrt{2\pi}} \int_{x}^{\infty} e^{-\frac{u^2}{2}} du \right) \\
&= \frac{1}{2} \left( \frac{2}{\sqrt{\pi}} \int_{\frac{x}{\sqrt{2}}}^{\infty} e^{-t^2} dt \right) \\
&= \frac{1}{2} \mathrm{erfc}\left( \frac{x}{\sqrt{2}} \right).
\end{aligned}
$$

# APPENDIX B

## Code

### B.1 Matlab Code

#### Main Function

```
clc;
clear all;
close all;
tic
%Initialize
docnostanttarget=0;                % 0 for same target point 1 for ...
    rotating target point
M=4;                               %constellation size
doMquam=0;                          %Mquam
doShapeCode=1;
nerrortocount=100;               %number of error to count at each SNR
SNRrange=0:1:8;
SNRrangeex=0:1:8;
SNRrangetheo=0:1:14;
GraphPsrange=10^-6;
A=1;
%create object
approach=2;                     %approach=1 using sum table;
                                %approach=2 using mod sum ;
                                %approach=3 dr Sodha's approach
shapecode_object=shapecode(docnostanttarget,M,A,approach);
shapecode_object.setdecoderinfo(1);                % 1=plot trellis ...
    structure of decoder (only for qpsk)
shapecode_object.resetshapecoder();
shapecode_object.resetshapedecoder();
R=1;                           %rate of uncoded symbol
Eb=2/3*A^2*(M-1)/(shapecode_object.Nbits*R)                           ...
    %energy per bit

Es=shapecode_object.Nbits*Eb*R; %Eneregy of uncoded symbol

Esscale=sqrt(3*Es/(2*(M-1)));
Rcoded=.5;                         %rate of coded symbol
%Escoded=sqrt(shapecode_object.Nbits*Eb*Rcoded*3/(2*(M-1)));        ...
    %Coded Amplitude
Escoded=2/3*(shapecode_object.M-1)*A^2;
%ploting thoeritical SNR (Plots are consistant with Figure 5-2-16 Digital ...
    Communication By John G.Proakis)
```

```
snrctr=0;

for SNR=SNRrangetheo
    %fprintf('Theoritical SNR=%d\n',SNR)
    snrctr=snrctr+1;
    N0=Eb*10^(-SNR/10);
    EbN0=Eb/N0;
    theorprobQPSK(snrctr) = 1 - (1-qf(sqrt(2*EbN0)))^2;
    %theorprobBPSK(snrctr) = qf(sqrt(2*EbN0));
    theoprobMquam(snrctr)=(1-(1-2*(1-sqrt(1/M))....
               *qf(sqrt((3/(M-1))*shapecode_object.Nbits*EbN0)))^2);
end

legendstrs={};
lct=1;
legendstrs{lct} = 'QPSK(th) symbol';
%legendstrs{lct+1} = 'BPSK(th) symbol';
%legendstrs{lct+2} = '64QAM(th) symbol';
%legendstrs{lct+1} = '16QAM(th) symbol';
lct=2;
if (doMquam)
    legendstrs{lct} = '16QAM(ex) symbol';
    lct=lct+1;
    legendstrs{lct}='16QAM(ex) bit';
    lct=lct+1;
end

if (doShapeCode)
    legendstrs{lct} = 'SPTC QPSK(symbol) state 10';
    lct=lct+1;
    legendstrs{lct} = 'SPTC QPSK(bit) state 10';
end
figure;
semilogy(SNRrangetheo,theorprobQPSK,'b:','linewidth',2);hold on;
%semilogy(SNRrangetheo,theorprobBPSK,'g:','linewidth',2);
%semilogy(SNRrangetheo,theoprobMquam,'k:','linewidth',2);
xticks(SNRrangetheo)
yticks([10^-6 10^-5 10^-4 10^-3 10^-2 10^-1 10^-0])
ylim([GraphPsrange 10^-0])
set(gca,'fontsize',15)
set(gcf, 'Position',  [2357 152 821 624])
xlabel('SNR_{db}')
ylabel('P_s')
grid on
%ploting Uncoded SNR (Plots are consistant with Figure 5-2-16 Digital ...
    Communication By John G.Proakis)
if (doMquam)
    snrctr=0;
    for SNR=SNRrangeex
        fprintf('Experimental SNR=%d\n',SNR)
        snrctr=snrctr+1;
        N0=Eb*10^(-SNR/10);
        EbN0=Eb/N0;
        sigma=sqrt(N0/2);
        errctr(snrctr)=0;
        errctr_bits(snrctr)=0;
```

```
        nsyms(snrctr)=0;
        while(errctr(snrctr)<nerrortocount)
            nsyms(snrctr)=nsyms(snrctr)+1;

            bits=randombits(shapecode_object.Nbits);
            symno=Binary2Decimal(bits);

            %constpts=Esscale*shapecode_object.const_pts(:,symno+1);
            constpts=shapecode_object.const_pts(:,symno+1);


            noise=sigma*randn(1,2);
            r(1)=constpts(1)+noise(1);
            r(2)=constpts(2)+noise(2);

            decodesymno=mquamdemod(r,shapecode_object.const_pts,....
            shapecode_object.const_axis);

            if(decodesymno≠symno)
                errctr(snrctr)=errctr(snrctr)+1;
                errctr_bits(snrctr) = errctr_bits(snrctr) + sum(bits ≠ ...
                    fliplr(de2bi(decodesymno,shapecode_object.Nbits)));
            end
        end
    end
end  %if do Mquam

if (doMquam)
   semilogy(SNRrangeex,errctr ./ nsyms,'r','linewidth',2);
   semilogy(SNRrangeex,(errctr_bits./(shapecode_object.Nbits*nsyms)),...
        'r:','linewidth',2);
end


Eb=Escoded/(Rcoded*log2(shapecode_object.M))

if (doShapeCode)
    snrctr=0;
    for SNR=SNRrange
        fprintf('ShapeCode SNR=%d\n',SNR)

        snrctr=snrctr+1;
        %N0=Eb*10^(-(SNR-10*log10(Rcoded))/10);
        N0=Eb*10^(-(SNR)/10);
        EbN0=Eb/N0;
        sigma=sqrt(N0/2);                 %calculate sigma for given SNR
        errctr_coded(snrctr)=0;
        errctr_bits_coded(snrctr)=0;
        nsyms_coded(snrctr)=0;
        while (errctr_coded(snrctr)<nerrortocount)
            nsyms_coded(snrctr)=nsyms_coded(snrctr)+1;    %calculate ...
                number of genarted symbols


            if(0)
```

```
            if (rem(nsyms_coded(snrctr),10000)==0 && ...
                (nsyms_coded(snrctr)≥ 10000))
                fprintf('number of symbol=%d \n',nsyms_coded(snrctr))
            end
            end


            bits=randombits(shapecode_object.Nbits);      %genarte random ...
                bits
            [sym1,sym2]=shapecode_object.encode(bits);    %encode the bits

            shapecode_object.bitsqueuein(bits);           %save the bits ...
                to compare the results

            %s1=Escoded*shapecode_object.const_pts(:,sym1+1);     ...
                %genrate signal points to transmit (symbol)
            %s2=Escoded*shapecode_object.const_pts(:,sym2+1);     ...
                %%genrate signal points to transmit (parity)
            s1=shapecode_object.const_pts(:,sym1+1);
            s2=shapecode_object.const_pts(:,sym2+1);

            r1=s1+sigma*randn(2,1);                             %add noise
            r2=s2+sigma*randn(2,1);                             %add noise

            %[bitsout,decodeout]=shapecode_object.viterbishapedecode...
        (r1/Escoded,r2/Escoded);   %decode
            [bitsout,decodeout]=shapecode_object.viterbishapedecode(r1,r2);

            if(decodeout)
                inbits=shapecode_object.bitqueueout();

                if(0)
                if (isequal(inbits,bitsout)==0)

                    %save('data_shapecode.mat')      %save workspace to file
                    fprintf('\nCoded SNR=%d Error ...
                        No=%d\n',SNR,errctr_coded(snrctr)+1)
%                   fprintf('InputBits=%d%d%d%d ...
    OutputBits=%d%d%d%d\n',inbits(1),inbits(2),....
%           inbits(3),inbits(4),bitsout(1),bitsout(2),bitsout(3),bitsout(4));
                end
                end



                errctr_coded(snrctr)=errctr_coded(snrctr)+any(inbits ≠ ...
                    bitsout);           %calculate number of symbol error
                errctr_bits_coded(snrctr) = errctr_bits_coded(snrctr) + ...
                    sum(inbits ≠ bitsout);     %calculate number of bit error
            end

%             if (errctr_coded(snrctr)≠0 && ...
    (errctr_coded(snrctr)/nsyms_coded(snrctr))≤GraphPsrange)
%                 break
%             end
           % errctr_coded(snrctr)
```

```
        end %while
%         if ((errctr_coded(snrctr)/nsyms_coded(snrctr))≤GraphPsrange)
%             break;
%         end

if (doShapeCode)

    semilogy((0:1:(length(errctr_coded)-1)),....
    (errctr_coded./nsyms_coded),'c','linewidth',2);
    semilogy((0:1:(length(errctr_coded)-1)),....
    (errctr_bits_coded./(shapecode_object.Nbits*nsyms_coded)),....
    'c:','linewidth',2)
    legend(legendstrs)
    pause(0.2)
    %saveas(gcf,'64QAMshapecodeBERrate12approach2','pdf');
end


    end  %for SNRRange
end  %doshapecode

toc
%saveas(gcf,'qpskshapecodeBERrate12approach2state10mult3march20','pdf');
%saveas(gcf,'16quamshapecodeBERrate12approach3march17','pdf');
%save('data_shapecodeqpskapproach2state10mult3march20.mat')
```

### Q function

```
function q=qf(x)
q=.5*erfc(x/sqrt(2));
```

### Function for Binary to Decimal Conversion

```
function symbol=Binary2Decimal(k)
symbol=0;
for ind=1:1:length(k)
    symbol=symbol+k(ind)*2^(length(k)-ind);
end
```

### Decoder for Uncoded QAM

```
function symbol=mquamdemod(r,const_ptsf,const_axis)
[¬,Ix]=min(abs(r(1)-const_axis));
[¬,Iy]=min(abs(r(2)-const_axis));
```

```
indxx=const_axis(Ix);
indyy=const_axis(Iy);
d=[indxx indyy];
[¬,Locb]=ismember(const_ptsf',d,'rows');
symbol=find(Locb==1)-1;
```

### Random Bit Generator Function

```
function bits=randombits(k)
bits=double(rand(1,k)≥.5);
```

### Shape Code Object

```
classdef shapecode< handle
    properties (Constant)
        viterbiwindowwidth=20;
    end
    properties
        M
        A
        Nstate
        Nepochs
        Nbits
        cumrotoffset
        fromtodist
        const_pts
        const_axis
        xcycle
        sumrotcount                    % cumulation of rotations(states)
        epoch
        doconstanttarget                % whether target rotation moves
        trellisoutput
        inputbits
        fromtostate
        metrics
        nextmetrics
        epoch_decode
        prevstatearray
        front
        countbranches
        beststate
        bitqueue
        bitqueuefront
        bitqueueback
        rotA
        vec
        AddRot
        testbits
```

```
        approach
        multipliedfactor
    end

%-----------------------------------------------------------------------
    methods
        function obj=shapecode(doconstanttarget,M,A,approach)
            obj.A=A;
            obj.doconstanttarget=doconstanttarget;
            obj.M=M;
            obj.Nstate=obj.M;
            obj.Nstate=6;
            obj.Nepochs=obj.M;
            obj.Nbits=log2(obj.M);
            obj.approach=approach;
            obj.multipliedfactor=1;
            %obj.multipliedfactor=1;
            if (obj.approach==3 && obj.M==16)
            %%% Dr Sodha's constellation point for 16 quam

             obj.const_pts=[-3 -3;-3 -1;-3 3;-3 1;-1 -3;-1 -1;-1 3;-1 1;3 ...
                 -3;3 -1;3 3;3 1;1 -3;1 -1;1 3;1 1]';  %points in the ...
                 constellation
             obj.const_axis=[-3 -1 3 1];


            elseif (obj.approach==3 && obj.M==4)
            %%% Dr Sodha's constellation point for 16 quam

             obj.const_pts=[-1 -1;1 -1;-1 1;1 1]';  %points in the ...
                 constellation
             obj.const_axis=[-1 1];
            else

            %%% My constellation point for square M quam

            obj.const_axis=-(2^(obj.Nbits/2)-1):2:2^(obj.Nbits/2)-1;
            obj.const_pts=[kron(ones(1,2^(obj.Nbits/2)),obj.const_axis); ...
                kron(obj.const_axis,ones(1,2^(obj.Nbits/2)))];
            end
            if (obj.M==16)
            obj.rotA=[0 7 15 8 1 6 14 9 3 4 12 11 2 5 13 10];      %from ...
                dr Sodha's code
            obj.vec=[2 6 14 10 11 15 7 3 1 5 13 9 8 12 4 0];       %from ...
                dr Sodha's code
            end
            obj.const_pts=obj.const_pts*obj.A;
            obj.const_axis=obj.const_axis*obj.A;

            if (obj.M==4)
            obj.rotA=[0 1 3 2];      %from dr Sodha's code
            obj.vec=[2 3 1 0];       %from dr Sodha's code
            end

            obj.AddRot=CodeTrellis(obj);                           % sum ...
                table from dr Sodha's code
```

```
            %obj.cumrotoffset=hankel(0:1:obj.M-1,[obj.M-1 0:1:obj.M-2]);
            obj.cumrotoffset=obj.AddRot;
            %obj.fromtodist=hankel(0:1:obj.M-1,[obj.M-1 0:1:obj.M-2]);
            obj.fromtodist=[0 7 15 8 1 6 14 9 3 4 12 11 2 5 13 10 ;
                            9 0 8 1 10 15 7 2 12 13 5 4 11 14 6 3 ;
                            1 8 0 9 2 7 15 10 4 5 13 12 3 6 14 11;
                            8 15 7 0 9 14 6 1 11 12 4 3 10 13 5 2;
                            15 6 14 7 0 5 13 8 2 3 11 10 1 4 12 9;
                            10 1 9 2 11 0 8 3 13 14 6 5 12 15 7 4;
                            2 9 1 10 3 8 0 11 5 6 14 13 4 7 15 12;
                            7 14 6 15 8 13 5 0 10 11 3 2 9 12 4 1;
                            13 4 12 5 14 3 11 6 0 1 9 8 15 2 10 7;
                            12 3 11 4 13 2 10 5 15 0 8 7 14 1 9 6;
                            4 11 3 12 5 10 2 13 7 8 0 15 6 9 1 14;
                            5 12 4 13 6 11 3 14 8 9 1 0 7 10 2 15;
                            14 5 13 6 15 4 12 7 1 2 10 9 0 3 11 8;
                            11 2 10 4 12 1 9 4 14 15 7 6 13 0 8 5;
                            3 10 2 11 4 9 1 12 6 7 15 14 5 8 0 13;
                            6 13 5 14 7 12 4 15 9 0 2 1 8 11 3 0 ];

            if (¬doconstanttarget)
                 if (obj.approach==3)
                 obj.xcycle=obj.rotA;           %set the target point
                 else
                 obj.xcycle=0:1:obj.M-1;
                 end
            else
                obj.xcycle=0;
            end

            obj.epoch=0;                       %number of epoch
            obj.sumrotcount=0;                 %total number of previous ...
                rotation


        end %end of contructor
%---------------------------------------------------------------------
% set AddRot Matrix



%          function temp=IntializeVec(obj,N)
%              for r=0:1:N-1
%                  for c=0:1:N-1
%                      temp(r+1,c+1)=0;
%                  end
%              end
%          end

        function temp=CircularShift(obj,A,shift)
            temp=ones(1,length(obj.vec));

            for r=0:1:length(A)-1

```

```
                if ((r+shift)<obj.M)

                    temp(r+1)=A(r+1+shift);
                else

                    temp(r+1)=A(r+1+shift-obj.M);
                end
            end
        end

        function AddRot=InsertVector(obj,AddRot,A,index,shift)
            newvec=obj.CircularShift(A,shift);
            for c=0:1:length(A)-1
                AddRot(index,c+1)=newvec(c+1);
            end
        end


        function temp=CodeTrellis(obj)
            temp=ones(obj.M,obj.M);
            for ind=0:1:length(obj.vec)-1
                temp=obj.InsertVector(temp,obj.vec,obj.vec(ind+1)+1,ind);
            end
        end
%--------------------------------------------------------------
       function resetshapecoder(obj)

          obj.epoch=0;
          obj.sumrotcount=0;
       end
 %-------------------------------------------------------------
     function resetshapedecoder(obj)
        obj.epoch_decode=0;
        obj.front=0;               % index into prevstatearray
        obj.countbranches=0;
        obj.bitqueuefront=0;
        obj.bitqueueback=0;
     end
%--------------------------------------------------------------
     function m=computemetric(obj,r1,r2,output)
        out11=output(1,1);
        out12=output(1,2);
        out21=output(2,1);
        out22=output(2,2);
        n1=(r1(1)-out11)^2+(r1(2)-out21)^2;
        n2=(r2(1)-out12)^2+(r2(2)-out22)^2;
        m=n1+n2;
     end
 %-------------------------------------------------------------
         function [bitsout,decodeout]=viterbishapedecode(obj,r1,r2)
             beststatemetric=inf;
             for state=0:obj.Nstate-1
                nextmetricbest=inf;
                for prevstate=obj.fromtostate(:,state+1,obj.epoch_decode+1)'
                    m=obj.computemetric(r1,r2,....
               obj.trellisoutput(:,:,prevstate+1,state+1,obj.epoch_decode+1));
```

```
                        nextmetric=obj.metrics(prevstate+1)+m;

                        if (nextmetric<nextmetricbest)
                            nextmetricbest=nextmetric;
                            prevstatebest=prevstate;
                            statebest=state;
                        end
                    end %for prevstate
                    if(nextmetricbest<beststatemetric)
                         beststatemetric=nextmetricbest;
                         obj.beststate=state;
                    end
                    obj.nextmetrics(state+1)=nextmetricbest;
                    obj.prevstatearray(state+1,obj.front+1)=prevstatebest;
                 end %for state
                 obj.metrics=obj.nextmetrics;
                 obj.countbranches=obj.countbranches+1;
                 decodeout=0;
                 bitsout=[0;0];


               if(obj.countbranches≥obj.viterbiwindowwidth)
                   % start from best state and work back
               state=obj.beststate;
               idx=obj.front;
               epoch1=obj.epoch_decode;

               for i=1:obj.viterbiwindowwidth
                   prevstate=obj.prevstatearray(state+1,idx+1);   % go back ...
                       to previous state
                   idx=mod(idx-1,obj.viterbiwindowwidth);         % circular ...
                       indexing
                   lastepoch=epoch1;
                   epoch1=mod(epoch1-1,obj.Nepochs);

                   if(i≠obj.viterbiwindowwidth)
                      state=prevstate;
                   end
               end
              bitsout=obj.inputbits(:,prevstate+1,state+1,lastepoch+1);
              decodeout=1;
              end
              obj.front=mod(obj.front+1,obj.viterbiwindowwidth);
              obj.epoch_decode=mod(obj.epoch_decode+1,obj.Nepochs);
           end

     %--------------------------------------------------------

             function bitsout=viterbishapedecodeflush(obj)
                 bitsout = [];
                 state = obj.beststate;

                 epoch1 = mod(obj.epoch_decode-1,obj.Nepochs);
                 idx = mod(obj.front-1,obj.viterbiwindowwidth);

                 for i = 1:obj.viterbiwindowwidth-1
```

```
                 prevstate = obj.prevstatearray(state +1, idx +1);   % go ...
                     back to previous state
                 lastepoch = epoch1;
                 bits1 = obj.inputbits(:,prevstate +1,state +1,lastepoch +1);
                 idx = mod(idx - 1,obj.viterbiwindowwidth);   % circular ...
                     indexing
                 epoch1 = mod(epoch1 - 1,obj.Nepochs);
                 bitsout = [bits1 bitsout];
                state = prevstate;
               end

        end

    %-------------------------------------------------------
         function bitsqueuein(obj,bits)
            obj.bitqueue(:,obj.bitqueuefront+1)=bits(:);
            obj.bitqueuefront=mod(obj.bitqueuefront+1,obj.viterbiwindowwidth);
         end
    %------------------------------------------------------

         function bits=bitqueueout(obj)
             bits=obj.bitqueue(:,obj.bitqueueback+1);
             obj.bitqueueback=mod(obj.bitqueueback+1,obj.viterbiwindowwidth);
         end


    %------------------------------------------------------
           function symbol=Bin2DecClass(obj,k)
             symbol=0;
                for ind=1:1:length(k)
                 symbol=symbol+k(ind)*2^(length(k)-ind);
                end
            end

    %------------------------------------------------------
             function [symnosave,coderot]=encode(obj,bits)
                        symno=obj.Bin2DecClass(bits);
                        symnosave=symno;


    %                 %%%% Encoding using Dr Sodha's approach
                          if (obj.approach==3)
                               newpoint=obj.AddRot(symno+1,obj.sumrotcount+1);

                               coderot=obj.rotA(newpoint+1);
                               %coderot is code symbol signal point

                               %update total number of previous rotation
    %                            if (obj.sumrotcount+coderot≥obj.M)
    %                              obj.sumrotcount=obj.sumrotcount+coderot-obj.M;
    %                            else
    %                              obj.sumrotcount=obj.sumrotcount+coderot;
    %                            end
                            obj.sumrotcount=mod(obj.sumrotcount+coderot,obj.M);
                               %obj.sumrotcount is the total rotation
                          end
```

```
%                  %%%%% end of Dr Shodha's approach


                 %%%% approach 1
                 if (obj.approach==1)
                    symno=obj.cumrotoffset(symno+1,obj.sumrotcount+1);
                 end

                 %%%% approach 2
                  if (obj.approach==2)
                    symno=mod(symno+obj.sumrotcount,obj.Nstate);
                  end

                 %%%%  used in  both approach 1 and 2:
                 if (obj.approach ==1 || obj.approach ==2)
                    if(¬obj.doconstanttarget)
                        xcyclecount=obj.epoch;
                    else
                        xcyclecount=0;
                    end

                    tosym=obj.xcycle(xcyclecount+1);
                 end
                %approach 1
                if (obj.approach==1)
                    coderot=obj.fromtodist(tosym+1,symno+1);
                end

                %approach 2
                if (obj.approach==2)
                    coderot=mod(symno+tosym,obj.Nstate);
                end


                %%%% used in both appraoch 1 and 2:
                if (obj.approach ==1 || obj.approach ==2)
                    obj.sumrotcount=mod(obj.multipliedfactor*....
            (obj.sumrotcount+coderot),obj.Nstate);   %to state
                    coderot=mod(coderot,obj.M);
                    obj.epoch=mod(obj.epoch+1,obj.Nepochs);                  ...
                        % move to the next epoch
                end
        end  %encode

%----------------------------------------------------------
        function testbits=CreateTestBitsClass(obj)
              testbits=zeros(1,obj.M*obj.Nbits);
              ctrr=0;
              ind=1;
              for q=0:1:obj.M-1
                while(q≠0)
                    r=rem(q,2);
                    q=fix(q/2);
                    testbits(ind*obj.Nbits-ctrr)=r;
                    ctrr=mod(ctrr+1,obj.Nbits);
                end
```

```
                ctrr=0;
                ind=ind+1;
             end
        end

%---------------------------------------------------------
        function states=allstateClass(obj)
            states=ones(obj.Nstate,obj.Nstate);
            for ind1=1:1:obj.Nstate
                for ind2=1:1:obj.Nstate
                    states(ind1,ind2)=ind1-1;
                end
            end
        end
%-------------------------------------------------------


 function obj=setdecoderinfo(obj,doplot)
  % set trellis information for decode using viterbi;
  % trellis is computed using each poissble from-state(initial
  % number of total rotation(sumrotcount)) and for each from-state
  % using every possible combination of bits;
  %algorithm
  if (doplot)
  figure(doplot)
  clf;
  set(gca,'ydir','reverse');
  yplotsep = 1;
  xplotsep = 1;
  markersize = 15;
  mulist = [0.15 0.45 0.60 0.75];
  xcyclesym = ['A' 'B' 'C' 'D'];
  end
  %initialize trellis parameters
  allstate=obj.allstateClass();
  obj.trellisoutput=Inf(2,2,obj.Nstate,obj.Nstate,obj.Nepochs);
  obj.inputbits=zeros(obj.Nbits,obj.Nstate,obj.Nstate,obj.Nepochs);
  obj.metrics=zeros(obj.Nstate,1);
  obj.nextmetrics=zeros(obj.Nstate,1);
  obj.fromtostate=zeros(obj.Nstate,obj.Nstate,obj.Nepochs);
  bj.testbits=obj.CreateTestBitsClass();  %genarate bit sequence ...
      [example, for M=4 bit sequence= 0 0; 0 1; 1 0; 1 1]
  for epoch1=0:obj.Nepochs-1
   if(doplot)
   plottime = epoch1; plotx1 = plottime; plotx2 = plottime + xplotsep;
   text(0.5*(plotx1 + plotx2),-0.2,xcyclesym(epoch1 +1)); hold on;
   end
   if (obj.doconstanttarget)
    xcyclecount=0;
   else
    xcyclecount=epoch1;
   end
   tosym=obj.xcycle(xcyclecount+1);
   for state=0:obj.Nstate-1
    bitctr=0;
    sumrotcountsave=state;
```

```
    randcolor=rand(1,3);
    while(bitctr<(obj.M*obj.Nbits))
     obj.sumrotcount=sumrotcountsave;
     symno=obj.Bin2DecClass(obj.testbits(bitctr+1:bitctr+obj.Nbits));
     symnosave=symno;
     %%%%% Dr Sodha's approach:
     if (obj.approach==3)
      newpoint=obj.AddRot(symno+1,obj.sumrotcount+1);
      coderot=obj.rotA(newpoint+1);
      %update total number of previous rotation
      sumrotcount=mod(obj.sumrotcount+coderot,obj.Nstate);
%     if (obj.sumrotcount+coderot≥obj.M)
%      sumrotcount=obj.sumrotcount+coderot-obj.M;           %to state
%     else
%      sumrotcount=obj.sumrotcount+coderot;                 %to state
%     end
     end
     %%%% End of Dr Sodha's approach:
     %%% approach 2
          if (obj.approach==2)
            symno=mod(symno+obj.sumrotcount,obj.Nstate);
            coderot=mod(symno+tosym,obj.Nstate);
           end
           %%%%% end of approach 2
           %%%% approach 1
           if (obj.approach==1)
           symno=obj.cumrotoffset(symno+1,obj.sumrotcount+1);
           coderot=obj.fromtodist(tosym+1,symno+1);
          end
          %%%%%end of approach 1
          %%% used in both approach 1 and 2:
          if (obj.approach ==1 || obj.approach ==2)
           sumrotcount=mod(obj.multipliedfactor*...
         (obj.sumrotcount+coderot),obj.Nstate);
          end
          coderot=mod(coderot,obj.M);
          fromstate=state;
          tostate=sumrotcount;
          obj.trellisoutput(:,1,fromstate+1,tostate+1,epoch1+1)=...
                     obj.const_pts(:,symnosave+1);
          obj.trellisoutput(:,2,fromstate+1,tostate+1,epoch1+1)=...
             obj.const_pts(:,coderot+1);
          obj.inputbits(:,fromstate+1,tostate+1,epoch1+1)=...
                     obj.testbits(bitctr+1:bitctr+obj.Nbits)';
          if (doplot)
           fromy = sumrotcountsave*yplotsep;
           toy =  sumrotcount*yplotsep;
           plot([plotx1 plotx2],[fromy,toy],'color',randcolor);
           plot(plotx1,fromy,'.',...
         'markersize',markersize,'color','k');
           plot(plotx2,fromy,'.','markersize',markersize,'color','k');
           %str = sprintf('%d%d/%d%d',obj.testbits(bitctr+1),...
       obj.testbits(bitctr+2),symnosave,coderot);
           if (obj.Nstate==4)
            str = sprintf('%d/%d',symnosave,coderot);
            mu = mulist(state +1);
```

```
                textx = (1-mu)*plotx1 + mu*plotx2;
                texty = (1-mu)*fromy + mu*toy;
                htext = text(textx,texty,str);
                tangle = -(180/pi)*atan((toy - fromy)/(plotx2 - plotx1));
                % - sign since y coords reversed
                set(htext,'VerticalAlignment','bottom');
                set(htext,'rotation',tangle);
               end
               %hAxes = gca;     %Axis handle
               %Changing 'LineStyle' to 'none'
               yticks([0:1:obj.Nstate-1])
               xticks([0 1])
               xlim([0 1])
               hold on
               plot([0 0],[0 obj.Nstate-1],'w')
               %ax1 = gca;
               %yruler = ax1.YRuler;
               %yruler.Axle.Visible = 'off';
               %xruler = ax1.XRuler;
               %xruler.Axle.Visible = 'off';
              end % if doplots
              bitctr=bitctr+obj.Nbits;
            end %while
           end% for state
           obj.fromtostate(:,:,epoch1+1)=allstate;
          end  % for epoch1
          obj.prevstatearray=zeros(obj.Nstate,obj.viterbiwindowwidth);
          obj.epoch_decode=0;
          obj.bitqueue=zeros(obj.Nbits,obj.viterbiwindowwidth);
          obj.bitqueuefront=0;
         obj.bitqueueback=0;
       end %setdecoderinfo
   %-----------------------------------------------------------------
    end
   end
```

## B.2 Julia Code

### Julia Code for Signal Point Target Code

```
tic()
using SpecialFunctions
using PyPlot
using ToeplitzMatrices
using LaTeXStrings

function qf(x)
    return .5*erfc(x/sqrt(2))
end

```

```
function mquamdemod(r,const_ptsf,const_axis)
  indxx=const_axis[indmin(abs.(r[1]-const_axis))]
  indyy=const_axis[indmin(abs.(r[2]-const_axis))]
  d=[indxx,indyy]
  lia=Int[ d == const_ptsf[:,i] for i=1:size(const_ptsf,2) ]
  symbol=(findin(lia,1))[1]
    return symbol-1
  end

function bin2dec(k)
  symbol=0
  for i=1:1:length(k)
    symbol=symbol+k[i]*2^(length(k)-i)
  end
  return symbol
end

function randombits(k)
  return bits=Int.(bitrand(k))
end #randombit
function createtestbits(M,Nbits)
  testbits=zeros(Int64,1,M*Nbits)
  ctrr=0
  i=1
  for q in collect(0:1:M-1)
    while (q!=0)
     q,r=divrem(q,2)
     testbits[i*Nbits-ctrr]=r
     ctrr=mod(ctrr+1,Nbits)
   end #while
     ctrr=0
     i=i+1
  end #for
  return testbits
end #createtestbits

doconstanttarget=false
doplot=false
M=64
Viterbiwindowwidth=10
doMquam=false
doshapecode=true
SNRrange=0:10
SNRrangetheo=0:15
GraphPsrange=10.0^-5
Nstate=M
Nepochs=M
Nbits=Int64(log2(M))
allstates=ones(Int64,Nstate,Nstate)
for i=1:1:Nstate
  for j=1:1:Nstate
  allstates[i,j]=i-1
  end
end
testbits=createtestbits(M,Nbits)
const_axis=collect(-(2^(Nbits/2)-1):2:(2^(Nbits/2)-1))
```

```
const_pts=Int.([kron(Int.(ones(2^(Nbits/2))),const_axis) ...
    kron(const_axis,Int.(ones(2^(Nbits/2))))]')

println("------------------------------------------------------------------")
if (doshapecode)
  #SNRrangeShapecode=0:10
  if ¬(doconstanttarget)
    xcycle=collect(0:1:(M-1))
  else
    xcycle=0
  end

#const_pts

  cumrotoffset=Int64.(Hankel(float.(collect(0:1:M-1)),float.([M-1; ...
      collect(0:1:M-2)])))
  fromtodist=Int64.(Hankel(float.(collect(0:1:M-1)),float.([M-1; ...
      collect(0:1:M-2)])))
  xcyclesym=['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J' 'K' 'L' 'M' 'N' 'O' 'P']
  trellisoutput=zeros(Int64,2,2,Nstate,Nstate,Nepochs)
  inputbits=zeros(Int64,Nbits,Nstate,Nstate,Nepochs)
  metrics=zeros(Nstate,1)
  nextmetrics=zeros(Nstate,1)
  prevstatearray=zeros(Int64,Nstate,Viterbiwindowwidth)
  fromtostate=zeros(Int64,Nstate,Nstate,Nepochs)
  bitqueue=zeros(Int64,Nbits,Viterbiwindowwidth)
  bitqueuefront=0
  bitqueueback=0

#------------------------------------------------------------------
  mutable struct ShapeCodeFun
  M::Int64
  Viterbiwindowwidth::Int64
  Nstate::Int64
  Nepochs::Int64
  Nbits::Int64
  fromtodist
  cumrotoffset
  const_pts
  xcycle
  sumrotcount
  epoch
  doconstanttarget::Bool
  trellisoutput
  inputbits
  fromtostate
  metrics
  nextmetrics
  epoch_decode
  prevstatearray
  front
  countbranches
  beststate
  bitqueue
  bitqueuefront
```

```
  bitqueueback
  end
#-----------------------------------------------------------------------------

  function setdecoderinfo(self::ShapeCodeFun,doplot,allstates,testbitsM)
  if (doplot)
    xcyclesym=['A' 'B' 'C' 'D' ];
    yplotsep=1
    xplotsep=1
    markersize=15
    mulist=[0.15; 0.45; 0.60; 0.75]
    fig,ax=subplots()
    ax[:axis]("off")
    ax[:invert_yaxis]()
  end
  testbits=testbitsM

  for epoch1=0:self.Nepochs-1
    if (doplot)
      plottime=epoch1
      plotx1=plottime
      plotx2=plottime+xplotsep


      ax[:text](plotx2,-.3,"$(xcyclesym[epoch1+1])")
    end
    if (self.doconstanttarget)
      xcyclecount=0
    else
      xcyclecount=epoch1
    end

    tosym=self.xcycle[xcyclecount+1]

    for state=0:self.M-1
      bitctr=0
      sumrotcountsave=state
      while (bitctr<(self.M*self.Nbits))
        self.sumrotcount=sumrotcountsave
        symno=bin2dec(testbits[bitctr+1:bitctr+self.Nbits])
        #println(symno) function [symnosave,coderot]=encode(obj,bits)
        symnosave=symno
        symno=mod(symno+self.sumrotcount,self.M)
        coderot=mod(symno+tosym,self.M)
        sumrotcount=mod(self.sumrotcount+coderot,self.M)
        fromstate=state
        tostate=sumrotcount
        self.trellisoutput[:,1,fromstate+1,tostate+1,epoch1+1]=...
            self.const_pts[:,symnosave+1]
        self.trellisoutput[:,2,fromstate+1,tostate+1,epoch1+1]=...
            self.const_pts[:,coderot+1]
        self.inputbits[:,fromstate+1,tostate+1,epoch1+1]=...
            testbits[bitctr+1:bitctr+self.Nbits]
        if (doplot)
                    fromy=sumrotcountsave*yplotsep
                    toy=sumrotcount*xplotsep
```

```

                        ax[:plot]([plotx1; plotx2],[fromy; toy],"k")
                        ax[:scatter](plotx1,fromy,c=:black)
                        ax[:scatter](plotx2,fromy,c=:black)

                        mu=mulist[state+1]

                        textx=(1-mu)*plotx1+mu*plotx2
                        texty=(1-mu)*fromy+mu*toy
                        tangle=(-180/pi)*atan((toy - fromy)/(plotx2 - plotx1))
                        ax[:text](textx,texty,"$(testbits[bitctr+1])...
                $(testbits[bitctr+2])/$(symnosave)$(coderot)",...
                rotation=tangle,horizontalalignment="left",...
                verticalalignment="bottom", rotation_mode="anchor")
            end  #doplot
            #println(bitctr)
            bitctr=bitctr+self.Nbits
          end #while
        end #end state
        self.fromtostate[:,:,epoch1+1]=allstates
      end #end epoch1
      end #end setdecoderinfo
    #----------------------------------------------------------------------

      function resetshapecoder(self::ShapeCodeFun)
      self.epoch=0
      self.sumrotcount=0
      end #end resetshapecoder
    #----------------------------------------------------------------------
      function resetshapedecoder(self::ShapeCodeFun)
      self.epoch_decode=0
      self.front=0
      self.countbranches=0
      self.bitqueuefront=0
      self.bitqueueback=0
      end  #end resetshapedecoder
    #----------------------------------------------------------------------

      function encode(self::ShapeCodeFun,bits)
      symno=bin2dec(bits)
      symnosave=symno
      symno=mod(symno+self.sumrotcount,self.M)
      if (¬self.doconstanttarget)
        xcyclecount=self.epoch
      else
        xcyclecount=0
      end
      tosym=self.xcycle[xcyclecount+1]
      coderot=mod(symno+tosym,self.M)
      self.sumrotcount=mod(self.sumrotcount+coderot,self.M)
      self.epoch=mod(self.epoch+1,self.Nepochs)
      return symnosave,coderot
      end #end encode

      function computemetric(r1,r2,output)
      out11=output[1,1]
```

```
out12=output[1,2]
out21=output[2,1]
out22=output[2,2]
n1=(r1[1]-out11)^2+(r1[2]-out21)^2
n2=(r2[1]-out12)^2+(r2[2]-out22)^2
m=n1+n2
return m
end #end computemetric

function viterbishapedecode(self::ShapeCodeFun,r1,r2)
beststatemetric=Inf
for state=0:self.Nstate-1
  nextmetricbest=Inf
  for prevstate=self.fromtostate[:,state+1,self.epoch_decode+1]'
    m=computemetric(r1,r2,...
     self.trellisoutput[:,:,prevstate+1,state+1,self.epoch_decode+1])
    nextmetric=self.metrics[prevstate+1]+m
    if (nextmetric<nextmetricbest)
      nextmetricbest=nextmetric
      global prevstatebest=prevstate
      statebest=state
    end
  end #for prevstate
  if (nextmetricbest < beststatemetric)
     beststatemetric = nextmetricbest
     self.beststate = state
  end
  self.nextmetrics[state+1]=nextmetricbest
  self.prevstatearray[state+1,self.front+1]=prevstatebest
end #for state
self.metrics=copy(self.nextmetrics)
self.countbranches=self.countbranches+1
decodeout=0
bitsout=[0;0]
if (self.countbranches≥self.Viterbiwindowwidth)
  state=self.beststate
  idx=self.front
  epoch1=self.epoch_decode
  prevstate=[]
  lastepoch=[]
  for i=1:self.Viterbiwindowwidth
    prevstate=self.prevstatearray[state+1,idx+1]
    idx=mod(idx-1,self.Viterbiwindowwidth)
    lastepoch=epoch1
    epoch1=mod(epoch1-1,self.Nepochs)
    if (i!=self.Viterbiwindowwidth)
      state=prevstate
    end
  end # i
    bitsout=self.inputbits[:,prevstate+1,state+1,lastepoch+1]
    decodeout=1
end #if
self.front=mod(self.front+1,self.Viterbiwindowwidth)
self.epoch_decode=mod(self.epoch_decode+1,self.Nepochs)
return decodeout,bitsout
end #shape decoder viterbi
```

```

# -----------------------------------------------------------------------

  function viterbishapecodeflush(self::ShapeCodeFun)
  bitsout=zeros(Int64,self.Nbits,self.Viterbiwindowwidth-1)
  state=self.beststate
  epoch1=mod(self.epoch_decode-1,self.Nepochs)
  idx=mod(self.front-1,self.Viterbiwindowwidth)
  for i=1:self.Viterbiwindowwidth-1
    prevstate=self.prevstatearray[state+1,idx+1]
    lastepoch=epoch1
    bits1=self.inputbits[:,prevstate+1,state+1,lastepoch+1]
    idx=mod(idx-1,self.Viterbiwindowwidth)
    epoch1=mod(epoch1-1,self.Nepochs)
    bitsout[:,self.Viterbiwindowwidth-1-i]=bits1
    state=prevstate
  end #for
  end #viterbishapecodeflush
#-------------------------------------------------------------------------

  function bitqueuein(self::ShapeCodeFun,bits)
  self.bitqueue[:,self.bitqueuefront+1]=bits
  self.bitqueuefront = mod(self.bitqueuefront + 1, self.Viterbiwindowwidth)
  end

  function bitqueueout(self::ShapeCodeFun)
  bits=self.bitqueue[:,self.bitqueueback+1]
  self.bitqueueback = mod(self.bitqueueback + 1, self.Viterbiwindowwidth)
  return bits
  end

#create object of ...
    ShapeCodeFun--------------------------------------------------


  a=ShapeCodeFun(M,Viterbiwindowwidth,Nstate,Nepochs,Nbits,
              fromtodist,cumrotoffset,const_pts,xcycle,0,
              0,doconstanttarget,trellisoutput,inputbits,fromtostate,
        metrics,nextmetrics,0,prevstatearray,0,0,0,bitqueue,0,0)


  setdecoderinfo(a,doplot,allstates,testbits)
  resetshapecoder(a)
  resetshapedecoder(a)

end #doshapecode

nerrortocount=100

Eb=1
R=1
Es=Nbits*Eb*R
Esscale=sqrt(3*Es/(2*(M-1)))
Rcoded=.5
Escoded=sqrt(Nbits*Eb*Rcoded*3/(2*(M-1)))
theorprobQPSK=[]
```

```
theorprobBPSK=[]
theoprobMquam=[]
for SNR in SNRrangetheo
  println("snr=$(SNR)")
  N0=Eb*10^(-SNR/10)
  EbN0=Eb/N0
  push!(theorprobQPSK,(1 - (1-qf(sqrt(2*EbN0)))^2))
  push!(theorprobBPSK,(qf(sqrt(2*EbN0))))
  push!(theoprobMquam,...
    (1-(1-2*(1-sqrt(1/M))*qf(sqrt((3/(M-1))*Nbits*EbN0)))^2))
end

r=[0.0 0.0]
if (doMquam)
  snrctr=0
  errctr=[]
  nsyms=[]
  for SNR in SNRrange
    println("snr=$(SNR)")
    snrctr=snrctr+1
    N0=Eb*10^(-SNR/10)
    EbN0=Eb/N0
    #println("N0=$(N0) EbN0=$(EbN0)")
    sigma2=N0/2
    sigma=sqrt(sigma2)
    push!(errctr,0)
    push!(nsyms,0)

    while(errctr[snrctr]<nerrortocount)
      nsyms[snrctr]=nsyms[snrctr]+1
      bits=randombits(Nbits)
      symno=bin2dec(bits)
      constpts=Esscale*const_pts[:,symno+1]
      noise=sigma*randn(1,2)
      r[1]=constpts[1]+noise[1]
      r[2]=constpts[2]+noise[2]
      decodesymno=mquamdemod(r/Esscale,const_pts,const_axis)
      #println("decodesymno=$(decodesymno) symno=$(symno)")

      if (decodesymno != symno)
        errctr[snrctr]=errctr[snrctr]+1
        println("SNR=$(SNR):uncoded error no=$(errctr[snrctr])")
        #println("decodesymno=$(decodesymno) symno=$(symno)")
      end
     if (errctr[snrctr]!=0 && errctr[snrctr]/nsyms[snrctr]≤GraphPsrange)
        break
      end   #end break while
    end #while errctr
    if (errctr[snrctr]/nsyms[snrctr]≤GraphPsrange)
      break
    end #end brek for
  end #for snr
end #if(doqpsk)

if (doshapecode)
    errctr_coded=[]
```

```
    errctr_bits_coded=[]
    nsyms_coded=[]
    snrctr=0
    for SNR in SNRrange
      println("SNRShapeCode=$(SNR)")
      snrctr=snrctr+1
      N0=Eb*10^(-SNR/10)
      EbN0=Eb/N0
      sigma2=N0/2
      sigma=sqrt(sigma2)
      push!(errctr_coded,0)
      push!(errctr_bits_coded,0)
      push!(nsyms_coded,0)
      while(errctr_coded[snrctr]<nerrortocount)
        nsyms_coded[snrctr]=nsyms_coded[snrctr]+1
        bits1=randombits(Nbits)
        sym1,sym2=encode(a,bits1)
        bitqueuein(a,bits1)
        s1=Escoded*a.const_pts[:,sym1+1]
        s2=Escoded*a.const_pts[:,sym2+1]
        noise1=sigma*randn(2,1)
        noise2=sigma*randn(2,1)
        r1=s1+noise1
        r2=s2+noise2
        decodeout, bitsout=viterbishapedecode(a,r1/Escoded,r2/Escoded)
        if (decodeout==1)
          inbits=bitqueueout(a)
          if (inbits!=bitsout)
            println("SNR=$(SNR):coded error no=$(errctr_coded[snrctr])")
          end
          println("inbits=$(inbits) bitsout=$(bitsout)")
          errctr_coded[snrctr]=errctr_coded[snrctr]....
        +Int64(any(inbits!=bitsout))
          errctr_bits_coded[snrctr] = errctr_bits_coded[snrctr] + ...
              sum(Int64.(inbits .!= bitsout))
        end #decodeout
        if (errctr_coded[snrctr]!=0 && ...
             errctr_coded[snrctr]/nsyms_coded[snrctr]≤GraphPsrange)
          break
        end   #end break while
    end #while error count
    if (errctr_coded[snrctr]/nsyms_coded[snrctr]≤GraphPsrange)
      break
    end #end break for

  end #for SNRrangeShapecode
end #doshapecode
legendstr=[]
push!(legendstr,"QPSK(th)")
push!(legendstr,"BPSK(th)")
push!(legendstr,"$(M)QUAM(th)")

if (doMquam)
  push!(legendstr,"$(M)Quam(ex)")
end
if (doshapecode)
```

```
  push!(legendstr,"Shapecoded$(M)Quam(sym)")
  push!(legendstr,"Shapecoded$(M)Quam(bit)")
end
len=length(SNRrange)

fig1,ax1=subplots()
ax1[:semilogy](collect(0:1:length(theorprobQPSK)-1),theorprobQPSK,"g:")
ax1[:semilogy](collect(0:1:length(theorprobBPSK)-1),theorprobBPSK,"b:")
ax1[:semilogy](collect(0:1:length(theoprobMquam)-1),theoprobMquam,"k:")

if (doMquam)
  ax1[:semilogy](collect(0:1:length(errctr)-1),errctr./(nsyms),"r:")
end

if (doshapecode)
  ax1[:semilogy](collect(0:1:length(errctr_coded)-1),...
    errctr_coded./nsyms_coded,"c")
  ax1[:semilogy](collect(0:1:length(errctr_coded)-1),...
    errctr_bits_coded./(Nbits*nsyms_coded),"c:")
end

ax1[:legend](legendstr)
ax1[:set_ylim]([GraphPsrange,10.0^-0])
ax1[:grid](color="k", linestyle="--")
xlabel(L"(SNR)_{db}")
ylabel(L"P_s" )
toc()
Ps=-1*Int(log10(GraphPsrange))
savefig("BER$(M)QUAMSCEbSNR$(SNRrange[end])Ps$(Ps)rate12.pdf",dpi=1500)
```

### Code for Rate 2/3 (SIE)

```
trellisoutput=zeros(Int64,2,3,M,Nstate,Nstate,Nepochs)
inputbits=zeros(Int64,Nbits,2,M,Nstate,Nstate,Nepochs)
metrics=zeros(Nstate,1)
nextmetrics=zeros(Nstate,1)
prevstatearray=zeros(Int64,Nstate,Viterbiwindowwidth)
fromtostate=zeros(Int64,Nstate,Nstate,Nepochs)
bitqueue=zeros(Int64,Nbits,2,Viterbiwindowwidth)
bitqueuefront=0
bitqueueback=0

bestsymbolarray=zeros(Int64,Nstate,Viterbiwindowwidth)



function setdecoderinfo(self::ShapeCodeFun,doplot,allstates,testbitsM)
if (doplot)
  xcyclesym=['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J' 'K' 'L' 'M' 'N' ...
      'O' 'P']
  yplotsep=1
  xplotsep=1
```

```
    markersize=15
    mulist=[.062 .12 .18 .25 .31 .37 .43 .5 .562 .6245 .687 .75 .81 .87 ...
        .93 .98]
    fig,ax=subplots()
    ax[:axis]("off")
    ax[:invert_yaxis]()
  end
  testbits=testbitsM

  for epoch1=0:self.Nepochs-1
    if (doplot)
      plottime=epoch1
      plotx1=plottime
      plotx2=plottime+xplotsep


      ax[:text](plotx2,-.3,"$(xcyclesym[epoch1+1])")
    end
    if (self.doconstanttarget)
      xcyclecount=0
    else
      xcyclecount=epoch1
    end

    tosym=self.xcycle[xcyclecount+1]

    for state=0:self.M-1
      bitctr=0
      sumrotcountsave=state
      while (bitctr<(self.M*self.Nbits))
        bitssymbol1=testbits[bitctr+1:bitctr+self.Nbits]
        symno=bin2dec(bitssymbol1)
        symnosave=symno
        self.sumrotcount=sumrotcountsave
        bitctr1=0
        while(bitctr1<(self.M*self.Nbits))
          bitssymbol2=testbits[bitctr1+1:bitctr1+self.Nbits]

          symno2=bin2dec(bitssymbol2)     #conver secend symbol to decimel

          symnosave2=symno2                                          #save ...
              to a memory

          state_int1=mod(symno+self.sumrotcount,self.M) ...
                                   #intermediate state after taking ...
              previosu number of rotation into account
          #coderot1=mod(state_int1+tosym,self.M)
          #state_final1=mod(self.sumrotcount+state_int1,self.M)

          #tosym=self.xcycle[xcyclecount+1]
          state_int2=mod(state_int1+symno2,self.M)
          #coderot2=mod(state_int2+tosym,self.M)
          #state_final2=mod(state_final1+state_int2,self.M)



```

```
            #tosym=self.xcycle[xcyclecount+1]
            coderot=mod(state_int2+tosym,self.M)

            #state_int3=mod(state_final2+coderot,self.M)
            #state_interfinal=mod(state_int3+tosym,self.M)
            sumrotcount=mod(self.sumrotcount+coderot,self.M)


            fromstate=state
            tostate=sumrotcount
            #println("symn1=$(symnosave)  sym2=$(symnosave2)  ...
                fromstate=$(fromstate) tostate=$(sumrotcount) ...
                epoch=$(epoch1) ")
            self.trellisoutput[:,1,symnosave+1,fromstate+1,tostate+1, ...
                epoch1+1]=self.const_pts[:,symnosave+1]

            self.trellisoutput[:,2,symnosave+1,fromstate+1,tostate+1, ...
                epoch1+1]=self.const_pts[:,symnosave2+1]
            self.trellisoutput[:,3,symnosave+1,fromstate+1,tostate+1, ...
                epoch1+1]=self.const_pts[:,coderot+1]

            #println(bitssymbol1)
            self.inputbits[:,1,symnosave+1,fromstate+1,tostate+1,epoch1+1] ...
                =bitssymbol1
            self.inputbits[:,2,symnosave+1,fromstate+1,tostate+1,epoch1+1 ...
                ]=bitssymbol2
            if (doplot)
                        fromy=sumrotcountsave*yplotsep
                        toy=sumrotcount*xplotsep

                        ax[:plot]([plotx1; plotx2],[fromy; toy],"k")
                        ax[:scatter](plotx1,fromy,c=:black)
                        ax[:scatter](plotx2,fromy,c=:black)

                        mu=mulist[state+1]
                        textx=(1-mu)*plotx1+mu*plotx2
                        texty=(641-mu)*fromy+mu*toy
                        tangle=(-180/pi)*atan((toy - fromy)/(plotx2 - plotx1))
                      ax[:text](textx,texty,"$(testbits[bitctr+1]) ...
                          $(testbits[bitctr+2])$(testbits[bitctr+3]) ...
                          $(testbits[bitctr+4])/$ (symnosave)$(coderot)", ...
                          rotation=tangle,horizontalalignment="left", ...
                          verticalalignment="bottom", rotation_mode="anchor")
              end  #doplot
                    #println(bitctr)
              bitctr1=bitctr1+self.Nbits
            end #while bitctr1
            bitctr=bitctr+self.Nbits
        end #while
      end #end state
      self.fromtostate[:,:,epoch1+1]=allstates
    end #end epoch1
    end #end setdecoderinfo
#------------------------------------------------------------

    function encode(self::ShapeCodeFun,bits1,bits2)
```

```
symno1=bin2dec(bits1)
symnosave1=symno1
symno2=bin2dec(bits2)
symnosave2=symno2
if (¬self.doconstanttarget)
  xcyclecount=self.epoch
else
  xcyclecount=0
end


tosym=self.xcycle[xcyclecount+1]

state_int1=mod(symno1+self.sumrotcount,self.M)
#coderot1=mod(state_int1+tosym,self.M)
#state_final1=mod(state_int1+self.sumrotcount,self.M)

state_int2=mod(symno2+state_int1,self.M)
#coderot2=mod(state_int2+tosym,self.M)
#state_final2=mod(state_int2+state_final1,self.M)

coderot=mod(state_int2+tosym,self.M)
#state_int3=mod(state_final2+coderot,self.M)
#state_interfinal=mod(state_int3+tosym,self.M)
self.sumrotcount=mod(coderot+self.sumrotcount,self.M)

self.epoch=mod(self.epoch+1,self.Nepochs)
return symnosave1,symnosave2,coderot
end #end encode

function computemetric(r1,r2,r3,output)
out11=output[1,1]
out12=output[1,2]
out13=output[1,3]
out21=output[2,1]
out22=output[2,2]
out23=output[2,3]
n1=(r1[1]-out11)^2+(r1[2]-out21)^2
n2=(r2[1]-out12)^2+(r2[2]-out22)^2
n3=(r3[1]-out13)^2+(r3[2]-out23)^2
m=n1+n2+n3
return m
end #end computemetric

function viterbishapedecode(self::ShapeCodeFun,r1,r2,r3)
beststatemetric=Inf
for state=0:self.Nstate-1
  nextmetricbest=Inf
  for prevstate=self.fromtostate[:,state+1,self.epoch_decode+1]'
    for symbol=0:self.Nstate-1
      m=computemetric(r1,r2,r3,self.trellisoutput ...
          [:,:,symbol+1,prevstate+1,state+1,self.epoch_decode+1])
      nextmetric=self.metrics[prevstate+1]+m
      if (nextmetric<nextmetricbest)
        nextmetricbest=nextmetric
        global prevstatebest=prevstate
```

```
          global symbolbest=symbol
          statebest=state
        end
      end #symbol
    end #for prevstate
    if (nextmetricbest < beststatemetric)
       beststatemetric = nextmetricbest
       self.beststate = state
    end
    self.nextmetrics[state+1]=nextmetricbest
    self.prevstatearray[state+1,self.front+1]=prevstatebest
    self.bestsymbolarray[state+1,self.front+1]=symbolbest
  end #for state
  self.metrics=copy(self.nextmetrics)
  self.countbranches=self.countbranches+1
  decodeout=0
  bitsout1=[0;0]
  bitsout2=[0;0]
  if (self.countbranches≥self.Viterbiwindowwidth)
    state=self.beststate
    idx=self.front
    epoch1=self.epoch_decode
    prevstate=[]
    lastepoch=[]
    prevstatbestsymbol=[]
    for i=1:self.Viterbiwindowwidth
      prevstate=self.prevstatearray[state+1,idx+1]

      prevstatbestsymbol=self.bestsymbolarray[state+1,idx+1]
      idx=mod(idx-1,self.Viterbiwindowwidth)
      lastepoch=epoch1
      epoch1=mod(epoch1-1,self.Nepochs)
      if (i!=self.Viterbiwindowwidth)
        state=prevstate
      end
    end # i
      bitsout1=self.inputbits[:,1,prevstatbestsymbol+1, ...
          prevstate+1,state+1,lastepoch+1]
      bitsout2=self.inputbits[:,2,prevstatbestsymbol+1, ...
          prevstate+1,state+1,lastepoch+1]
      decodeout=1
  end #if
  self.front=mod(self.front+1,self.Viterbiwindowwidth)
  self.epoch_decode=mod(self.epoch_decode+1,self.Nepochs)
  return decodeout,bitsout1,bitsout2
  end #shape decoder viterbi

# ---------------------------------------------------------------
```

**Code for Rate $3/4$ (SIE)**

```
trellisoutput=zeros(Int64,2,n,M,M,Nstate,Nstate,Nepochs)
inputbits=zeros(Int64,Nbits,k,M,M,Nstate,Nstate,Nepochs)
metrics=zeros(Nstate,1)
nextmetrics=zeros(Nstate,1)
prevstatearray=zeros(Int64,Nstate,Viterbiwindowwidth)
fromtostate=zeros(Int64,Nstate,Nstate,Nepochs)
bitqueue=zeros(Int64,Nbits,k,Viterbiwindowwidth)
bitqueuefront=0
bitqueueback=0

bestsymbol1array=zeros(Int64,Nstate,Viterbiwindowwidth)
bestsymbol2array=zeros(Int64,Nstate,Viterbiwindowwidth)



function setdecoderinfo(self::ShapeCodeFun,doplot,allstates,testbitsM)
if (doplot)

end
testbits=testbitsM

for epoch1=0:self.Nepochs-1
  if (doplot)

  end
  if (self.doconstanttarget)
    xcyclecount=0
  else
    xcyclecount=epoch1
  end

  tosym=self.xcycle[xcyclecount+1]

  for state=0:self.Nstate-1
    bitctr=0
    sumrotcountsave=state
    while (bitctr<(self.M*self.Nbits))
      bitssymbol1=testbits[bitctr+1:bitctr+self.Nbits]
      symno=bin2dec(bitssymbol1)
      symnosave=symno
      self.sumrotcount=sumrotcountsave
      bitctr1=0
      while (bitctr1<(self.M*self.Nbits))
        bitssymbol2=testbits[bitctr1+1:bitctr1+self.Nbits]

        symno2=bin2dec(bitssymbol2)     #conver secend symbol to decimel

        symnosave2=symno2                                           #save ...
            to a memory


        bitctr2=0
        while (bitctr2<(self.M*self.Nbits))
          bitssymbol3=testbits[bitctr2+1:bitctr2+self.Nbits]
          symno3=bin2dec(bitssymbol3)
          symnosave3=symno3
```

```
            #println(symno3)

            An=mod(symno+self.sumrotcount,self.M)                    ...
                #intermediate state after taking previosu number of ...
                rotation into account
            An1=mod(An+symno2,self.M)
            An2=mod(An1+symno3,self.M)
            coderot=(An2+tosym)
           sumrotcount=mod(self.sumrotcount+coderot,self.Nstate)


           fromstate=state
           tostate=sumrotcount
           #println("symn1=$(symnosave)  sym2=$(symnosave2) ...
               sym3=$(symnosave3) parity=$(coderot) ...
               fromstate=$(fromstate) tostate=$(sumrotcount) ...
               epoch=$(epoch1) ")

           self.trellisoutput[:,1,symnosave+1,symnosave2+1,fromstate+1, ...
               tostate+1,epoch1+1]=self.const_pts[:,symnosave+1]
           self.trellisoutput[:,2,symnosave+1,symnosave2+1,fromstate+1, ...
               tostate+1,epoch1+1]=self.const_pts[:,symnosave2+1]
           self.trellisoutput[:,3,symnosave+1,symnosave2+1,fromstate+1, ...
               tostate+1,epoch1+1]=self.const_pts[:,symnosave3+1]
           self.trellisoutput[:,4,symnosave+1,symnosave2+1,fromstate+1, ...
               tostate+1,epoch1+1]=self.const_pts[:,coderot+1]

           #println(bitssymbol1)
           self.inputbits[:,1,symnosave+1,symnosave2+1,fromstate+1, ...
               tostate+1,epoch1+1]=bitssymbol1
           self.inputbits[:,2,symnosave+1,symnosave2+1,fromstate+1, ...
               tostate+1,epoch1+1]=bitssymbol2
           self.inputbits[:,3,symnosave+1,symnosave2+1,fromstate+1, ...
               tostate+1,epoch1+1]=bitssymbol3


                   #println(bitctr)
          bitctr2=bitctr2+self.Nbits
          end #while bitctr2
            bitctr1=bitctr1+self.Nbits
          end #while bitctr1
          bitctr=bitctr+self.Nbits
      end #while
    end #end state
    self.fromtostate[:,:,epoch1+1]=allstates
  end #end epoch1
  end #end setdecoderinfo

#----------------------------------------------------------------

  function encode(self::ShapeCodeFun,bits1,bits2,bits3)
  symno1=bin2dec(bits1)
  symnosave1=symno1
  symno2=bin2dec(bits2)
  symnosave2=symno2
  symno3=bin2dec(bits3)
```

```
symnosave3=symno3
if (¬self.doconstanttarget)
  xcyclecount=self.epoch
else
  xcyclecount=0
end
tosym=self.xcycle[xcyclecount+1]
An=mod(symno1+self.sumrotcount,self.M)
An1=mod(An+symno2,self.M)
An2=mod(symno3+An1,self.M)
coderot=mod(An2+tosym,self.M)
self.sumrotcount=mod(coderot+self.sumrotcount,self.Nstate)
self.epoch=mod(self.epoch+1,self.Nepochs)
return symnosave1,symnosave2,symnosave3,coderot
end #end encode
function computemetric(r1,r2,r3,r4,output)
out11=output[1,1]
out12=output[1,2]
out13=output[1,3]
out14=output[1,4]
out21=output[2,1]
out22=output[2,2]
out23=output[2,3]
out24=output[2,4]
n1=(r1[1]-out11)^2+(r1[2]-out21)^2
n2=(r2[1]-out12)^2+(r2[2]-out22)^2
n3=(r3[1]-out13)^2+(r3[2]-out23)^2
n4=(r4[1]-out14)^2+(r4[2]-out24)^2
m=n1+n2+n3+n4
return m
end #end computemetric
function viterbishapedecode(self::ShapeCodeFun,r1,r2,r3,r4)
beststatemetric=Inf
for state=0:self.Nstate-1
  nextmetricbest=Inf
  for prevstate=self.fromtostate[:,state+1,self.epoch_decode+1]'
    for symbol1=0:self.M-1
      for symbol2=0:self.M-1
        m=computemetric(r1,r2,r3,r4,self.trellisoutput[:,:,symbol1+1, ...
            symbol2+1,prevstate+1,state+1, self.epoch_decode+1])
        nextmetric=self.metrics[prevstate+1]+m
        if (nextmetric<nextmetricbest)
          nextmetricbest=nextmetric
```

```
            global prevstatebest=prevstate
            global symbolbest1=symbol1
            global symbolbest2=symbol2
            statebest=state
          end
        end #symbol2
      end #symbol1
    end #for prevstate
    if (nextmetricbest < beststatemetric)
       beststatemetric = nextmetricbest
       self.beststate = state
    end
    self.nextmetrics[state+1]=nextmetricbest
    self.prevstatearray[state+1,self.front+1]=prevstatebest
    self.bestsymbol1array[state+1,self.front+1]=symbolbest1
    self.bestsymbol2array[state+1,self.front+1]=symbolbest2
  end #for state
  self.metrics=copy(self.nextmetrics)
  self.countbranches=self.countbranches+1
  decodeout=0
  bitsout1=[0;0]
  bitsout2=[0;0]
  bitsout3=[0;0]
  if (self.countbranches≥self.Viterbiwindowwidth)
    state=self.beststate
    idx=self.front
    epoch1=self.epoch_decode
    prevstate=[]
    lastepoch=[]
    prevstatbestsymbol1=[]
    prevstatbestsymbol2=[]
    for i=1:self.Viterbiwindowwidth
      prevstate=self.prevstatearray[state+1,idx+1]

      prevstatbestsymbol1=self.bestsymbol1array[state+1,idx+1]
      prevstatbestsymbol2=self.bestsymbol2array[state+1,idx+1]
      idx=mod(idx-1,self.Viterbiwindowwidth)
      lastepoch=epoch1
      epoch1=mod(epoch1-1,self.Nepochs)
      if (i!=self.Viterbiwindowwidth)
        state=prevstate
      end
    end # i
      bitsout1=self.inputbits[:,1,prevstatbestsymbol1+1, ...
          prevstatbestsymbol2+1,prevstate+1, state+1,lastepoch+1]
      bitsout2=self.inputbits[:,2,prevstatbestsymbol1+1, ...
          prevstatbestsymbol2+1,prevstate+1, state+1,lastepoch+1]
      bitsout3=self.inputbits[:,3,prevstatbestsymbol1+1, ...
          prevstatbestsymbol2+1,prevstate+1, state+1,lastepoch+1]
      decodeout=1
  end #if
  self.front=mod(self.front+1,self.Viterbiwindowwidth)
  self.epoch_decode=mod(self.epoch_decode+1,self.Nepochs)
  return decodeout,bitsout1,bitsout2,bitsout3
  end #shape decoder viterbi
```

### Code for Rate 2/3 (PPE)

```

trellisoutput=zeros(Int64,2,3,M,Nstate,Nstate,Nepochs)
inputbits=zeros(Int64,Nbits,2,M,Nstate,Nstate,Nepochs)
metrics=zeros(Nstate,1)
nextmetrics=zeros(Nstate,1)
prevstatearray=zeros(Int64,Nstate,Viterbiwindowwidth)
fromtostate=zeros(Int64,Nstate,Nstate,Nepochs)
bitqueue=zeros(Int64,Nbits,2,Viterbiwindowwidth)
bitqueuefront=0
bitqueueback=0

bestsymbolarray=zeros(Int64,Nstate,Viterbiwindowwidth)


function setdecoderinfo(self::ShapeCodeFun,doplot,allstates,testbitsM)
if (doplot)
  xcyclesym=['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J' 'K' 'L' 'M' 'N' ...
      'O' 'P']
  yplotsep=1
  xplotsep=1
  markersize=15
  mulist=[.062 .12 .18 .25 .31 .37 .43 .5 .562 .6245 .687 .75 .81 .87 ...
      .93 .98]
  fig,ax=subplots()
  ax[:axis]("off")
  ax[:invert_yaxis]()
end
testbits=testbitsM

for epoch1=0:self.Nepochs-1
  if (doplot)
    plottime=epoch1
    plotx1=plottime
    plotx2=plottime+xplotsep


    ax[:text](plotx2,-.3,"$(xcyclesym[epoch1+1])")
  end
  if (self.doconstanttarget)
    xcyclecount=0
  else
    xcyclecount=epoch1
  end

  tosym=self.xcycle[xcyclecount+1]

  for state=0:self.M-1
    bitctr=0
    sumrotcountsave=state
    while (bitctr<(self.M*self.Nbits))
      bitssymbol1=testbits[bitctr+1:bitctr+self.Nbits]
      symno=bin2dec(bitssymbol1)
      symnosave=symno
```

```
        self.sumrotcount=sumrotcountsave
        bitctr1=0
        while(bitctr1<(self.M*self.Nbits))
          bitssymbol2=testbits[bitctr1+1:bitctr1+self.Nbits]

          symno2=bin2dec(bitssymbol2)     #conver secend symbol to decimel

          symnosave2=symno2                                         #save ...
              to a memory

          state_int1=mod(symno+self.sumrotcount,self.M) ...
                                #intermediate state after taking ...
              previosu number of rotation into account
          coderot1=mod(state_int1+tosym,self.M)
          state_final1=mod(self.sumrotcount+coderot1,self.M)

          #tosym=self.xcycle[xcyclecount+1]
          state_int2=mod(state_final1+symno2,self.M)
          coderot2=mod(state_int2+tosym,self.M)
          state_final2=mod(state_final1+coderot2,self.M)



          #tosym=self.xcycle[xcyclecount+1]
          coderot=mod(coderot1+coderot2,self.M)

          state_int3=mod(state_final2+coderot,self.M)
          state_interfinal=mod(state_int3+tosym,self.M)
          sumrotcount=mod(state_final2+state_interfinal,self.M)


          fromstate=state
          tostate=sumrotcount
          #println("symn1=$(symnosave)  sym2=$(symnosave2)  ...
              fromstate=$(fromstate) tostate=$(sumrotcount) ...
              epoch=$(epoch1) ")
          self.trellisoutput[:,1,symnosave+1,fromstate+1,tostate+1 ...
              ,epoch1+1]=self.const_pts[:,symnosave+1]

          self.trellisoutput[:,2,symnosave+1,fromstate+1,tostate+1, ...
              epoch1+1]=self.const_pts[:,symnosave2+1]
          self.trellisoutput[:,3,symnosave+1,fromstate+1,tostate+1, ...
              epoch1+1]=self.const_pts[:,coderot+1]

          #println(bitssymbol1)
          self.inputbits[:,1,symnosave+1,fromstate+1,tostate+1, ...
              epoch1+1]=bitssymbol1
          self.inputbits[:,2,symnosave+1,fromstate+1,tostate+1, ...
              epoch1+1]=bitssymbol2
          if (doplot)
                      fromy=sumrotcountsave*yplotsep
                      toy=sumrotcount*xplotsep

                      ax[:plot]([plotx1; plotx2],[fromy; toy],"k")
                      ax[:scatter](plotx1,fromy,c=:black)
                      ax[:scatter](plotx2,fromy,c=:black)
```

```

                          mu=mulist[state+1]
                          textx=(1-mu)*plotx1+mu*plotx2
                          texty=(641-mu)*fromy+mu*toy
                          tangle=(-180/pi)*atan((toy - fromy)/(plotx2 - plotx1))
                        ax[:text](textx,texty,"$(testbits[bitctr+1]) ...
                            $(testbits[bitctr+2])$ (testbits[bitctr+3])$ ...
                            (testbits[bitctr+4])/$(symnosave)$(coderot)", ...
                            rotation=tangle,horizontalalignment="left", ...
                            verticalalignment="bottom",  rotation_mode="anchor")
                end  #doplot
                      #println(bitctr)
                bitctr1=bitctr1+self.Nbits
              end #while bitctr1
              bitctr=bitctr+self.Nbits
          end #while
        end #end state
        self.fromtostate[:,:,epoch1+1]=allstates
      end #end epoch1
      end #end setdecoderinfo

      function encode(self::ShapeCodeFun,bits1,bits2)
      symno1=bin2dec(bits1)
      symnosave1=symno1
      symno2=bin2dec(bits2)
      symnosave2=symno2
      if (¬self.doconstanttarget)
        xcyclecount=self.epoch
      else
        xcyclecount=0
      end


      tosym=self.xcycle[xcyclecount+1]

      state_int1=mod(symno1+self.sumrotcount,self.M)
      coderot1=mod(state_int1+tosym,self.M)
      state_final1=mod(coderot1+self.sumrotcount,self.M)

      state_int2=mod(symno2+state_final1,self.M)
      coderot2=mod(state_int2+tosym,self.M)
      state_final2=mod(coderot2+state_final1,self.M)

      coderot=mod(coderot1+coderot2,self.M)
      state_int3=mod(state_final2+coderot,self.M)
      state_interfinal=mod(state_int3+tosym,self.M)
      self.sumrotcount=mod(state_interfinal+state_final2,self.M)

      self.epoch=mod(self.epoch+1,self.Nepochs)
      return symnosave1,symnosave2,coderot
      end #end encode

      function computemetric(r1,r2,r3,output)
      out11=output[1,1]
      out12=output[1,2]
      out13=output[1,3]
```

```
out21=output[2,1]
out22=output[2,2]
out23=output[2,3]
n1=(r1[1]-out11)^2+(r1[2]-out21)^2
n2=(r2[1]-out12)^2+(r2[2]-out22)^2
n3=(r3[1]-out13)^2+(r3[2]-out23)^2
m=n1+n2+n3
return m
end #end computemetric

function viterbishapedecode(self::ShapeCodeFun,r1,r2,r3)
beststatemetric=Inf
for state=0:self.Nstate-1
  nextmetricbest=Inf
  for prevstate=self.fromtostate[:,state+1,self.epoch_decode+1]'
    for symbol=0:self.Nstate-1
      m=computemetric(r1,r2,r3,self.trellisoutput[:,:,symbol+1, ...
          prevstate+1,state+1,self.epoch_decode+1])
      nextmetric=self.metrics[prevstate+1]+m
      if (nextmetric<nextmetricbest)
        nextmetricbest=nextmetric
        global prevstatebest=prevstate
        global symbolbest=symbol
        statebest=state
      end
    end #symbol
  end #for prevstate
  if (nextmetricbest < beststatemetric)
     beststatemetric = nextmetricbest
     self.beststate = state
  end
  self.nextmetrics[state+1]=nextmetricbest
  self.prevstatearray[state+1,self.front+1]=prevstatebest
  self.bestsymbolarray[state+1,self.front+1]=symbolbest
end #for state
self.metrics=copy(self.nextmetrics)
self.countbranches=self.countbranches+1
decodeout=0
bitsout1=[0;0]
bitsout2=[0;0]
if (self.countbranches≥self.Viterbiwindowwidth)
  state=self.beststate
  idx=self.front
  epoch1=self.epoch_decode
  prevstate=[]
  lastepoch=[]
  prevstatbestsymbol=[]
  for i=1:self.Viterbiwindowwidth
    prevstate=self.prevstatearray[state+1,idx+1]

    prevstatbestsymbol=self.bestsymbolarray[state+1,idx+1]
    idx=mod(idx-1,self.Viterbiwindowwidth)
    lastepoch=epoch1
    epoch1=mod(epoch1-1,self.Nepochs)
    if (i!=self.Viterbiwindowwidth)
      state=prevstate
```

```
      end
    end # i
      bitsout1=self.inputbits[:,1,prevstatbestsymbol+1, ...
          prevstate+1,state+1,lastepoch+1]
      bitsout2=self.inputbits[:,2,prevstatbestsymbol+1, ...
          prevstate+1,state+1,lastepoch+1]
      decodeout=1
  end #if
  self.front=mod(self.front+1,self.Viterbiwindowwidth)
  self.epoch_decode=mod(self.epoch_decode+1,self.Nepochs)
  return decodeout,bitsout1,bitsout2
  end #shape decoder viterbi
```

## Code for Rate 3/4 (PPE)

```



  trellisoutput=zeros(Int64,2,n,M,M,Nstate,Nstate,Nepochs)
  inputbits=zeros(Int64,Nbits,k,M,M,Nstate,Nstate,Nepochs)
  metrics=zeros(Nstate,1)
  nextmetrics=zeros(Nstate,1)
  prevstatearray=zeros(Int64,Nstate,Viterbiwindowwidth)
  fromtostate=zeros(Int64,Nstate,Nstate,Nepochs)
  bitqueue=zeros(Int64,Nbits,k,Viterbiwindowwidth)
  bitqueuefront=0
  bitqueueback=0

  bestsymbol1array=zeros(Int64,Nstate,Viterbiwindowwidth)
  bestsymbol2array=zeros(Int64,Nstate,Viterbiwindowwidth)


  function setdecoderinfo(self::ShapeCodeFun,doplot,allstates,testbitsM)
  if (doplot)

  end
  testbits=testbitsM

  for epoch1=0:self.Nepochs-1
    if (doplot)

    end
    if (self.doconstanttarget)
      xcyclecount=0
    else
      xcyclecount=epoch1
    end

    tosym=self.xcycle[xcyclecount+1]

    for state=0:self.Nstate-1
      bitctr=0
```

```
      sumrotcountsave=state
      while (bitctr<(self.M*self.Nbits))
        bitssymbol1=testbits[bitctr+1:bitctr+self.Nbits]
        symno=bin2dec(bitssymbol1)
        symnosave=symno
        self.sumrotcount=sumrotcountsave
        bitctr1=0
        while (bitctr1<(self.M*self.Nbits))
          bitssymbol2=testbits[bitctr1+1:bitctr1+self.Nbits]

          symno2=bin2dec(bitssymbol2)     #conver secend symbol to decimel

          symnosave2=symno2                                         #save ...
              to a memory


          bitctr2=0
          while (bitctr2<(self.M*self.Nbits))
            bitssymbol3=testbits[bitctr2+1:bitctr2+self.Nbits]
            symno3=bin2dec(bitssymbol3)
            symnosave3=symno3
            #println(symno3)

            An=mod(symno+self.sumrotcount,self.M)                       ...
                #intermediate state after taking previosu number of ...
                rotation into account
            coderot1=mod(An+tosym,self.M)
            An1=mod(self.sumrotcount+coderot1,self.M)

            #tosym=self.xcycle[xcyclecount+1]
            An2=mod(An1+symno2,self.M)
            coderot2=mod(An2+tosym,self.M)
            An3=mod(An1+coderot2,self.M)

            An4=mod(An3+symno3,self.M)
            coderot3=mod(An4+tosym,self.M)
            An5=mod(coderot3+An3,self.M)

           #tosym=self.xcycle[xcyclecount+1]
           coderot=mod(coderot1+coderot2+coderot3,self.M)

           An6=mod(An5+coderot,self.M)
           An7=mod(An6+tosym,self.M)
           sumrotcount=mod(An5+An7,self.Nstate)


           fromstate=state
           tostate=sumrotcount
           #println("symn1=$(symnosave)  sym2=$(symnosave2) ...
               sym3=$(symnosave3) parity=$(coderot) ...
               fromstate=$(fromstate) tostate=$(sumrotcount) ...
               epoch=$(epoch1) ")

           self.trellisoutput[:,1,symnosave+1,symnosave2+1,fromstate+1, ...
               tostate+1,epoch1+1]=self.const_pts[:,symnosave+1]
```

```
              self.trellisoutput[:,2,symnosave+1,symnosave2+1,fromstate+1, ...
                  tostate+1,epoch1+1]=self.const_pts[:,symnosave2+1]
              self.trellisoutput[:,3,symnosave+1,symnosave2+1,fromstate+1, ...
                  tostate+1,epoch1+1]=self.const_pts[:,symnosave3+1]
              self.trellisoutput[:,4,symnosave+1,symnosave2+1,fromstate+1, ...
                  tostate+1,epoch1+1]=self.const_pts[:,coderot+1]

              #println(bitssymbol1)
              self.inputbits[:,1,symnosave+1,symnosave2+1,fromstate+1, ...
                  tostate+1,epoch1+1]=bitssymbol1
              self.inputbits[:,2,symnosave+1,symnosave2+1,fromstate+1, ...
                  tostate+1,epoch1+1]=bitssymbol2
              self.inputbits[:,3,symnosave+1,symnosave2+1,fromstate+1, ...
                  tostate+1,epoch1+1]=bitssymbol3


                      #println(bitctr)
             bitctr2=bitctr2+self.Nbits
             end #while bitctr2
               bitctr1=bitctr1+self.Nbits
             end #while bitctr1
             bitctr=bitctr+self.Nbits
         end #while
       end #end state
       self.fromtostate[:,:,epoch1+1]=allstates
     end #end epoch1
     end #end setdecoderinfo


     function encode(self::ShapeCodeFun,bits1,bits2,bits3)
     symno1=bin2dec(bits1)
     symnosave1=symno1
     symno2=bin2dec(bits2)
     symnosave2=symno2
     symno3=bin2dec(bits3)
     symnosave3=symno3
     if (¬self.doconstanttarget)
       xcyclecount=self.epoch
     else
       xcyclecount=0
     end


     tosym=self.xcycle[xcyclecount+1]

     An=mod(symno1+self.sumrotcount,self.M)
     coderot1=mod(An+tosym,self.M)
     An1=mod(coderot1+self.sumrotcount,self.M)

     An2=mod(symno2+An1,self.M)
     coderot2=mod(An2+tosym,self.M)
     An3=mod(coderot2+An1,self.M)

     An4=mod(symno3+An3,self.M)
     coderot3=mod(An4+tosym,self.M)
     An5=mod(coderot3+An3,self.M)
```

```

coderot=mod(coderot1+coderot2+coderot3,self.M)


An6=mod(An5+coderot,self.M)
An7=mod(An6+tosym,self.M)
self.sumrotcount=mod(An7+An5,self.Nstate)

self.epoch=mod(self.epoch+1,self.Nepochs)
return symnosave1,symnosave2,symnosave3,coderot
end #end encode

function computemetric(r1,r2,r3,r4,output)
out11=output[1,1]
out12=output[1,2]
out13=output[1,3]
out14=output[1,4]
out21=output[2,1]
out22=output[2,2]
out23=output[2,3]
out24=output[2,4]
n1=(r1[1]-out11)^2+(r1[2]-out21)^2
n2=(r2[1]-out12)^2+(r2[2]-out22)^2
n3=(r3[1]-out13)^2+(r3[2]-out23)^2
n4=(r4[1]-out14)^2+(r4[2]-out24)^2
m=n1+n2+n3+n4
return m
end #end computemetric

function viterbishapedecode(self::ShapeCodeFun,r1,r2,r3,r4)
beststatemetric=Inf
for state=0:self.Nstate-1
  nextmetricbest=Inf
  for prevstate=self.fromtostate[:,state+1,self.epoch_decode+1]'
    for symbol1=0:self.M-1
      for symbol2=0:self.M-1
        m=computemetric(r1,r2,r3,r4,self.trellisoutput[:,:,symbol1+1, ...
            symbol2+1,prevstate+1,state+1, self.epoch_decode+1])
        nextmetric=self.metrics[prevstate+1]+m
        if (nextmetric<nextmetricbest)
          nextmetricbest=nextmetric
          global prevstatebest=prevstate
          global symbolbest1=symbol1
          global symbolbest2=symbol2
          statebest=state
        end
      end #symbol2
    end #symbol1
  end #for prevstate
  if (nextmetricbest < beststatemetric)
     beststatemetric = nextmetricbest
     self.beststate = state
  end
  self.nextmetrics[state+1]=nextmetricbest
  self.prevstatearray[state+1,self.front+1]=prevstatebest
  self.bestsymbol1array[state+1,self.front+1]=symbolbest1
```

```
      self.bestsymbol2array[state+1,self.front+1]=symbolbest2
    end #for state
    self.metrics=copy(self.nextmetrics)
    self.countbranches=self.countbranches+1
    decodeout=0
    bitsout1=[0;0]
    bitsout2=[0;0]
    bitsout3=[0;0]
    if (self.countbranches≥self.Viterbiwindowwidth)
      state=self.beststate
      idx=self.front
      epoch1=self.epoch_decode
      prevstate=[]
      lastepoch=[]
      prevstatbestsymbol1=[]
      prevstatbestsymbol2=[]
      for i=1:self.Viterbiwindowwidth
        prevstate=self.prevstatearray[state+1,idx+1]

        prevstatbestsymbol1=self.bestsymbol1array[state+1,idx+1]
        prevstatbestsymbol2=self.bestsymbol2array[state+1,idx+1]
        idx=mod(idx-1,self.Viterbiwindowwidth)
        lastepoch=epoch1
        epoch1=mod(epoch1-1,self.Nepochs)
        if (i!=self.Viterbiwindowwidth)
          state=prevstate
        end
      end # i
        bitsout1=self.inputbits[:,1,prevstatbestsymbol1+1, ...
            prevstatbestsymbol2+1,prevstate+1, state+1,lastepoch+1]
        bitsout2=self.inputbits[:,2,prevstatbestsymbol1+1, ...
            prevstatbestsymbol2+1,prevstate+1, state+1,lastepoch+1]
        bitsout3=self.inputbits[:,3,prevstatbestsymbol1+1 ...
            ,prevstatbestsymbol2+1,prevstate+1, state+1,lastepoch+1]
        decodeout=1
    end #if
    self.front=mod(self.front+1,self.Viterbiwindowwidth)
    self.epoch_decode=mod(self.epoch_decode+1,self.Nepochs)
    return decodeout,bitsout1,bitsout2,bitsout3
    end #shape decoder viterbi

  # -------------------------------------------------------------

    function viterbishapecodeflush(self::ShapeCodeFun)
    bitsout=zeros(Int64,self.Nbits,self.Viterbiwindowwidth-1)
    state=self.beststate
    epoch1=mod(self.epoch_decode-1,self.Nepochs)
    idx=mod(self.front-1,self.Viterbiwindowwidth)
    for i=1:self.Viterbiwindowwidth-1
      prevstate=self.prevstatearray[state+1,idx+1]
      lastepoch=epoch1
      bits1=self.inputbits[:,prevstate+1,state+1,lastepoch+1]
      idx=mod(idx-1,self.Viterbiwindowwidth)
      epoch1=mod(epoch1-1,self.Nepochs)
      bitsout[:,self.Viterbiwindowwidth-1-i]=bits1
      state=prevstate
```

```
  end #for
  end #viterbishapecodeflush
#------------------------------------------------------------------

  function bitqueuein(self::ShapeCodeFun,bits1,bits2,bits3)
  self.bitqueue[:,1,self.bitqueuefront+1]=bits1
  self.bitqueue[:,2,self.bitqueuefront+1]=bits2
  self.bitqueue[:,3,self.bitqueuefront+1]=bits3
  self.bitqueuefront = mod(self.bitqueuefront + 1, self.Viterbiwindowwidth)
  end

  function bitqueueout(self::ShapeCodeFun)
  bits1=self.bitqueue[:,1,self.bitqueueback+1]
  bits2=self.bitqueue[:,2,self.bitqueueback+1]
  bits3=self.bitqueue[:,3,self.bitqueueback+1]
  self.bitqueueback = mod(self.bitqueueback + 1, self.Viterbiwindowwidth)
  return bits1,bits2 ,bits3
  end

#create object of ShapeCodeFun--------------------------------------


  a=ShapeCodeFun(M,Viterbiwindowwidth,Nstate,Nepochs,Nbits,
              fromtodist,cumrotoffset,const_pts,xcycle,0,
              0,doconstanttarget,trellisoutput,inputbits, ...
                  fromtostate,metrics,
              nextmetrics,0,prevstatearray,0,0,0,bitqueue,0,0, ...
                  bestsymbol1array,bestsymbol2array)


  setdecoderinfo(a,doplot,allstates,testbits)
  resetshapecoder(a)
  resetshapedecoder(a)

end #doshapecode

nerrortocount=100

Eb=1
R=1
Es=Nbits*Eb*R
Esscale=sqrt(3*Es/(2*(M-1)))
Rcoded=k/n
Escoded=sqrt(Nbits*Eb*Rcoded*3/(2*(M-1)))
theorprobQPSK=[]
theorprobBPSK=[]
theoprobMquam=[]
for SNR in SNRrangetheo
  println("snr=$(SNR)")

  N0=Eb*10^(-SNR/10)
  EbN0=Eb/N0

  push!(theorprobQPSK,(1 - (1-qf(sqrt(2*EbN0)))^2))
  push!(theorprobBPSK,(qf(sqrt(2*EbN0))))

```

```
  push!(theoprobMquam, ...
      (1-(1-2*(1-sqrt(1/M))*qf(sqrt((3/(M-1))*Nbits*EbN0)))^2))
end


r=[0.0 0.0]
if (doMquam)
  snrctr=0

  errctr=[]
  errctr_bits=[]
  nsyms=[]
  for SNR in SNRrangeex
    println("snr=$(SNR)")
    snrctr=snrctr+1
    N0=Eb*10^(-SNR/10)
    EbN0=Eb/N0
    #println("N0=$(N0) EbN0=$(EbN0)")

    sigma2=N0/2
    sigma=sqrt(sigma2)

    push!(errctr,0)
    push!(errctr_bits,0)

    push!(nsyms,0)


    while(errctr[snrctr]<nerrortocount)
      nsyms[snrctr]=nsyms[snrctr]+1
      bits=randombits(Nbits)
      symno=bin2dec(bits)
      constpts=Esscale*const_pts[:,symno+1]

      noise=sigma*randn(1,2)

      r[1]=constpts[1]+noise[1]
      r[2]=constpts[2]+noise[2]

      decodesymno=mquamdemod(r/Esscale,const_pts,const_axis)

      #println("decodesymno=$(decodesymno) symno=$(symno)")

      if (decodesymno != symno)

        errctr[snrctr]=errctr[snrctr]+1
        errctr_bits[snrctr] = errctr_bits[snrctr] + ...
            sum(Int64.(decodesymno .!= symno))

        println("SNR=$(SNR):uncoded error no=$(errctr[snrctr])")
        #println("decodesymno=$(decodesymno) symno=$(symno)")
      end
     if (errctr[snrctr]!=0 && errctr[snrctr]/nsyms[snrctr]≤GraphPsrange)
        break
      end   #end break while
    end #while errctr
```

```
    if (errctr[snrctr]/nsyms[snrctr]≤GraphPsrange)
      break
    end #end brek for
  end #for snr
end #if(doqpsk)

if (doshapecode)
    errctr_coded=[]
    errctr_bits_coded=[]
    nsyms_coded=[]
    snrctr=0
    for SNR in SNRrange
      println("SNRShapeCode=$(SNR)")
      snrctr=snrctr+1
      N0=Eb*10^(-SNR/10)
      EbN0=Eb/N0
      sigma2=N0/2
      sigma=sqrt(sigma2)
      push!(errctr_coded,0)
      push!(errctr_bits_coded,0)
      push!(nsyms_coded,0)
      while(errctr_coded[snrctr]<nerrortocount)
        nsyms_coded[snrctr]=nsyms_coded[snrctr]+3

        if (rem(nsyms_coded[snrctr],10000)==0 && (nsyms_coded[snrctr]≥ ...
            10000))
          println("number of symbol=$(nsyms_coded[snrctr])")
        end





        bits1=randombits(Nbits)
        bits2=randombits(Nbits)
        bits3=randombits(Nbits)
        sym1,sym2,sym3,parity=encode(a,bits1,bits2,bits3)
        bitqueuein(a,bits1,bits2,bits3)
        s1=Escoded*a.const_pts[:,sym1+1]
        s2=Escoded*a.const_pts[:,sym2+1]
        s3=Escoded*a.const_pts[:,sym3+1]
        s4=Escoded*a.const_pts[:,parity+1]
        noise1=sigma*randn(2,1)
        noise2=sigma*randn(2,1)
        noise3=sigma*randn(2,1)
        noise4=sigma*randn(2,1)
        r1=s1+noise1
        r2=s2+noise2
        r3=s3+noise3
        r4=s4+noise4

        decodeout,bitsout1,bitsout2,bitsout3=viterbishapedecode ...
            (a,r1/Escoded,r2/Escoded,r3/Escoded,r4/Escoded)

        if (decodeout==1)
          inbits1,inbits2,inbits3=bitqueueout(a)
```

```
          #println("inbits 1=$(inbits1) bitsout1=$(bitsout1) inbits ...
              2=$(inbits2) bitsout2=$(bitsout2) inbits 3=$(inbits3)   ...
              bitsout3=$(bitsout3)")
          if (inbits1!=bitsout1 || inbits2!=bitsout2 || inbits3!=bitsout3)
            println("SNR=$(SNR):coded error no=$(errctr_coded[snrctr])")
          end
          #println("inbit+sum(Int64.(inbits2 .!= bitsout2))s=$(inbits) ...
              bitsout=$(bitsout)")
          errctr_coded[snrctr]=errctr_coded[snrctr]+ ...
              Int64(any(inbits1!=bitsout1))+ ...
              Int64(any(inbits2!=bitsout2))+Int64(any(inbits3!=bitsout3))
          errctr_bits_coded[snrctr] = errctr_bits_coded[snrctr] + ...
              sum(Int64.(inbits1 .!= bitsout1))+ sum(Int64.(inbits2 .!= ...
              bitsout2))+ sum(Int64.(inbits3 .!= bitsout3))
        end #decodeout
        if (errctr_coded[snrctr]!=0 && ...
            errctr_coded[snrctr]/nsyms_coded[snrctr]≤GraphPsrange)
          break
        end   #end break while
    end #while error count
    if (errctr_coded[snrctr]/nsyms_coded[snrctr]≤GraphPsrange)
      break
    end #end break for
    @save "$(M)QAMrate34method1$(datetoday).jld"
  end #for SNRrangeShapecode
end #doshapecode
legendstr=[]
push!(legendstr,"Uncoded QPSK(th) SER")
push!(legendstr,"Uncoded BPSK(th) SER")
push!(legendstr,"Uncoded $(M)QAM(th) SER")

if (doMquam)
  push!(legendstr,"Uncoded $(M)QAM(ex) SER")
  push!(legendstr,"Uncoded $(M)QAM(ex) BER")
end
if (doshapecode)
  push!(legendstr,"Rate 3/4 CAC $(M)QAM SER")
  push!(legendstr,"Rate 3/4 CAC $(M)QAM BER")
end
len=length(SNRrange)

fig1,ax1=subplots()
ax1[:semilogy](collect(0:1:length(theorprobQPSK)-1), ...
    theorprobQPSK,"g:",marker="^")
ax1[:semilogy](collect(0:1:length(theorprobBPSK)-1), ...
    theorprobBPSK,"b:",marker="<")
ax1[:semilogy](collect(0:1:length(theoprobMquam)-1), ...
    theoprobMquam,"k:",marker=">")

if (doMquam)
  ax1[:semilogy](collect(0:1:length(errctr)-1), ...
      errctr./(nsyms),"r",marker="v")
  ax1[:semilogy](collect(0:1:length(errctr_bits)-1), ...
      errctr_bits./(nsyms*Nbits),"r:",marker="d")
end

```

```
if (doshapecode)
  ax1[:semilogy](collect(0:1:length(errctr_coded)-1), ...
      errctr_coded./nsyms_coded,"c",marker="s")
  ax1[:semilogy](collect(0:1:length(errctr_coded)-1), ...
      errctr_bits_coded./(Nbits*nsyms_coded),"c:",marker="o")
end


ax1[:legend](legendstr)
ax1[:set_ylim]([GraphPsrange,10.0^-0])
ax1[:grid](color="k", linestyle="--")
xlabel(L"(SNR)_{dB}")
ylabel(L"P_s, P_b" )
xticks(SNRrangetheo)
toc()
Ps=-1*Int(log10(GraphPsrange))
savefig("BER$(M)QUAMSCSNR$(SNRrange[end])Ps$(Ps) ...
    Rate34method2$(datetoday).pdf",dpi=1500)
@save "$(M)QAMrate34method2$(datetoday).jld"
```

## B.3 Python Code

### Python Code for Signal Point Target Code Using QPSK Constellation

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Jun 27 08:58:24 2018

@author: munibun
"""

SNRrange=8  # It will count SNRrange-1
import time
start_time = time.time()

import numpy as np
import math
import matplotlib.pyplot as plt

def qf(x):
    return .5*math.erfc(x/math.sqrt(2))

def qpskdemod(r):
    if (r[1]≥0):
        bit1=1
    else:
        bit1=0
    if (r[0]≥0):
```

```
            bit2=1
        else:
            bit2=0

        return 2*bit1+bit2

def randombits():
    genrand=np.random.rand(2,1)
    bits=np.zeros(shape=(2,1),dtype=int).reshape(2,)
    if genrand[0]>.5:
        bits[0]=1
    else:
        bits[0]=0
    if genrand[1]>.5:
        bits[1]=1
    else:
        bits[1]=0

    return bits

class ShapeCodeFunctions1:
    M=int(4)
    viterbiwindowwidth=int(10)

    def __init__(self,doconstanttarget):      #basic constructor

        self.doconstanttarget=doconstanttarget
        self.Nstate= self.M
        self.Nepoch=self.M

        self.Nbits=int(math.log2(self.M))


        self.const_pts=np.transpose(np.array([[-1,-1],      #D(0)
                                              [1,-1],       #C(1)
                                              [-1,1],       #A(2)
                                              [1,1]]).astype(int))     #B(3))


        self.fromtodist=np.array([[0,3,1,2],
                                  [1,0,2,3],
                                  [3,2,0,1],
                                  [2,1,3,0]]).astype(int)


        self.cumrotoffset=np.array([[0,2,3,1],
                                    [1,0,2,3],
                                    [2,3,1,0],
                                    [3,1,0,2]]).astype(int)


        if (not(doconstanttarget)):
            self.xcycle=np.array([2,3,1,0]).astype(int)
        else:
            self.xcycle=np.array([2]).astype(int)

```

```
        self.epoch=0
        self.sumrotcount=0

        #end of constructor-----------------------------

    def resetshapecoder(self):
        self.epoch=0
        self.sumrotcount=0

        #end of resetshapecoder-------------------------

    def resetshapedecoder(self):
        self.epoch_decode=0
        self.front=0
        self.countbranches=0
        self.bitqueuefront=0
        self.bitqueueback=0

        #end of resershapedeocder-----------------------

    def encode(self,bits):
            symno=2*bits[0]+bits[1]
            symnosave=symno
            symno=self.cumrotoffset[symno,self.sumrotcount]
            if(not(self.doconstanttarget)):
                xcyclecount=self.epoch
            else:
                xcyclecount=0
            tosym=self.xcycle[xcyclecount]
            coderot=self.fromtodist[symno,tosym]
            self.sumrotcount=(self.sumrotcount+coderot)%self.M
            self.epoch=(self.epoch+1)%4
            return symnosave,coderot

            #end of encode ----------------------------

    def setdecoderinfo(self,doplot):
        if(doplot):
            fig, ax = plt.subplots(1,1)

            plt.clf()


            for spine in plt.gca().spines.values():
                spine.set_visible(False)


            ax.set_xlim(-.8, 4.2)
            ax.set_ylim(-.8, 3.2)

            plt.xticks(range(0, 5, 1), fontsize=14)
            plt.yticks(range(0, 4, 1), fontsize=14)

            #plt.tick_params(
            #        axis='x',          # changes apply to the x-axis
```

```
            #        which='both',      # both major and minor ticks are ...
                affected
            #        bottom=False,      # ticks along the bottom edge are off
            #        top=False,         # ticks along the top edge are off
            #        labelbottom=False) # labels along the bottom edge ...
                are off
            plt.gca().invert_yaxis()

            xplotsep=1
            yplotsep=1
            #markersize = 15
            mulist = [0.15, 0.45, 0.60, 0.75]
            xcyclesym=['A','B','C','D']

        self.trellisoutput=np.ndarray(
        shape=(2,2,self.Nstate,self.Nstate,self.Nepoch),dtype=int)
        self.inputbits=np.ndarray(
        shape=(self.Nbits,self.Nstate,self.Nstate,self.Nepoch),dtype=int)
        self.metrics=np.zeros(shape=(self.Nstate,1),dtype=int)
        self.nextmetrics=np.zeros(shape=(self.Nstate,1),dtype=int)

        self.fromtostate=np.ndarray(
        shape=(self.Nstate,self.Nstate,self.Nepoch),dtype=int)

        testbits=[0,0,0,1,1,0,1,1]

        for epoch1 in range(self.Nepoch):
            if(doplot):
                    plottime=epoch1

                    plotx1=plottime
                    plotx2=plottime+xplotsep
                    plt.text( plotx2,-.3,xcyclesym[plottime])

            if (self.doconstanttarget):
                xcyclecount=0
            else:
                xcyclecount=epoch1

            targetsymbol=self.xcycle[xcyclecount]

            for init_total_rot in range(self.M):
                bitcount=0
                init_total_rot_save=init_total_rot
                #print("\n initial sum rot count(state)={0}  target ...
                    symbol={1}".format(init_total_rot,self.xcycle[epoch1]))

                while (bitcount< len(testbits)):
                    self.sumrotcount=init_total_rot_save
                    sumno=2*testbits[bitcount]+testbits[bitcount+1]
                    currentposition=self.cumrotoffset[sumno,self.sumrotcount]
                    codeoutput=self.fromtodist[currentposition,targetsymbol]
                    self.sumrotcount=(self.sumrotcount+codeoutput)%self.M
                   # print("bits={0}{1}  out=({2},{3})  ...
                       sumrotcount={4}".format(testbits[bitcount] ...
```

```
                       ,testbits[bitcount+1],sumno, ...
                       codeoutput,self.sumrotcount))
                    fromstate=init_total_rot
                    tostate=self.sumrotcount
                    self.trellisoutput[:,0,fromstate,tostate,epoch1]= ...
                        self.const_pts[:,sumno]
                    self.trellisoutput[:,1,fromstate,tostate,epoch1]= ...
                        self.const_pts[:,codeoutput]
                    self.inputbits[:,fromstate,tostate,epoch1]= ...
                        np.array([testbits[bitcount], ...
                        testbits[bitcount+1]]).astype(int)

                    if(doplot):
                        from_pos=init_total_rot_save*yplotsep
                        to_pos=self.sumrotcount*yplotsep

                        plt.plot([plotx1,plotx2],[from_pos,to_pos],c='k')
                        plt.plot(plotx1,from_pos,'ko')
                        plt.plot(plotx2,from_pos,'ko')

                        strng="{0}{1}/{2}{3}".format(
                testbits[bitcount],testbits[bitcount+1],sumno,codeoutput)

                        mu=mulist[init_total_rot]
                        textx = (1-mu)*plotx1 + mu*plotx2
                        texty = (1-mu)*from_pos + mu*to_pos
                        tangle = (180/math.pi)*math.atan((to_pos - ...
                            from_pos)/(plotx2 - plotx1));
                        trans_angle = ...
                            plt.gca().transData.transform_angles( ...
                            np.array((tangle,)), ...
                            np.array([textx,texty]).reshape((1, 2)))[0]
                        plt.text(textx, texty, strng,color='r', ...
                            fontsize=8,rotation=trans_angle, ...
                            horizontalalignment='left', ...
                            verticalalignment='bottom', ...
                            rotation_mode='anchor')
                    bitcount=bitcount+2
            self.fromtostate[:,:,epoch1]=np.array([[0,0,0,0],
                                                   [1,1,1,1],
                                                   [2,2,2,2],
                                                   [3,3,3,3]]).astype(int)
        if(doplot):

            plt.savefig('trellisfromshapecodefunction.png',dpi=1500)
            plt.show()

        self.epoch_decode=0
        self.bitqueuefront=0
        self.bitqueueback=0
        self.front=0
        self.countbranches=0
        self.prevstatearray=np.zeros((self.Nstate, ...
            self.viterbiwindowwidth)).astype(int)
        self.bitqueue=np.zeros((self.Nbits, ...
            self.viterbiwindowwidth)).astype(int)
```

```
        #end of setdecoderinfo---------------------------------

    def computemetric(self,r1,r2,output):
         m=(r1[0]-output[0,0])**2+(r1[1]-output[1,0])**2 ...
             +(r2[0]-output[0,1])**2 +(r2[1]-output[1,1])**2
         return m

    #end of compute metric---------------------

    def viterbishapedecoder(self,r1,r2):
        beststatemetric=math.inf
        for state in range(self.Nstate):
            bestnextmetric=math.inf
            for prevstate in self.fromtostate[:,state,self.epoch_decode]:
                m=self.computemetric(r1,r2, ...
                    self.trellisoutput[:,:,prevstate,state,self.epoch_decode])
                nextmetric=self.metrics[prevstate]+m

                if(nextmetric<bestnextmetric):
                    bestnextmetric=nextmetric
                    bestprevstate=prevstate
                    statebest=state
            if(bestnextmetric<beststatemetric):
                beststatemetric=bestnextmetric
                self.beststate=state

            self.nextmetrics[state]=bestnextmetric
            self.prevstatearray[state,self.front]=bestprevstate
            #print('state: {0} bestprevstate={1} beststate={2} ...
                bestmetric={3}\n'.format(state,bestprevstate,statebest, ...
                int(bestnextmetric)))
        self.metrics[:]=self.nextmetrics[:]
        self.countbranches=self.countbranches+1
        decodeout=0
        bitsout=[0,0]

        if(self.countbranches>=self.viterbiwindowwidth):
            state=self.beststate
            idx=self.front
            epoch1=self.epoch_decode
            for i in range(self.viterbiwindowwidth):

                prevstate=self.prevstatearray[state,idx]

                lastepoch=epoch1
                epoch1=(epoch1-1)%self.Nepoch
                idx=(idx-1)%self.viterbiwindowwidth

                if(i!=self.viterbiwindowwidth-1):
                    state=prevstate
            decodeout=1
            bitsout=self.inputbits[:,prevstate,state,lastepoch]

        self.front=(self.front+1)%self.viterbiwindowwidth
        self.epoch_decode=(self.epoch_decode+1)%self.Nepoch
```

```
        return decodeout,bitsout

#end of viterbishapedecoder---------------------------------------

    def viterbishapedecoderflush(self):
        bitsout=np.zeros((self.Nbits,self.viterbiwindowwidth-1)).astype(int)
        state=self.beststate
        epoch1=(self.epoch_decode-1)%self.Nepoch
        idx=(self.front-1)%self.viterbiwindowwidth
        for i in range(self.viterbiwindowwidth-2):
            prevstate=self.prevstatearray[state,idx]
            lastepoch=epoch1
            bits=self.inputbits[:,prevstate,state,lastepoch]
            idx=(idx-1)%self.viterbiwindowwidth
            epoch1=(epoch1-1)%self.Nepoch
            bitsout[:,self.viterbiwindowwidth-2-i]=bits
            state=prevstate

#end of viterbishapedecoderflush-----------------------------------

#end of viterbishapedecoderflush-----------------------------------

    def bitqueuein(self,bits):
        self.bitqueue[:,self.bitqueuefront]=bits[:]
        self.bitqueuefront=(self.bitqueuefront+1)%self.viterbiwindowwidth
#end of bitqueuein--------------------------------------------------
    def bitqueueout(self):
        bits=self.bitqueue[:,self.bitqueueback]
        self.bitqueueback=(self.bitqueueback+1)%self.viterbiwindowwidth
        return bits
#end of bitqueueout -------------------------------------------------

shapecodeclass1=ShapeCodeFunctions1(1)
shapecodeclass1.resetshapecoder()
shapecodeclass1.resetshapedecoder()
shapecodeclass1.setdecoderinfo(0)

nerrortocount=100
Eb=1
R=1
Es=2*Eb*R
Esscale=math.sqrt(Es/2)
Rcoded=.5
Escoded=math.sqrt(2*Eb*Rcoded/2)
doqpsk=1
doshapecode=1
snrctr=0
theorprobQPSK=[0]*SNRrange
theorprobBPSK=[0]*SNRrange
errctr=[0]*SNRrange
nsyms=[0]*SNRrange
errctr_coded=[0]*SNRrange
errctr_bits_coded=[0]*SNRrange
nsyms_coded=[0]*SNRrange

r=[0,0]
```

```
for SNR in range(SNRrange):
    print('snr={0}\n'.format(SNR))
    N0=Eb*10**(-SNR/10)
    EbN0=Eb/N0
    theorprobQPSK[snrctr]=(1-(1-qf(math.sqrt(2*EbN0)))**2)
    theorprobBPSK[snrctr]=(qf(math.sqrt(2*EbN0)))
    sigma=math.sqrt(N0/2)

    if(doqpsk):
        #errctr.append(0)
        #nsyms.append(0)
        while (errctr[snrctr]<nerrortocount):
            nsyms[snrctr]=nsyms[snrctr]+1
            bits1=np.random.rand(2)>.5
            bits2=bits1.astype(int)
            symno=2*bits2[0]+bits2[1]
            constpt=Esscale*shapecodeclass1.const_pts[:,symno]
            noise1=sigma*np.random.randn(2)
            r[0]=constpt[0]+noise1[0]
            r[1]=constpt[1]+noise1[1]
            decodesymno=qpskdemod(r)
            if (decodesymno != symno):
                errctr[snrctr]=errctr[snrctr]+1

    if(doshapecode):
        #errctr_coded.append(0)
        #errctr_bits_coded.append(0)
        #nsyms_coded.append(0)
        while(errctr_coded[snrctr]<nerrortocount):
            nsyms_coded[snrctr]=nsyms_coded[snrctr]+1
            bits3=np.random.rand(2)>.5
            bits4=bits3.astype(int)
            bits4=bits4.reshape((2,))
            sym1,sym2=shapecodeclass1.encode(bits4)
            shapecodeclass1.bitqueuein(bits4)
            s1 = Escoded*shapecodeclass1.const_pts[:,sym1]
            s2 = Escoded*shapecodeclass1.const_pts[:,sym2]

            noise2=sigma*np.random.randn(2)
            noise3=sigma*np.random.randn(2)
            r1=s1+noise2
            r2=s2+noise3
            decodeout,bitsout= shapecodeclass1.viterbishapedecoder(r1,r2)
            if(decodeout):
                inbits = shapecodeclass1.bitqueueout()
                errctr_coded[snrctr] = errctr_coded[snrctr] + ...
                    np.any((inbits) != (bitsout)).astype(int)
                errctr_bits_coded[snrctr]=errctr_bits_coded[snrctr] ...
                    +sum(((inbits) != (bitsout)).astype(int))

    snrctr=snrctr+1

lenn=len(range(SNRrange))
legendstrs=[]
```

```
plt.semilogy(list(range(SNRrange)),theorprobQPSK,'g:')
plt.semilogy(list(range(SNRrange)),theorprobBPSK,'b:')

legendstrs.append('BPSK(th)')
legendstrs.append( 'QPSK(th)')

if(doqpsk):
    plt.semilogy(list(range(SNRrange)),np.array(errctr)/np.array(nsyms),'r:')
    legendstrs.append( 'QPSK(ex)')

if(doshapecode):
    plt.semilogy(list(range(SNRrange)), ...
        np.array(errctr_coded)/np.array(nsyms_coded),'c')
    legendstrs.append( 'Shapecode(sym)')
    plt.semilogy(list(range(SNRrange)), ...
        np.array(errctr_bits_coded)/(2*np.array(nsyms_coded)),'c:')
    legendstrs.append( 'Shapecode(bit)')
plt.legend(legendstrs)
print("--- %s seconds ---" % (time.time() - start_time))
if (shapecodeclass1.doconstanttarget):
    plt.savefig('BERcurveconstatnttarget.png',dpi=1500)
else:
    plt.savefig('BERcurve.png',dpi=1500)
plt.show()
```

### Python Code for finding Free Euclidean Distance

```


from collections import defaultdict
import numpy as np
import matplotlib.pyplot as plt
import math
class CreatePath:

    def __init__(self,totalstates):
        self.vertices= totalstates
        self.statenode = defaultdict(list)
        self.branchlist=defaultdict(list)
        self.index=0

    def AddBranch(self,initstate,finalstate):
        self.statenode[initstate].append(finalstate)

    def CheckVisitedBranchs(self, initstate, targetstate, branchvisited, ...
        branch):


        branchvisited[initstate]= True
        branch.append(initstate)
        if (initstate ==targetstate):

```

```
            branchh=list(branch)
            self.branchlist[self.index]=branchh
            self.index=self.index+1
        else:

            for ind in self.statenode[initstate]:
                if branchvisited[ind]==False:
                    self.CheckVisitedBranchs(ind, targetstate, ...
                        branchvisited, branch)
        branch.pop()
        branchvisited[initstate]= False

    def MakePaths(self,startingstate, targetstate):
        branchvisited =[False]*(self.vertices)
        branch = []
        self.CheckVisitedBranchs(startingstate, ...
            targetstate,branchvisited, branch)




def distancecal(r1,r2):
    return (r1[0]-r2[0])**2+(r1[1]-r2[1])**2

doplot=1
if doplot:

    for spine in plt.gca().spines.values():
                spine.set_visible(False)

    plt.gca().invert_yaxis()


loopcontrol=1
Nepochinit=0
M=16
Nstate=128
Nbits=math.log2(M)
target=0
mulipliedfactor=19
const_axis=np.arange(-(2**(Nbits/2)-1),(2**(Nbits/2)+1),2,dtype=int)
axis1=np.kron(np.ones(shape=(1,int(2**(Nbits/2))),dtype=int),const_axis)
axis2=np.kron(const_axis,np.ones(shape=(1,int(2**(Nbits/2))),dtype=int))
const_pts=np.concatenate((axis1, axis2), axis=0)
print(const_pts)
#const_pts=np.array([[ -3, -3, -3, -3, -1, -1, -1, -1, 3, 3, 3, 3, 1, 1, ...
    1, 1],[-3, -1, 3, 1, -3, -1, 3, 1, -3, -1, 3, 1, -3, -1, 3, 1]],np.int32)
#----------------------0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
#const_pts=const_pts[:,[1,0,15,5,14,2,8,6,9,10,4,13,7,11,12,3]]
print(const_pts)


if M==4:
    const_pts=np.array([[-1,1,1,-1],[-1,-1,1,1]],np.int32)
#bleow is dr sodha const for 16 QAM
#if M==16:
```

```
#    const_pts=np.array([[ -3, -3, -3, -3, -1, -1, -1, -1, 3, 3, 3, 3, 1, ...
    1, 1, 1],[-3, -1, 3, 1, -3, -1, 3, 1, -3, -1, 3, 1, -3, -1, 3, ...
    1]],np.int32)
symbollist=list(range(M))
bestmindistance=0
#const_pts=np.transpose(const_pts)
#np.random.shuffle(const_pts)
#const_pts=np.transpose(const_pts)
#while(loopcontrol==1):

for s in list(range(Nstate)):

    for offset in list(range(0-s,Nstate-s,1)):

        while(loopcontrol<3):

            Nepochinit=Nepochinit+1

            Nepoch=Nepochinit+1

            trellisoutput=np.zeros(shape=(2,2,Nstate,Nstate,Nepoch), ...
                dtype=int)*10000
            trellisbranch=np.zeros(shape=(2,Nstate,Nstate,Nepoch),dtype=int)


            g = CreatePath(Nstate*(Nepoch+1))
            initnode=0
            if doplot:
                plt.xticks(range(0, Nepoch+1, 1), fontsize=14)
                plt.yticks(range(0, Nstate+1, 1), fontsize=14)

            for epoch in list(range(Nepoch)):

                for state in list(range(Nstate)):
                    fromstate=state
                    symbolnum=0
                    #print(fromstate)

                    for symbol in symbollist:
                        #print(initnode)

                        newpoint=(symbol+state)%M
                        parity=(newpoint+target)%M
                        tostate=(mulipliedfactor*(parity+state)) %Nstate

                        fromy=fromstate
                        toy=tostate
                        trellisoutput[:,0,fromstate,tostate,epoch]= ...
                            const_pts[:,symbol]
                        trellisoutput[:,1,fromstate,tostate,epoch]= ...
                            const_pts[:,parity]
                        trellisbranch[0,fromstate,tostate,epoch]=symbol
                        trellisbranch[1,fromstate,tostate,epoch]=parity
                        g.AddBranch(fromy+initnode,initnode+Nstate+toy)
                initnode=initnode+Nstate

```

```


            d = Nepoch*Nstate+s+offset

            g.MakePaths(s, d)

            branchlist=defaultdict(list)
            revisedbranchlist=defaultdict(list)
            index=0


            for row in g.branchlist:

                for element in g.branchlist[row]:
                    branchlist[index].append(element% Nstate)

                index=index+1


            index=0
            for keyy in branchlist:
                checkk=branchlist[keyy]
                if checkk.count(s)==2 or checkk.count(s)==Nepoch+1:
                    for element in checkk:
                        revisedbranchlist[index].append(element)
                    index=index+1


            indexsave=index

            if doplot:
                for epoch in list(range(Nepoch)):
                    plotx1=epoch
                    plotx2=epoch+1
                    for key in revisedbranchlist:
                        fromy=revisedbranchlist[key][plotx1]
                        toy=revisedbranchlist[key][plotx2]
                        plt.plot([plotx1,plotx2],[fromy,toy],c='k')
                        plt.plot(plotx1,fromy,'ko')
                        plt.plot(plotx2,fromy,'ko')



            #phase 2

            if(Nepoch>2):
                Nepoch=Nepoch-1
                g = CreatePath(Nstate*(Nepoch+1))
                initnode=0




                for epoch in list(range(Nepoch)):

                    for state in list(range(Nstate)):
```

```
            fromstate=state
            symbolnum=0
            #print(fromstate)

            for symbol in symbollist:
                #print(initnode)

                newpoint=(symbol+state)%M
                parity=(newpoint+target)%M
                tostate=(mulipliedfactor*(parity+state)) %Nstate

                fromy=fromstate
                toy=tostate
                g.AddBranch(fromy+initnode,initnode+Nstate+toy)
        initnode=initnode+Nstate



    d = Nepoch*Nstate+s+offset
    branchlist=defaultdict(list)
    g.MakePaths(s, d)

    index1=0


    for row in g.branchlist:

        for element in g.branchlist[row]:
            branchlist[index1].append(element% Nstate)
        index1=index1+1

    for keyy in branchlist:
        checkk=branchlist[keyy]
        if checkk.count(s)==2 or checkk.count(s)==Nepoch+1:
            for element in checkk:
                revisedbranchlist[index].append(element)
            index=index+1



    if doplot:
        for epoch in list(range(Nepoch)):
            plotx1=epoch
            plotx2=epoch+1
            for key in ...
                list(range(indexsave,len(revisedbranchlist))):
                fromy=revisedbranchlist[key][plotx1]
                toy=revisedbranchlist[key][plotx2]
                plt.plot([plotx1,plotx2],[fromy,toy],c='k')
                plt.plot(plotx1,fromy,'ko')
                plt.plot(plotx2,fromy,'ko')

distancelist=[]
for key in revisedbranchlist:
    temp_list=revisedbranchlist[key]
```

```
            if temp_list.count(s) != Nepoch+1 and temp_list.count(s) ...
                != Nepoch+2:
                distance=0
                for ind in list(range(len(temp_list)-1)):
                    distance=distance+ distancecal(trellisoutput[:,0, ...
                        temp_list[ind], ...
                        temp_list[ind+1],0],trellisoutput[:,0,s,s,0])+ ...
                        distancecal(trellisoutput[:,1, ...
                        temp_list[ind],temp_list[ind+1],0], ...
                        trellisoutput[:,1,s,s,0])
                distancelist.append(distance)
            else:
                distancelist.append(1000)


       minimumpath=revisedbranchlist[distancelist. ...
           index(min(distancelist))]

        if doplot:
            for epoch in list(range(len(minimumpath)-1)):
                    plotx1=epoch
                    plotx2=epoch+1

                    fromy=minimumpath[plotx1]
                    toy=minimumpath[plotx2]
                    plt.plot([plotx1,plotx2],[fromy,toy],c='g')
                    plt.plot(plotx1,fromy,'g')
                    plt.plot(plotx2,fromy,'g')

        alldistancelist=[]
        bestmindistance=1000
        for key1 in revisedbranchlist:
            ref_path=revisedbranchlist[key1]
            for key2 in revisedbranchlist:
                current_path=revisedbranchlist[key2]
                if (current_path != ref_path and len(current_path)== ...
                    len(ref_path)):
                    distance=0
                    for ind in list(range(len(current_path)-1)):
                        distance=distance+distancecal( ...
                            trellisoutput[:,0, current_path[ind], ...
                            current_path[ind+1],0], ...
                            trellisoutput[:,0, ...
                            ref_path[ind],ref_path[ind+1],0]) \
                        +distancecal( trellisoutput[:,1, ...
                            current_path[ind], ...
                            current_path[ind+1],0], ...
                            trellisoutput[:,1, ref_path[ind], ...
                            ref_path[ind+1],0])
                    if (distance<bestmindistance):
                        bestmindistance=distance
                        final_reference_path=ref_path
                        final_current_path=current_path
```

```

            loopcontrol=len(revisedbranchlist)

if doplot:
    for epoch in list(range(len(final_reference_path)-1)):
                plotx1=epoch
                plotx2=epoch+1

                fromy=final_reference_path[plotx1]
                toy=final_reference_path[plotx2]
                plt.plot([plotx1,plotx2],[fromy,toy],c='b')
                plt.plot(plotx1,fromy,'b')
                plt.plot(plotx2,fromy,'b')
                fromy=final_current_path[plotx1]
                toy=final_current_path[plotx2]
                plt.plot([plotx1,plotx2],[fromy,toy],c='m')
                plt.plot(plotx1,fromy,'m')
                plt.plot(plotx2,fromy,'m')

print(min(distancelist))
print(bestmindistance)
print(final_reference_path)
print(final_current_path)
plt.figure()

plt.plot(const_pts[0,:],const_pts[1,:],'ko')
for ind in list(range(len(final_reference_path)-1)):
    initstate_ref=final_reference_path[ind]
    finalstate_ref=final_reference_path[ind+1]
    initstate_cur=final_current_path[ind]
    finalstate_cur=final_current_path[ind+1]
    print("epoch={}: Blue {}/{} magenta ...
        {}/{}".format(ind,trellisbranch[0,initstate_ref, ...
        finalstate_ref,0],trellisbranch[1,initstate_ref,finalstate_ref,0], ...
        trellisbranch[0,initstate_cur,finalstate_cur,0],trellisbranch[1, ...
        initstate_cur,finalstate_cur,0]))

    print(trellisbranch[0,initstate_ref,finalstate_ref,0],
        const_pts[:,trellisbranch[0,initstate_ref,finalstate_ref,0]])
    print(trellisbranch[1,initstate_ref,finalstate_ref,0],
        const_pts[:,trellisbranch[1,initstate_ref,finalstate_ref,0]])
    print(trellisbranch[0,initstate_cur,finalstate_cur,0],
        const_pts[:,trellisbranch[0,initstate_cur,finalstate_cur,0]])
    print(trellisbranch[1,initstate_cur,finalstate_cur,0], ...
        const_pts[:,trellisbranch[1,initstate_cur,finalstate_cur,0]])
    x=[const_pts[0,trellisbranch[0,initstate_ref,finalstate_ref,0]], ...
        const_pts[0,trellisbranch[0,initstate_cur,finalstate_cur,0]]]
    y=[const_pts[1,trellisbranch[0,initstate_ref,finalstate_ref,0]], ...
        const_pts[1,trellisbranch[0,initstate_cur,finalstate_cur,0]]]
    plt.plot(x,y, 'r')
    x=[const_pts[0,trellisbranch[1,initstate_ref,finalstate_ref,0]], ...
        const_pts[0,trellisbranch[1,initstate_cur,finalstate_cur,0]]]
    y=[const_pts[1,trellisbranch[1,initstate_ref,finalstate_ref,0]], ...
        const_pts[1,trellisbranch[1,initstate_cur,finalstate_cur,0]]]
    plt.plot(x,y, 'g--')

```

```
for i in list(range(len(symbollist))):
    plt.annotate(i, (const_pts[0,i]+.07,const_pts[1,i]+.07))
```