

Pairwise Optimization of Modulation Constellations for Non-Uniform Sources

by

Brendan F.D. Moore

A thesis submitted to the
Department of Mathematics and Statistics
in conformity with the requirements for
the degree of Master of Applied Science

Queen's University
Kingston, Ontario, Canada
September, 2009

Copyright © Brendan Moore, 2009

Abstract

The design of two-dimensional signal constellations for the transmission of binary non-uniform memoryless sources over additive white Gaussian noise channels is investigated. The main application of this problem is the implementation of improved constellations where transmitted data is highly non-uniform. A simple algorithm, which optimizes a constellation by re-arranging its points in a pairwise fashion (i.e., two points are modified at a time, with all other points remaining fixed), is presented. In general, the optimized constellations depend on both the source statistics and the signal-to-noise ratio (SNR) in the channel. We show that constellations designed with source statistics considered can yield symbol error rate (SER) performance that is substantially better than rectangular quadrature amplitude modulation signal sets used with either Gray mapping or more recently developed maps. SER gains as high as 5 dB in E_b/N_0 SNR are obtained for highly non-uniform sources.

Symbol mappings are also developed for the new constellations using a similar pairwise optimization method whereby we assign and compare a weighted score for each pair. These maps, when compared to the mappings used in conjunction with

the standard rectangular QAM constellation, again achieve considerable performance gains in terms of bit error rate (BER). Gains as high as 4 dB were achieved over rectangular QAM with Gray mapping, or more than 1 dB better than previously improved mappings.

Finally, the uncoded Pairwise Optimized system is compared to a standard tandem source and channel coding system. Neither system is universally better, and the trade-offs between the systems are investigated.

Acknowledgments

I would like to thank my supervisors, Dr. Fady Alajaji and Dr. Glen Takahara, for their ideas, suggestions, criticisms, edits and other help throughout this process. The majority of my expenses were covered by funding supplied by the Natural Sciences and Engineering Research Council of Canada, the H. K. Walter Award and the R. S. McLaughlin Fellowship, and I am grateful for that support. Where that was not enough, my parents happily (grudgingly?) helped with my tuition, rent and bills when I needed it, and I am lucky to have had their support as well.

Thanks to everybody else for offering time and brains to improve the work, and for tolerating me being in another city for two more years.

Contents

Abstract	ii
Acknowledgments	iv
List of Figures	xii
List of Tables	xiv
1 Introduction	1
1.1 Literature Review	2
1.2 Problem Statement	4
1.3 Contributions	5
1.4 Thesis Outline	6
2 Background	8
2.1 Source and Channel Models	8
2.1.1 Non-Uniform <i>I.I.D.</i> Binary Source	9
2.1.2 AWGN Channel	9

<i>CONTENTS</i>	vi
2.2 Modulation and Demodulation	11
2.2.1 Quadrature Amplitude Modulation	12
2.2.2 ML vs. MAP Decoding	13
2.3 Source and Channel Coding	15
2.3.1 Huffman Code	16
2.3.2 Convolutional Channel Coding	17
3 Pairwise Optimization of M-ary Constellations	19
3.1 Pairwise Optimization Algorithm and Design	20
3.1.1 Algorithm	24
3.2 Initial Constellations	28
3.2.1 Rectangular QAM	28
3.2.2 Concentric Circles	29
3.2.3 Bad Constellations	32
3.3 Results and Performance	32
3.3.1 Impact of Initial Constellations	33
3.3.2 Binary and Quaternary Constellations	33
3.3.3 16-ary Constellations and Robustness	35
3.3.4 64-ary and 256-ary Constellations	39
4 Designing Maps for the PO Constellations	57
4.1 Initialization and Probability Constraint	58

<i>CONTENTS</i>	vii
4.2 Objectives	59
4.2.1 Defining the Neighbourhood and Weighted Hamming Score . .	60
4.3 Map Improvement Algorithm	62
4.4 Results and Performance	64
4.4.1 16-ary Constellations	64
4.4.2 64-ary and 256-ary Constellations	66
5 Comparison to Source and Channel Coding	75
5.1 System for Comparison	76
5.2 Performance Comparison	78
5.3 Complexity Comparison	81
6 Conclusions	86
6.1 Summary	86
6.2 Future Work	87
Bibliography	89
A Constellation Coordinates	92
B Source Code	104
B.1 Optimization	104
B.1.1 Initializing a Constellation	104
B.1.2 Setting Up and Running the GUI	105

CONTENTS

viii

B.1.3	Optimizing Pairs	106
B.2	Simulation	108
B.2.1	Loading Constellations	108
B.2.2	Running the Simulations	109
B.2.3	Tandem Scheme	110

List of Figures

3.1	Example circles over which \vec{s}_1 and \vec{s}_2 are optimized, with all other points remaining constant.	22
3.2	Convergence of union upper bound of SER for PO16.	25
3.3	Convergence of union upper bound of SER for PO64.	26
3.4	Standard 16-QAM modulation constellation.	28
3.5	Constellation to minimize conditional probability of error for \vec{s}_u	30
3.6	Initial constellation placing more likely points closer to the origin on concentric circles (here $M = 16$).	42
3.7	PO4 constellation for $p = 0.9$. Designed for $SNR = 0 dB$	43
3.8	Performance of size $M = 4$ constellations for $p = 0.9$. Optimized from [10] and PO4 are both designed for $SNR = 0 dB$	44
3.9	Pairwise optimized constellation for $M = 16$, $p = 0.9$ and design $SNR = 1 dB$	45

3.10 Performance of size $M = 16$ constellations for $p = 0.9$ and design $SNR = 1$ dB. Performance of a specialized constellation (i.e., with design SNR identical to true SNR) also shown.	46
3.11 Pairwise optimized constellation for $M = 16$, $p = 0.9$ and design $SNR = 0$ dB.	47
3.12 Pairwise optimized constellation for $M = 16$, $p = 0.9$ and design $SNR = -3$ dB.	48
3.13 Pairwise optimized constellation for $M = 16$, $p = 0.9$ and design $SNR = -5$ dB.	49
3.14 Pairwise optimized constellation for $M = 16$, $p = 0.9$ and design $SNR = -10$ dB.	50
3.15 Pairwise optimized constellation for $M = 16$, $p = 0.9$ and design $SNR = -20$ dB.	51
3.16 Pairwise optimized constellation for $M = 16$, $p = 0.9$ and design $SNR = 10$ dB.	52
3.17 Performance of $M = 16$ constellations for varying values of p and design $SNR = 1$ dB.	53
3.18 Pairwise optimized constellation for $M = 64$, $p = 0.9$ and design $SNR = 2$ dB.	54

3.19	Performance of constellations for $M = 64$, $p = 0.9$ and design $SNR = 2dB$ and the pairwise optimized constellation for $M = 256$ with design $SNR = 4 dB$. BPSK also shown for reference.	55
3.20	Pairwise optimized constellation for $M = 256$, $p = 0.9$ and design $SNR = 4 dB$	56
4.1	Standard 16-QAM modulation constellation with a Gray mapping. . .	60
4.2	Two example neighbourhoods are shown for PO16 ($M = 16$ and $k = 4$). . .	61
4.3	PO constellation for $M = 16$ with improved mapping.	65
4.4	BER Performance of 16-ary constellations. PO constellation simulated with mapping seen in Fig. 4.3.	69
4.5	PO constellation for $M = 64$ with improved mapping.	70
4.6	PO constellation for $M = 256$ with improved mapping.	71
4.7	BER Performance of 64-ary constellations (and PO256). PO constellations simulated with mappings seen in Fig. 4.5 and 4.6.	72
4.8	BER Performance of all M -ary PO constellations presented.	73
4.9	PO constellation for $M = 8$ (designed for $p = 0.9$ and $SNR = 0 dB$) with symbol mapping.	74
5.1	State machine representing the convolutional channel code with constraint length $k = 3$ and rate $r = 1/2$	77

5.2 Performance of tandem source and channel coding scheme for various block lengths. Selected PO constellation performance shown for reference. 84

List of Tables

5.1	Fourth-order Huffman code used for the tandem scheme.	83
5.2	Average tail corruption length and occurrence rate (within the Viterbi decoder) for various message block sizes. Results shown only at <i>Crossover SNR</i> (the point at which the tandem scheme overall BER performance matches that of PO4). The tandem scheme does not outperform PO4 for block size 12, so this Crossover SNR is where it surpasses PO16. .	85
A.1	Coordinates and bit mapping for PO4 constellation, for $p = 0.9$ and design SNR of 0 <i>dB</i>	92
A.2	Coordinates and bit mapping for PO8 constellation, for $p = 0.9$ and design SNR of 0 <i>dB</i>	93
A.3	Coordinates and bit mapping for PO16 constellation, for $p = 0.9$ and design SNR of 1 <i>dB</i>	93
A.4	Coordinates for PO16 constellation, for $p = 0.5$ and design SNR of 1 <i>dB</i> .	94
A.5	Coordinates for PO16 constellation, for $p = 0.6$ and design SNR of 1 <i>dB</i> .	95

A.6	Coordinates for PO16 constellation, for $p = 0.7$ and design SNR of 1 <i>dB</i> .	95
A.7	Coordinates for PO16 constellation, for $p = 0.8$ and design SNR of 1 <i>dB</i> .	96
A.8	Coordinates and bit mapping for PO64 constellation, for $p = 0.9$ and design SNR of 2 <i>dB</i>	97
A.9	Coordinates and bit mapping for PO256 constellation, for $p = 0.9$ and design SNR of 4 <i>dB</i>	99

Chapter 1

Introduction

Communication in the modern world means digitizing and transmitting practically every type of information imaginable. Data from a *source* is generally converted into binary data (strings of ones and zeros) using a variety of methods for easier manipulation across different systems. For various sources of information, the binary data can end up with different proportions of bits being zero and one. When the split is approximately 50/50 in the long run, we call this *uniform* data. Transmitting data from uniform sources is fairly well understood, and there are many existing schemes and modulation constellations for accomplishing the task, such as rectangular quadrature amplitude modulation (QAM) using Gray mapping. When we consider non-uniform data, however, these well-known constellations and maps are not optimal. For these non-uniform sources, we wish to consider designing constellations which can achieve better performance than the standard systems by exploiting our knowledge

of the non-uniformity of the source.

1.1 Literature Review

For uniformly distributed sources, rectangular QAM using Gray mapping is known to perform well, and the Gray map is shown as the optimal map in terms of bit error rate (BER) for high enough signal-to-noise ratios (SNR) [1]. As noted in [14], however, there are many real world examples of data sources which are highly non-uniform, such as text (email and instant/short messages), medical images and encoded voice data [2]. Compression will often have residual redundancy in the output due to non-ideal coding methods [3]. Rather than using traditional source and channel coding (which can be sensitive to noise-related errors in decoding if optimal variable-length source coding is used), we can choose instead to directly exploit the non-uniformity of the source via the modulation scheme, while gaining noise-resiliency in many cases and significantly reducing system complexity and delay [3]. Such an approach can be quite attractive for complexity-constrained and delay-sensitive applications such as wireless sensor networks. In these non-uniform situations, the performance of Gray mapped M -ary rectangular QAM is sub-optimal, since regular discrete spacing does not account for the source distribution [7]. One simple improvement is to exploit the knowledge of symbol probability by implementing (optimal) maximum *a posteriori* (MAP) decoding (instead of maximum-likelihood decoding) at the receiver.

In [14], new $M1$ -mappings were developed to improve performance of M -ary rect-

angular QAM and phase-shift keying constellations. It is also noted in [14] that performance can be improved by translating each mapped constellation so that it has zero mean, which we confirm. Here we consider making further changes to the constellations in order to achieve lower symbol error rate (SER). In [6], such a constellation design problem was considered for uniform sources under additive white Gaussian noise (AWGN). The uniformity of the source yields symmetric constellations, and also tends to have equal separation between points within the constellations. This is because the equiprobable points each get the same “share” of the available space. When considering non-uniform symbols, one desires more distance between the likely points and its neighbours to allow correct decoding in the presence of larger than average noise.

The idea of exploring constellations for non-uniform data was originally considered in [8] for the two-point, one-dimensional constellation. In [8], they solve for the optimal binary pulse amplitude modulation (BPAM) symbol amplitudes (the results of our work presented within match these existing results). In [10, 5], optimal constellations are considered for $M = 4$ (for non-uniform sources). [10] considers a general non-uniform source, and presents constellations for various degrees of non-uniformity as well as different levels of noise. We extend the ideas of these searches for non-uniform constellations up to larger constellations, and compare the results of our methods to those previously developed.

1.2 Problem Statement

We consider a memoryless source $\{X_n\}$ which generates independent binary symbols $\{0, 1\}$ non-uniformly with $p = Pr\{X_n = 0\} > \frac{1}{2}$. We wish to transmit this data over an AWGN channel with noise variance of $\frac{N_0}{2}$ per dimension. We assume that an M -ary two-dimensional (2-D) modulation scheme is to be used, and that it is desirable to maximize data throughput per transmission while achieving the lowest possible SER. For convenience, we assume M to be a power of two. Binary symbols are grouped into sequences of $\log_2 M$ bits, forming a new symbol sequence $\{Y_n\}$ having M distinct values $\{s_1, s_2, \dots, s_M\}$ with probabilities $\{p_1, p_2, \dots, p_M\}$. The probabilities are defined by the number of zeros in the bit sequence. If sequence s_i has n_i zeros, then $p_i = p^{n_i}(1-p)^{\log_2 M - n_i}$. (In the constellation diagrams that come later, we refer to equiprobable symbols by the number of zeros, n , they have in their corresponding binary sequence.) Each channel symbol is then mapped to a signal point, \vec{s}_i , in some initial M -ary constellation, where $\vec{s}_i = (s_{i,x}, s_{i,y})$. Our objective is then to change the arrangement of the points in that constellation to achieve the lowest SER possible at a given SNR E_b/N_0 , where E_b is the average energy per bit, $E_s/\log_2 M$.

The search space to be considered is continuous and consists of all collections of points $\{\vec{s}_1, \vec{s}_2, \dots, \vec{s}_M\}$ satisfying

(i) a zero mean constraint: $\sum_{i=1}^M p_i \vec{s}_i = 0$; and

(ii) an average symbol energy constraint: $\sum_{i=1}^M p_i \|\vec{s}_i\|^2 = E$,

where the average energy, E , is given. Note that E and E_b are related by $E_b = \frac{E}{\log_2 M}$. Our objective function is the SER. For $M = 2$, the optimal constellation was found analytically in [8], but as the constellation size grows, so does the complexity of problem. In [10], the authors design optimal constellations for $M = 4$ by numerically evaluating tight error bounds developed in [9]. Our first goal is to design signal point arrangements (constellations) that are near-optimal for larger constellation sizes, such as $M = 16, 64, 256$, under MAP decoding. Our second goal is to design appropriate symbol maps for these new constellations to achieve the best performance when considering bit errors.

1.3 Contributions

The contributions of this thesis are as follows:

1. A Pairwise Optimization (PO) method is presented for developing new modulation constellations for transmitting data from non-uniform sources. Performance (in terms of SER) is compared to standard modulation constellations for sizes $M = 2, 4, 16, 64, 256$, as well as constellations from existing literature.
2. A method is introduced for designing good maps for the asymmetric and irregular constellations created by the PO algorithm. Performance (in terms of BER) is again compared to standard modulation constellations and maps, and some previously improved maps.

3. Trade-offs (in terms of both performance and complexity) between this uncoded transmission scheme and a tandem source and channel coded system are investigated.

1.4 Thesis Outline

The remainder of this thesis is organized as follows:

In Chapter 2, we introduce the source and channel models we will be using in our designs and simulations. We also describe the fundamentals of modulation and demodulation, and two metrics used for demodulation decisions. Included here are descriptions of some standard constellations, which we will use for comparison. We also present some basics of source and channel coding as groundwork for comparison to a completely different type of system. In Chapter 3, we develop our PO process for designing constellation for non-uniform sources. Before considering larger constellations, we compare our findings to the existing literature for small constellations. From there, we evaluate the performance of our system in terms of SER. In Chapter 4, we describe an iterative method for designing symbol mappings for the PO constellations, and compare the results to existing maps for traditional constellations. We compare the uncoded PO constellations and maps to a coded system in Chapter 5, using a tandem source and channel coding scheme (separately, but simultaneous). Finally, we draw our conclusions in Chapter 6 and present some possible future work. Appendix B includes some selected samples of important parts of our source code for running

the PO algorithm, as well as some descriptions of the use of the code. Appendix A includes tables of point coordinates and symbol mappings for the constellations presented in the earlier chapters.

Chapter 2

Background

2.1 Source and Channel Models

A *source* can be anything that outputs data. This data can be text that is input by a user, measurements observed by a thermometer, a voice signal on a mobile phone or any other information stream. Sources can come from just about anywhere, and be expressed in many different ways, but are mainly either analog signals or digitized data.

A *channel* is any conduit or medium over which the information/data from a source may be transmitted. Some common channels are the copper lines used for traditional telephones, fiber optic links used by major Internet connections, radio waves used by your favorite music or news stations, or the higher frequency electromagnetic waves used by your mobile phone.

In general, we are concerned with data and transmission that might typically be used in a wireless scenario. The specific source and channel models we will be considering are explained here.

2.1.1 Non-Uniform *I.I.D.* Binary Source

In particular, we want to consider a non-uniform binary source. This means that whatever form the data originally took, it has been converted into a binary stream $\{X_1, X_2, \dots\}$, where $X_i \in \{0, 1\}$. The bits $\{X_i\}$ are generated from *independent and identically distributed* (i.i.d.) Bernoulli trials with probability $p = Pr\{X_i = 0\} \geq \frac{1}{2}$.

When $p = \frac{1}{2}$, such a source is known as a uniform source, which has approximately the same number of zeros and ones in a long sequence of bits. We are interested in non-uniform sources, where $p > \frac{1}{2}$ and the majority of bits in a long sequence are zeros. As mentioned in Chapter 1, there are many examples of such non-uniform data, such as uncompressed images (medical scans like MRI or x-ray), email and text messages, and most sub-optimally compressed redundant sources [3].

For the purpose of our constellation designs and simulations, we generally consider a source with probability $p = 0.9$ in what follows, except in Section 3.3.3.

2.1.2 AWGN Channel

To understand the fundamental ways that constellations can be designed when considering source statistics, the channel model we used is the relatively simple *additive*

white Gaussian noise (AWGN) channel. This channel model essentially simulates the background noise that can be expected when dealing with any wireless communications system but does not consider more specific obstacles, such as fading or multipath interference. The AWGN channel works by perturbing each transmission a single time with the addition of noise following a Gaussian distribution. That is to say when the sender transmits signal X , the receiver will observe a signal Y , where

$$X + Z = Y, \text{ where } Z \sim N(0, N_0).$$

Here, Z is the additive noise with a zero-mean Gaussian distribution having variance N_0 , the noise power.

Since we generally consider 2D transmission in the work that follows, we employ the 2D form of the above. Given total noise power of N_0 , the noise power *per dimension* will be $\frac{N_0}{2}$. When we transmit $\vec{X} = (x_x, x_y)$, we will observe a signal $\vec{Y} = (y_x, y_y)$, where

$$\vec{X} + \vec{n} = \vec{Y}, \text{ where } \vec{n} = (n_x, n_y), \text{ and } n_x, n_y \sim N(0, \frac{N_0}{2}).$$

This model allows us to use well understood approximations of the errors caused by noisy transmission conditions during the design process. Additionally, it keeps our simulations fairly straightforward while providing a reasonable measurement of system performance.

2.2 Modulation and Demodulation

Modulation is what is done to our source data to prepare it for transmission over the channel. This is effectively converting the digital data into an analog waveform which is modified in some pre-defined way so that it can carry data. For a given carrier waveform, data may be added by modifying the phase, amplitude or frequency of the wave, or any combination of the three. These modifications of the signal are limited to a finite list of M symbols, $\{\vec{s}_1, \vec{s}_2, \dots, \vec{s}_M\}$, known as the modulation alphabet. The choice of M is decided by how many bits we want to send per transmission, m , and is defined by $M = 2^m$, so that each of the M symbols represents one of the 2^m unique sequences of m bits. For [hase and amplitude modulation, each of these M symbols can be thought of as a point in the two-dimensional (2D) Cartesian plane, where $\vec{s}_i = (s_{i,x}, s_{i,y})$. It is possible to consider higher dimensional transmissions, but this thesis is limited to the 2D case, which is most common.

Taken together, these M symbols form what is known as a *modulation constellation* – the collection of points in the 2D plane we will be examining for the remainder of this thesis. Our objective will be to design the modulation constellation (i.e., to specify the M points in the plane) under appropriate constraints, so as to minimize the symbol error rate (SER) for a given source distribution and noise power (Chapter 3). Given an optimized constellation, a further design goal is to determine a good mapping of bit sequences to constellation symbols so as to reduce the bit error rate (BER) (Chapter 4).

2.2.1 Quadrature Amplitude Modulation

Perhaps the most common modulation constellation used in practice are those of quadrature amplitude modulation (QAM) is a 2D transmission scheme to transmit data. As described in [12, §7.3.3], the two pieces of information carried by QAM can be considered as either:

- (i) the symbol amplitude (or, energy) and the symbol phase (or angle); or
- (ii) the x -axis coordinate, and the y -axis coordinate.

For the purposes of both the design and simulation within this thesis, we choose the latter option. During simulation, we transmit the x and y coordinates separately. Since we are dealing with Gaussian noise with variance N_0 *in total*, during simulation our QAM transmissions experience noise power of $\frac{N_0}{2}$ *per dimension* [9]. This applies to both our new PO constellations as well as all of the standard constellations for comparison.

For transmissions using QAM, standard constellations fall on a rectangular grid. We call this *rectangular QAM*, and an example of such a constellation is shown in Fig. 3.4 for $M = 16$. The grid layout makes intuitive sense for uniform data, since points are evenly spaced (no preference given) and which points lay closer to the origin is not important. The idea presented in this thesis is to design new constellations which do not use this grid, but instead have their points arranged to better exploit the non-uniformity of the source data. This can be done both by using less energy in

clever ways by placing more likely points nearer to the origin, and given more space around likely points to reduce the conditional SER for those points.

2.2.2 ML vs. MAP Decoding

In an ideal world, a receiver would be able to observe exactly the same signal that the transmitter sent. Unfortunately, we must deal with the fact that noise distorts our transmissions and causes our receiver to detect a signal (hopefully) close to the original symbol, but not necessarily exactly the same. The signal that is actually received when \vec{s}_i is sent is

$$\vec{s}_r = \vec{s}_i + \vec{n},$$

where \vec{n} is the white Gaussian noise added by the channel. From that observation, the receiver must pick which symbol from the alphabet it considers as the *intended* symbol, and decode to the corresponding data. To accomplish this, there are many possible methods.

Maximum likelihood (ML) decoding considers only the observed signal position, \vec{s}_r , and the positions of the points in the constellation. The signal decoded, \vec{s}_d , is that which lies closest (in terms of Euclidian distance) to what was received. That is,

$$\vec{s}_d = \underset{\vec{s}_u \in \{\vec{s}_1, \vec{s}_2, \dots, \vec{s}_M\}}{\operatorname{argmin}} \|\vec{s}_r - \vec{s}_u\|.$$

This is optimal (in terms of minimizing symbol decoding probability of error) for uniform sources, since no point is weighted more heavily than any other. This type of decoding is generally used with the standard rectangular QAM constellations we

previously described, as well as many others created for uniform sources. But what about non-uniform sources?

Maximum a posteriori (MAP) decoding takes into account both the distance to the received signal (as in ML decoding) as well as the original symbol probabilities. In this case, we want to weight our decision appropriately by the probability of each possible symbol which may have been transmitted. Unlike the case of ML decoding, we want to find the symbol \vec{s}_u *maximizing* the probability of being sent conditional on to the received signal. With MAP decoding it is assumed that the distribution of the AWGN noise and the *a priori* symbol probabilities are known, and the decoded symbol is chosen as

$$\vec{s}_d = \underset{\vec{s}_u \in \{\vec{s}_1, \vec{s}_2, \dots, \vec{s}_M\}}{\operatorname{argmax}} p_u \cdot \exp\left(\frac{-\|\vec{s}_r - \vec{s}_u\|^2}{N_0}\right), \quad (2.1)$$

where p_u is the probability that symbol \vec{s}_u is sent (without considering the received signal) according to the source distribution.

The advantage of MAP decoding is that we favour more strongly those source symbols which are more likely. Because of this, the constellation is able to withstand greater noise during transmission of likely symbols. In designing our PO constellations, we take further advantage of the MAP metric by placing more space around the more likely symbols, greatly increasing the area of their decision regions, and reducing the possibility of a decoding error for these high-probability symbols. For a non-uniform source, MAP decoding is optimal; it reduces to ML decoding if the source is in fact uniform.

2.3 Source and Channel Coding

In Chapter 5 we compare the PO to a tandem source and channel coded system. Here we explain what this means.

Source coding is the act of compressing the source symbols to remove redundancy in the data, expressing the original message in the fewest possible bits by encoding it using a dictionary of *codewords* corresponding to sequences of message bits. This compression can be either lossless (the original message can be identically decoded) or lossy (some aspects of the original message are lost, and some close approximation is decoded). The output of an optimal compression scheme is (asymptotically) perfectly uniform data, exhibiting no residual redundancy. In practice, most compression schemes are sub-optimal and have some level of residual redundancy.

Channel coding can be thought of as the opposite of source coding. Channel coding *adds* bits to the message in a controlled manner which can be used to detect, and possibly correct, errors incurred during transmission. As a very simple example, consider the use of bit duplication. For each bit in the message to be transmitted, we transmit that same bit twice rather than just once. If the receiver decodes two *different* bits, it knows there must have been an error during transmission. In other words, we added bits to protect the data from errors, and used those bits to detect a problem.

The tandem scheme employed in Chapter 5 uses a fourth-order Huffman code, together with a convolutional channel code. These are described here.

2.3.1 Huffman Code

The *Huffman code* used here is a variable-length prefix code which minimizes the expected codeword length $\left(\sum_{i=1}^M p_i l_i\right)$ for the source alphabet being compressed. As explained in [4, §5.6], it is constructed by iteratively grouping the two least likely symbols (or groups) together under a branch of a binary tree (since we wish to have binary data), until the last grouping unifies the entire alphabet, then assigns binary codewords following the splits of the tree. By doing so, we arrive at a list of codewords for our source symbols having the following properties:

1. No codeword is the prefix of any other codeword.
2. The lengths are inversely ordered with the symbol probabilities (i.e., if $p_i > p_j$, then $l_i \leq l_j$).
3. The two longest codewords have the same length.
4. Two of the longest codewords differ only in the last bit and correspond to the least likely symbols.

While there are other optimal codes, the Huffman code gives one optimal code [4, §5.8]. For our comparison, we use a fourth-order Huffman code, meaning we consider groups of four source bits at a time, resulting in the Huffman code represented in Table 5.1 (tailored to our non-uniform source for $p = 0.9$).

2.3.2 Convolutional Channel Coding

The *convolutional* channel code works by linking blocks of message bits together in something that could be roughly described as discrete convolution. More specifically, overlapping sequences of bits are used to generate parity bits, and these parity bits are transmitted over the channel, rather than the message itself. The parity bits are determined by a set of *generator functions*, and serve to indicate the next bit of the message by representing a transition within a state machine, as explained in [11, §8.2]. This state-transition method is advantageous because it becomes very unlikely that individual errors during transmission lead to decoding errors.

The optimal decoder for the convolution code, which is a sequence ML decoder, is known as the *Viterbi decoder*. Its function is to reconstruct the message data from the received parity bits. It accomplishes this by calculating a metric for every possible path through the state machine, based on the observed parity bits, and then working backwards to reconstruct the most likely path. The details can be found in [11]. The Viterbi decoder is able to “smooth over” transmission errors by linking together paths on either side of the error which are likely to match up. The Viterbi decoder we implement uses *soft decoding*, meaning that it does not decode its observations into bits before looking at the paths, but rather considers the actual received signals during processing. It is known that this yields approximately 2 dB gain when compared to an equivalent *hard* Viterbi decoder.

At high enough SNRs, the convolutional code performs much better for longer

block lengths, since it allows the Viterbi decoder to measure greater differentiation between individual path metrics. This is reflected in the data presented in Chapter 5.

Chapter 3

Pairwise Optimization of M-ary Constellations

In this chapter, we consider a new method for developing improved signal constellations for 2-D transmission. The possible constellations are essentially any set of M -points in two dimensions. While the search space is continuous, the zero mean and average power constraints may be used to reduce the search complexity. The zero mean constraint is a necessary property of any optimal (in terms of minimal SER) constellation with constrained average energy, since SER performance under MAP decoding is not affected by translation or rotation of the constellation (e.g., [10]); it is only affected by changing the relative distances between points. It is of note that for non-uniform sources, rectangular (symmetric) constellations such as 16-, 64- and

256-QAM are not zero mean. Since the variance

$$\begin{aligned}
\sum_{i=1}^M p_i \|\vec{s}_i - \bar{s}\|^2 &= \sum_{i=1}^M p_i (\vec{s}_i - \bar{s})^T (\vec{s}_i - \bar{s}) \\
&= \sum_{i=1}^M p_i (\vec{s}_i^T \vec{s}_i - 2\vec{s}_i^T \bar{s} + \bar{s}^T \bar{s}) \\
&= \|\bar{s}\|^2 + \sum_{i=1}^M p_i \|\vec{s}_i\|^2 - 2\bar{s}^T \sum_{i=1}^M p_i \vec{s}_i \\
&= \sum_{i=1}^M p_i \|\vec{s}_i\|^2 - \|\bar{s}\|^2
\end{aligned} \tag{3.1}$$

is constant under translations of the constellation, we minimize the average energy in (3.1), $\sum_{i=1}^M p_i \|\vec{s}_i\|^2$, by shifting the constellation to be zero mean (i.e., $\|\bar{s}\|^2 = 0$). So, it is possible to improve such non-zero mean constellations slightly by translating them to be zero mean and scaling them up to their original average energy. This will increase the separation between all points, which subsequently improves the resiliency of the constellation in the presence of noise.

3.1 Pairwise Optimization Algorithm and Design

For a given initial constellation, it is not possible to change the position of a single point while still adhering to the zero mean and average energy constraints. Changing the coordinates of a single point would certainly shift the left hand side of 3.2 to be non-zero. Taking any pair of points, however, allows us to move those two points around in concert while still adhering to both of the constraints we have imposed. If

\vec{s}_1 and \vec{s}_2 are the two selected points, then the zero mean constraint

$$\sum_{i=1}^M p_i \vec{s}_i = 0 \quad (3.2)$$

implies that

$$p_1 \vec{s}_1 + p_2 \vec{s}_2 = - \sum_{i=3}^M p_i \vec{s}_i.$$

So, if we let $\vec{b} = \sum_{i=3}^M p_i \vec{s}_i$, then

$$\vec{s}_1 = \frac{1}{p_1} (-\vec{b} - p_2 \vec{s}_2)$$

or

$$\vec{s}_1 = \vec{a} - c \vec{s}_2$$

where $\vec{a} = -\frac{\vec{b}}{p_1}$ and $c = \frac{p_2}{p_1}$. Thus, the x and y coordinates ($s_{1,x}$ and $s_{1,y}$) of \vec{s}_1 are determined by the coordinates ($s_{2,x}$, $s_{2,y}$) of \vec{s}_2 by:

$$s_{1,x} = a_x - c \cdot s_{2,x} \text{ and } s_{1,y} = a_y - c \cdot s_{2,y}. \quad (3.3)$$

Then the average energy constraint

$$\sum_{i=1}^M p_i \|\vec{s}_i\|^2 = E, \quad (3.4)$$

where E is the average symbol energy, implies the following:

$$p_1 \|\vec{s}_1\|^2 + p_2 \|\vec{s}_2\|^2 = E - \sum_{i=3}^M p_i \|\vec{s}_i\|^2. \quad (3.5)$$

Letting the constant $d = \sum_{i=3}^M p_i \|\vec{s}_i\|^2$ and substituting (3.3) in (3.5) yields

$$\begin{aligned} p_1 \left((a_x - c \cdot s_{2,x})^2 + (a_y - c \cdot s_{2,y})^2 \right) \\ + p_2 (s_{2,x}^2 + s_{2,y}^2) = E - d. \end{aligned} \quad (3.6)$$

Expanding and completing the square gives us

$$\left(s_{2,x} - \frac{p_1 a_x}{p_1 + p_2}\right)^2 + \left(s_{2,y} - \frac{p_1 a_y}{p_1 + p_2}\right)^2 = r^2 \quad (3.7)$$

where $r^2 = \frac{p_1(E-d)}{p_2(p_1+p_2)} - p_1 p_2 (a_x^2 + a_y^2)$.

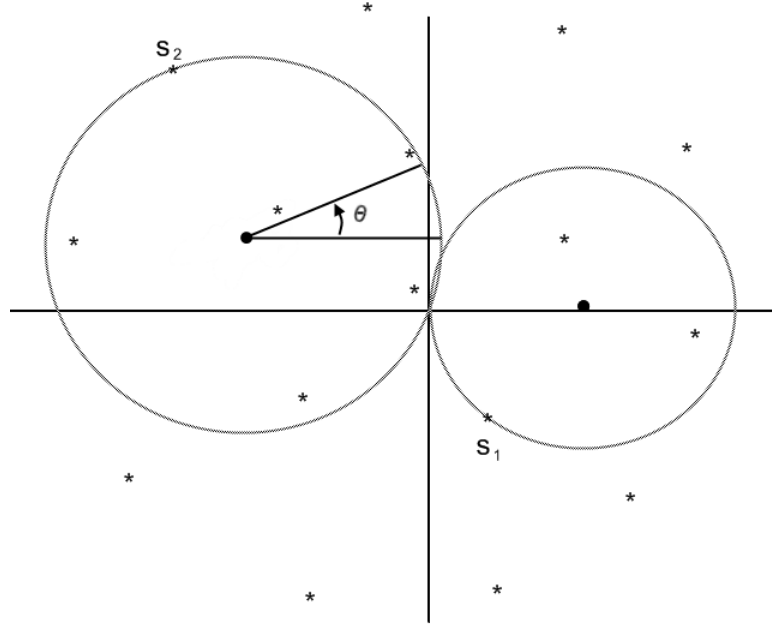


Figure 3.1: Example circles over which \vec{s}_1 and \vec{s}_2 are optimized, with all other points remaining constant.

Under the imposed constraints, Eqn. (3.7) defines a circle, centered at $\left(\frac{p_1 a_x}{p_1 + p_2}, \frac{p_1 a_y}{p_1 + p_2}\right)$ with radius r , on which \vec{s}_2 may travel, and the relationship given by Eqn. (3.3) defines a second, corresponding, circle for \vec{s}_1 to travel around. With (3.7), for each pair of signals (\vec{s}_1, \vec{s}_2) , the problem of searching over four variables $(s_{1,x}, s_{1,y}, s_{2,x}, s_{2,y})$ is effectively reduced to searching over a single variable, θ , which is the angle parameterizing the circle for \vec{s}_2 , measured counterclockwise relative to the positive x -axis for

the center of the circle (see Fig. 3.1). For a given value of θ , \vec{s}_2 is defined, and \vec{s}_1 has a corresponding position. It is over this parameter θ that each pair of points can be optimized for performance.

It is interesting to consider why it is we get these circles. In dealing with M constellation points, we are optimizing in $2M$ dimensions. The zero mean constraint restricts two of those dimensions, leaving a $2M - 2$ dimensional hyperplane. The average energy constraint similarly represents a $2M$ dimensional ellipsoid, the intersection of which with the hyperplane is a smaller ellipsoid. The circles over which we are performing our pairwise search come from taking the $2 - D$ projection of this higher dimensional ellipsoid.

With regard to the performance for a potential constellation, we consider the union upper bound on the SER P_s . The union bound can be inaccurate for low SNRs, but it is fairly tight for medium to high SNRs. The tight upper and lower bounds of [9] can also be used to improve the accuracy of SER calculated during the design stage. However, since the union bound is used only during the iterative design stage (not to evaluate performance), it is accurate enough for our purposes and has the additional benefit of computational speed and simplicity:

$$P_s = \sum_{u=1}^M P(\epsilon|\vec{s}_u)P(\vec{s}_u) \quad (3.8)$$

$$= \sum_{u=1}^M P\left(\bigcup_{i \neq u} \epsilon_{iu}\right) P(\vec{s}_u)$$

$$\leq \sum_{u=1}^M \sum_{i \neq u} P(\epsilon_{iu})P(\vec{s}_u) \quad (3.9)$$

where ϵ is the event indicating any decoding error has occurred, ϵ_{iu} is the event that \vec{s}_i is decoded erroneously given that \vec{s}_u is transmitted,

$$P(\epsilon_{iu}) = Q\left(\frac{\|\vec{s}_i - \vec{s}_u\|}{\sqrt{2N_0}} + \frac{\sqrt{2N_0} \ln \frac{P(\vec{s}_u)}{P(\vec{s}_i)}}{2\|\vec{s}_i - \vec{s}_u\|}\right)$$

(as in [9]) and $Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-y^2/2} dy$ is the Gaussian Q -function. Note that $P(\epsilon_{iu})$ is the probability that \vec{s}_i has a larger MAP decoding metric (refer back to (2.1)) than \vec{s}_u given that \vec{s}_u was sent.

When considering only the pair of points \vec{s}_1 and \vec{s}_2 , we can ignore the terms in Eqn. (3.9) for $u \neq 1, 2$ and $i \neq 1, 2$ as they will remain constant even as \vec{s}_1 and \vec{s}_2 move about their respective circles. The remaining terms we need to use to calculate the upper bound are

$$\begin{aligned} F_{12} &= \sum_{i \neq 1} P(\epsilon_{i1})P(\vec{s}_1) + \sum_{i \neq 2} P(\epsilon_{i2})P(\vec{s}_2) \\ &\quad + \sum_{u=3}^M P(\vec{s}_u) (P(\epsilon_{1u}) + P(\epsilon_{2u})) \end{aligned} \quad (3.10)$$

which is the objective function to be minimized for each pair of points being optimized.

3.1.1 Algorithm

The Pairwise Optimization (PO) algorithm is implemented as follows:

1. Configure some initial constellation, ensuring it adheres to the zero mean and average energy constraints.
2. Randomly (uniformly) select a pair of points (\vec{s}_1, \vec{s}_2) .

3. Calculate the constrained circles from (3.7) and (3.3).
4. Find the positions of (\vec{s}_1, \vec{s}_2) minimizing (3.10).
5. Go back to Step 2 and repeat until the constellation stabilizes.

Whereas earlier treatments of this topic typically used gradient search methods [5, 6, 10], we instead employ our randomized pairwise search. While the gradient search is effective for smaller constellations, it becomes increasingly troublesome for larger constellations, as the number of local optima which can *catch* the gradient search increases significantly. Using this random pair selection allows us to be more robust against local solutions, by letting the pair of symbols in the constellation take bigger jumps at each iteration.

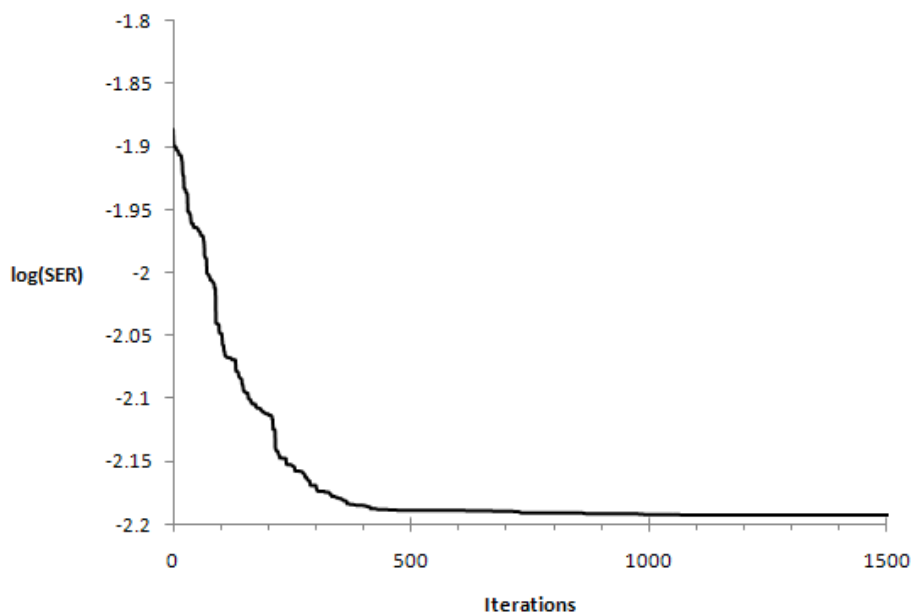


Figure 3.2: Convergence of union upper bound of SER for PO16.

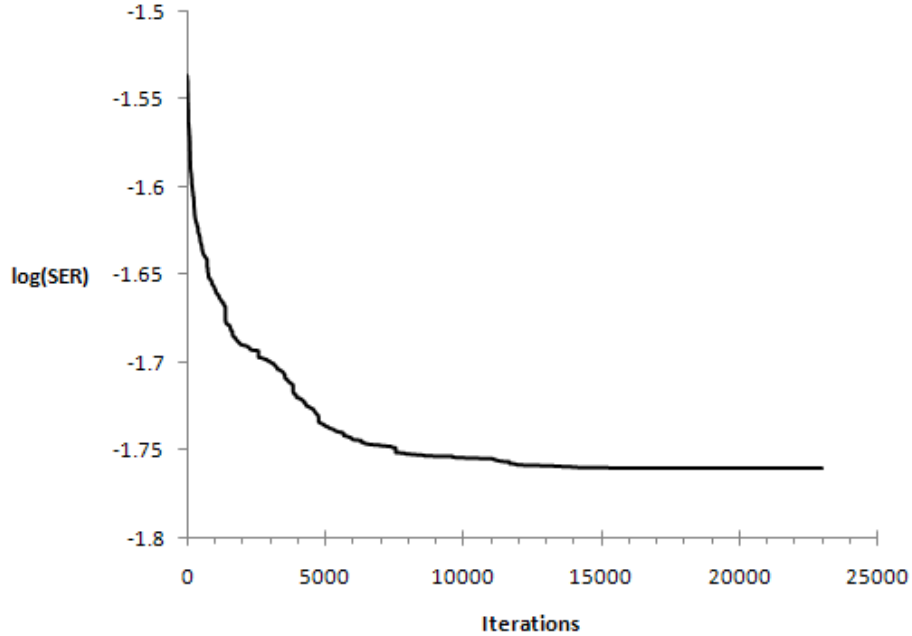


Figure 3.3: Convergence of union upper bound of SER for PO64.

The initial constellation used in Step 1 contains the source information implicitly through the symbol probabilities. Tests using different initial constellations (rectangular, circular, asymmetric) all yielded similar results, although with quite varying convergence rates (bad constellations were slower). The initial constellations chosen will be discussed in more detail in Section 3.2. In Step 4, we calculate the circle noted in Eqn. (3.7) and set angle θ to be 0 relative to the x -axis, and take discrete steps counterclockwise. At each step of θ , F_{12} is calculated using the corresponding \vec{s}_1 and \vec{s}_2 on their respective circles, and the design SNR (E_b/N_0), which is set as a constant. This is a simple and brute-force approach, but it works well enough for our intentions. The complexity of the algorithm can be approximated by the number of

times we calculate the Gaussian Q -function. For each pair of points being optimized, we calculate F_{12} for 50 steps of θ , each of which requires $4M$ calls to $Q(\cdot)$ as in (3.10), or $200M$ calls per pair. We need roughly $2M^2$ pairs before good constellations are achieved, for a total of $400M^3$ calls (each call takes approx. $3 \mu s$ on our 3.0 GHz AMD hardware, for $M = 16$). The convergence to a stable union upper bound of SER can be seen in Fig. 3.2 for PO16 and in Fig. 3.2 for PO64. When executed, our algorithm stabilizes in a matter of seconds for sizes up to $M = 16$, and scales up to three or four hours for $M = 256$.

Stabilization, as used in Step 5, means visual inspection of the constellation at this point. When considering the speed of convergence, it is difficult to be precise, since we do not know what the optimal constellation looks like, nor the final PO constellation for larger sizes. In general, the more likely symbols settle quickly, but the large number of unlikely symbols in large constellations tend to continue rearranging themselves (with better performance at each step) for much longer. The PO algorithm must converge on a final constellation (possibly a local minimum) since each iteration can only decrease the union upper bound, and SER in a non-negative quantity. Since we have that $UnionBound(i) \geq 0$ for all i and the PO algorithm is such that $UnionBound(i) \geq UnionBound(i + 1)$, we must have the union bound converging to some stable value as the number of iterations goes to infinity.

3.2 Initial Constellations

The PO algorithm needs to have some starting constellation given as input before it can begin to optimize individual pairs. With that in mind, we must consider what types of constellations we will use to initialize the algorithm.

3.2.1 Rectangular QAM

One obvious place to begin is with a constellation used in standard implementations: rectangular QAM.

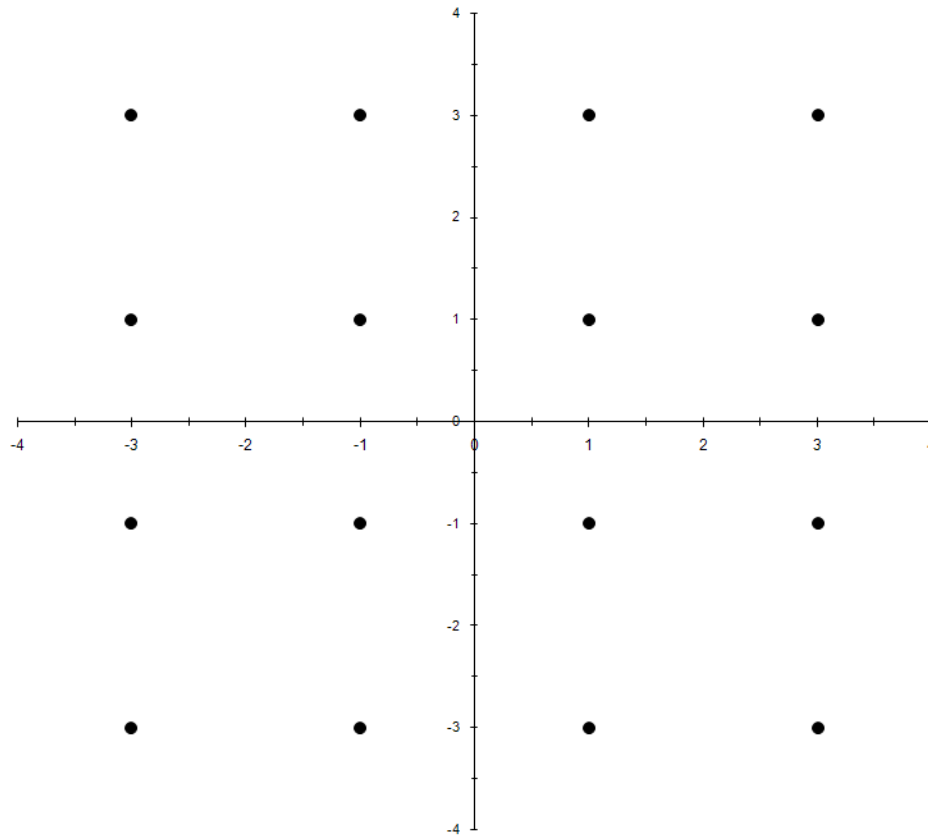


Figure 3.4: Standard 16-QAM modulation constellation.

In Fig. 3.4 we can see the geometry of the rectangular 16-QAM constellation. This constellation puts uniform spacing between adjacent points, and adheres to a grid pattern which allows for a simpler hardware implementation. However, it was designed with a uniform source (i.e., with $p = 0.5$) in mind. While in [14] new maps were presented which considers the non-uniformity of the source, the constellation itself is not changed. The results of this rigidity is a non-zero mean constellation for non-uniform sources. As discussed, a trivial improvement to these constellations is to translate them to zero mean, and to scale them up to their original average energy.

3.2.2 Concentric Circles

Here we intend to present some heuristic improvements to the initial constellation, which will generally be in line with our expectation for the final results. In order to get the most out of the average energy constraint, we feel that it makes sense as a general design principle to place symbols with higher probabilities closer to the center of the constellation. This will “free up” some of the available energy, and allow the lower probability symbols to sit farther away from the origin and other points.

More specifically, consider the individual conditional probabilities of error which contribute to the total SER shown in (3.8). How can we try to minimize $P(\epsilon|\vec{s}_u)$ for a given symbol \vec{s}_u ? If we only care about minimizing the errors associated with \vec{s}_u , we should try to keep it as far from the other symbols as possible, subject to the constraints and MAP decoding. Our first instinct here is to put \vec{s}_u by itself on the

positive x-axis, and cluster the remaining $M - 1$ points together at a common point on the negative x-axis. For a uniform source, this constellation would minimize the conditional probability of error. For a nonuniform source, however, we can do better.

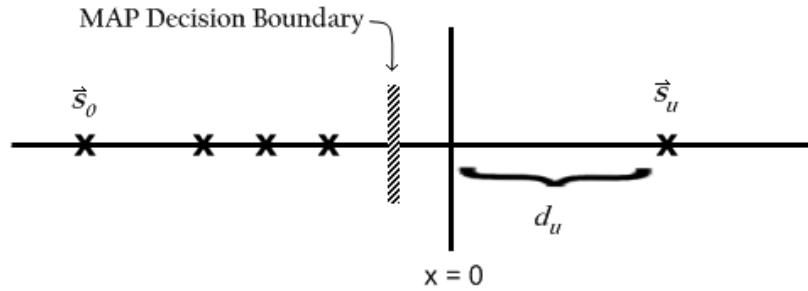


Figure 3.5: Constellation to minimize conditional probability of error for \vec{s}_u .

Since, in general, some of those clustered points have higher probability than others in the cluster, the MAP decoding decision boundary to dominate will be that corresponding to the most likely point in the cluster, which we will call \vec{s}_0 . That is to say, the MAP decoding metric (see (2.1)) of \vec{s}_0 will be greater than that of any of the other clustered symbols, regardless of the received signal. In order to minimize the conditional probability, we can move some of those lower probability symbols closer to \vec{s}_u until we have coincident MAP decision boundaries for every other point, resulting in a constellation as seen in Fig. 3.5, where \vec{s}_u lies d_u from the origin. If this is then done for each symbol, we have a set of distances $\{d_1, \dots, d_M\}$ where $d_u = d_i$ for $p_u = p_i$, for $i = 1, \dots, M$.

We can argue this as the optimal constellation to minimize the conditional prob-

ability of error for \vec{s}_u , $P(\epsilon|\vec{s}_u)$, as follows. In general, a larger decoding region will lead to a smaller conditional probability of error for that symbol. So, we wish to maximize the decoding region for \vec{s}_u . This immediately implies that we must have all other symbols lying along a line passing through \vec{s}_u (without loss of generality, we take this line to be the x-axis) and must all be on one side of \vec{s}_u (as seen in Fig. 3.5). If any point were to be moved *off* the x-axis, then its decision boundary could cut a diagonal through the constellation and reduce the decision region for \vec{s}_u . Similarly, if any point were to lay on the opposite side of \vec{s}_u from the rest of the points, then the decision region for \vec{s}_u would be reduced to a vertical strip in the plane, instead of the entire right side. As to the spacing of points, we have already mentioned that the error is dominated by the decision boundary closest to \vec{s}_u . In order to maximize the decision region, we must move the closest boundary farther away from \vec{s}_u . Since we are bound by the average energy constraint, we must move some other symbol \vec{s}_j *closer* to \vec{s}_u (and subsequently its decision boundary will get closer to \vec{s}_u) in order to move \vec{s}_0 (or any other symbol) and its decision boundary *farther* from \vec{s}_u . The termination of this procedure is when all of the decision boundaries are coincident, at which point no symbol may be moved farther from \vec{s}_u without requiring some other point be moved closer.

We then use the distances $\{d_1, \dots, d_M\}$ to create an initial constellation, as seen in Fig. 3.6, of concentric circles of symbols. Within a given layer, all points are equiprobable. The most likely point(s) is nearest the origin, and the least likely

points are the farthest away from the origin, lying on correspondingly larger circles. It is our expectation that the optimal constellation for a non-uniform source will at least resemble this constellation, with the ordering of points conserved (i.e., more likely points lie closer to the origin).

3.2.3 Bad Constellations

As a test of the robustness of the algorithm, we will also test using some deliberately bad constellations, of the type seen in Fig. 3.5, which favours a single symbol, but ignores the performance for the rest. All symbols, except one, will be clustered near one another (even laying coincident), with just a single symbol removed to a distance. It is our expectation that the algorithm will produce good output even from this state.¹

3.3 Results and Performance

We consider the memoryless non-uniform binary source with $Pr0 = p$ for transmission over an AWGN channel using constellation sizes of $M = 2, 4, 16, 64, 256$ and compare the SER performance (via simulations), under symbol-by-symbol MAP decoding, of our pairwise optimized constellations (which are denoted by PO2, PO4, \dots , PO256)

¹The algorithm will fail if all points lie in exactly the same spot, since this is not really a constellation, and there is no energy available to be redistributed once it is centered to be zero mean.

to existing constellations. We use $p = 0.9$ in all of the simulations, except for the discussion in Section 3.3.3. In the plots of our constellations for this chapter, we note the symbol probabilities using n , the number of zeros in the corresponding binary sequence.

3.3.1 Impact of Initial Constellations

The configuration of the initial constellation does not appear to contribute towards deciding the geometry of the constellation reached by the PO algorithm. However, the initial constellation *does* affect the time until convergence. Starting with bad constellations (those which have many closely clustered points) forces many early iterations to be spent spreading those symbols apart. Initial constellations with all symbols well-spaced essentially give the PO algorithm a head start, allowing pairs to be optimized immediately, or much sooner. While the impact of this delay is minimal for small constellations, it becomes a considerable time cost for larger constellations (such as $M = 64$ and $M = 256$).

3.3.2 Binary and Quaternary Constellations

We begin by comparing our results with the known optimal constellation presented in [8] for $M = 2$. Our algorithm directly arrives at the same final constellation as the work in [8], as shown in Eqn. (3.6) with both $\vec{a} = \vec{0}$ and $d = 0$ (since we have no

symbols beyond \vec{s}_1 and \vec{s}_2):

$$p((-c \cdot s_{2,x})^2 + (-c \cdot s_{2,y})^2) + (1-p)(s_{2,x}^2 + s_{2,y}^2) = E$$

and we choose the point with $s_{2,y} = 0$. Thus

$$\begin{aligned} p(-c \cdot s_{2,x})^2 + (1-p)s_{2,x}^2 &= E \\ s_{2,x} &= \sqrt{\frac{E}{pc^2 + (1-p)}} = \sqrt{\frac{E \cdot p}{(1-p)}} \\ s_{1,x} &= -c \cdot s_{2,x} = -\sqrt{\frac{E \cdot (1-p)}{p}} \end{aligned}$$

which is the result obtained in [8]. Note that for $M = 2$, the union bound in (3.9) yields the exact SER. While the pairwise algorithm is not limited to one dimension, the results are equivalent after rotation. Simulation confirms an exact SER performance match, as expected. There is no consideration of design SNR for $M = 2$ because the constraints alone fix the relative positions of \vec{s}_1 and \vec{s}_2 , and we have no other points with respect to which we may optimize.

We next consider the constellations found in [10] for $M = 4$. When the pairwise optimization stabilizes, the resulting constellation, seen in Fig. 3.7, is nearly identical to those arrived at in [10] for the given design parameters (in this case $p = 0.9$ and $SNR = 0$ dB), up to a rotation and/or reflection. In Fig. 3.8, it is clear that the pairwise optimized constellation PO4 performs identically to the optimized $M = 4$ constellation of [10]. Both constellations perform considerably better than quaternary phase shift keying (QPSK) for highly non-uniform sources, with nearly 5 dB gain for $SER \leq 10^{-2}$. The above results indicate that the algorithm does in fact tend

toward optimal constellations, and we may proceed to apply it to larger modulation constellations, where optimal constellations are not known.

3.3.3 16-ary Constellations and Robustness

Before investigating large constellations, we will examine the performance of the 16 point constellation. The PO constellation is shown in Fig. 3.9. In Fig. 3.10, the M1 mapping of [14] already improves the performance of (rectangular) 16-QAM by approximately 1 dB. We can also see that the pairwise optimized constellation PO16 achieves a further improvement of 2 dB over the M1 mapping, for a total gain of 3 dB over Gray mapped 16-QAM. This PO16 constellation was designed for a noise level of $SNR = 1$ dB, but perhaps overall performance across all noise levels is not as good as it could be. To examine this, also included in Fig. 3.10 is the performance at each true SNR step of a specialized constellation designed specifically for that noise level. It is clear that this specialized configuration does not provide considerable gains over a constellation designed at a single SNR that is carefully selected (in this case, 1 dB) and used for transmission across all noise levels. This shows that a constellation designed using a single appropriately chosen design SNR can provide robust performance *vis-à-vis* changes in the true SNR, and that it is not necessary to have a set of constellations tuned to every channel noise level.

Effect of Design for Decreasing SNR

We have just mentioned that it is sufficient to design a PO constellation using a single, well chosen SNR. It is interesting to consider, though, what happens to the constellations when designed at very low SNR (i.e., in the presence of a great deal of noise). We will consider the constellation of Fig. 3.9 a starting point, as this is essentially the constellation that emerges for any design SNR above 1 dB. We first take a small step down to design at $SNR = 0$ dB, and arrive at the constellation seen in Fig. 3.11. We do not see many differences yet, only small discrepancies in symbol placement. Dropping down to design $SNR = -3$ dB we start to see some changes, shown in Fig. 3.12. Here we see one of the lower probability points has been placed in close proximity to the most likely point near the center of the constellation. As this point gets closer to the origin, it will effectively never be decoded, since the most likely symbol should always have a higher MAP metric. In Fig. 3.13, the design SNR has been lowered again to -5 dB. While the general geometry of the constellation remains the same, we see more of the clustering phenomenon that was originally noted in [15]. In this constellation, there are two lower probability symbols clustered near the most likely point, both of which will rarely be the decoded symbol.

At design $SNR = -10$ dB, Fig. 3.14 show that the resulting constellation is now quite different than previous examples. The less likely point clustered at the center is effectively coincident to the most likely symbol. As such, it will never be decoded. We also see that there is clustering among the less likely symbols farther from the center.

Instead of being more evenly spread out, they are gathered into small groups that are farther apart. As [15] notes, and we are inclined to agree, this allows the errors between groups to be greatly reduced at the expense of more likely errors within a cluster. Since we are dealing with a highly non-uniform source, we have the advantage of being able to err on the safe side with MAP decoding, and will simply decode the most likely symbol of a given cluster.

Our final constellation for this discussion, shown in Fig. 3.15, has been designed at $SNR = -20$ dB. It is quite similar to the previous constellation, but with three symbols clustered towards the center as seen earlier. Again we see the clustering throughout most of the other points. These constellations are not particularly useful in practice, however, as no constellation is going to perform well in the presence of such high noise power; but, they will considerably outperform standard Gray-mapped QAM constellations.

Effect of Design for Higher SNR

When designing constellations for higher SNR, the PO algorithm creates constellations similar to the one seen in Fig. 3.9 for mid to high SNR values. As we move up to quite high SNR, however, the resulting configuration tends to rely less and less on the source distribution, since all points are very likely to be decoded correctly, regardless of placement. This situation essentially turns into ML decoding (since the probabilities do not significantly affect the MAP metric for very small noise), and

the resulting constellation reflects this by having points placed equidistant from one another. For design at 10 dB , the constellation is shown in Fig. 3.16, and is nearly identical to the constellation seen in Fig. 6(d) of [6].

Gains for Other Source Distributions

While we have been using $p = 0.9$ for all optimizations and simulations so far, there are also gains achieved over Gray-mapped 16-QAM by PO16 for smaller values of p . Looking at Fig. 3.17, we can see that this is the case. For $p = 0.5$ (a uniform binary source), the gain achieved was negligible. This is exactly as expected, since a uniform source does not allow us to take advantage of the source characteristics in the constellation design. With a small step up to $p = 0.6$, the gain achieved is about 0.25 dB . There is not much non-uniformity to take advantage of. For $p = 0.7$ and $p = 0.8$, more significant gains of 0.5 dB and 1.5 dB are achieved, respectively. Again, this is in line with expectations, since the greater non-uniformity in the source provides more opportunities to exploit in the constellation design. The combined effect of allowing more likely points to have more space around them with the increased noise recovery of MAP decoding greatly improves results for the most likely points in the constellation. We will continue to use $p = 0.9$ for the remaining results to show the largest gain possible, but this demonstrates that the method is viable for lower values.

Practical Considerations

When considering the possibility of designing for a range of both p and SNR values, we must address the practicality of doing so. Since we have shown the PO constellations to be robust over a reasonable range of true SNRs using only a single design SNR, we could arguably provide constellations suited to large steps in noise power. Both the transmitter and receiver would need to know the constellations in advance, and be capable of measuring noise power in the channel (e.g., using a pilot tone) and signaling when a constellation switch should happen. This can be accomplished with a very small overhead scheme which transmits the constellation to be used after every several thousand transmissions, using only a few bits each time (i.e., a single transmission) to do so. Similarly, we can implement multiple constellations to address a range of source distributions for reasonable steps in p . Doing so we would arrive at a set of constellations for each of a small number of SNRs and perhaps five to ten different p -values. This array of solutions would allow the best performance gains possible for the current source and channel characteristics, using only a small transmission overhead to signal the switches. This switching scheme has not been implemented in this thesis, but is a viable option for any real-world implementation.

3.3.4 64-ary and 256-ary Constellations

The result of pairwise optimization of a 64 point constellation using design $SNR = 2$ dB is shown in Fig. 3.18. Again we see the tendency of more likely points to lay

closer to the origin. This keeps the average energy low, allowing less likely points to sit farther away, thus creating more distance between points overall. In Fig. 3.19, we compare the performance of this constellation to 64-QAM with the M1 and Gray mappings. The M1 64-QAM mapping developed in [14] already outperforms Gray mapped 64-QAM by approximately 3.5 dB for any given SER. The pairwise optimized constellation we develop here, PO64, outperforms 64-QAM with M1 mapping by another 1.5 dB at a given SER, for a total improvement of about 5 dB over 64-QAM with Gray mapping. It is interesting to note that for medium and high SNRs (above 4 dB), the PO64 constellation achieves better SER than the BER of binary phase shift keying (BPSK). It is likely that the BER of PO16 will be lower than that of BPSK for sufficiently high values of p . The performance of the pairwise optimized constellation for $M = 256$ (PO256 in Fig. 3.20) is better than 64-QAM with Gray map by approximately 2 dB for any SER (see Fig. 3.19). Note that PO256 has both a higher data rate (two more bits per symbol) and a lower SER than Gray mapped rectangular 64-QAM at all SNRs, thus improving both system performance and throughput.

It is worth noting that the relative SER performance of the various PO constellation was monotonic with respect to M . That is, as the size of the constellation, M , increases, so does the SER (meaning worse performance), with increasing step sizes between the larger constellations. This pattern is to be expected, since the larger constellations correspond to higher transmission rates (source bits per channel use),

so more power is needed to achieve the same SER performance. This is interesting to note here for SER, because it is not the case when we move to considering symbol mapping and measuring BER, as we will do in the next chapter.

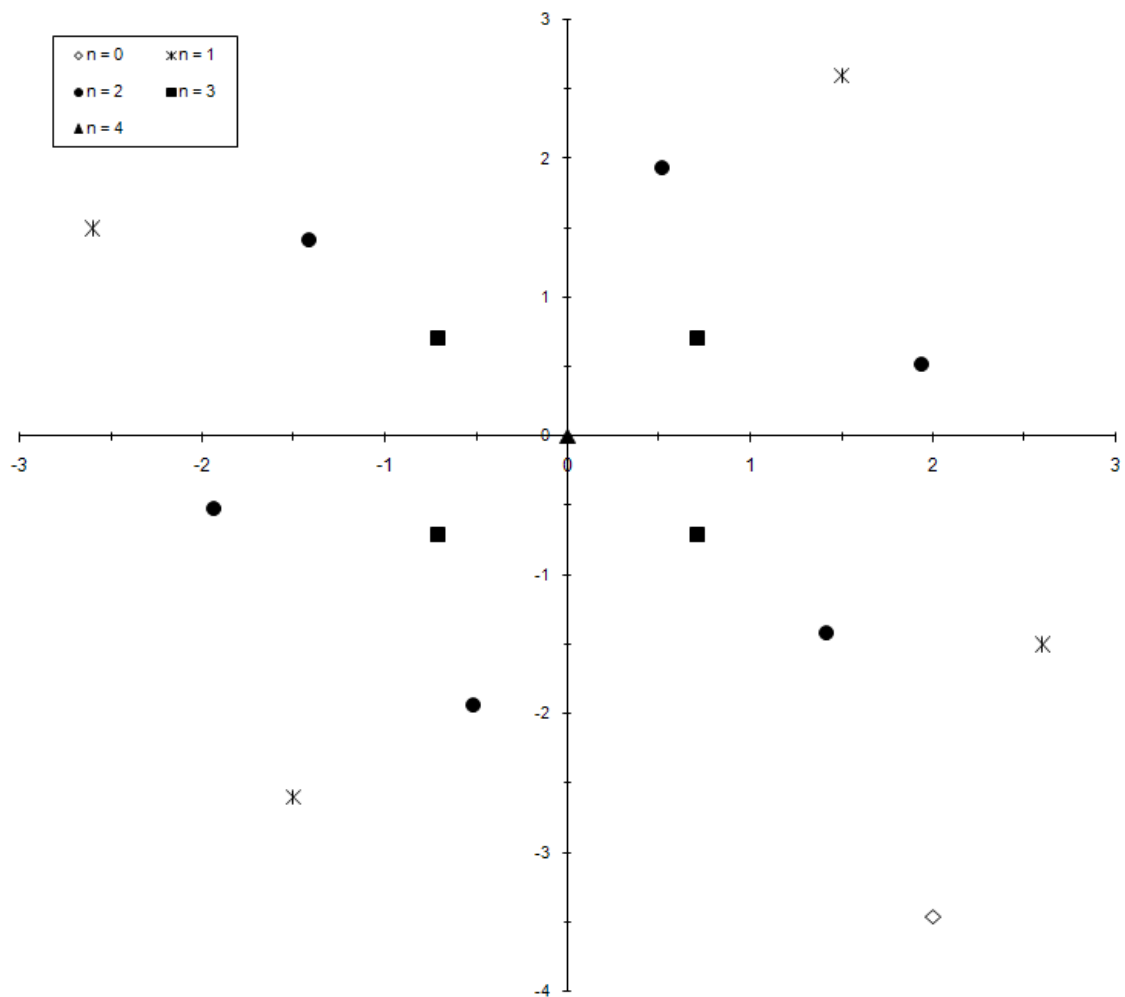


Figure 3.6: Initial constellation placing more likely points closer to the origin on concentric circles (here $M = 16$).

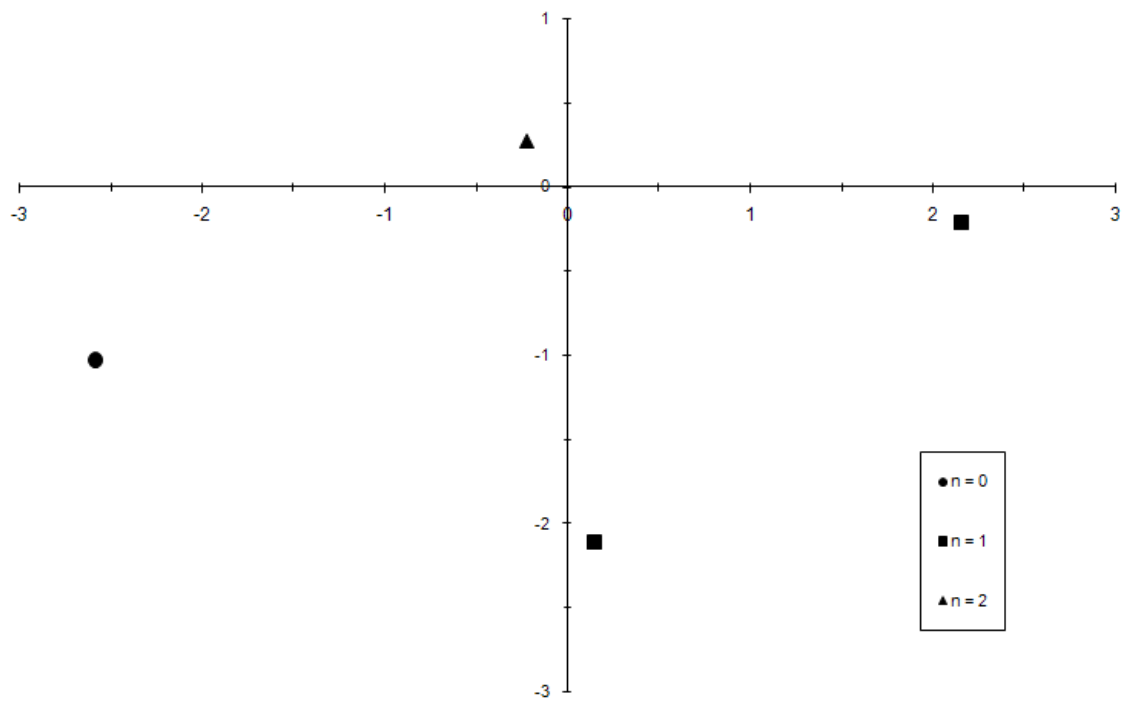


Figure 3.7: PO4 constellation for $p = 0.9$. Designed for $SNR = 0$ dB.

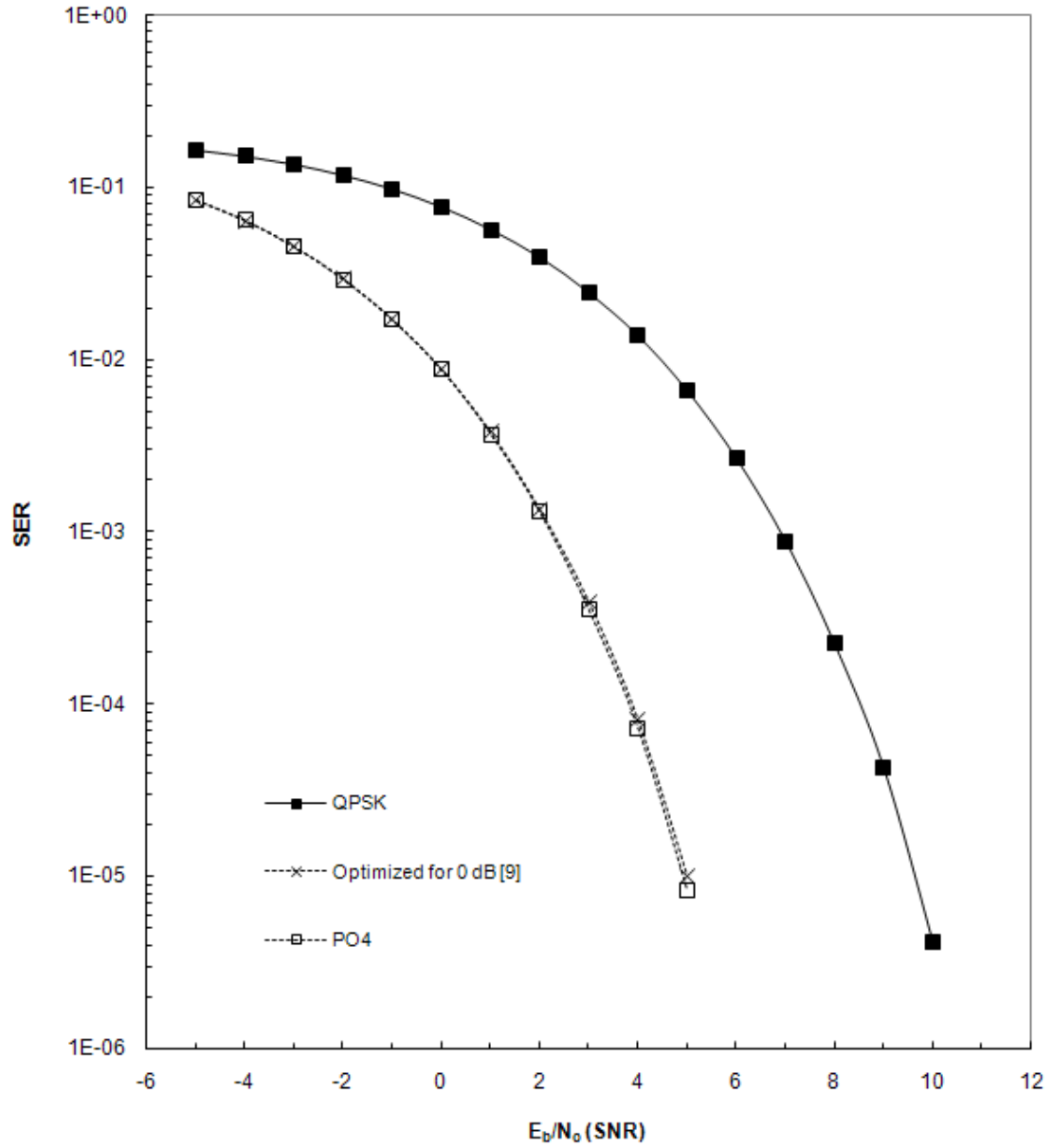


Figure 3.8: Performance of size $M = 4$ constellations for $p = 0.9$. Optimized from [10] and PO4 are both designed for $SNR = 0$ dB.

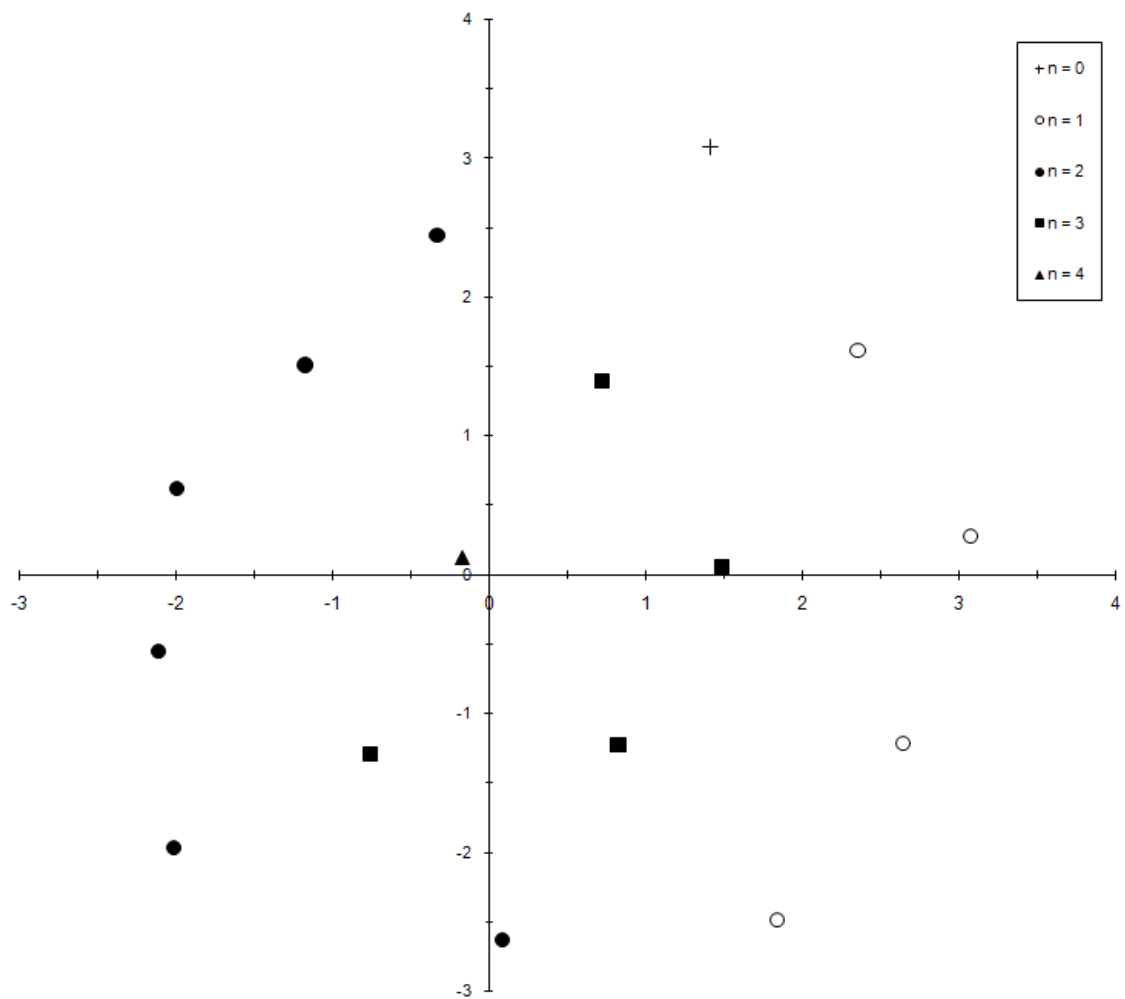


Figure 3.9: Pairwise optimized constellation for $M = 16$, $p = 0.9$ and design $SNR = 1$ dB.

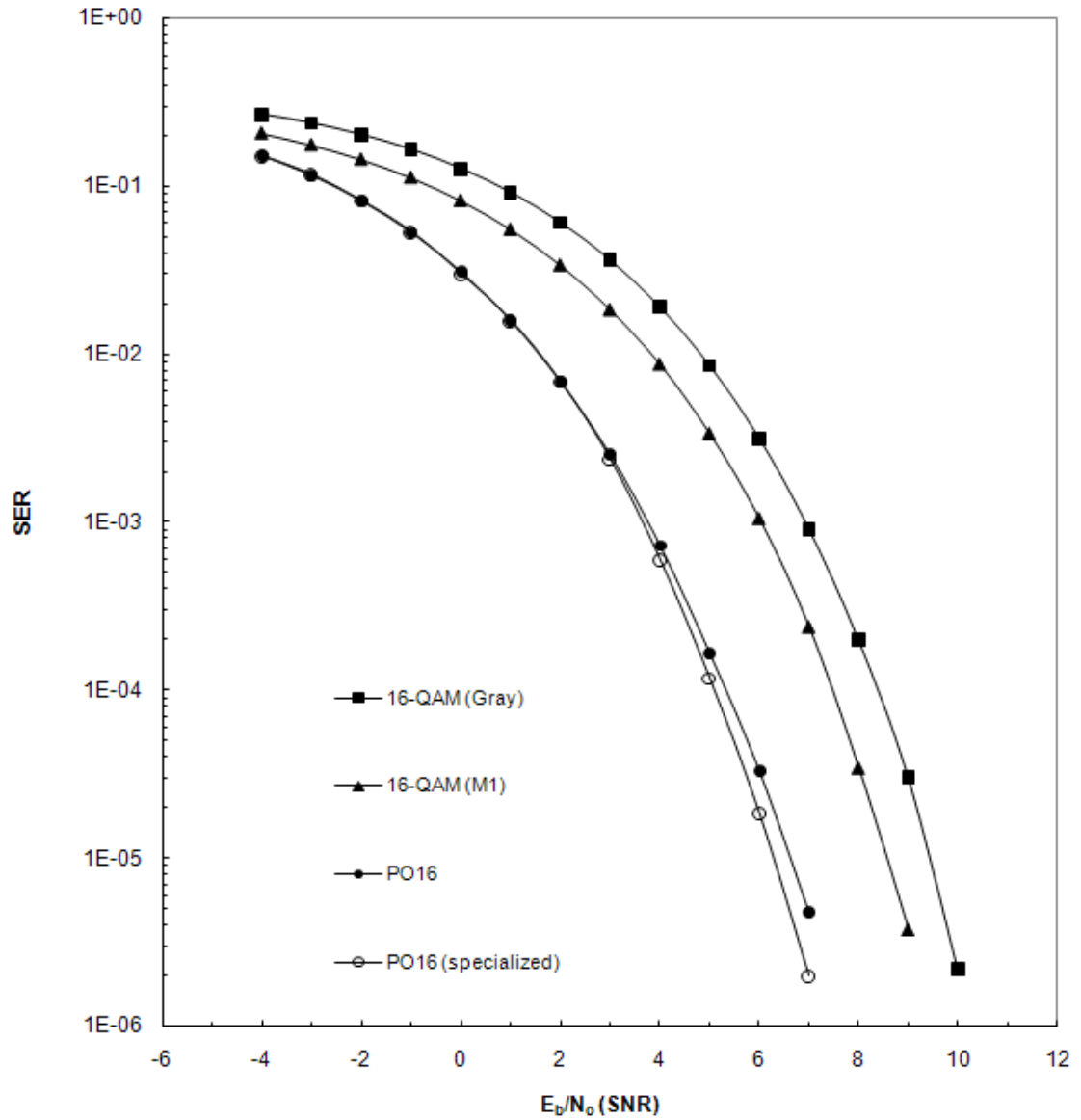


Figure 3.10: Performance of size $M = 16$ constellations for $p = 0.9$ and design $SNR = 1$ dB. Performance of a specialized constellation (i.e., with design SNR identical to true SNR) also shown.

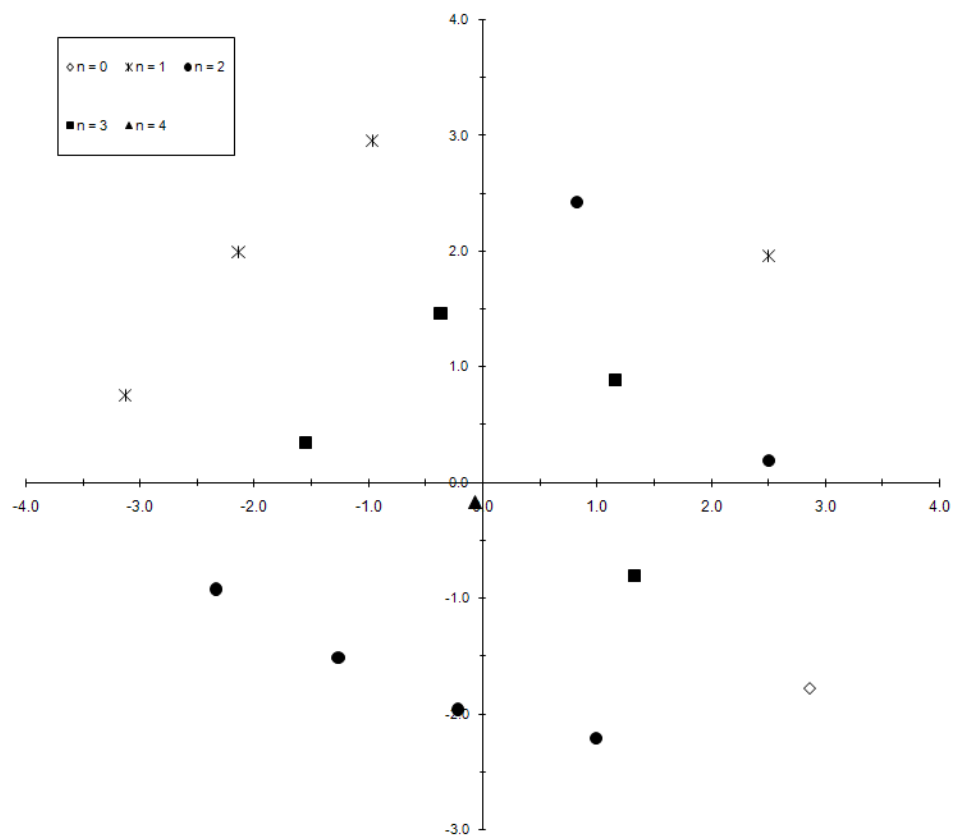


Figure 3.11: Pairwise optimized constellation for $M = 16$, $p = 0.9$ and design $SNR = 0$ dB.

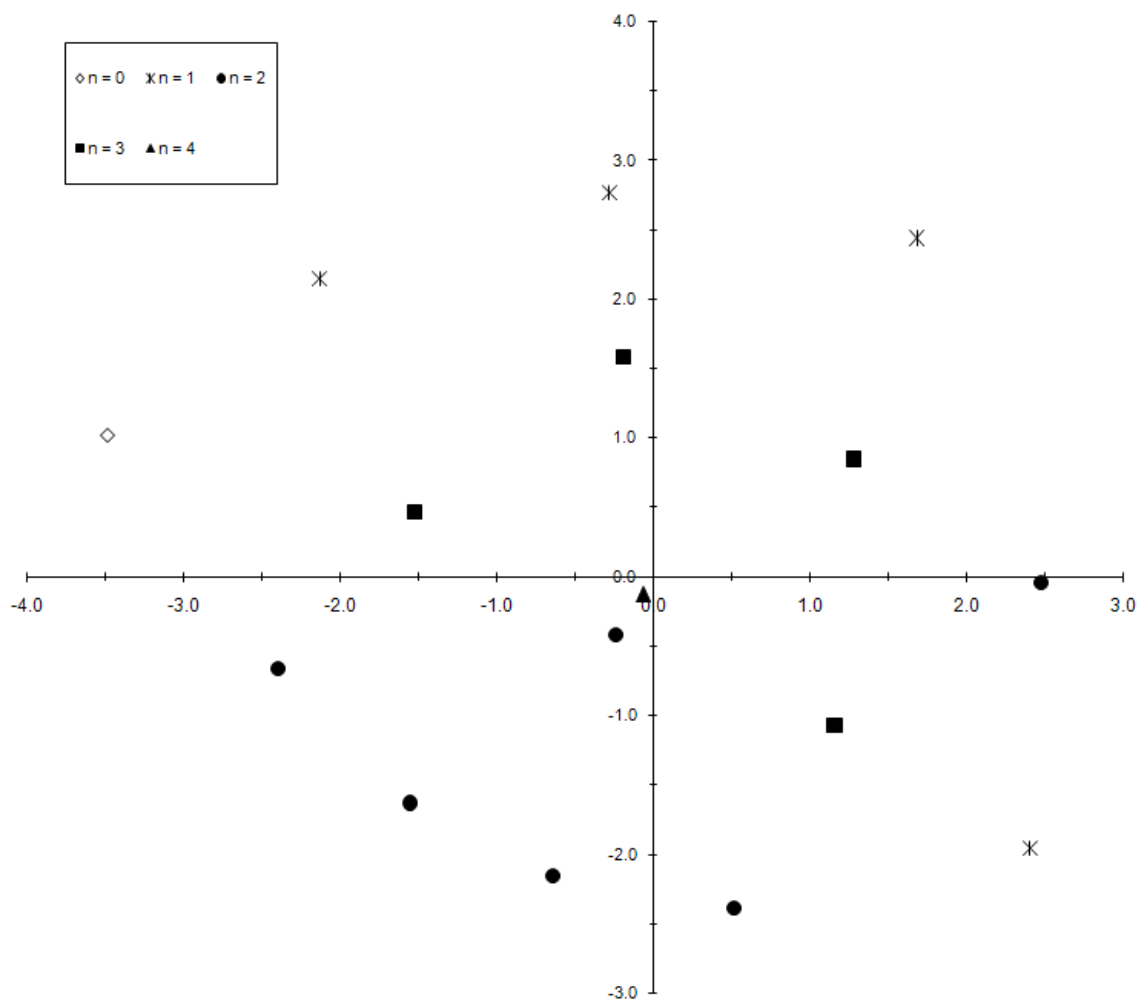


Figure 3.12: Pairwise optimized constellation for $M = 16$, $p = 0.9$ and design $SNR = -3$ dB.

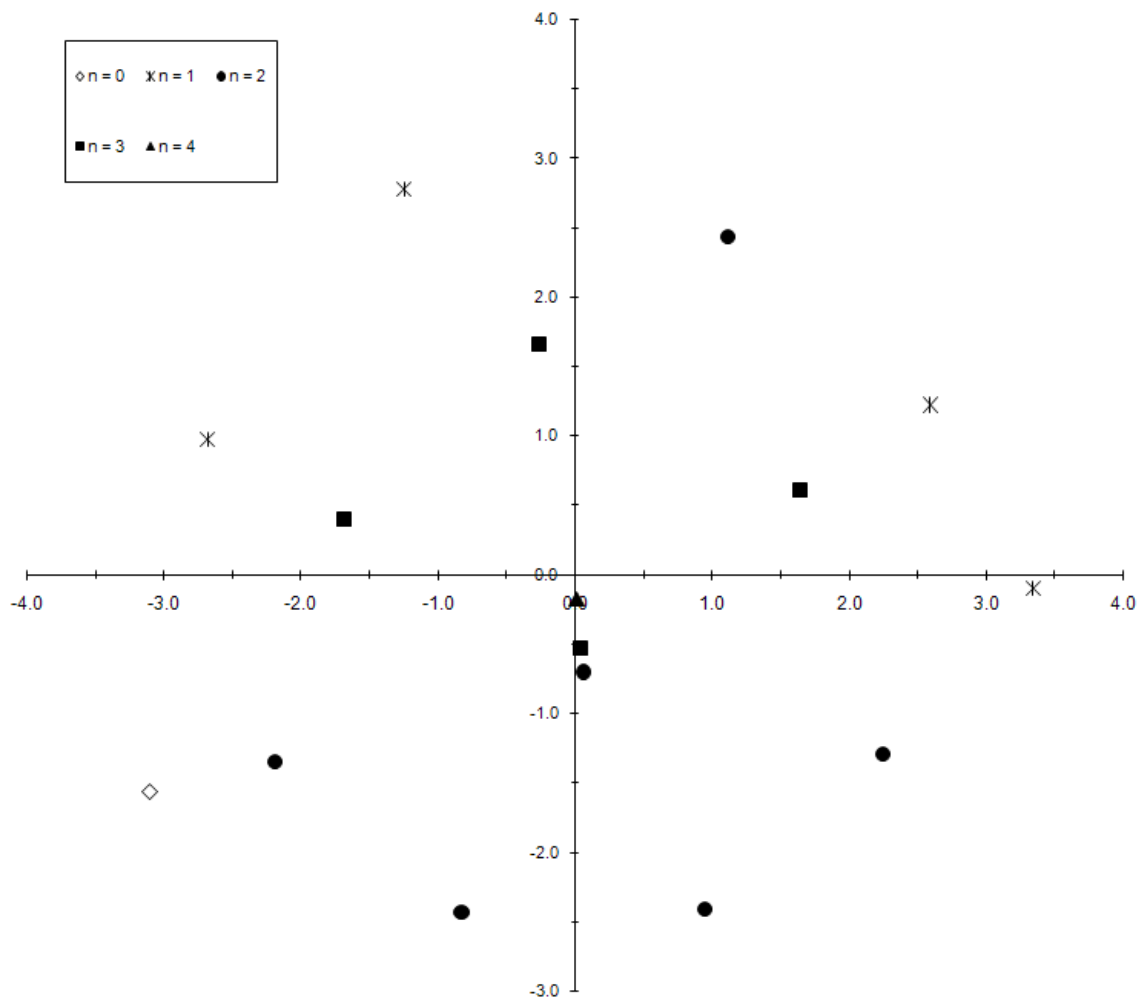


Figure 3.13: Pairwise optimized constellation for $M = 16$, $p = 0.9$ and design $SNR = -5$ dB.

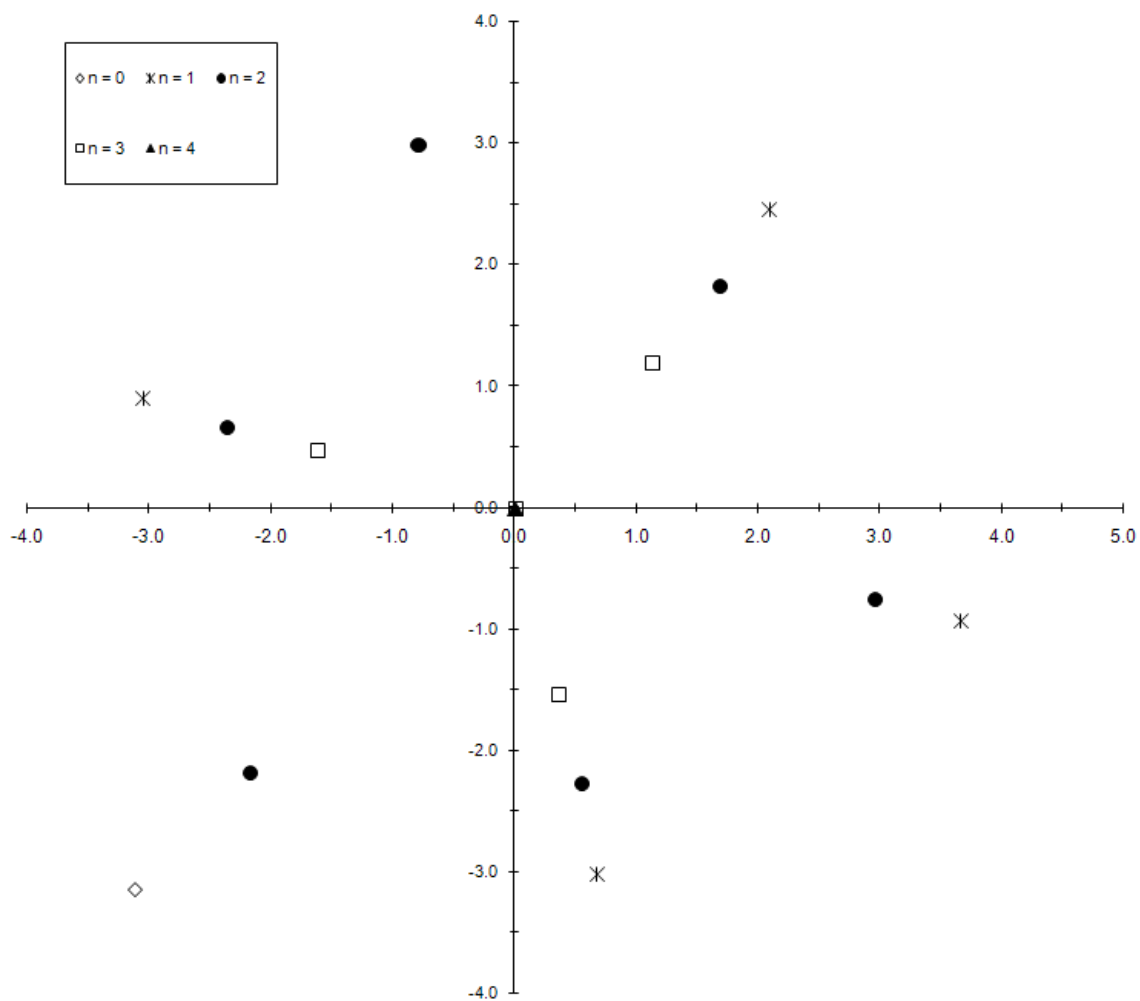


Figure 3.14: Pairwise optimized constellation for $M = 16$, $p = 0.9$ and design $SNR = -10$ dB.

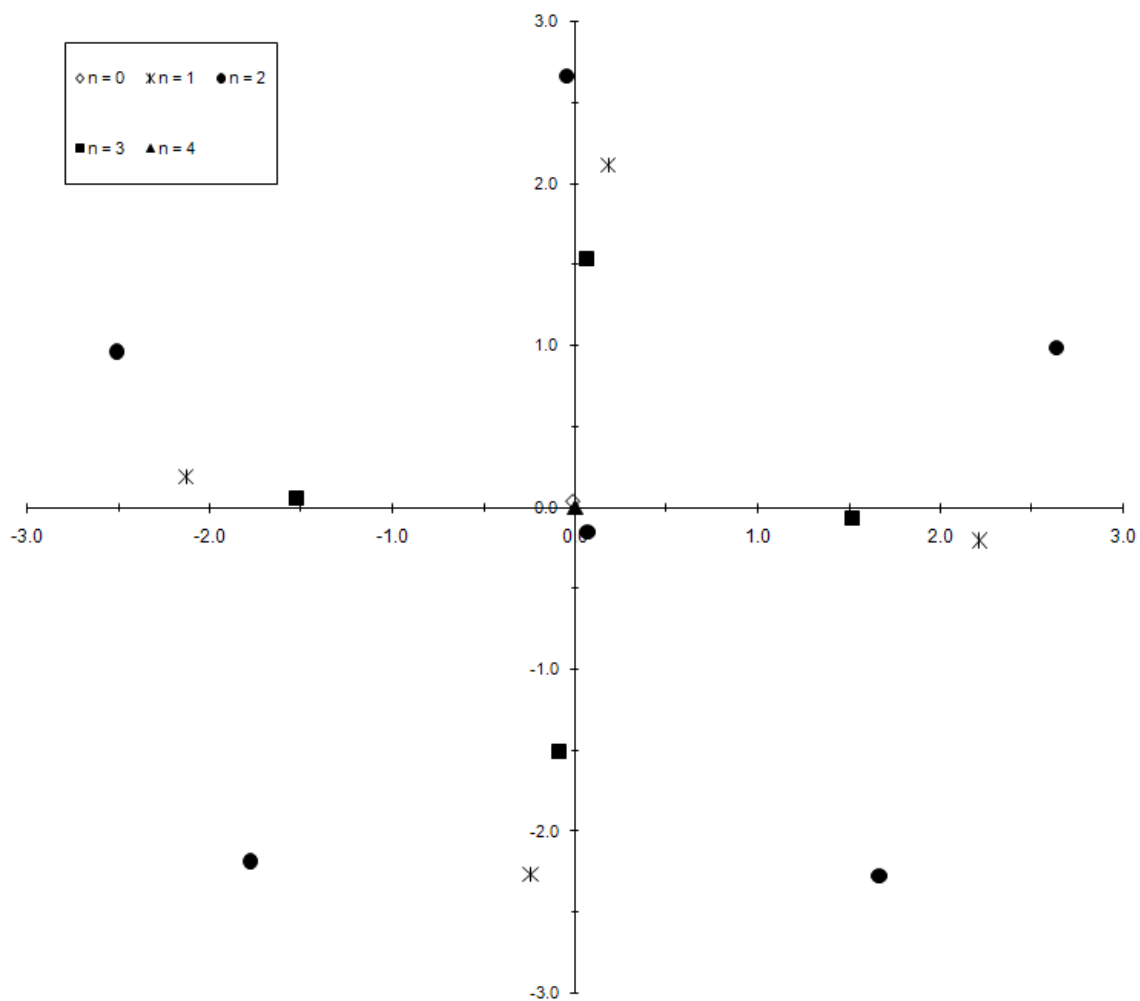


Figure 3.15: Pairwise optimized constellation for $M = 16$, $p = 0.9$ and design $SNR = -20$ dB.

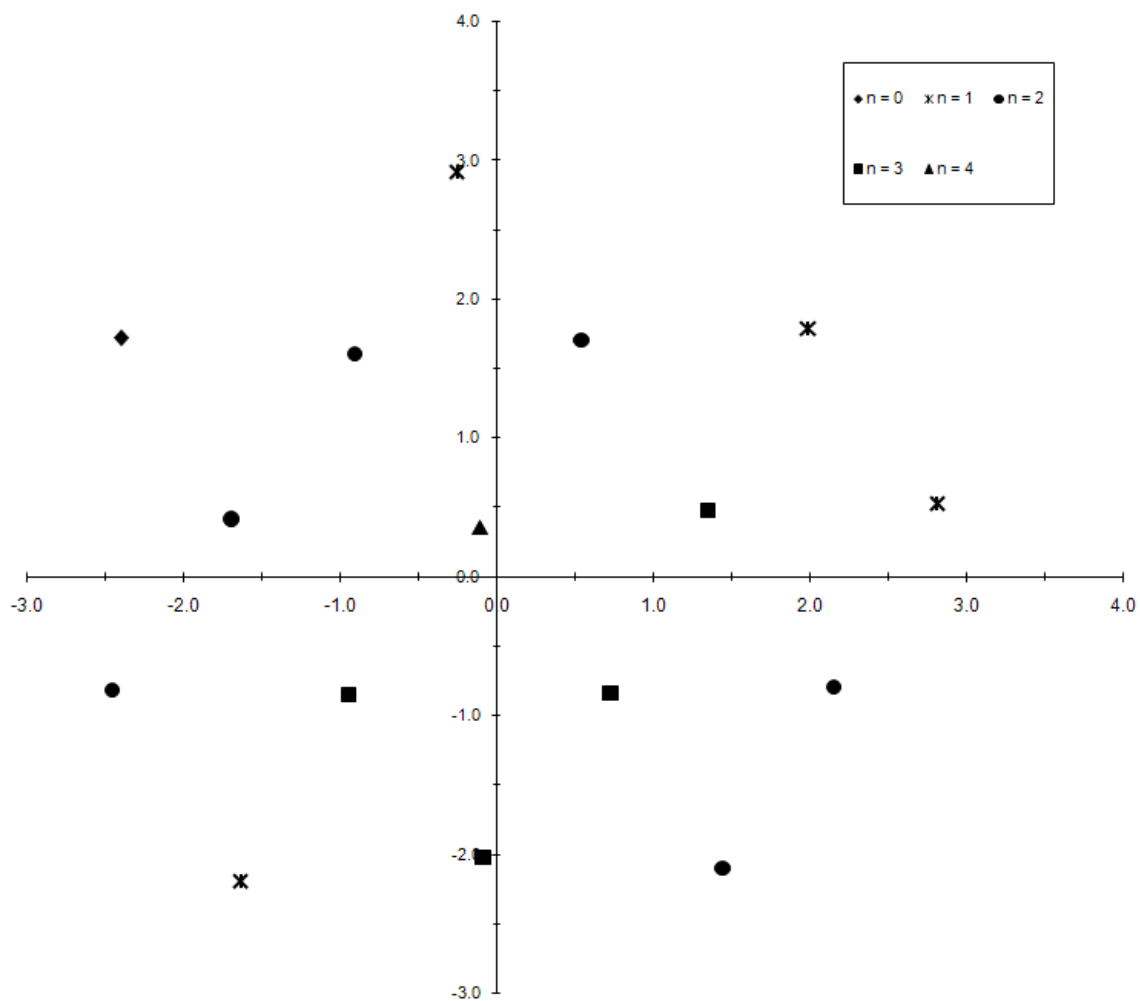


Figure 3.16: Pairwise optimized constellation for $M = 16$, $p = 0.9$ and design $SNR = 10$ dB.

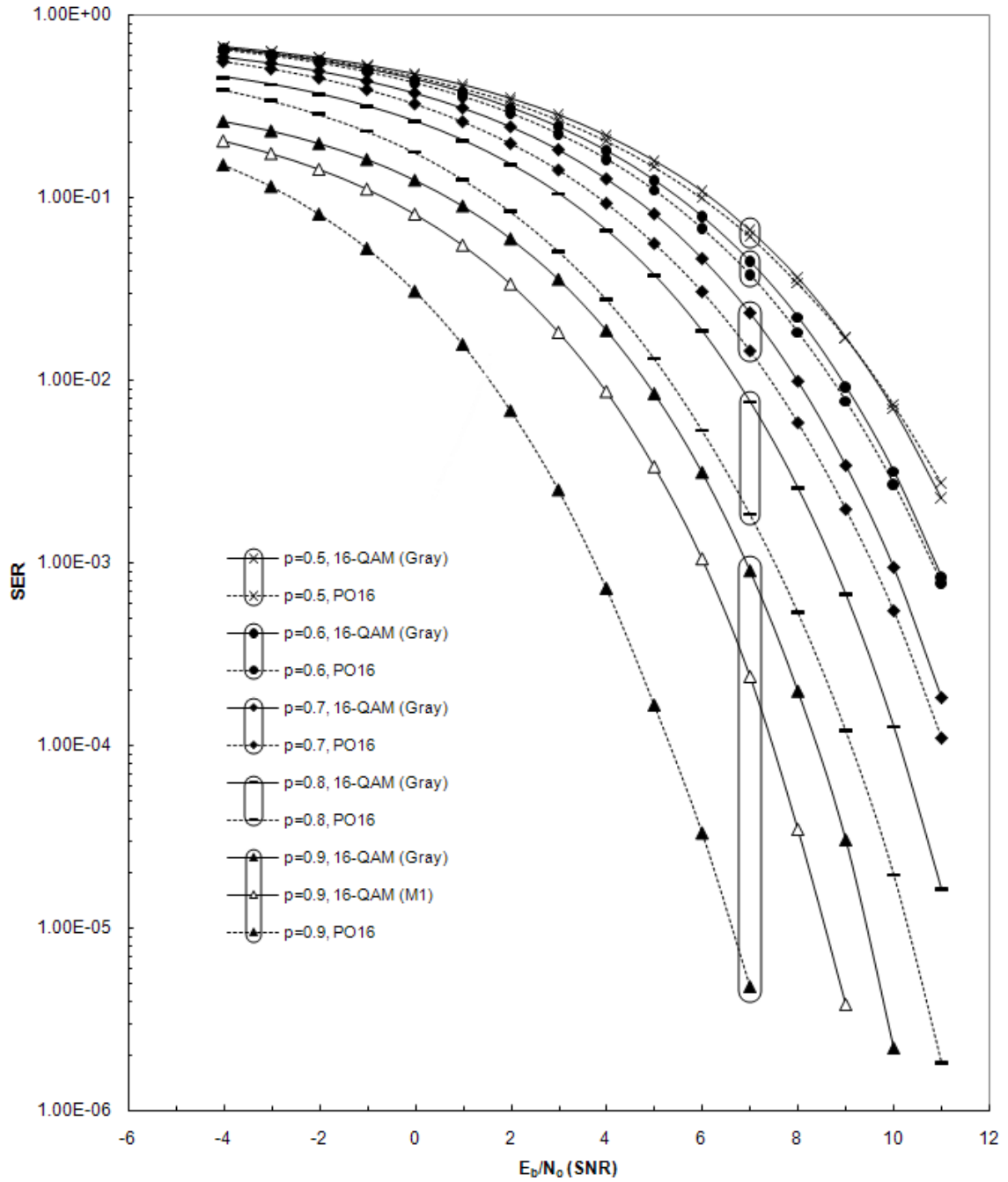


Figure 3.17: Performance of $M = 16$ constellations for varying values of p and design $SNR = 1$ dB.

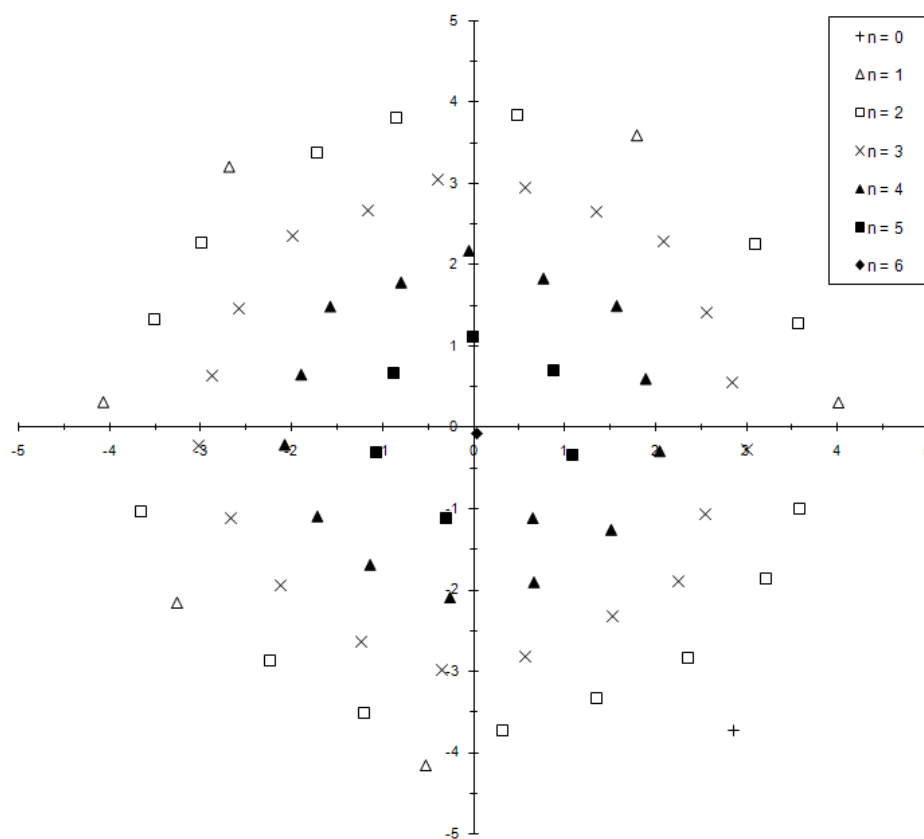


Figure 3.18: Pairwise optimized constellation for $M = 64$, $p = 0.9$ and design $SNR = 2$ dB.

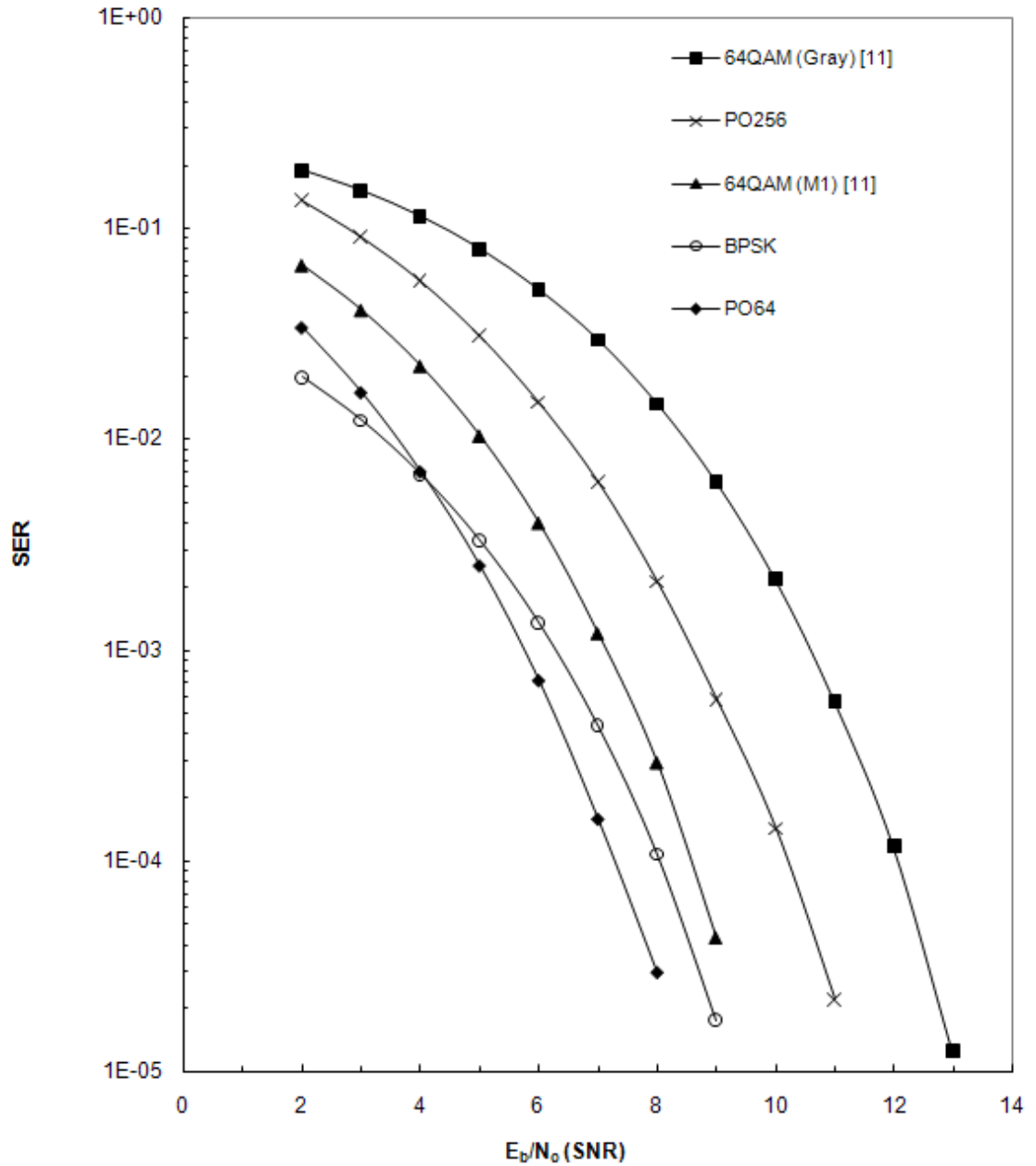


Figure 3.19: Performance of constellations for $M = 64$, $p = 0.9$ and design $SNR = 2 dB$ and the pairwise optimized constellation for $M = 256$ with design $SNR = 4 dB$. BPSK also shown for reference.

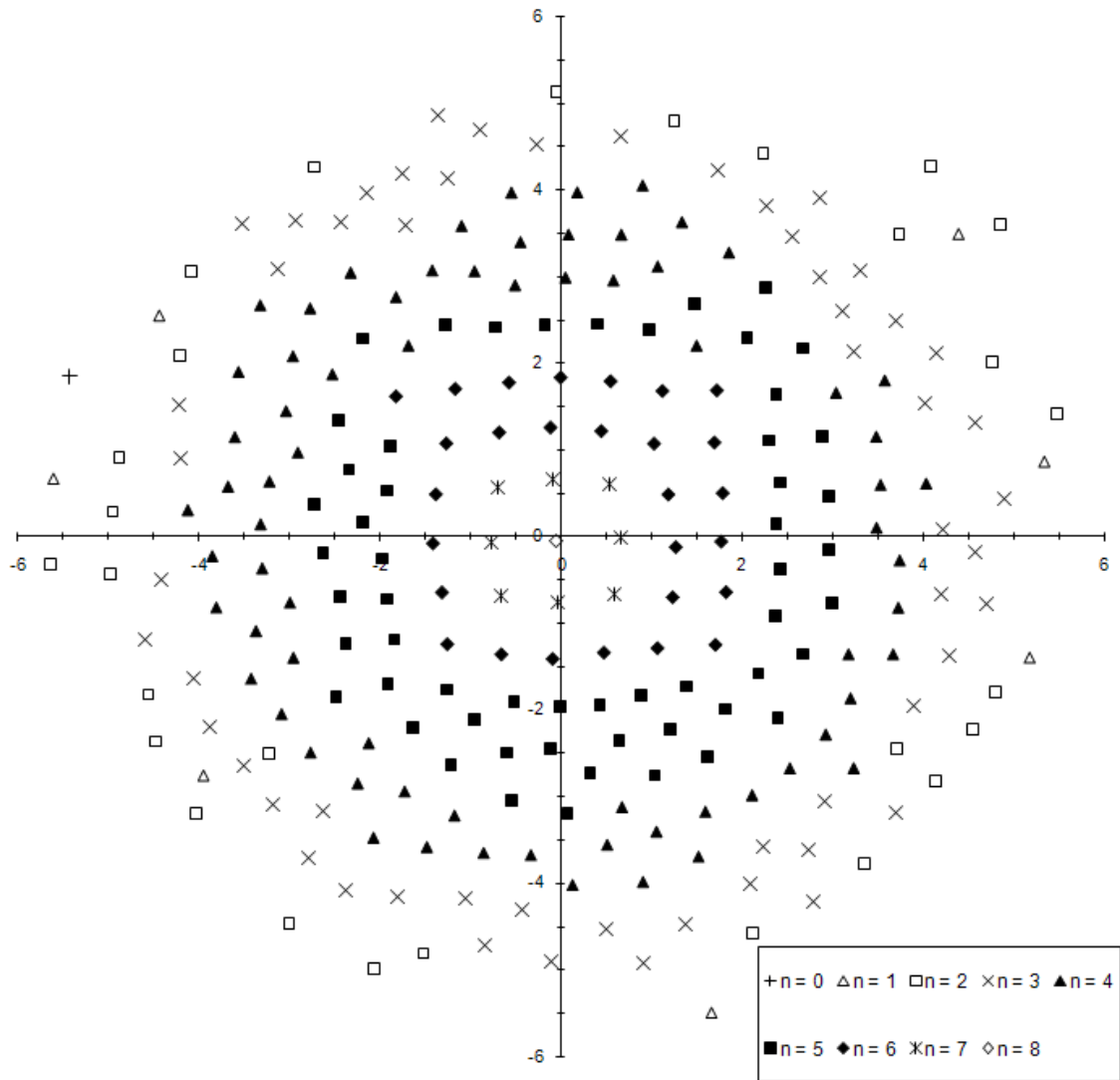


Figure 3.20: Pairwise optimized constellation for $M = 256$, $p = 0.9$ and design $SNR = 4$ dB.

Chapter 4

Designing Maps for the PO Constellations

In the previous chapter, we developed a method for designing improved modulation constellations for non-uniform sources, based on the SER performance of those constellations. But, knowing the probability of each symbol is not enough to allow implementation. In order to be able to use these constellations in any real systems involving binary streams of data, we must have a direct mapping for each possible $\log_2 M$ -bit symbol. In this chapter, we wish to design maps for the PO constellation we have designed in order to assign a specific bit pattern to each point in the constellation.

The terms “point” and “symbol” were used somewhat interchangeably in the previous chapter. Now we must be careful to distinguish them. When we say *point*,

we are referring to a Cartesian coordinate pair which has been selected for transmission by the PO algorithm. *Symbol* now refers to a binary sequence of $\log_2 M$ bits, corresponding to data from the source.

The PO constellations themselves only determine the points where the set of symbols of a given probability should lay, but not the arrangement amongst those equiprobable symbols. When rearranging the individual symbols, we need only consider moving any symbol to another point in the same set. We must now consider exactly how to arrange these points within each set to achieve the best possible BER performance.

4.1 Initialization and Probability Constraint

By the nature of the PO algorithm, the mapping can be initiated and modified under a fairly strong (and helpful) constraint. For a given symbol, we know from its source probability that we can immediately reduce the possible mappings for that symbol to a subset of the points found in the PO constellation – those corresponding to the same source probability.

For each constellation treated, the map is initialized arbitrarily, but such that it conforms to the probability constraint. We define a *layer* as a set of binary symbols taken together with a set of constellation points which are equiprobable. For our binary non-uniform source with source distribution $Pr\{0\} = p$, and an M -ary constellation, the symbols (bit patterns) in layer l will each have l zeros in their bi-

nary sequences. Layer l will also have exactly $\binom{m}{l}$ symbols, each with probability $p^l(1-p)^{m-l}$, where $m = \log_2 M$ is the number of bits in each symbol. During the initialization of the map optimization procedure (described later), the symbols and points within each layer are assigned randomly.

4.2 Objectives

Our guiding objective is to minimize the BER of the constellation and mapping pair. We aim to achieve a BER much lower than the SER by trying to minimize the number of bit errors that occur, even in the presence of noise which causes a symbol error. We can achieve this by minimizing the Hamming distance of each symbol to its neighbours. For a uniform source, using rectangular QAM, this can be achieved using Gray mapping, as seen in Fig. 4.1 [14]. This mapping is arranged such that the Hamming distance of any symbol to any of its neighbours is always one. To exploit the source statistic of the non-uniform source, improved mappings were developed in [14] without modifying the geometry of the underlying constellation. The challenge we face when dealing with the non-uniform source and our PO constellations is that we have neither equiprobable nor equidistant (in terms of the Euclidian distance) neighbours. We still wish to create a “Gray-like” mapping, but it is not so simple as the rectangular QAM case. If not all nearby points are equidistant (Euclidian distance-wise), how do we choose which points we will consider as neighbours? Since, however we select them, those neighbours will not necessarily be equiprobable, we

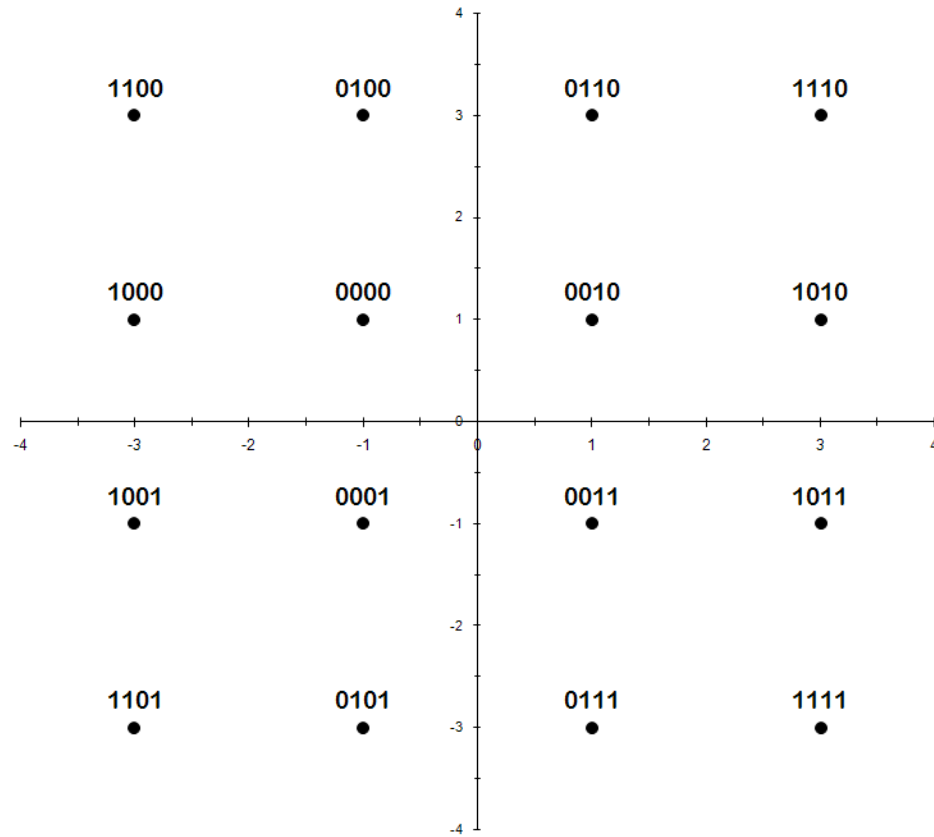


Figure 4.1: Standard 16-QAM modulation constellation with a Gray mapping.

must consider how to measure the Hamming distance to the entire neighbourhood.

4.2.1 Defining the Neighbourhood and Weighted Hamming Score

We must first decide and define what symbols we will consider as neighbours. We initially considered setting the neighbourhood of a point \vec{s}_u to be all points which lay inside a circle of radius r (*i.e.*, all points \vec{s}_i such that $\|\vec{s}_u - \vec{s}_i\| \leq r$). But how do we choose an appropriate r ? Should it change depending on the probability of the

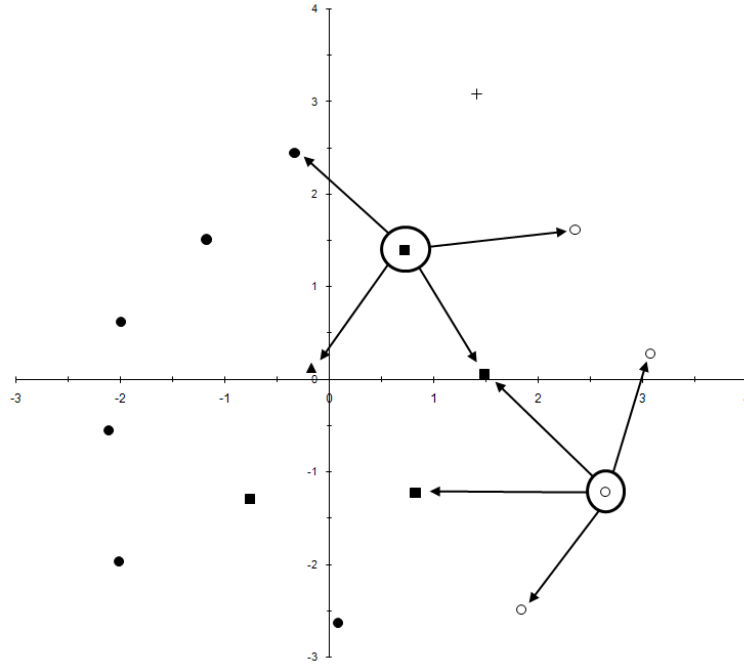


Figure 4.2: Two example neighbourhoods are shown for PO16 ($M = 16$ and $k = 4$).

symbol being considered? Without a way to determine what r should be, we moved on to the idea of selecting some k points which lay closest (in terms of Euclidian distance) to \vec{s}_u (i.e., the k nearest neighbours). This method provided a natural solution to the flexible radius problem, as we would always include the closest k points, despite those points laying farther away for the less likely symbols towards the outside of the constellation. Looking at the PO constellations we had obtained so far, we selected $k = \sqrt{M}$ as the neighbourhood size. This value provided a good balance between limiting the neighbourhood to those points which would most likely be decoded in error, but which also would include enough points to ensure we indeed had an appropriate neighbourhood “around” the symbol in question. For example,

two neighbourhoods of different points in the PO16 constellation can be seen in Fig. 4.2.

Now we must define the *Weighted Hamming Score* we will use when considering the suitability of a given symbol for its neighbourhood. To do this, we disregard the Euclidian distance between the symbol and each of its neighbours, and instead consider the Hamming distance and probability of each other point. For symbol \vec{s}_u and its k nearest neighbours (indexed by $\{n_1, \dots, n_k\}$), the Weighted Hamming Score, $WHS(\cdot)$, is then defined as:

$$WHS(\vec{s}_u) = \sum_{i=1}^k p_{n_i} d(b_{n_i}, b_u) \quad (4.1)$$

where p_{n_i} is the probability of \vec{s}_{n_i} , $d(\cdot, \cdot)$ is Hamming distance and b_{n_i} is the binary sequence (symbol) currently assigned to \vec{s}_{n_i} .

4.3 Map Improvement Algorithm

The Map Improvement algorithm is implemented as follows:

1. Configure some initial mapping (as described above).
2. Loop through set of all layers several times.
 - 2.1. Loop through each layer individually, proceeding outwards.
 - 2.1.1. For current layer, generate many pairs of symbols.
 - 2.1.1.1 For each pair of symbols, determine both neighbourhoods.

2.1.1.2 Compare total Weighted Hamming Score with and without switching the symbol assignment, using (4.2).

2.1.1.3 If switching the symbol assignment decreases WHS, then switch the mapping of binary sequences to selected points.

As described above, the initial mapping of Step 1 is arbitrary aside from conforming to the probability requirements. For Steps 2 through to 2.1.1, we adjusted the number of times to repeat each loop to achieve what appeared to be the best results. The values used in our code were as follows:

We looped through the set of all layers $4m$ times for Step 2, where $m = \log_2 M$ is the number of bits in each sequence (and is also the number of layers). When stepping through each individual layer in Step 2.1, we start with the center and proceed outwards. We only “loop” each layer once per overall loop, since repeating a single layer immediately would be equivalent to simply generating more pairs. Finally, for Step 2.1.1, we achieved favourable results when considering $2L^2$ pairs, where $L = \binom{m}{l}$ is the number of symbols in layer l (members of which have l zeros in their binary sequences).

When checking each pair of symbols in Step 2.1.1.2 to see if they should be switched, we calculate the sum of the WHD for each symbol in both neighbourhoods $((n_{u,1}, \dots, n_{u,k})$ is the neighbourhood of \vec{s}_u , and $(n_{v,1}, \dots, n_{v,k})$ is the neighbourhood of

\vec{s}_v). We want to know if

$$\sum_{i=1}^k (p_{n_{u,i}} d(b_{n_{u,i}}, b_u) + p_{n_{v,i}} d(b_{n_{v,i}}, b_v)) > \sum_{i=1}^k (p_{n_{u,i}} d(b_{n_{u,i}}, b_v) + p_{n_{v,i}} d(b_{n_{v,i}}, b_u)) \quad (4.2)$$

and, if so, we switch the mappings b_u and b_v for \vec{s}_u and \vec{s}_v .

4.4 Results and Performance

The results of this procedure are now considered. For small constellation sizes, the mapping is not particularly important. For $M = 2$, there is no map to be considered at all – the symbols are exactly determined by the constellation. For $M = 4$, the only consideration is the placement of the mapping 01 versus 10. Indeed these are different, but the resulting BER is identical, since each configuration has identical Hamming distance to its neighbours. As such, we immediately move to considering 16-ary constellations and larger.

4.4.1 16-ary Constellations

We present the result of the mapping improvement algorithm in Fig. 4.3 for the PO constellation we developed in the previous chapter. We can see immediately that the Hamming distance between many close neighbours is 2, where the Gray had put all distances to just 1. By the design of the PO constellations, we cannot achieve distances as small as those of the Gray map. This is because we force symbols of equal probability to be near one another. For instance, consider the layer $l = 3$ in Fig. 4.3.

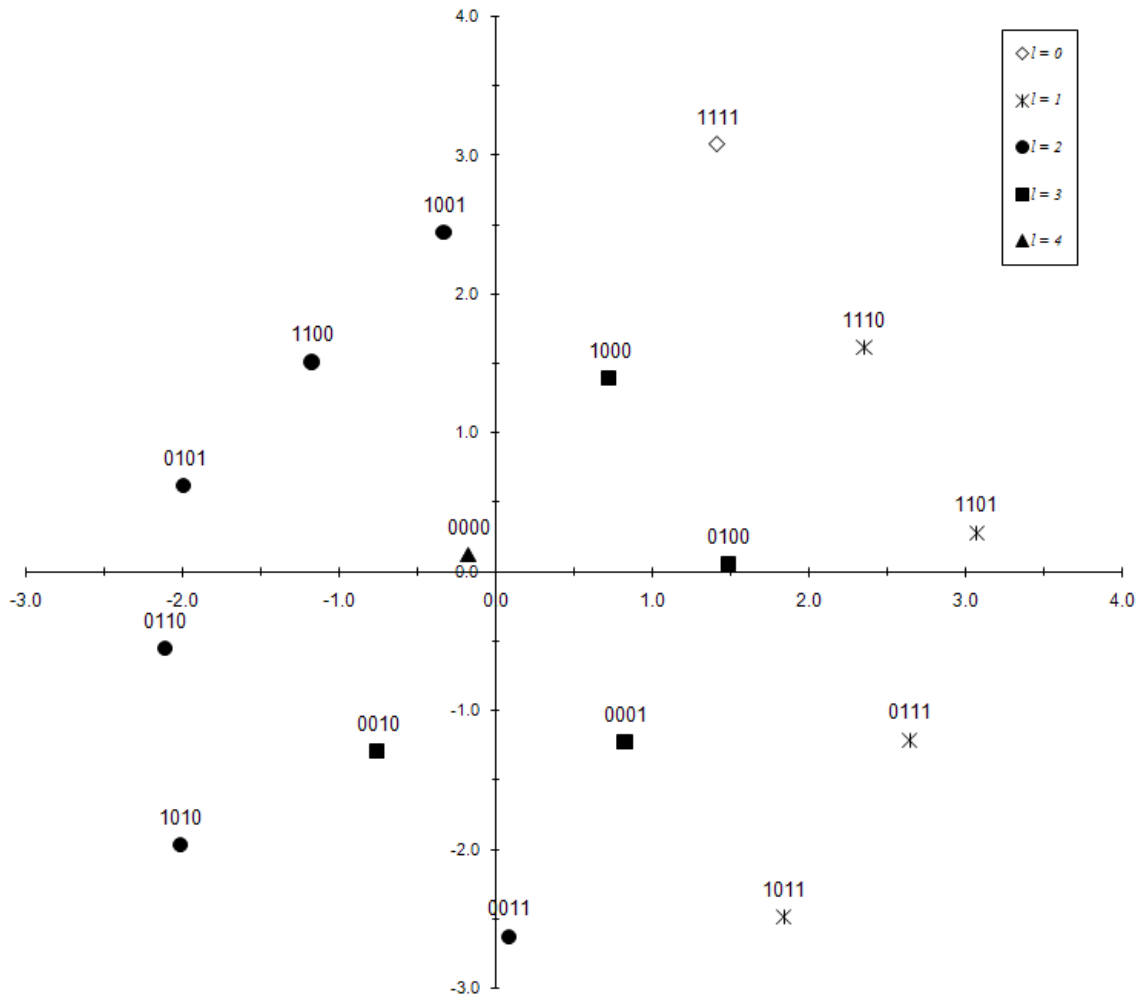


Figure 4.3: PO constellation for $M = 16$ with improved mapping.

All symbols in this layer have exactly three zeros in their binary sequences (there are four of these symbols). Since they must be a neighbour to at least one other point in their own layer, and we do not have any repeated symbols, the Hamming distance of these close neighbours must be 2. However, you will also find that this mapping also has a *maximum* Hamming distance of 2, aside from the distance involving the symbol 1111 to layer $l = 3$ (which cannot be avoided, again by the design of the PO

constellation). Despite having greater Hamming distance between some neighbours, the PO constellation with the associated map performs very well.

Looking at Fig. 4.4, the performance improvement on the PO constellation with its designed map is clear. There are considerable gains over the standard rectangular QAM constellation with both maps considered. While the optimized $M1$ maps of [14] achieve approximately 1 dB gain of the Gray map, the PO constellation and map makes a further improvement of more than 2 dB over the $M1$, where the difference is greatest. This best improvement occurs at mid-range SNRs of 2 to 5 dB , and is approximately in line with the gains we saw when considering SER performance. It is also interesting to note that the SER performance of PO16 is in fact superior to the BER performance of both rectangular 16-QAM maps for mid-high SNRs, so even without the map improvement procedure, the BER of PO16 would be considerably better than the standard constellations.

We can also see that the performance difference between the BER and SER of PO16 shrinks as SNR increase to high levels. This is to be expected, as when the noise is very small compared to the signal, we will make very few errors on the most likely symbols, and this is where the reducing bit errors has the greatest impact.

4.4.2 64-ary and 256-ary Constellations

We see the resulting maps for the PO64 and PO256 constellations in Figures 4.5 and 4.6, respectively. Looking first at Fig. 4.5, again we see Hamming distances

of close neighbours larger than those seen in a Gray map (as we saw for the PO16 constellation). But we also see again that most symbols (especially the group of more likely symbols near the center) are assigned so that they are quite similar to their neighbours. Many of the most likely symbols have a Hamming distance of only 1 or 2 to their neighbours, but we also have the Hamming distance between *any* neighbours being at most 3. For the 256-ary PO constellation and mapping in Fig. 4.6, we see the same situation in general. There are some neighbours with greater Hamming distances, but most are kept quite low (2 or 3), especially when compared to the longer sequence length (8 bits) of symbols in this constellation.

The BER performance of the 64- and 256-ary PO constellations is shown in Fig. 4.7. We can see that the 64-QAM M1-map of [14] provides gains of approximately $3dB$ over Gray mapped 64-QAM, and even $0.25 dB$ over BPSK. The PO64 constellation with the improved map further provides more than $1 dB$ over the BER performance of the M1-mapped QAM. Again, we find that this is approximately in line with the gains seen when examining SER performance. Interestingly, we can see that the PO256 constellation and map has better BER performance than Gray mapped 64-QAM, despite the constellation density and the larger Hamming distances seen in PO256. As was the case with SER performance, PO256 also simultaneously achieves superior BER performance and higher data throughput.

As a final note, we present in Fig. 4.8 all the BER performance curves of the PO constellations and maps developed here. It is interesting to note that PO4 outper-

forms PO2 for all tested SNRs. This is important because it has achieved better BER performance while simultaneously doubling the data rate. We cannot justify this phenomenon definitively, but have two plausible explanations. First, when moving from $M = 2$ to $M = 4$, there is “plenty of space” available on the plane, so the additional points do not suffer much from the “crowding” that larger constellations experience. Second, PO4 cuts its SER in half when considering BER, since any probable decode error would have only one of two bits wrong (11 would be erroneously decoded to 00 very rarely, and *vice-versa*), while PO2 sees no gain by considering BER. Not only is PO4 superior to PO2 at all noise levels, but in fact PO8 (which we did not look at closely, but is shown in Fig. 4.9) also has superior BER performance to PO2 for sufficiently high SNR (above 2 dB), and has triple the data throughput of PO2.

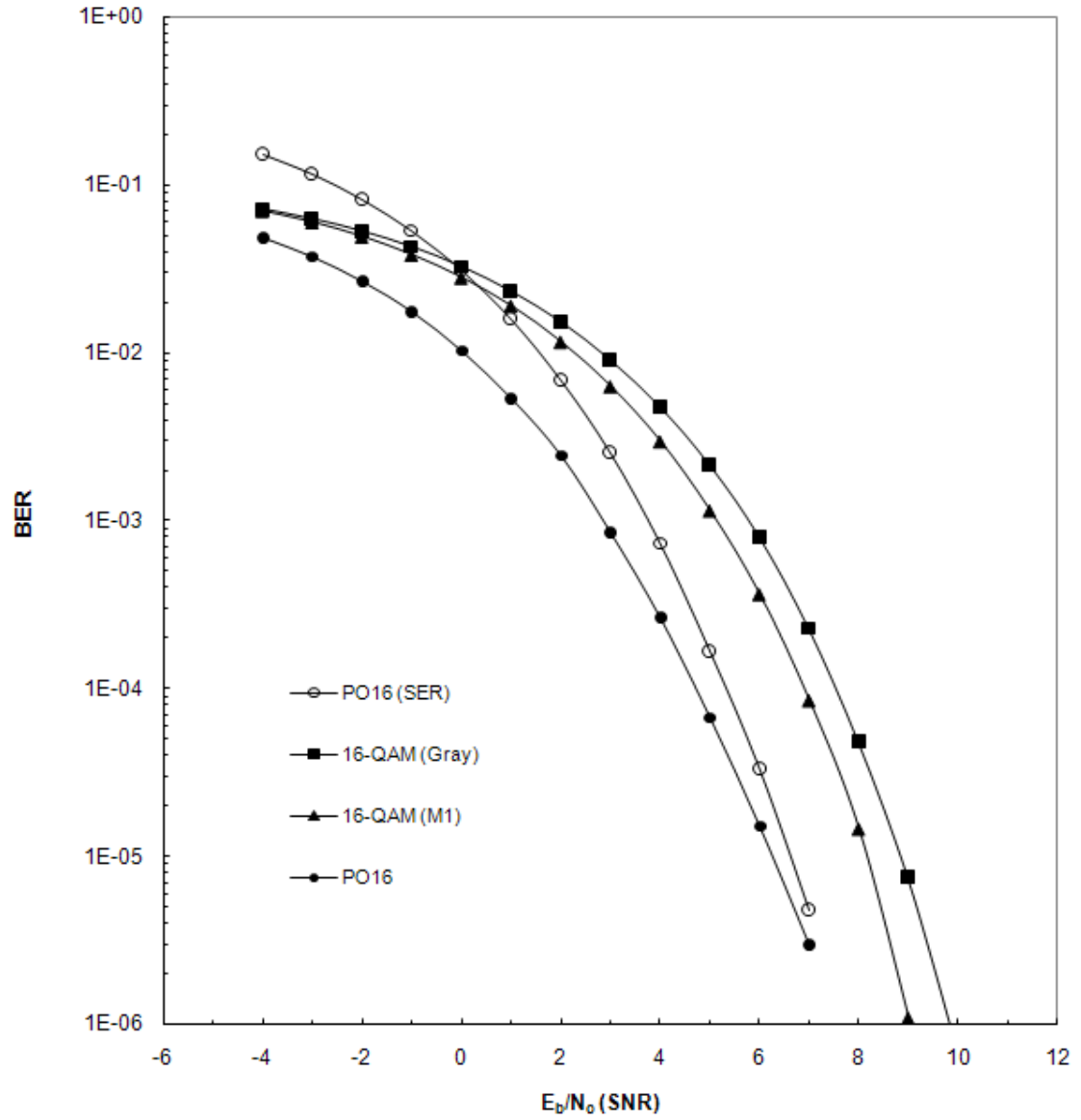


Figure 4.4: BER Performance of 16-ary constellations. PO constellation simulated with mapping seen in Fig. 4.3.

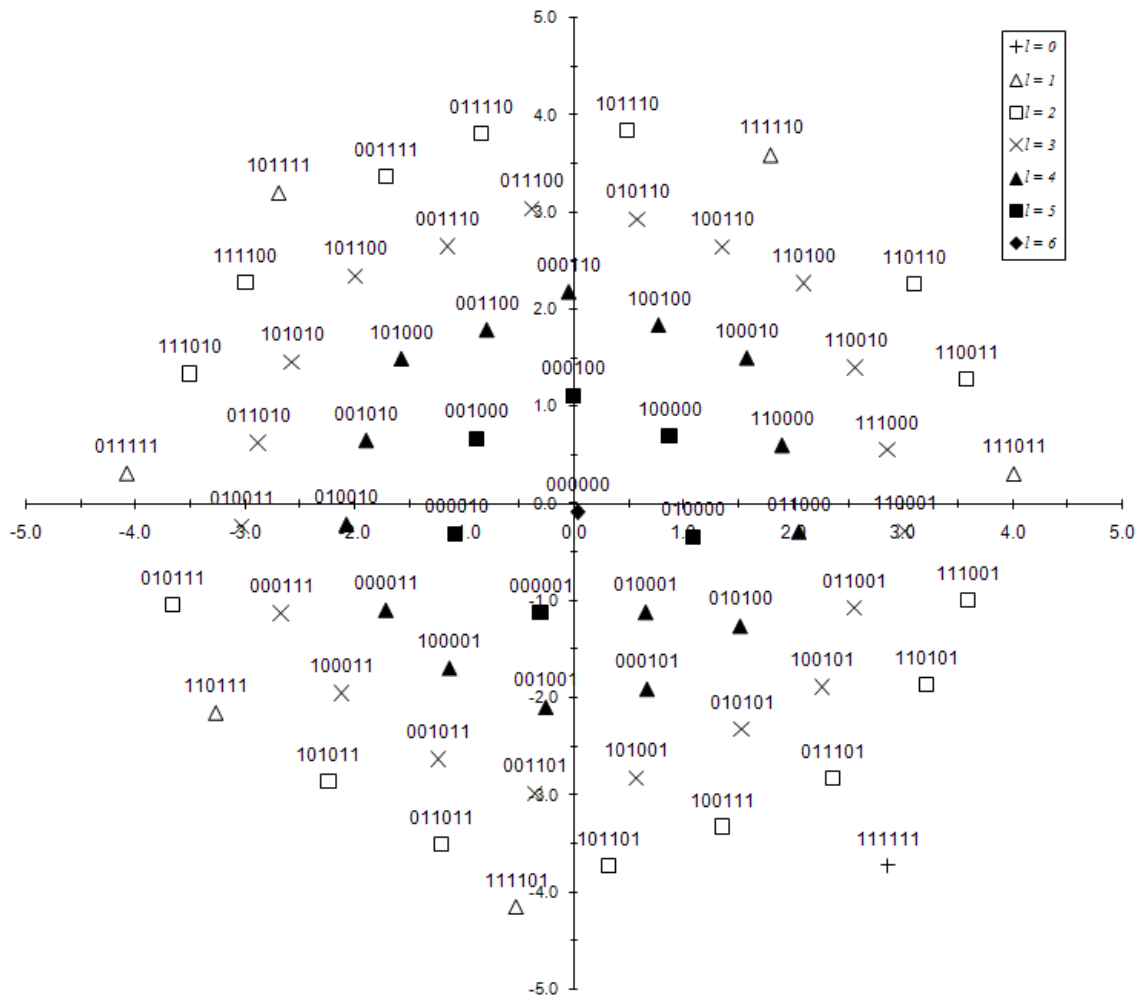


Figure 4.5: PO constellation for $M = 64$ with improved mapping.

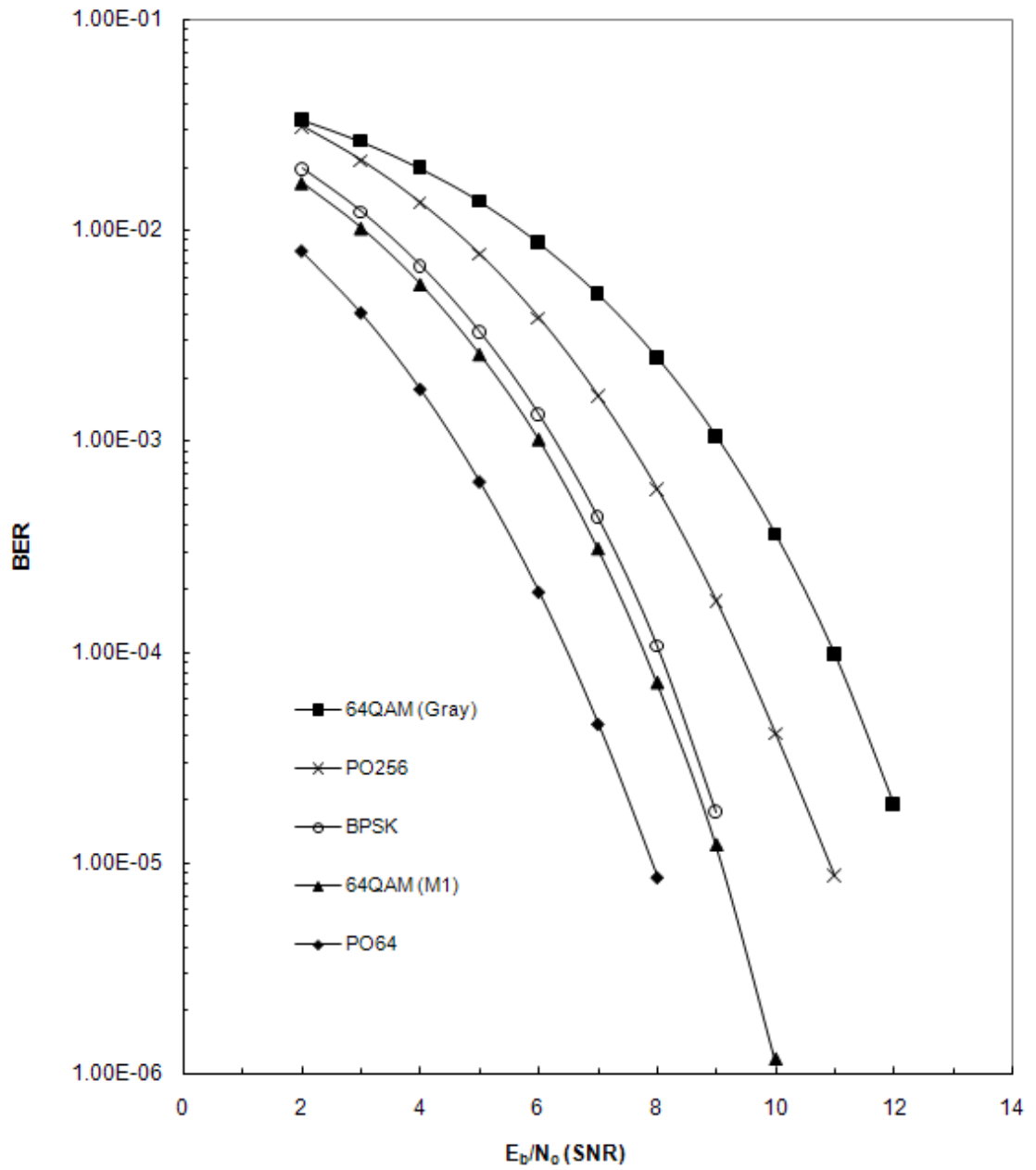
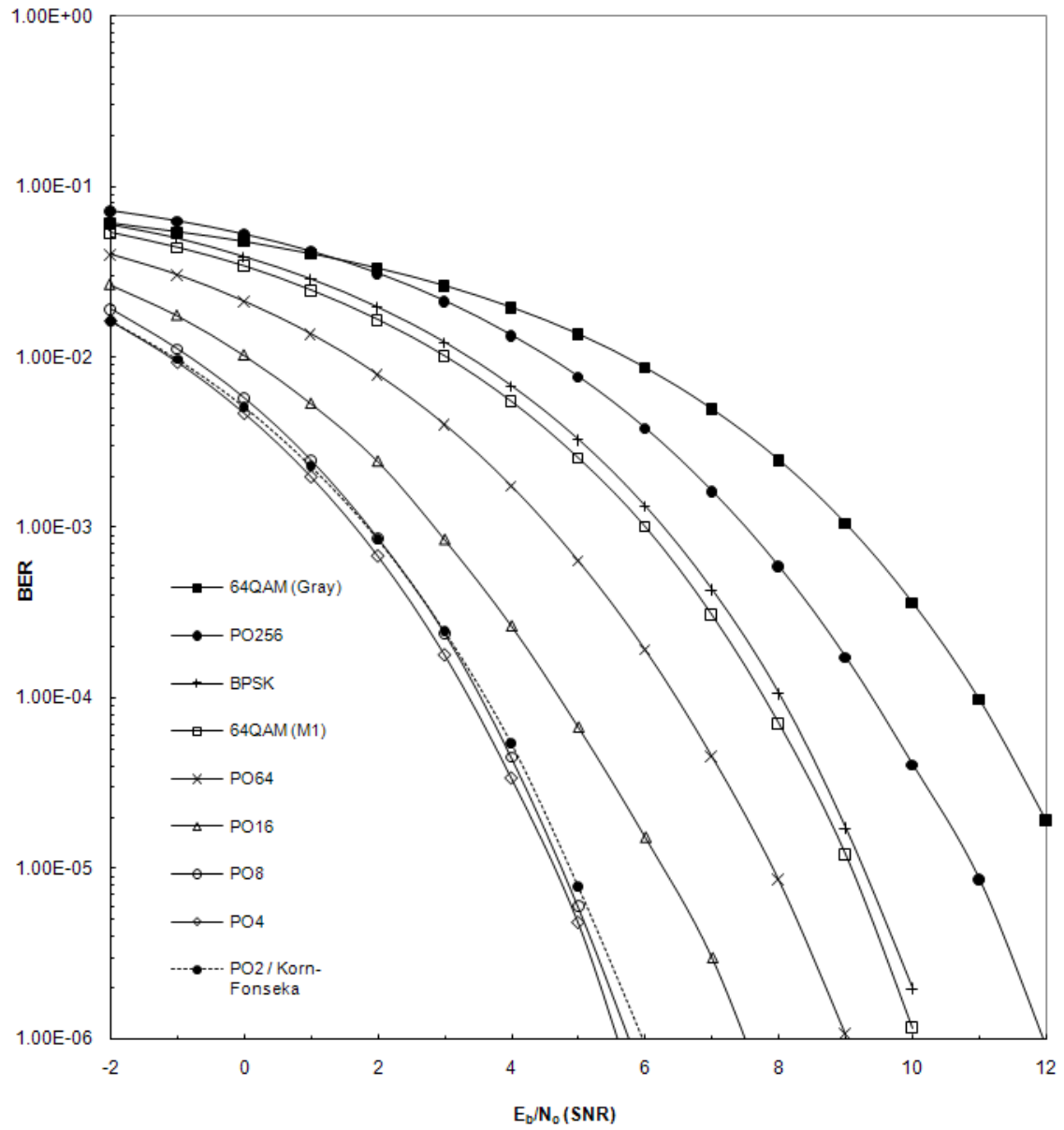


Figure 4.7: BER Performance of 64-ary constellations (and PO256). PO constellations simulated with mappings seen in Fig. 4.5 and 4.6.

Figure 4.8: BER Performance of all M -ary PO constellations presented.

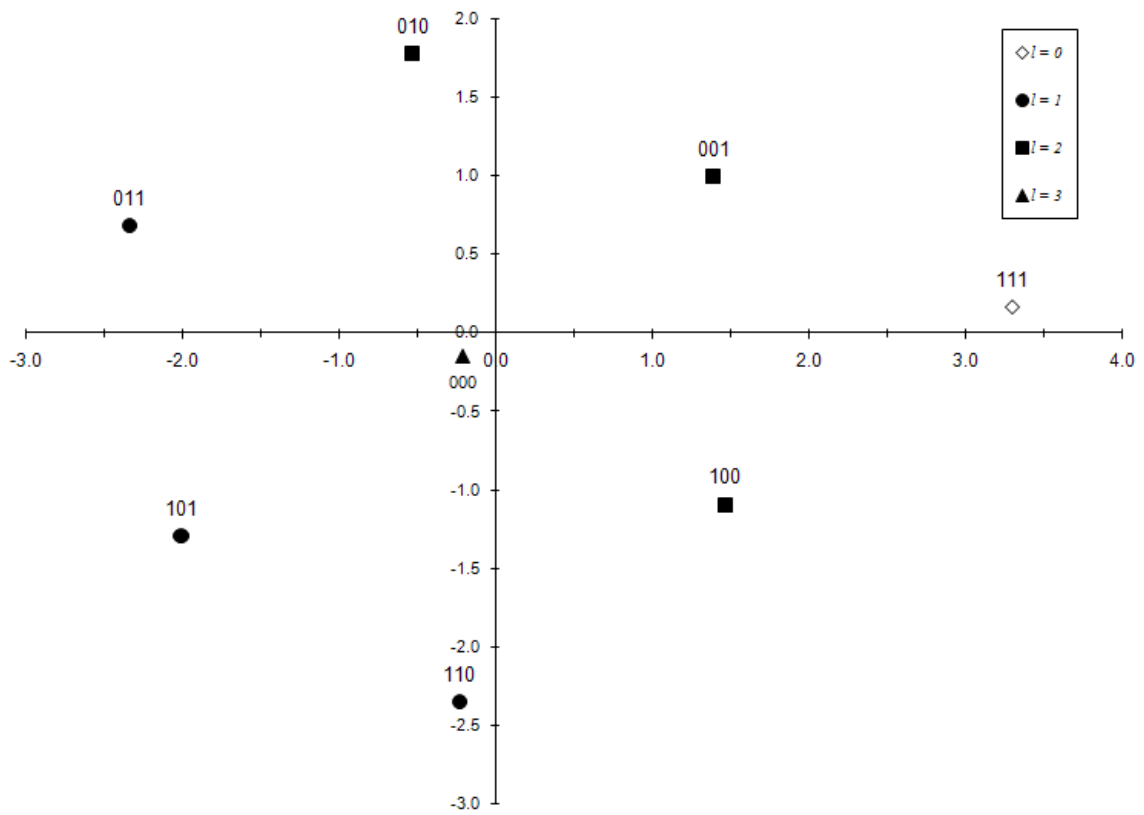


Figure 4.9: PO constellation for $M = 8$ (designed for $p = 0.9$ and $SNR = 0$ dB) with symbol mapping.

Chapter 5

Comparison to Source and Channel Coding

Thus far we have only compared our PO system to similar transmission schemes: mainly BPSK and rectangular QAM. We have been considering only the case of transmitting directly the modulated source symbols, without trying to compress the source, or protect the message with parity.

We will now consider another possible transmission scheme: using in tandem both source and channel coding on the data (separate source and channel coding) to first compress the message, and then protect it during transmission. Using this tandem scheme has a trade-off relative to our uncoded system, and different performance profiles. We will explore these differences in this chapter.

5.1 System for Comparison

We begin by describing the tandem coding scheme to which we will compare the PO system. The goal is to compare to a system that is reasonably representative of real world systems employing such tandem coding schemes.

As in all of the test cases, we generate bits according to the source distribution. For each of T trials, we generate messages in blocks of N bits at a time to be processed. The message will first be compressed (losslessly) using a fourth-order Huffman code. This means we will look at four bit symbols from our non-uniform binary source, and design a Huffman code for them. For $p = 0.9$, the resulting Huffman code is as seen in Table 5.1. Using this Huffman code results in an average code rate of 0.49255. This code rate is close to the entropy of our binary source, which is 0.46, so we know this code is appropriate. Given this code rate close to 0.5, the compressed message will be approximately $\frac{N}{2}$ bits long.

The channel code we will be using is a convolutional code with constraint length $k = 3$ and rate $r = 1/2$. The generator functions used for the convolutional code are $G_0 = 101$ and $G_1 = 111$, in binary representation form [11, pp 470-477]. This leads the output to follow the state machine described in Fig. 5.1. The states represent the two previous input bits, $X_{n-1}X_{n-2}$, and the transition labels indicate the current input bit and the two channel coded parity bits which will be transmitted, X_n/g_0g_1 . To allow us to know how the transmission begins, we always reset the initial state to 00 at the beginning of each message block, and record the transitions from there

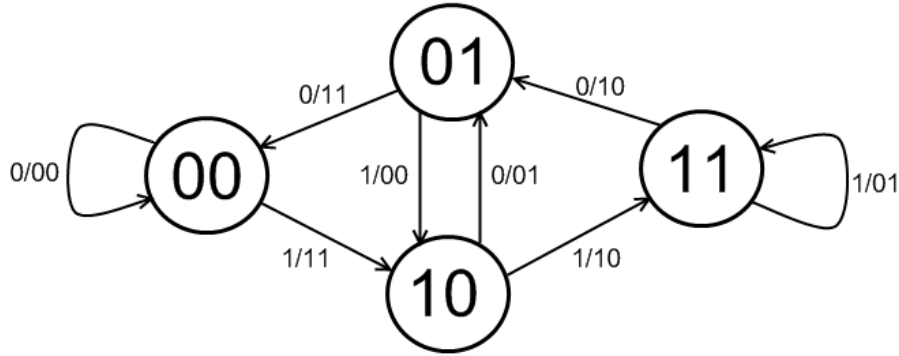


Figure 5.1: State machine representing the convolutional channel code with constraint length $k = 3$ and rate $r = 1/2$.

forward. So for each compressed message bit to be sent, two coded parity bits are sent representing the state change. The channel coded bits are transmitted using BPSK with AWGN, and the receiver collects the observed voltages in the channel.

Once the entire block (approximately N bits) has been received, the observed voltages are passed to a soft Viterbi decoder to recover the transmitted message. The output of the Viterbi algorithm is a best guess of the bit sequence (by minimizing the sequence error probability) which comprises the encoded message (approximately $\frac{N}{2}$ bits). This binary sequence is then passed to the Huffman decoder to be converted back into the original message (N bits long). Note that the overall rate of this tandem coding system is 1 source bit per channel use.

5.2 Performance Comparison

We will now examine the performance of this tandem coding scheme in comparison to our uncoded PO system. To test the tandem coded system, messages of N source bits were generated and passed through the system described in Section 5.1. Tests were conducted for a range of values of N , as the performance of the Viterbi decoder does indeed depend heavily on the length of the blocks received. Other values were tested, but the values we have selected for interesting performance comparison to the PO systems are $N = 12, 100, 200, 800, 5000$. For smaller values of N , more trials (T) were performed to simulate approximately ten to twenty million bits¹ total for each SNR.

Looking at the data presented in Fig. 5.2, we first notice that performance is quite poor at low SNR for all block sizes, but is slightly worse for larger N . For small block length ($N = 12$) we see that the tandem scheme does not outperform PO2 and PO4 even at relatively high SNR, and in fact only beats out PO16 at a mid-high SNR of 5 dB. Keep in mind that at that level of noise, PO16 is communicating four times as much data as the tandem scheme. Stepping the block size up to $N = 100$, we start to get performance approaching PO2 and PO4 for mid-high SNRs. In fact, at 3 dB, the tandem scheme with $N = 100$ matches the performance of PO4 (which is already slightly better than PO2), but then fails to overtake it. To beat out PO4 completely,

¹For $N = 5000$, the total number of bits simulated was considerably higher (approx. 100M rather than 20M), since the results produced were inconsistent using only a few thousand trials.

we must further increase block size.

Once we move up to $N = 800$, we note that it is possible to beat the performance of PO4. Here the tandem scheme surpasses the BER performance of PO4 at slightly above 3 dB, and remains superior from then on. Considering $N = 5000$, we see the tandem scheme surpassing the performance of PO4 just beyond 4 dB, and subsequently surpassing the tandem scheme for $N = 800$ at approximately 4.5 dB.

These results indicate that PO4 is clearly superior in performance for low and mid-range SNRs. At mid-high SNRs and beyond, it is possible for the tandem scheme to surpass the performance of PO4 for sufficiently long block sizes. It should be noted, however, that the gains over PO4 are less than 1 dB at SNR around 5 dB, even for a very large block size, and that PO4 is achieving twice as much data throughput for its performance.

There are some additional trade-offs to be considered here. The relative complexity will be discussed in the next section, but there are performance trade-offs to be considered, as well. For instance, the long block length necessary to beat the performance of PO4 entails a considerable amount of decoder delay. The receiver must wait for the entire block to be transmitted before passing the observations to the Viterbi decoder, which then must process the data. The long block length, while resulting in better BER performance overall, is susceptible to long runs of corrupted data in individual messages. That is, when the Viterbi decoder outputs an incorrect bit, the message from that point onward (the tail) is heavily corrupted (high concentration

of bit errors) due to desynchronization of the Huffman decoder.

The average corruption run lengths measured during simulation (at the Viterbi decoder output) are shown in Table 5.2. The average lengths presented are over the number of transmissions where an extended data corruption has occurred (rather than average over all trials). The rate of the appearance of these types of errors is given by the occurrence rate. So for $N = 800$, in 0.5% of trials, there were long runs of corrupted data which had an average length of 184 bits. For $N = 5000$, the average corrupted run length was 1285 bits, occurring in 0.2% of trials. While this is a rare occurrences in both cases, it is a considerable portion of the message when it does happen – over 20% of the message (the tail end) is corrupted on average. While the PO constellation has a higher BER at high SNRs, it does not suffer from these long runs of corrupted data, and instead has its bit errors spread more evenly throughout the messages. Whether this concentration of errors is important depends entirely on the specific application, and the tolerance or sensitivity towards different types of failures.

The careful reader will note that manipulation of the average corruption run length and occurrence rate in Table 5.2 does not yield the same average bit error rate as the data presented in Fig. 5.2. This is not a data discrepancy, but rather one of presentation. For Table 5.2, we have a view *inside the machine*, and are able to check when the Viterbi decoder has made a bit error in the sequence it returns, and consider the message bits decoded from that point onward to be corrupted (high probability

of error). The actual result of such a bit error has two possible manifestations. The less likely case is that the error causes the Huffman decoder to decode an incorrect codeword *of the same length* as the intended codeword. It will then continue to decode the rest of the message correctly as if there were no error. The more likely scenario is that the Viterbi bit error will cause the Huffman decoder to decode an incorrect codeword *of a different length* than the intended codeword. This causes the Huffman decoder to become out of synchronization with the true data, but continues to decode incorrect codewords until near the end of the message, when it may find a non-existent codeword. It is likely, given the source distribution under consideration, that the corrupted output will still have many of the bits of the original message correct (the zeros). Since this is essentially the decoder *getting lucky* with its mistakes, we have considered the entire tail of the message to be corrupted when such a Viterbi bit error occurs, as the decoded message is unreliable and contains a higher-than-normal concentration of bit errors, but the measurement of BER for Fig. 5.2 considers only the individual bits (the lucky bits are counted as correct data).

5.3 Complexity Comparison

The most fundamental difference between the two systems is one of hardware complexity versus software complexity. The PO system requires more complicated hardware design (in the transmitter), whereas the tandem source and channel coding needs a processor capable of the calculations required to run the system.

The PO constellation requires more hardware for transmission; it essentially needs a unique pair of amplifiers per constellation point in order to generate the required voltages to send that point. The tandem scheme is simplified in hardware by only transmitting using BPSK, and therefore only needs a single amplifier to be used for both symbols.

The tandem scheme is relatively quite complicated in software implementation. The PO system requires only simple instructions to translate the source data to a constellation point, and a fairly straightforward system for MAP demodulation at the receiver. The tandem scheme requires considerable processing, particularly the receiver for performing the Viterbi calculations to detect the encoded message sequence. This processing could be quite taxing for a small wireless node running off limited battery power, such as those found in a wireless sensor network.

Symbol	Codeword	Codeword Length	Frequency
0000	0	1	0.6561
1000	101	3	0.0729
0100	110	3	0.0729
0010	111	3	0.0729
0001	1000	4	0.0729
1100	100111	6	0.0081
1010	1001000	7	0.0081
1001	1001001	7	0.0081
0110	1001010	7	0.0081
0101	1001011	7	0.0081
0011	1001100	7	0.0081
1110	100110101	9	0.0009
1101	100110110	9	0.0009
1011	100110111	9	0.0009
0111	1001101000	10	0.0009
1111	1001101001	10	0.0001

Table 5.1: Fourth-order Huffman code used for the tandem scheme.

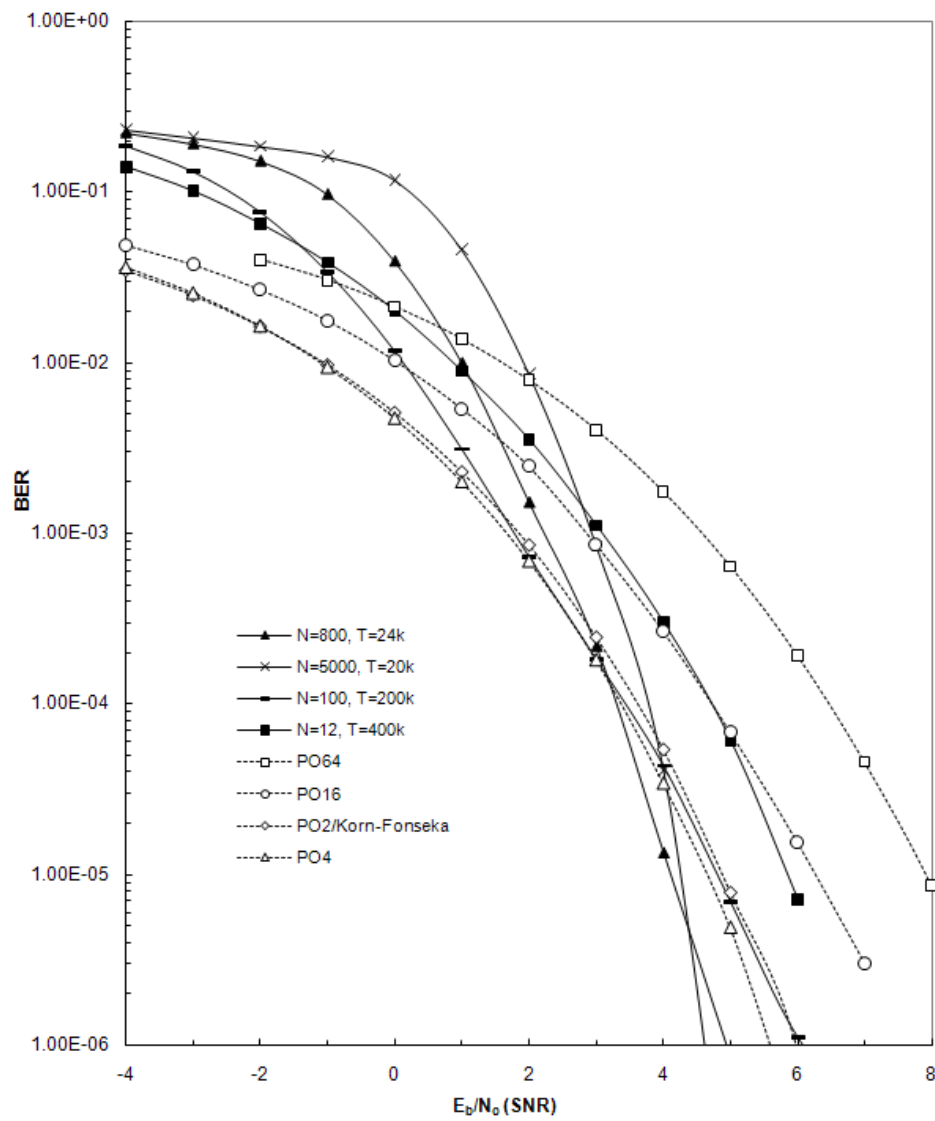


Figure 5.2: Performance of tandem source and channel coding scheme for various block lengths. Selected PO constellation performance shown for reference.

Block Size, N	Crossover SNR	Average Corruption Length	Occurrence
12	5 dB	4	0.02%
100	3 dB	9	0.35%
800	3 dB	184	0.50%
5000	4 dB	1285	0.205%

Table 5.2: Average tail corruption length and occurrence rate (within the Viterbi decoder) for various message block sizes. Results shown only at *Crossover SNR* (the point at which the tandem scheme overall BER performance matches that of PO4). The tandem scheme does not outperform PO4 for block size 12, so this Crossover SNR is where it surpasses PO16.

Chapter 6

Conclusions

6.1 Summary

It is clear that the pairwise optimized constellations offer significant gains over traditional (rectangular QAM) modulation constellations for highly non-uniform sources. This is especially true for high rate constellations where a great deal of energy is “wasted” by placing likely symbols far from the origin. We recognize that asymmetric non-rectangular constellations introduce additional complexity both in the hardware of the transmitter (for modulation) and in the calculations required for demodulation. Smaller improvements can be easily obtained by re-centering the traditional rectangular constellations to be zero mean, and scaling them up to their original average energy.

The gains achieved when considering SER were not only sustained, but improved,

as good maps were designed for the PO constellation and BER performance was measured. Despite higher Hamming distances between neighbouring symbols, the BER performance of the PO constellation was significantly better than standard Gray-mapped rectangular QAM.

When comparing the uncoded PO system to a tandem source and channel coded system, neither was universally superior. PO was general better at low SNR, while the tandem scheme (for large enough block size) was superior at high SNRs. Where the BER performance of the two systems matched, it is notable that the PO constellations transmitted at a higher rate. Each system had its advantages and disadvantages, and making a choice between them for implementation would depend entirely on the application.

6.2 Future Work

Here we present some areas where the work we have described in this paper could be improved or advanced in some way. Some of these would be relatively simple steps, such as improving the approximation of the SER used to design the PO constellations in Chapter 3. As it stands, we employed the Union Upper Bound to the SER, which can be loose for low SNRs. We deemed this acceptable since we did not need an accurate measure of the SER, just some indication of where to place signal points to try to improve SER performance, and we needed it to be very fast to keep the speed reasonable for the repeated iterations. It would be interesting to learn if the

extra computational cost of a closer approximation would yield significantly different constellations.

Additionally, some of the larger constellations are quite irregular in appearance. Several points, particularly in PO256, are placed out of line with the overall layer pattern. It might be possible to *massage* these constellations manually to make them more regular in appearance. This would not be simply for aesthetic value. If the constellation were slightly adjusted in places with the intent of aligning small sets of points, it would be possible to simplify a hardware implementation by sharing amplifiers for multiple points.

Furthermore, we suggest in Chapter 3 that the optimum constellation for a nonuniform source will obey the strict ordering of points by probability (more likely points lie closer to the origin). We were unsuccessful in our attempts to prove this, but could not construct a counterexample. It would certainly be of value to prove this ordering, since it would considerably reduce the search space when looking for optimum constellations.

Finally, it would be interesting to explore the continuity of the optimal constellation (in terms of SER performance) with respect to changes in p , to determine if a continuous path of optimal constellations exists.

Bibliography

- [1] E. Agrell, E.G. Strom, and T. Ottosom. Gray Coding for Multilevel Constellations in Gaussian Noise. *IEEE Transactions on Information Theory*, IT-53(1):224–235, January 2007.
- [2] F. Alajaji, N. Phamdo, and T. Fuja. Channel Codes that Exploit the Residual Redundancy in CELP-Encoded Speech. *IEEE Transactions on Speech Audio Processing*, 4:325–336, Sept. 1996.
- [3] F. Behnamfar, F. Alajaji, and T. Linder. MAP Decoding for Multi-Antenna Systems with Non-Uniform Sources: Exact Pairwise Error Probability and Applications. *IEEE Transactions on Communications*, 57(1):242–254, January 2009.
- [4] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience, second edition, 2006.
- [5] S. Emami and S. L. Miller. Nonsymmetric Sources and Optimum Signal Selection. *IEEE Transactions on Communications*, 44(4):440–447, April 1996.

- [6] G. J. Foschini, R. D. Gitlin, and S. B. Weinstein. Optimization of Two-Dimensional Signal Constellations in the Presence of Gaussian Noise. *IEEE Transactions on Communications*, COM-22(1):28–38, January 1974.
- [7] J. Huang, S. Meyn, and M. Médard. Error Exponents for Channel Coding With Applications to Signal Constellation Design. *IEEE Journal on Selected Areas of Communications*, 24(8):1647–1661, August 2006.
- [8] I. Korn, J. P. Fonseka, and S. Xing. Optimal Binary Communication With Nonequal Probabilities. *IEEE Transactions on Communications*, 51(9):1435–1438, September 2003.
- [9] H. Kuai, F. Alajaji, and G. Takahara. Tight Error Bounds for Nonuniform Signaling over AWGN Channels. *IEEE Transactions on Information Theory*, 46(7):2712–2718, November 2000.
- [10] H. Nguyen and T. Nechiporenko. Quarternary Signal Sets for Digital Communications with Nonuniform Sources. *IEEE CCECE/CCGEI*, pages 2085–2088, May 2005.
- [11] J. G. Proakis. *Digital Communications*. McGraw-Hill, fourth edition, 2000.
- [12] J. G. Proakis and M. Salehi. *Communication Systems Engineering*. Pearson Prentice Hall, second edition, 2002.

- [13] Y. Sun. Stochastic Iterative Algorithms for Signal Set Design for Gaussian Channels and Optimality of the L2 Signal Set. *IEEE Transactions on Information Theory*, 43(5):1574–1587, September 1997.
- [14] G. Takahara, F. Alajaji, N. C. Beaulieu, and H. Kuai. Constellation Mappings for Two-Dimensional Signaling of Nonuniform Sources. *IEEE Transactions on Communications*, 51(3):400–408, March 2003.
- [15] N. Wei and Y. Wan. Optimal Constellation for General Rectangular PAM/QAM with Arbitrary Code Mapping. In *Proceedings of IEEE International Conference on Communications*, pages 2749–2754, Glasgow, Scotland, June 24-28 2007.

Appendix A

Constellation Coordinates

Included here are tables listing the point coordinates and bit mappings assigned for the PO constellation tested in this thesis. For the case of $M = 16$ where many design SNRs were considered, we include only the “main” PO16 constellation compared to the traditional constellations using.

Table A.1: Coordinates and bit mapping for PO4 constellation, for $p = 0.9$ and design SNR of 0 dB.

Probability	X Coord.	Y Coord.	Bit Mapping
0.81	-0.2237	0.2702	00
0.09	2.1540	-0.2112	01
0.09	0.1465	-2.1067	10
0.01	-2.5832	-1.0291	11

Table A.2: Coordinates and bit mapping for PO8 constellation, for $p = 0.9$ and design SNR of 0 dB.

Probability	X Coord.	Y Coord.	Bit Mapping
0.729	-0.2073	-0.1505	000
0.081	1.3941	0.9974	001
0.081	-0.5297	1.7736	010
0.081	1.4689	-1.0913	100
0.009	-0.2304	-2.3443	110
0.009	-2.3356	0.6856	011
0.009	-2.0093	-1.2882	101
0.001	3.2953	0.1616	111

Table A.3: Coordinates and bit mapping for PO16 constellation, for $p = 0.9$ and design SNR of 1 dB.

Probability	X Coord.	Y Coord.	Bit Mapping
0.6561	-0.1743	0.126	0000
0.0729	0.7246	1.3984	1000
0.0729	0.8259	-1.2234	0001
0.0729	-0.7552	-1.291	0010
0.0729	1.4862	0.0598	0100
0.0081	-2.1079	-0.5451	0110
0.0081	-1.1739	1.5154	1100
0.0081	-1.9892	0.626	0101
0.0081	-2.0126	-1.9601	1010
0.0081	-0.3329	2.4461	1001
0.0081	0.0828	-2.6222	0011
0.0009	3.0669	0.2873	1101

Continued on next page

Table A.3 – continued from previous page

Probability	X Coord.	Y Coord.	Bit Mapping
0.0009	2.3512	1.6216	1110
0.0009	2.6403	-1.2074	0111
0.0009	1.8417	-2.4818	1011
0.0001	1.4126	3.0828	1111

Table A.4: Coordinates for PO16 constellation, for $p = 0.5$ and design SNR of 1 dB.

Probability	X Coord.	Y Coord.
0.0625	-1.2032	0.0847
0.0625	1.2335	-0.13
0.0625	0.1746	0.3778
0.0625	-0.3461	0.4311
0.0625	-0.9965	-0.6428
0.0625	-0.4436	-1.0984
0.0625	0.678	0.1487
0.0625	-0.932	0.7585
0.0625	-0.1572	-0.5011
0.0625	0.3682	1.0974
0.0625	-0.3007	1.1351
0.0625	-0.546	-0.0978
0.0625	0.2633	-1.1483
0.0625	0.3614	-0.3469
0.0625	0.8686	-0.7887
0.0625	0.9776	0.7207

Table A.5: Coordinates for PO16 constellation, for $p = 0.6$ and design SNR of 1 dB.

Probability	X Coord.	Y Coord.
0.1296	-0.2295	-0.1986
0.0864	-0.6293	0.3912
0.0864	0.3681	-0.3495
0.0864	-0.0184	0.6597
0.0864	0.5575	0.3436
0.0576	0.9057	-0.8426
0.0576	-0.9085	-0.7539
0.0576	1.1837	-0.1803
0.0576	-1.1531	-0.1366
0.0576	0.2607	-1.167
0.0576	-0.3621	-1.1069
0.0384	-0.786	1.1415
0.0384	0.7395	1.1516
0.0384	-0.0269	1.3746
0.0384	1.288	0.5646
0.0256	-1.4326	0.5514

Table A.6: Coordinates for PO16 constellation, for $p = 0.7$ and design SNR of 1 dB.

Probability	X Coord.	Y Coord.
0.2401	-0.1544	-0.1987
0.1029	-0.0897	0.832
0.1029	-0.7677	0.3916
0.1029	0.5599	-0.3815
0.1029	0.6111	0.5224
0.0441	-1.3246	-0.2084
0.0441	-0.9503	-0.8764
Continued on next page		

Table A.6 – continued from previous page

Probability	X Coord.	Y Coord.
0.0441	0.3442	-1.2997
0.0441	1.3927	0.0429
0.0441	-0.3538	-1.2426
0.0441	1.1422	-0.8746
0.0189	0.6228	1.5463
0.0189	1.362	0.9689
0.0189	-0.3359	1.6217
0.0189	-1.1765	1.1777
0.0081	-1.8702	0.4302

Table A.7: Coordinates for PO16 constellation, for $p = 0.8$ and design SNR of 1 dB.

Probability	X Coord.	Y Coord.
0.4096	-0.1362	-0.1846
0.1024	0.8053	-0.52
0.1024	0.7716	0.6373
0.1024	-1.0024	0.4216
0.1024	-0.1263	1.0628
0.0256	0.4889	-1.5915
0.0256	-0.3522	-1.4604
0.0256	-1.6432	-0.3512
0.0256	-1.0807	-1.0523
0.0256	1.5501	-0.9567
0.0256	1.739	0.267
0.0064	-1.6778	1.2254
0.0064	-0.8021	1.89
0.0064	1.4251	1.5348
0.0064	0.4097	2.0307

Continued on next page

Table A.7 – continued from previous page

Probability	X Coord.	Y Coord.
0.0016	-2.4538	0.3331

Table A.8: Coordinates and bit mapping for PO64 constellation, for $p = 0.9$ and design SNR of 2 dB .

Probability	X Coord.	Y Coord.	Bit Mapping	Probability	X Coord.	Y Coord.	Bit Mapping
0.531441	0.0376	-0.0845	000000	0.000729	1.5265	-2.325	010101
0.059049	0.8744	0.695	100000	0.000729	-0.3529	-2.9831	001101
0.059049	-0.0049	1.1104	000100	0.000729	-1.2423	-2.6314	001011
0.059049	1.0865	-0.3444	010000	0.000729	0.5765	2.9316	010110
0.059049	-0.8841	0.6711	001000	0.000729	2.8538	0.5536	111000
0.059049	-1.0774	-0.3131	000010	0.000729	3.0146	-0.2801	110001
0.059049	-0.3044	-1.1257	000001	0.000729	2.5637	1.4012	110010
0.006561	-2.0783	-0.2146	010010	0.000729	1.3493	2.6449	100110
0.006561	1.517	-1.2613	010100	0.000729	-2.6746	-1.1252	000111
0.006561	-0.2578	-2.0942	001001	0.000729	-2.8825	0.6296	011010
0.006561	-0.7973	1.7838	001100	0.000081	0.4801	3.8333	101110
0.006561	0.6541	-1.1171	010001	0.000081	-0.8479	3.8026	011110
0.006561	1.8988	0.5972	110000	0.000081	-1.2126	-3.5133	011011
0.006561	2.0525	-0.2929	011000	0.000081	2.3583	-2.8328	011101
0.006561	-1.8971	0.6481	001010	0.000081	-3.5106	1.3302	111010
0.006561	0.7711	1.8321	100100	0.000081	-3.6627	-1.0441	010111
0.006561	1.5779	1.4945	100010	0.000081	1.3513	-3.3348	100111
0.006561	-1.1376	-1.6919	100001	0.000081	-3.0006	2.2664	111100
0.006561	0.6667	-1.9069	000101	0.000081	3.0996	2.2522	110110
0.006561	-1.7187	-1.0957	000011	0.000081	3.2116	-1.8678	110101
0.006561	-0.0484	2.1727	000110	0.000081	3.5861	-1.0021	111001

Continued on next page

Table A.8 – continued from previous page

Probability	X Coord.	Y Coord.	Bit Mapping	Probability	X Coord.	Y Coord.	Bit Mapping
0.006561	-1.5769	1.4863	101000	0.000081	0.315	-3.7329	101101
0.000729	0.5691	-2.8256	101001	0.000081	3.5758	1.2715	110011
0.000729	2.2609	-1.8888	100101	0.000081	-1.7224	3.3658	001111
0.000729	2.0906	2.2706	110100	0.000081	-2.2429	-2.8696	101011
0.000729	-3.0274	-0.2339	010011	0.000009	-4.0779	0.3131	011111
0.000729	-1.9937	2.3444	101100	0.000009	1.794	3.5924	111110
0.000729	-2.1234	-1.9502	100011	0.000009	-0.5284	-4.1554	111101
0.000729	-2.5764	1.4582	101010	0.000009	-3.2663	-2.1555	110111
0.000729	-0.3873	3.0373	011100	0.000009	4.0121	0.3069	111011
0.000729	-1.1577	2.6547	001110	0.000009	-2.6927	3.2045	101111
0.000729	2.5521	-1.0764	011001	0.000001	2.8666	-3.7328	111111

Table A.9: Coordinates and bit mapping for PO256 constellation, for $p = 0.9$ and design SNR of 4 dB.

Probability	X Coord.	Y Coord.	Bit Mapping	Probability	X Coord.	Y Coord.	Bit Mapping
0.43046721	-0.0486	-0.0556	00000000	0.00006561	-1.417	3.0637	10001110
0.04782969	0.5378	0.6043	10000000	0.00006561	-2.9498	-1.4022	01100110
0.04782969	0.6671	-0.0058	00001000	0.00006561	0.0867	3.4766	10100011
0.04782969	-0.0859	0.6569	00000010	0.00006561	2.1126	-2.987	00011110
0.04782969	0.6032	-0.6578	00010000	0.00006561	-3.8447	-0.2313	00110101
0.04782969	-0.7714	-0.071	00000001	0.00006561	-0.504	2.8924	10100110
0.04782969	-0.0288	-0.7599	01000000	0.00006561	3.0395	1.653	11101000
0.04782969	-0.6633	-0.68	00100000	0.00006561	-1.0939	3.5761	10011100
0.04782969	-0.7059	0.5624	00000100	0.00006561	-3.2148	0.629	00011101
0.00531441	1.7884	0.5035	10100000	0.00006561	2.9243	-2.288	01011010
0.00531441	-1.8285	1.62	01000100	0.00006561	-1.6815	2.196	10000111
0.00531441	1.8263	-0.6386	00010001	0.00006561	-1.8164	2.7554	11000110
0.00531441	1.1222	1.6818	10000001	0.00006561	3.2331	-2.6756	00001111
0.00531441	1.2694	-0.1191	00011000	0.00006561	-0.3284	-3.6768	01110010
0.00531441	-1.2708	1.0766	00000011	0.00006561	-2.9872	-0.7666	01101100
0.00531441	0.4743	-1.3353	00010100	0.00006561	-3.4165	-1.6434	01001110
0.00531441	0.5485	1.7944	10010000	0.00006561	1.5211	-3.6955	01010110
0.00531441	1.1868	0.488	00010010	0.00006561	-3.0793	-2.0524	01011001
0.00531441	1.0667	-1.2848	01010000	0.00006561	0.1833	3.9638	00111001
0.00531441	-0.6626	-1.3567	01100000	0.00006561	-3.2942	-0.3711	00101101
0.00531441	-1.3192	-0.641	01000001	0.00006561	0.9041	4.0421	10110001
0.00531441	-0.5769	1.779	00000110	0.00006561	0.6685	3.4758	10011010
0.00531441	1.7243	1.6891	00001001	0.00006561	0.5819	2.9491	10010011
0.00531441	-0.6855	1.203	01000010	0.00006561	-0.4448	3.3895	10101010
0.00531441	-1.3853	0.4903	00000101	0.00006561	-0.9518	3.0535	10010110
0.00531441	-0.1172	1.2606	10000010	0.00006561	-3.3632	-1.095	00101110
0.00531441	-1.1713	1.706	10000100	0.00006561	3.1758	-1.3637	00101011

Continued on next page

Table A.9 – continued from previous page

Probability	X Coord.	Y Coord.	Bit Mapping	Probability	X Coord.	Y Coord.	Bit Mapping
0.00531441	1.6967	1.0868	11000000	0.00006561	1.5965	-3.1777	00111100
0.00531441	-1.4181	-0.0762	00100100	0.00006561	-0.8501	-3.6529	11100010
0.00531441	-0.0951	-1.4095	00110000	0.00006561	0.1306	-4.022	11110000
0.00531441	-1.2607	-1.2391	00100001	0.00006561	1.8565	3.269	11001001
0.00531441	1.0288	1.0743	10001000	0.00006561	2.5293	-2.676	01011100
0.00531441	1.235	-0.6976	01001000	0.00006561	-1.7212	-2.9448	01100011
0.00531441	0.444	1.2189	00001010	0.00006561	-2.2402	-2.8524	01010011
0.00531441	1.7089	-1.2456	00001100	0.00006561	0.6766	-3.1234	11010010
0.00531441	-0.0055	1.8375	00100010	0.00000729	-0.4259	-4.3091	10110110
0.00531441	1.7718	-0.0551	00101000	0.00000729	-2.7954	-3.7071	11110001
0.00059049	2.0642	2.2822	10001001	0.00000729	3.9004	-1.9574	01001111
0.00059049	2.2681	2.8663	10000011	0.00000729	4.6908	-0.7853	01111001
0.00059049	2.3827	0.1406	10011000	0.00000729	-0.1127	-4.9053	10110101
0.00059049	2.4245	-0.385	00111000	0.00000729	-4.5989	-1.1886	01101101
0.00059049	0.326	-2.7279	01010010	0.00000729	-2.3882	-4.0917	01100111
0.00059049	-0.0009	-1.9606	01110000	0.00000729	-0.8467	-4.7231	00110111
0.00059049	2.684	2.1621	00001011	0.00000729	2.7341	-3.6192	11011010
0.00059049	-1.254	-1.7679	01100001	0.00000729	-1.7492	4.1898	11011100
0.00059049	0.98	2.3823	10001010	0.00000729	2.9141	-3.0597	01011110
0.00059049	-2.4825	-1.8487	01001010	0.00000729	4.5749	1.3161	11110010
0.00059049	2.3851	1.6304	11000001	0.00000729	2.7892	-4.2216	00111101
0.00059049	-1.2729	2.4289	11000100	0.00000729	-3.5122	-2.6455	01011011
0.00059049	2.1891	-1.5783	00011001	0.00000729	2.5462	3.45	10011011
0.00059049	1.8212	-1.9934	01011000	0.00000729	3.3088	3.0581	11000111
0.00059049	-0.5915	-2.4952	01100010	0.00000729	2.8535	3.902	10001111
0.00059049	-0.9573	-2.1081	10100001	0.00000729	-1.7227	3.5885	11001110
0.00059049	-2.7262	0.3693	00010101	0.00000729	-1.3682	4.8561	10011101
0.00059049	2.2991	1.101	11001000	0.00000729	-3.1845	-3.0991	11101001
0.00059049	-2.3768	-1.2394	01001001	0.00000729	0.6565	4.6133	10111001

Continued on next page

Table A.9 – continued from previous page

Probability	X Coord.	Y Coord.	Bit Mapping	Probability	X Coord.	Y Coord.	Bit Mapping
0.00059049	-1.882	1.0372	01000110	0.00000729	2.8609	2.9917	11011001
0.00059049	-0.1135	-2.4431	00110010	0.00000729	-1.2572	4.1354	10101110
0.00059049	-0.173	2.4314	10100010	0.00000729	-4.2125	0.8918	01110101
0.00059049	1.2152	-2.2178	10010100	0.00000729	3.6955	-3.1969	00011111
0.00059049	-0.5436	-3.0462	00100110	0.00000729	-1.8151	-4.1604	11010011
0.00059049	-1.9075	-1.7075	01000011	0.00000729	4.5812	-0.194	01111010
0.00059049	3.004	-0.7756	00010011	0.00000729	4.1917	-0.6672	01110011
0.00059049	-1.9719	-0.2558	00100101	0.00000729	-3.5202	3.611	11010101
0.00059049	1.0394	-2.7529	00010110	0.00000729	-4.4177	-0.5008	11100101
0.00059049	1.6259	-2.5398	00011010	0.00000729	-2.6357	-3.1735	01101011
0.00059049	-2.4547	1.3351	01000101	0.00000729	4.8947	0.4289	11111000
0.00059049	1.4842	2.681	10010001	0.00000729	3.1033	2.6017	10101011
0.00059049	2.888	1.1463	11000010	0.00000729	-1.059	-4.1792	11100110
0.00059049	-1.9173	0.5261	00000111	0.00000729	-2.9423	3.6456	11001101
0.00059049	1.3893	-1.7325	00011100	0.00000729	1.3768	-4.4745	01111100
0.00059049	0.0778	-3.1958	10110000	0.00000729	2.0946	-4.0201	01011101
0.00059049	2.4016	-2.0974	00001110	0.00000729	4.1351	2.1198	00101111
0.00059049	-1.8362	-1.1896	01100100	0.00000729	-2.141	3.9654	11010110
0.00059049	-2.3411	0.7677	10000101	0.00000729	1.7214	4.2225	10110011
0.00059049	0.647	-2.3584	00110100	0.00000729	-0.2668	4.5221	10111100
0.00059049	-2.1815	2.2753	10001100	0.00000729	0.9136	-4.93	01110110
0.00059049	-1.2101	-2.6392	01010001	0.00000729	-2.4426	3.6144	10011110
0.00059049	-0.5131	-1.9109	00110001	0.00000729	2.2734	3.8174	10010111
0.00059049	2.6736	-1.3587	01001100	0.00000729	3.2265	2.1317	11001011
0.00059049	-2.4387	-0.6923	10100100	0.00000729	-3.1347	3.0896	10101101
0.00059049	-1.6264	-2.2017	00100011	0.00000729	-0.8922	4.6893	10100111
0.00059049	0.4113	2.4449	10010010	0.00000729	4.2077	0.0664	10111010
0.00059049	-2.6278	-0.1936	00101100	0.00000729	2.2295	-3.5783	00111110
0.00059049	2.9697	-0.1545	00101010	0.00000729	-4.2309	1.5143	01010111

Continued on next page

Table A.9 – continued from previous page

Probability	X Coord.	Y Coord.	Bit Mapping	Probability	X Coord.	Y Coord.	Bit Mapping
0.00059049	-0.717	2.4056	1000110	0.00000729	3.688	2.4871	11100011
0.00059049	-2.1821	0.1623	00001101	0.00000729	0.5037	-4.5472	11110100
0.00059049	2.9608	0.462	10101000	0.00000729	4.0104	1.5266	11101010
0.00059049	0.8897	-1.8333	01010100	0.00000729	-4.066	-1.6463	11101100
0.00059049	2.4247	0.6169	11100000	0.00000729	-3.8863	-2.2014	01101110
0.00059049	2.376	-0.9256	01101000	0.00000729	4.279	-1.3894	00111011
0.00059049	-1.9236	-0.7214	00101001	0.00000081	-2.7283	4.2552	11011110
0.00059049	0.4314	-1.9494	11010000	0.00000081	-0.056	5.124	10111011
0.00006561	-2.1183	-2.3875	11100001	0.00000081	4.1336	-2.8294	01011111
0.00006561	-3.3155	2.6617	10001101	0.00000081	-4.9748	-0.4399	10101111
0.00006561	1.4994	2.1951	10011001	0.00000081	-3.2235	-2.515	01101111
0.00006561	-2.3175	3.0375	11001100	0.00000081	2.1172	-4.5882	10111110
0.00006561	-1.4764	-3.5851	11010001	0.00000081	-1.5197	-4.8173	11110110
0.00006561	3.53	0.5921	10111000	0.00000081	2.2307	4.4161	11011011
0.00006561	0.0525	2.9815	10110010	0.00000081	1.2495	4.7906	10011111
0.00006561	3.6678	-1.3628	01101001	0.00000081	-4.0818	3.0454	11011101
0.00006561	3.7251	-0.8247	01111000	0.00000081	4.793	-1.8105	01111101
0.00006561	0.9087	-3.9864	10110100	0.00000081	5.4751	1.4035	11111010
0.00006561	-1.172	-3.221	00110011	0.00000081	-4.5569	-1.8304	11111100
0.00006561	-3.8001	-0.8226	10101100	0.00000081	3.7078	-2.4634	00111111
0.00006561	4.0332	0.6043	01101010	0.00000081	3.352	-3.7831	01111110
0.00006561	-0.5438	3.9604	10101001	0.00000081	-2.0595	-5	11110011
0.00006561	-2.8996	0.963	01000111	0.00000081	-4.2101	2.0803	11101101
0.00006561	-2.7599	-2.4954	01001011	0.00000081	-4.03	-3.2078	11010111
0.00006561	0.5128	-3.5586	00110110	0.00000081	-4.9497	0.2784	10111101
0.00006561	3.741	-0.2793	00111010	0.00000081	4.0804	4.2673	10110111
0.00006561	-3.5958	1.1413	01010101	0.00000081	4.7593	2.0023	11101011
0.00006561	-3.5545	1.8935	01100101	0.00000081	3.7293	3.4825	11001111
0.00006561	1.0696	3.1085	10001011	0.00000081	-3.0021	-4.4793	11111001

Continued on next page

Table A.9 – continued from previous page

Probability	X Coord.	Y Coord.	Bit Mapping	Probability	X Coord.	Y Coord.	Bit Mapping
0.00006561	-2.9546	2.0739	11010100	0.00000081	4.842	3.584	11100111
0.00006561	-3.6701	0.5711	00100111	0.00000081	-4.8791	0.9054	01110111
0.00006561	-2.5201	1.8641	11000101	0.00000081	-4.4793	-2.3754	11101110
0.00006561	-3.3116	0.1374	10100101	0.00000081	4.5419	-2.2344	01111011
0.00006561	3.4818	1.1454	11001010	0.00000081	-5.6384	-0.3315	11110101
0.00006561	1.0571	-3.4085	01110100	0.00000009	1.6651	-5.4917	11111110
0.00006561	-2.0661	-3.4774	01110001	0.00000009	5.3404	0.8641	11111011
0.00006561	-2.7651	2.6266	11100100	0.00000009	-3.9444	-2.7553	11101111
0.00006561	3.4856	0.0957	11011000	0.00000009	5.1786	-1.3959	01111111
0.00006561	3.5757	1.7936	11000011	0.00000009	4.3924	3.488	11011111
0.00006561	-4.115	0.2999	00010111	0.00000009	-4.4326	2.5487	11111101
0.00006561	3.1992	-1.8705	00011011	0.00000009	-5.6009	0.6667	10111111
0.00006561	-3.031	1.4423	01001101	0.00000009	-7.7859	1.4055	11110111
0.00006561	1.3385	3.6193	10010101	0.00000001	-5.4258	1.8394	11111111

Appendix B

Source Code

This Chapter describes and includes selected portions of the source code used in designing and simulating the PO constellations. The Python Programming Language was used and the code is commented throughout following the `#` character (see www.python.org for more information).

B.1 Optimization

Code included here deals with creating the PO constellations.

B.1.1 Initializing a Constellation

```
##### INIT CONSTELLATION BLOCK #####
N = 16
print "N=",N
#number of bits needed
M = int(math.ceil(math.log(N,2)))
print "M=",M
# probability of 0
P0=0.9
#Average energy (not sure this works properly right now, but needs to be set anyway)
```

```

Pav=1.0
N_0 = 0.33
SNR = 2.0
N_0 = (Pav/(1.0*M)) / (10.0**(SNR/10.0))
print "N_0 =",N_0
os.chdir("saved-constellations")
#set initial constellation
C = initconst.newConst_circles(M,P0,Pav)
C = initconst.zeromeanCon(C)
C = initconst.avgenergyCon(C,Pav)
C2 = initconst.grid(M,P0)
C2 = initconst.zeromeanCon(C2)
C2 = initconst.avgenergyCon(C2,Pav)
ugly = [(0.85,-1.0,-1.0),(0.01,0.01,0.01),(0.01,0.02,0.02),(0.01,0.01,0.02),
(0.01,0.02,0.01),(0.01,0.03,0.01),(0.01,0.03,0.02),(0.01,0.01,0.03),
(0.01,0.03,0.03),(0.01,0.04,0.01),(0.01,0.02,0.04),(0.01,0.01,0.04),
(0.01,0.04,0.04),(0.01,0.05,0.01),(0.01,0.02,0.05),(0.01,0.01,0.05)]
ugly = initconst.zeromeanCon(ugly)
ugly = initconst.avgenergyCon(ugly,Pav)
#print C2
global con
con = C2
#con = ugly      #C for circles, C2 for grid. circles better for small N, grid better for large N
#print "using UGLY!!"
print "Energy is",getEnergy(con)

```

B.1.2 Setting Up and Running the GUI

This is the code used to initialize and operate the display and interface which shows the constellations changing over time and allows the user to stop/start/save/load constellations.

```

#===== SETTING UP GUI =====
clen = len(con)
crange = range(0,clen)
canW = 1000
canH = 1000

g = math.floor(50.0/P0) #graphical scaling constant
g=100 #defaults value if not set/overwritten

ds = 3 #graphical dot size
x0 = math.floor(canW/2)
y0 = math.floor(canH/2)

#initialize tkinter GUI components
root = Tk()
canvas = Canvas(root,width=canW, height=canH)
canvas.pack()

#bind input events to canvas to allow user interaction
canvas.bind_all("<Key>",eventkeypress)      #interrupt to deal with a keyboard key press

#set up canvas
bg = canvas.create_rectangle(0,0,canW,canH,fill="white")
xaxis = canvas.create_line(0,y0,canW,y0,fill="black")
yaxis = canvas.create_line(x0,0,x0,canH,fill="black")

#set up initial constellation display
pts = range(0,clen)

```

```

labs = range(0,clen)
for i in crange:
    pts[i] = canvas.create_oval(x0+con[i][1]*g - ds,y0-con[i][2]*g - ds,x0+con[i][1]*g + ds,
        y0-con[i][2]*g + ds,fill="red")
    prob = math.floor(con[i][0]*100000.0)/100000.0
    labs[i] = canvas.create_text(x0+con[i][1]*g + ds,y0-con[i][2]*g + ds,anchor="nw",fill="black",text=prob)

#=====

#start the app

print "\nKeyboard commands:"
print "Press g to start pairwise optimization"
print "Press s to stop operation"
print "Press o to save current constellation to txt/ps file pair"
print "Press i to load a constellation from txt file"
print "Press q to quit"
flag = 0

while flag>=0:
    try:
        canvas.update()           #always update canvas
        if flag==1:               #if in optimization state,
            doPair()              #run optimize pair code
    except tkinter.TclError:
        print "You closed the display window. Exiting without error."
        flag = -2                 #set flag to not try closing window (user already closed)
    pass
print "Exiting ... "
if flag!=-2:                     #do not try to close if user close manually already (flag -2)
    canvas.destroy()             #close the tkinter display to avoid window buildup
print "Goodbye."

```

B.1.3 Optimizing Pairs

The GUI code above repeatedly calls “doPair()” which is where the pair work happens. This function is another manager for the GUI (essentially updates the new positions of the points once they have been optimized), but it does call the actual optimization code, “optimizepair(),” for a random pair of points in the constellation.

```

def doPair():
    global con
    global N_0
    s1 = random.randrange(0,clen)
    s2 = random.randrange(0,clen)
    while s2==s1:
        s2 = random.randrange(0,clen)

    con = optpair.optimizepair(con,N_0,Pav,s1,s2)
    canvas.coords(pts[s1],x0+con[s1][1]*g - ds,y0-con[s1][2]*g - ds,
        x0+con[s1][1]*g + ds,y0-con[s1][2]*g + ds)
    canvas.coords(pts[s2],x0+con[s2][1]*g - ds,y0-con[s2][2]*g - ds,
        x0+con[s2][1]*g + ds,y0-con[s2][2]*g + ds)
    canvas.coords(labs[s1],x0+con[s1][1]*g + ds,y0-con[s1][2]*g + ds)
    canvas.coords(labs[s2],x0+con[s2][1]*g + ds,y0-con[s2][2]*g + ds)

def optimizepair(con,N_0,Pav,s1,s2):
    E=Pav

```

```

conrange = range(0, len(con))
#measuring correct E
emeas=0.0
for i in conrange:
    emeas += con[i][0]*normsq(con[i][1],con[i][2])
    #pass
E = emeas

#perform 2-D minimization of SER moving only 2 points
#get circle of S2
rt2n = math.sqrt(2*N_0)
p1 = con[s1][0] #load data to local vars from con to simplify code
x1 = con[s1][1]
y1 = con[s1][2]
p2 = con[s2][0]
x2 = con[s2][1]
y2 = con[s2][2]
# find vector a = b / p1, and energy of other pts d
ax = 0.0
ay = 0.0
d = 0.0
c = 1.0*p2/p1
for i in conrange:
    if (i!=s1) and (i!=s2):
        ax += -1.0*con[i][0]*con[i][1]/p1
        ay += -1.0*con[i][0]*con[i][2]/p1
        d += con[i][0]*normsq(con[i][1],con[i][2])
        #pass
    #pass
# calculate centre and radius of circle for s2
xcen = (p1*ax*c)/(p1*c*c+p2)
ycen = (p1*ay*c)/(p1*c*c+p2)
h = p1*ax*ax - ((p1*ax*c)**2)/(p1*c*c+p2) + p1*ay*ay - ((p1*ay*c)**2)/(p1*c*c+p2)
r2 = (E-d-h)/(p1*c*c+p2)
#print E,d,h,(p1*c*c+p2)
#print "r2 is ",r2
r = math.sqrt(r2)

#optimize s1 s2 over the constrained circle (x2 - xc)^2 + (y2-yc)^2 = r^2,
# and x1 = ax - c*x2, y1 = ay - c*y2

theta = math.atan((y2-ycen)/(x2-xcen))
maxtheta = theta + 2*math.pi
minflag = 1
min = 0.0
mintheta = 0.0
while theta <= maxtheta:
    x2n = xcen + r*math.cos(theta)
    y2n = ycen + r*math.sin(theta)
    x1n = ax - c*x2n
    y1n = ay - c*y2n

    #get sum of SER parts that change as s1 s2 change location
    sum1 = 0.0
    sum2 = 0.0
    sumi = 0.0
    for i in conrange: #this is the calculation of F_12
        if i!=s1:
            if i==s2:
                sum1+=p1*qfunc.normp((distance(x1n,y1n,x2n,y2n)/rt2n)
                    + (rt2n*math.log(p1/p2)/(2*distance(x1n,y1n,x2n,y2n))))
            else:
                sum1+=p1*qfunc.normp((distance(x1n,y1n,con[i][1],con[i][2])/rt2n)
                    + (rt2n*math.log(p1/con[i][0])/(2*distance(x1n,y1n,con[i][1],con[i][2]))))

```

```

if i!=s2:
    if i==s1:
        sum2+=p2*qfunc.normp((distance(x2n,y2n,x1n,y1n)/rt2n)
        + (rt2n*math.log(p2/p1)/(2*distance(x2n,y2n,x1n,y1n))))
        #pass
    else:
        sum2+=p2*qfunc.normp((distance(x2n,y2n,con[i][1],con[i][2])/rt2n)
        + (rt2n*math.log(p2/con[i][0])/(2*distance(x2n,y2n,con[i][1],con[i][2]))))
        #pass
if (i!=s1) and (i!=s2):
    sumi += con[i][0]*(qfunc.normp((distance(con[i][1],con[i][2],x1n,y1n)/rt2n)
    + (rt2n*math.log(con[i][0]/p1)/(2*distance(con[i][1],con[i][2],x1n,y1n))))))
    sumi += con[i][0]*(qfunc.normp((distance(con[i][1],con[i][2],x2n,y2n)/rt2n)
    + (rt2n*math.log(con[i][0]/p2)/(2*distance(con[i][1],con[i][2],x2n,y2n))))))
    #pass
#pass
sersum = sum1+sum2+sumi
if minflag == 1 :
    min = sersum
    mintheta = theta
    minflag = 0
else:
    if sersum < min:
        min = sersum
        mintheta = theta
theta+= math.pi/50.0
#size of divisor here has crucial role affecting speed and accuracy of process. (Larger is slower but better)
#pass
#print "min theta is ",mintheta," with min ",min

#replace old pair with new pair
x2new = xcen + r*math.cos(mintheta)
y2new = ycen + r*math.sin(mintheta)
x1new = ax - c*x2new
y1new = ay - c*y2new

newcon = replacepoint(con,s1,x1new,y1new)
newcon = replacepoint(newcon,s2,x2new,y2new)

#return new constellation
return newcon

```

B.2 Simulation

Code included here deals with the simulations used to test the PO constellations.

B.2.1 Loading Constellations

The following code is used to list the available constellation files in the working directory that the user can choose to load for simulation.

```

#list constellations in load directory
print os.path
os.chdir("C:\Users\Brendan\Documents\Eclipse_Workspace\constellation\src\simulator\load-constellations")
#switch to input directory (may need to play with this for linux operation)

```

```

dirList=os.listdir(os.path.curdir)           #get list of files in input directory
print "Available Constellations:"
for i in range(0,len(dirList)):             #show user a list of available files
    print i,",",dirList[i]
#get user input to choose constellation to load, validate input
choice =-1
while (choice<0) or (choice>=len(dirList)):
    print "Please enter choice [ 0 -",len(dirList)-1,"]:"
    instr = raw_input()                     #get keyboard input
    try:
        choice = int(instr)                 #try to cast as integer
        if (choice<0) or (choice>=len(dirList)):#check range of integer
            print "Out of acceptable range."
    except ValueError:                       #if input NOT integer, loop back
        print "Enter integer values only, please."
        choice = -1

filename = dirList[choice]                  #take filename according to user selection
print "Loading from file:",filename

#read saved constellation file
fileread = open(filename,"r")              #open input file
rawcon = []
line = fileread.readline()                 #read first line
while line!="":                             #continue until end of file
    rawcon.append(line[0:len(line)-1])       #store current line
    line = fileread.readline()              #read next line

#parse raw file data into constellation structure
con = []
for s in rawcon:                             #for each raw string...
    p=0.0
    x=0.0
    y=0.0
    c1=0
    c2=0
    for c in range(0,len(s)):                #step through the raw string
        if s[c]==',' and c1!=0:              #tokenize with commas
            c2=c
        if s[c]==',' and c1==0:
            c1=c
    p = float(s[0:c1])                       #set (p,x,y) from string sections
    x = float(s[c1+1:c2])
    y = float(s[c2+1:len(s)])
    print "p=",p, " x=",x, " y=",y
    con.append((p,x,y))                      #add point to constellation

```

B.2.2 Running the Simulations

The following code is used for the BER simulations (it also reports SER performance) for the constellation “con” being tested over the range of SNR set in the code.

```

lencon = len(con)
conrange = range(0,lencon)

Es = getEnergy(con) #get energy form constellation. Should be 1.0 in all our tests.
print "Es =",Es
Eb = Es / math.ceil(math.log(lencon,2))
print "Eb =",Eb
minSNR = 0.0
maxSNR = 12.0                                     #set SNR range to test over, and step size

```

```

stepSNR = 1.0

numsym = 10000000          #number of symbols to simulate at each SNR step
print "Testing performance with",numsym,"points."
curSNR = minSNR

while curSNR <= maxSNR:
    No = Eb / (10.0**(curSNR/10.0))    #step up through reasonable SNR values
    print "SNR =",curSNR," No=",No     #set noise energy for this SNR
    errorcount = 0                    #reset count of errors
    symcount=0                        #reset number of symbols for this SNR test
    while symcount<numsym:
        rnum = random.random()         #loops through numsym generated symbols
        tsum = 0.0                     #random number
        pick=-1                        #cumulative temporary sum to pick
        while tsum<rnum:
            pick+=1                    #loop through con until cumulative prob. hits random number
            tsum+=con[pick][0]
            pass
        #print pick                    #pick is the selected symbol
        xsent = con[pick][1]           #set target x,y of transmitted pt
        ysent = con[pick][2]
        xrec = xsent + random.gauss(0,math.sqrt(No/2.0)) #add noise during transmission
        yrec = ysent + random.gauss(0,math.sqrt(No/2.0)) #No/2 because of 2 dimensions
        #decode the received x,y to a symbol
        #rec = decodeML(con,xrec,yrec) #decode received with ML
        rec = decodeMAP(con,No,xrec,yrec) #decode with MAP
        if rec!=pick: #if decode does not match sent, error
            errorcount+=1
        symcount+=1                    # go to next test symbol
    print "Error rate =",((1.0*errorcount)/numsym) #print error rate at this SNR
    curSNR+=stepSNR                   #step up to next SNR
    pass

```

B.2.3 Tandem Scheme

These two bits of code were used to test the performance of the various tandem scheme codes used. The main simulation code is shown first, followed by the code used in the soft Viterbi decoder.

```

countoverallerrorruns = 0
totaloverallerrorruns = 0

SNR = float(startSNR)
while SNR<=stopSNR:
    No = Eb / (10.0**(SNR/10.0))
    averageBER = 0
    totalcorrect = 0
    trynum = 0
    counterrorruns = 0
    totalerrorruns = 0
    while trynum<trials:    #loop through number of trials indicated
        #print "Trial #",trynum
        #generate random message according to source distribution
        message = ""       #initialize message string
        i=0
        while i<N:        #loop through message bits
            rnum = random.random()
            if rnum>p:
                message = message + "1"

```

```

else:
    message = message + "0"
    i+=1
#print "Message:",message

#compress (lossless) message with Huffman encoder
encodedmessage=""
byte = 0
while byte*nb <N:                                #loop through message one symbol at a time
    word = message[byte*nb:(byte+1)*nb]          #get next source symbol of nb bits
    encodedmessage=encodedmessage+huffEncode[word] #encode using huffman code
    byte+=1                                       #increment counter
#print "Encoded:",encodedmessage

# #check statistics of encoded message
# l=0
# countzero=0
# while l<len(encodedmessage):
#     if encodedmessage[l]=="0":
#         countzero+=1
#     l+=1
# p0enc = 1.0*countzero/len(encodedmessage)
# print "Compressed message has p=",p0enc

#encode compressed message with convolution encoder
encodedmessage2 = "00"+encodedmessage #init to state 00

convmessage=""
j=0
while j+2<len(encodedmessage2):
    convinput=encodedmessage2[j:j+K]
    bit1=int(convinput[0])*int(G1[2]) + int(convinput[1])*int(G1[1]) + int(convinput[2])*int(G1[0])
    if bit1==0 or bit1==2:
        convmessage=convmessage+"0"
    else:
        convmessage=convmessage+"1"
    bit2=int(convinput[0])*int(G2[2]) + int(convinput[1])*int(G2[1]) + int(convinput[2])*int(G2[0])
    if bit2==0 or bit2==2:
        convmessage=convmessage+"0"
    else:
        convmessage=convmessage+"1"
    j+=1
#print "Convo'd:",convmessage

#"transmit" the convolution encoded message over BPSK channel with AWGN
received = []
r = 0
while r<len(convmessage):
    if convmessage[r]=="0":
        r1 = -1.0 + random.gauss(0,math.sqrt(No/2))
    else:
        r1 = 1.0 + random.gauss(0,math.sqrt(No/2))
    if convmessage[r+1]=="0":
        r2 = -1.0 + random.gauss(0,math.sqrt(No/2))
    else:
        r2 = 1.0 + random.gauss(0,math.sqrt(No/2))
    received.append((r1,r2))
    r+=2
#print "Received:",received

probencmessage = viterbi.softViterbi(received)
#print "Viterbi:",probencmessage

#decode output of Viterbi algorithm with Huffman decode

```

```

decodedmessage = ""
j=0
flagerrorrrun = 0
errorrunstart = 0
while j<len(probencmessage):
    if probencmessage[j]!=encodedmessage[j]:
        errorrunstart=int(j)
        flagerrorrrun=1
        j=len(probencmessage)
        #counterrorrruns+=1
        #totalerrorrruns+=len(message)-len(decodedmessage)
        pass
    j+=1

starthd = 0
endhd = starthd+1
while starthd<len(probencmessage) and (endhd-starthd)<11:
    if flagerrorrrun==1 and endhd > errorrunstart:
        counterrorrruns+=1
        totalerrorrruns+=len(message)-len(decodedmessage)
        flagerrorrrun=0
    cword = probencmessage[starthd:endhd]
    if cword in huffDecode:
        decodedmessage = decodedmessage + huffDecode[cword]
        starthd = endhd
        endhd = starthd+1
    else:
        endhd+=1
#   if endhd-starthd==11:
#       #print "End of message is lost to noise corruption"
#       counterrorrruns+=1
#       totalerrorrruns+=len(message)-len(decodedmessage)
#       pass
#   #print "Decoded:",decodedmessage

#get BER of this trial
i=0
trialcorrect = 0
#startrunfailure = 0
while i<len(decodedmessage) and i<len(message):
    if decodedmessage[i]==message[i]:
        trialcorrect+=1
        #startrunfailure+=1
    i+=1
trialber = 1.0 - (1.0*trialcorrect/len(message))
#print "Trial BER =",trialber
#if startrunfailure<len()

totalcorrect +=trialcorrect
averageBER +=trialber

trynum+=1      #increment trial counter

averageBER = 1.0*averageBER/trials
#print "For SNR ",SNR," , Average BER =",averageBER #, " or ",1.0-(1.0*totalcorrect/(N*trials))
#using totalcorrect not necessary, adding trialber is accurate
print "SNR = ",SNR
print averageBER
if counterrorrruns>0:
    print "Average block error length =",1.0*totalerrorrruns/counterrorrruns
    print "(or, ",(1.0*totalerrorrruns)/(trials*1.0),")"
    pass
else: print "No run errors"
totaloverallerrorrruns+=totalerrorrruns

```

```

countoverallerrorruns+=counterrorruns
SNR+=stepSNR
if countoverallerrorruns>0:
    print "Overall average block error length =",1.0*totaloverallerrorruns/countoverallerrorruns

```

The input to the soft Viterbi decoder below is simply a list of observed voltages from the channel after being disturbed by noise.

```

def softViterbi(obs):
    states=[0,1,2,3]
    possible_preds = [ #[target][source]
        [0,1],
        [2,3],
        [0,1],
        [2,3]
    ]

    #[source][target] expected voltages to be observed given transition from source to target
    expected_obs = [
        [(-1.0,-1.0), (None,None), (1.0,1.0), (None,None)],
        [(1.0,1.0), (None,None), (-1.0,-1.0), (None,None)],
        [(None,None), (-1.0,1.0), (None,None), (1.0,-1.0)],
        [(None,None), (1.0,-1.0), (None,None), (-1.0,1.0)]
    ]

    transition_bits = [ #[from][to]
        ["0","","1",""],
        ["0","","1",""],
        [","0","","1"],
        [","0","","1"],
    ]

    pathmetrics =[      #know initial state is 0 (pad the message with two 0's to get this)
        [0],
        [1000],
        [1000],
        [1000]
    ]

    predecessors = [
        [0],
        [0],
        [0],
        [0],
    ]

    time=1
    while time<len(obs)+1:
        for s in states:
            possPMs = []
            possPMpred = []
            for posspred in possible_preds[s]:
                possPMpred.append(posspred)
                possPMs.append(pathmetrics[posspred][time-1]
                    +(obs[time-1][0]-expected_obs[posspred][s][0])**2+(obs[time-1][1]-expected_obs[posspred][s][1])**2)
            if possPMs[0]<possPMs[1]:
                pathmetrics[s].append(possPMs[0])
                predecessors[s].append(possPMpred[0])
            else:
                pathmetrics[s].append(possPMs[1])
                predecessors[s].append(possPMpred[1])

        time+=1

```

```
revmessage=""
trace = len(pathmetrics[0])-1
curstate = 0
minPM = 100000000
for s in states:
    if pathmetrics[s][trace]<minPM:
        curstate=s
        minPM = pathmetrics[s][trace]

while trace > 0:
    prevstate = predecessors[curstate][trace]
    revmessage = transition_bits[prevstate][curstate]+revmessage
    curstate = int(prevstate)
    trace -= 1
#message=str(revmessage.reverse())
return revmessage
```