Unequal Error Protection Codes Based on Trellis Shaping

by

Kevin C. Yu

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1995

Author .....
Department of Electrical Engineering and Computer Science
May 26, 1995

Certified by .....
Mitchell D. Trott
Assistant Professor
Thesis Supervisor

Accepted by .....
F. R. Morgenthaler
Chairman, Departmental Committee on Graduate Students

Unequal Error Protection Codes Based on Trellis Shaping

by

Kevin C. Yu

Submitted to the Department of Electrical Engineering and Computer Science
on May 26, 1995, in partial fulfillment of the
requirements for the degree of
Masters of Science in Computer Science and Engineering

Abstract

Transmitting information across communication channels results in errors in any practical system. Unequal error protection (UEP) arises when data can be classified by importance. Complexity and performance can be traded off at the expense of the less important data by using different degrees of coding to yield a lower probability of error for the more important information. For example, when transmitting an image, the coarse resolution image could be the important data and the fine resolution image could be the less important data. The UEP code could be designed so that a receiver with poor channel quality (low SNR) can still decode the coarse image reliably while a receiver with a higher quality channel (high SNR) could also decode the fine resolution image.

A new UEP coding scheme is developed based on trellis shaping and is compared to the codes of Wei [10] and Calderbank [1]. . The novel approach uses the coarse code to both encode the coarse data and to shape the inner, fine code. The shaping produces a cloud of fine codewords in the Voronoi regions surrounding each of the coarse codewords. In normal trellis shaping, the uncoded bits of the inner code are fixed by the shaping code. In the UEP trellis code, the shaped bits are reused when encoding the coarse information, thus creating an overlap in the bits. Reusing the shaped bits is allowed because the Voronoi regions of each coarse codeword are shrunk before the bits are reused.

The overlap of uncoded fine bits and coarse bits allows the UEP trellis code to achieve a higher rate fine code for a fixed constellation size. The expected improvement in the fine coding gain over Wei's codes, which encode the fine and coarse information independently, is 3dB per overlap bit per two dimensions. This coding gain, however, comes at the cost of decoding complexity.

The stack algorithm is used for decoding the UEP trellis code. The stack algorithm is modified to compensate for the decoding delay of the shaping code. The shaping decoder delay prevents the stack decoder from making a decision on the correct output symbol for each hypothesized branch. Thus, all possible symbols for every branch must be considered.

A simple UEP trellis code is implemented. Simulations confirm the performance

of the UEP trellis code as compared to a similar code constructed by Wei's methods. Wei's code achieves a higher coarse coding gain than the UEP code, but as expected, Wei's fine code performed slightly worse. The degraded coarse code performance of the UEP trellis code results from codewords which lie on a boundary between two adjacent Voronoi regions. Wei's codes are always arranged so this situation never occurs.

Thesis Supervisor: Mitchell D. Trott
Title: Assistant Professor

Acknowledgments

I would like to thank the many people who helped and encouraged me throughout the last three years and the course of this thesis. First off, I would like to thank my parents and my sisters for their support and advice. Their advice kept me going during the rough times and their support and confidence were greatly appreciated. Their encouragement was critical in my finishing this thesis.

I would also like to thank my friends at MIT for providing a glimpse of sanity (and sometimes, insanity) in such a dismal setting. In particular, I would like to thank James Sarvis for his patience with answering stupid questions which I was too lazy to think about. I would also like to thank Alex Wu for encouraging me to pursue my interests and always letting me talk his ear off when I had problems. Even though Alex was away for the last year of this thesis, his encouragement and friendship were valuable to me and often put many problems and situation into a larger, more stress free view. Rajeev Surati has also been a great friend throughout the last few years. He has provided countless hours of humor and guidance. His advice has helped me through some difficult times at the beginning of my graduate studies. He was always ready to give excellent advice when I got into trouble. In addition, I would like to thank all my officemates for putting up with all my junk and conversations when I didn't feel like working.

In addition to the friends at MIT, I would like to thank Rolando Cabrera, Alvin Chung, Benson Chung, and Anthony Vu for making my summer so much fun and encouraging me to pursue my academic goals. They provided advice on problems ranging from personal matters to career goals. Their friendships are truly special and are deeply cherished.

I would like to acknowledge the MIT Varsity hockey coaches, Joe Quinn, Tom Keller, and Bill McBrine. Hockey has provided a great sense of accomplishment for me while still offering many challenges. The coaches have provided me with the opportunity to pursue an activity which provided a nice change from the hectic, academic atmosphere at MIT.

Finally, I would like to acknowledge the single biggest influence on my life in the last year, Professor Mitchell Trott. His guidance and understanding have been essential to this thesis and my research. His humor and personality have created an atmosphere which has helped me finish this thesis while still enjoying my work. His countless hours answering questions and providing suggestions before the writing of this thesis were beyond the call of duty and when the thesis was finally written, his immaculate proofreading made the entire writing process much faster and more painless for me. I truly feel honored and lucky to have him as an advisor.

In addition to being an advisor, Mitchell Trott has also been a good friend. His friendship has been a constant help and source of encouragement. I am constantly impressed by his knowledge and personality. His friendship has been a huge positive factor on my life in the past year and I would like to sincerely thank him for all his efforts, both as an advisor and as a friend.

The research in this thesis was supported by NSF grant NCR-9314341.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The goal of communication systems is to transmit information across channels reliably. In many applications such as HDTV, a crucial design parameter is the ability to maintain communication with the receivers (TV sets) as the channel quality varies. Thus, a tradeoff between data rate and reliability can be made. A natural classification exists for many types of data, such as speech and image signals. This classification separates the data into more important data and less important data. By sacrificing some of the less important data, the more important data can be transmitted more reliably across the channel.In the HDTV example, a coarse-grained image could be the more important data and the fine detail could be the less important data. Receivers which have a less favorable channel, *i.e.*, more noise on the channel, can still receive a minimal resolution picture. Receivers which have a more favorable channel can receive the less important data which can be used to enhance the image quality. Unequal error protection deals with the problem of providing multiple levels of protection by providing more protection from channel noise for the more important data by trading off complexity and performance.

Throughout the development of unequal error protection (UEP) codes, the data will assumed to take on two forms: more important data and less important data. The more important data is transmitted with higher levels of coding, thus achieving a lower probability of error for a given signal-to-noise ratio (SNR). The less important data is transmitted with a less powerful code, thus achieving a higher probability of

error for a given SNR. In the schemes presented here, the transmit power is assumed to be much higher than the noise power. Multilevel signaling is therefore appropriate.

Binary signaling is capable of achieving capacity for low signal to noise ratios while multilevel signaling can achieve capacity for high signal to noise ratios. Low SNR UEP schemes using binary signaling have been widely studied, see for example [8, 6]. However, the only apparent research on the the high SNR case is that of Wei [10] and Calderbank [1].

A new UEP coding scheme was developed based on trellis shaping and will be described in the following chapters. The motivation for this scheme comes from Wei's codes. However, Wei's UEP codes use coding independently on the less and more important data. The UEP trellis codes that are proposed here overlap the less and more important codes using the more important code to shape the less important code. The overlap allows a higher rate code for a given constellation size. In other words, constellation expansion is not as severe since some bits overlap when encoding. The expected improvement in coding gain with respect to Wei's codes is 3dB per overlap bit per two dimensions. However, the increased coding gain comes at the cost of decoding complexity.

The remainder of the thesis is organized as follows. Chapter 2 provides a quick overview of some basic terminology and results which are needed for the development of the UEP trellis code. The topics discussed include lattice codes, trellis codes, fundamental regions of both lattice and trellis codes, and trellis shaping. Chapter 3 begins with a simple example of a UEP code analogous to uncoded QAM. The chapter then describes the construction of the UEP trellis code and provides an analysis of performance and interpretation of the coding scheme. The chapter concludes with an example of a simple UEP trellis code and a detailed analysis of the code's performance. Chapter 4 discusses many problems caused by the increased complexity in decoding the UEP trellis code. Decoding the UEP trellis code cannot be accomplished with the Viterbi Algorithm. Instead, a variation of the stack algorithm is used. The chapter discusses particular design choices used in implementing the stack algorithm and their effects on performance. Chapter 5 presents the Wei's and Calderbank's codes in detail

and compares their performance against the UEP trellis code's performance. Results from simulations of the UEP trellis code example at the end of Chapter 3 are analyzed and interpreted. Finally, Chapter 6 summarizes the performance of the UEP trellis code and provides some conclusions about the UEP trellis coding scheme. Future work is also discussed in this chapter.

Chapter 2

Background for Unequal Error Protection and Trellis Coding

This chapter begins with basic definitions and background results on trellis codes. Readers who desire a more detailed description of trellis codes and lattices should consult [5]. Section 2.1 addresses lattices and provides several definitions of basic quantities and terms. Section 2.2 addresses trellis codes and their behavior. Finally, Section 2.3 develops the idea of trellis shaping originally presented by Forney [3].

2.1 Lattices

A lattice Λ is an infinite set of discrete, regularly spaced points in R^N which forms an algebraic group under vector addition. Every lattice contains the origin. Each lattice point has a set of neighbors in the lattice. The set of distances associated with these neighbors is the distance profile of the lattice. The distance profile of every point is the same. In other words, the lattice looks the same from every point within the lattice. The minimum distance d_Λ of the lattice is the least distance in the distance profile.

A sublattice Λ' of a lattice Λ is a subset of points which is also a lattice. Notice, the sublattice Λ' must contain the origin and have infinite extent in order to also be a

lattice. For any lattice Λ and sublattice Λ', a coset $\Lambda' + a$ of Λ' in Λ is a set of points

$$\Lambda' + a = \{\lambda' + a \,|\lambda' \in \Lambda'\} \tag{2.1}$$

where $a \in \Lambda$. The element a is called a *coset representative* since it uniquely identifies the coset of Λ'.

A partition of a lattice is the set of all cosets of a sublattice in the original lattice. The partition induced by $\Lambda' \subseteq \Lambda$ is denoted Λ/Λ'. The number of distinct cosets of a partition is the *order* of the partition and is denoted $|\Lambda/\Lambda'|$. Let $[\Lambda/\Lambda']$ denote a set of distinct coset representatives. The size of this set, $|[\Lambda/\Lambda']|$, is the same as the order of the partition, $|\Lambda/\Lambda'|$.

An example of a lattice in R is the set of integers $\Lambda = Z$. A sublattice of Z is $\Lambda' = 2Z$, the set of even integers. The cosets of $2Z$ are $2Z$ and $2Z + 1$, the sets of even and odd integers respectively. The order of the partition is $|\Lambda/\Lambda'| = 2$. One possible set of coset representative is $[\Lambda/\Lambda'] = \{0, 1\}$.

A lattice partition chain of length n, denoted $\Lambda/\Lambda'/\Lambda''/\cdots/\Lambda^{(n)}$, consists of a lattice Λ and a series of sublattices $\Lambda', \Lambda'', \ldots, \Lambda^{(n)}$. Each partition has order r, thus $|\Lambda/\Lambda'| = |\Lambda'/\Lambda''| = \ldots = |\Lambda^{(n-1)}/\Lambda^{(n)}| = r$. The only partition chains considered here have $r = 2$. In the case of $\Lambda = Z^2$, the integer lattice, squared distances double within each sublattice partition in the partition chain. An example of a lattice partition chain in R of length 4 is $Z/2Z/4Z/8Z$.

Every lattice has a fundamental region, $\mathcal{R}_F^\Lambda$ defined below.

Definition 1 (Fundamental Regions of Lattices) *A fundamental region of Λ is a region in R^N formed by taking one element from each coset of Λ in R^N.*

The fundamental region is not unique. The particular fundamental region, $\mathcal{R}_F^\Lambda$, used here is the parallelepiped formed by the basis vectors of the lattice and centered at the origin. Define $\mathcal{R}_F^\Lambda(x)$ to be the region $\mathcal{R}_F^\Lambda + x$ where $x \in \Lambda$. The regions $\mathcal{R}_F^\Lambda(x)$ $\forall x \in \Lambda$ are congruent by construction and tile R^N. The *fundamental volume* is the volume of $\mathcal{R}_F^\Lambda$.

Another region of interest in R^N is the Voronoi region.

Definition 2 (Voronoi Regions of Lattices) *A Voronoi region for a lattice point $x \in \Lambda$, $\mathcal{R}_V^{\Lambda}(x)$, is defined as the set of points in R^N which are closer to x than to any other lattice point $y \in \Lambda$. The boundary of the Voronoi region consists of points equidistant from several lattice points and the boundary points must be assigned to a Voronoi region by a tie-breaking decision rule.*

All Voronoi regions are also congruent. The Voronoi region of a lattice is an N-dimensional polytope where each face of the polytope is an $(N-1)$-dimensional planar boundary region between two different lattice points. The planar region is formed by the set of points equidistant from adjacent lattice points. Each face of the polytope intersects another face, forming an $(N-2)$-dimensional edge of the polytope. An arbitrary but consistent tie-breaking rule assigns all points of a polytope face to one of the nearest lattice points. A set of edges is also assigned to the same lattice point. From the congruency[1]of Voronoi regions under translations, the other polytope faces are uniquely assigned once this tie-breaking rule is specified for some set of polytope faces. Also notice that the Voronoi regions tile R^N since every point in R^N has nearest lattice point or is assigned by the tie-breaking rule.

2.2 Trellis Codes

A signal space code is a set of points in a high dimensional signal space. A trellis code Ω is a signal space code which uses a finite state machine to select points from a lattice Λ in R^N. Each point selected by Ω is a sequence of points from a lower dimensional space. For instance, if each point in a signal space consists of L symbols from R^N, then the resulting signal space is R^{LN} formed from the Cartesian product of the L symbols. The set of allowable sequences code Ω is *not* the full Cartesian product of symbols in R^N. Dependencies are introduced between successive symbols

[1]In general, for a geometrically uniform signal set or code, not all symmetries are translations. Congruence can not always be guaranteed in such cases because the tie breaking rule does not uniquely assign points on a boundary under the symmetry of reflection. However, for translations, which are the only symmetry which will require a tie-breaking rule in the UEP trellis code, the tie-breaking rule is unique for all points.

which disallow certain L-sequences. The lower density of points results in a larger minimum distance. Coding gain is achieved when the minimum distance increases faster than the fundamental volume [5]. Trellis codes are infinite dimensional codes since sequences are assumed to have infinite extent.

The dependencies between symbols are introduced through a finite state machine (FSM). The FSM of the trellis code has a certain rate defined as the number of input bits divided by the number of output bits. For a rate-k/n code, k bits are accepted at each time and n bits are output. For example, a rate-1/2 FSM takes one bit in and produces 2 bits out. Since $n > k$, the number of points selected by the FSM is 2^{n-k} times larger than an uncoded system. The n output bits of the FSM select the coset of $\Lambda^{(n)}$ from the length-n lattice partition chain in the following manner. The least significant bit of the n bits selects a coset representative of Λ' in Λ. The next bit selects the coset representative of Λ'' in Λ' and so forth until the most significant bit selects the coset representative of $\Lambda^{(n)}$ in $\Lambda^{(n-1)}$. Then, the vector sum of all the coset representatives selects the coset of $\Lambda^{(n)}$ in Λ. Additional uncoded bits can also be used in a trellis code. These bits do not enter the FSM. Instead, they select a point from the coset of $\Lambda^{(n)}$ which was selected by the coded n bits. The set of points selected by the trellis code falls into a region S called the shaping region of the trellis code. Equivalently, the constellation used by the trellis code at every time i consists of the points $\Lambda \bigcap S$.

The FSM is often represented in the form of a trellis where the states are connected by transitions (an example with 4 states is shown in Figure 3-7(a)). The transitions are labeled with the input which causes the transition and the coset representative of $\Lambda^{(n)}$ in Λ selected by the FSM. FSM's are often implemented with a convolutional encoder.

A code Ω is *geometrically uniform* if there exists a symmetry group $\Gamma(\Omega)$ which maps any codeword in Ω to any other codeword in Ω while preserving the code Ω. Define $\mathbf{\Phi}$ to be a subgroup of $\Gamma(\Omega)$ such that for all $\boldsymbol{x} \in \Omega$ and $\boldsymbol{x}' \in \Omega$, there exists exactly one $\boldsymbol{\phi} \in \mathbf{\Phi}$ such that $\boldsymbol{x}' = \boldsymbol{\phi}(\boldsymbol{x})$. Denote the specific mapping in $\mathbf{\Phi}$ which maps $\boldsymbol{x} \in \Omega$ to $\boldsymbol{x}' \in \Omega$ as $\boldsymbol{\phi}_{\boldsymbol{x} \mapsto \boldsymbol{x}'}(\cdot)$. All codes considered later for UEP trellis coding

are geometrically uniform. Geometric uniformity ignores the effects of the shaping region S. In general, shaped codes are almost never geometrically uniform.

The performance of a trellis code is governed by two factors, the increase in minimum distance, and the increase in the fundamental volume. The minimum distance of a trellis code Ω is the distance between the two closest points in Ω. An error event occurs when a decoder chooses a different codeword than the correct (transmitted) codeword.

The distance between two sequences (codewords) is the sum of the squared distances between the sequence elements. The two minimum distance codewords will differ in only a finite number of elements. Usually, the majority of the distance between the two codewords is contributed by the first and last differing sequence elements. For geometrically uniform codes, only one codeword designated as the correct codeword needs to be analyzed. The minimum distance of a trellis code is denoted d_Ω. The gain in squared minimum distance is $d^2_{\text{gain},\Omega} = d^2_\Omega / d^2_\Lambda$.

The second performance factor is the increase in the fundamental volume of a trellis code (or normalized redundancy) [5]. This increase is due to the increased number of bits used by the FSM to select points from the lattice Λ. For good codes, the distance gain of d^2_Ω exceeds the energy increase caused by redundancy.

The coding gain of a rate-k/n trellis code versus an uncoded square lattice (Z^2) with square shaping region is the ratio of the energy-normalized squared distance of the trellis code to the energy-normalized squared distance of the uncoded system. This coding gain can be decomposed into contributions from 3 different factors. These factors are the gain induced by using the particular lattice Λ (γ_Λ), the gain induced by the shaping region (γ_S), and the gain induced by the redundancy of the FSM (γ_Ω) [5]. The overall coding gain of a trellis code over a code using a square shaping region on the integer lattice is

$$\gamma = \gamma_\Lambda \cdot \gamma_S \cdot \gamma_\Omega. \tag{2.2}$$

For a derivation of this result, see [5]. More precise definitions are given in Sec-

tion 3.4.3 for UEP trellis codes.

In order to define the fundamental regions and Voronoi regions for general trellis codes, an equivalence class[2] must be defined first. The equivalence class is defined by a particular set of mapping $\mathbf{\Phi} \subseteq \Gamma(\Omega)$ that is minimally sufficient to map any $\boldsymbol{x} \in \Omega$ to any other $\boldsymbol{x}' \in \Omega$.

Definition 3 (Equivalence Class) *The equivalence class of $\boldsymbol{x}$ is the set $\mathbf{\Phi}(\boldsymbol{x})$ of points which can be obtained by applying the mappings $\mathbf{\Phi}$ to $\boldsymbol{x}$. In other words, if $\boldsymbol{x}$ and $\boldsymbol{x}'$ are in the same equivalence class, there exists a mapping $\boldsymbol{\phi} \in \mathbf{\Phi}$ such that $\boldsymbol{x} = \boldsymbol{\phi}(\boldsymbol{x}')$.*

Using Definition 3, a fundamental region for geometrically uniform trellis code may defined as follows.

Definition 4 (Fundamental region of a trellis code) *A fundamental region, $\boldsymbol{\mathcal{R}}_F^{\Omega}$, of a code Ω is the set of points which is formed by taking one element from each equivalence class of R^{∞}.*

A fundamental region of most trellis codes can be constructed using the time-zero lattice. For all good trellis codes, the transitions leaving any state σ of the FSM selects a coset of Λ_0, the *time-zero lattice* of Ω [3]. The lattice Λ_0 forms the partition chain $\Lambda/\Lambda_0/\Lambda^{(n)}$ where $\Lambda^{(n)}$ is the last partition of Λ used by Ω. The order of the partitions are $|\Lambda/\Lambda_0| = 2^r$ and $|\Lambda_0/\Lambda^{(n)}| = 2^k$ where $r = n - k$. The coset of Λ_0 selected from state σ is the union of the cosets of $\Lambda^{(n)}$ selected by the 2^k transitions leaving σ. From Definition 1, $\boldsymbol{\mathcal{R}}_F^{\Lambda}(x)$ is the fundamental region of Λ centered at $x \in \Lambda$. Selecting a set of coset representatives of Λ_0 in Λ, $[\Lambda/\Lambda_0]$, a basic region $\boldsymbol{\mathcal{R}}_{\Lambda_0}$ is defined as

$$\boldsymbol{\mathcal{R}}_{\Lambda_0} = \bigcup_{a \in [\Lambda/\Lambda_0]} \boldsymbol{\mathcal{R}}_F^{\Lambda}(a).$$

[2] An equivalence class is not defined for points which lie on the boundary regions between lattice (or codewords) points. The definition does not hold under the symmetry of reflection for boundary points where the number of equivalence classes is fewer than the number of boundary points.

A fundamental region of the trellis code with the time-zero lattice Λ_0 is

$$\begin{aligned}\mathcal{R}_F^{\Omega} &= \cdots \times \mathcal{R}_{\Lambda_0} \times \mathcal{R}_{\Lambda_0} \times \mathcal{R}_{\Lambda_0} \times \cdots \\ &= (\mathcal{R}_{\Lambda_0})^{\infty}.\end{aligned}$$

Since fundamental regions are constructed from the full Cartesian product of $\mathcal{R}_{\Lambda_0}$ which, in turn, is composed of fundamental regions of Λ, boundary problems of $\mathcal{R}_F^{\Lambda}$ under certain symmetries are also inherited by $\mathcal{R}_{\Lambda_0}$. Except at the boundaries, no point in a fundamental region of Ω can be mapped to another point in the same fundamental region by any $\boldsymbol{\phi} \in \boldsymbol{\Phi}$ (see Forney [3] or Section 2.3).

Voronoi regions also exists for trellis codes. Since the constellation of Ω in R^{∞} is not the full Cartesian product of points in $\Lambda \in R^N$, Voronoi regions of the trellis code are *not* simply full Cartesian product of lattice Voronoi regions. For any sequence $\boldsymbol{x}$, define $\boldsymbol{x}_L$ to be the $2L+1$ length sequence $\boldsymbol{x}_L$ formed by taking the components of $\boldsymbol{x}$ from $-L$ to L.

Definition 5 (Voronoi region of a trellis code) *Let $\boldsymbol{x}$ be any codeword in Ω and denote the Voronoi region of $\boldsymbol{x}$ as $\mathcal{R}_V^{\Omega}(\boldsymbol{x})$. Let $\boldsymbol{x}'$ be any sequence in R^{∞}. Define $N_L(\boldsymbol{x}')$ to be the nearest $2L+1$ length neighbor of $\boldsymbol{x}'$ in Ω,* i.e.,

$$\|N_L(\boldsymbol{x}') - \boldsymbol{x}'_L\|^2 \leq \|\boldsymbol{y} - \boldsymbol{x}'_L\|^2 \quad \forall\, \boldsymbol{y} \in \Omega. \tag{2.3}$$

Then, $\boldsymbol{x}' \in \mathcal{R}_V^{\Omega}(\boldsymbol{x})$ if $\forall L\ \exists L' > L$ such that $[N_{L'}(\boldsymbol{x}')]_L = \boldsymbol{x}_L$.

Intuitively stated, $\mathcal{R}_V^{\Omega}(\boldsymbol{x})$ is the set of sequences which will be decoded to $\boldsymbol{x}$ by a maximum likelihood sequence decoder.

Voronoi regions are also fundamental regions of Ω since no non-identity mapping $\boldsymbol{\phi} \in \boldsymbol{\Phi}$ can map any point within a Voronoi region to another point in the same Voronoi region[3]. Since the Voronoi region is a fundamental region, all the properties

[3]If there exists a map $\boldsymbol{\phi}_{\boldsymbol{x}\mapsto\boldsymbol{y}} \in \boldsymbol{\Phi}$, where $\boldsymbol{x} \in \Omega$ and $\boldsymbol{y} \in \Omega$, which also maps $\boldsymbol{x}_1 \in \mathcal{R}_V^{\Omega}(\boldsymbol{x})$ to $\boldsymbol{x}_2 \in \mathcal{R}_V^{\Omega}(\boldsymbol{x})$, then if $\boldsymbol{x}_1$ decodes to $\boldsymbol{x}$, $\boldsymbol{x}_2$ must decode to $\boldsymbol{y}$ (and vice versa), thus $\boldsymbol{x}_2 \notin \mathcal{R}_V^{\Omega}(\boldsymbol{x})$, a contradiction.

of the fundamental regions apply. Points on the boundary of a Voronoi region are also not well defined under certain symmetries such as reflection. However, decision rules can be made if only certain "good" symmetries are used such as translations. Voronoi regions of Ω are also congruent and tile all of R^∞.

2.3 Trellis Shaping

The major results from trellis shaping needed for UEP trellis coding are outlined in this section. For a more detailed and in-depth treatment of shaping, the reader should see [3].

The motivation for trellis shaping begins by considering any channel code, Ω_c. The code Ω_c selects points from a constellation $\mathcal{C}_c = \Lambda_c \cap S$ at every time i. The coding gain for any such code can be decomposed into the three contributions outlined in Section 2.2. The gain which comes from the shaping region S arises because the ideal distribution on the constellation for a power-limited additive white Gaussian (AWG) noise channel is Gaussian [4]. The objective of shaping is to approach this distribution and the ultimate shaping gain over uniform square shaping is 1.53dB.

The idea of trellis shaping arises for signal space codes. Shaping is done on sequences of infinite dimension rather than finite dimensional points and the shaping region is the Voronoi region of the shaping code. In other words, trellis shaping is a technique for converting a sequence of i.i.d. random variables uniformly distributed over some region into a sequence which approximates Gaussian random variables over a slightly expanded region.

The basic setup for a trellis shaping code uses a lattice partition chain $\Lambda_c/\Lambda_c'/\Lambda_s/\Lambda_s'$, a geometrically uniform inner channel code Ω_c which selects points from Λ_c, and a shaping code Ω_s which selects points from Λ_s. At each time i, the code Ω_c takes k_c bits in and produces n_c bits to select a coset of Λ_c' from the partition Λ_c/Λ_c'. Then, k_u bits select a point from within the coset of Λ_c'. The k_u bits are referred to as the uncoded bits and these bits are the ones affected by the shaping code. A second code, the shaping code Ω_s, takes k_s bits in and produces n_s bits to select a coset of

Λ'_s from the partition Λ_s/Λ'_s. The orders of the lattice partitions are $|\Lambda_c/\Lambda'_c| = 2^{n_c}$, $|\Lambda'_c/\Lambda_s| = 2^{k_u}$, and $|\Lambda_s/\Lambda'_s| = 2^{n_s}$.

Trellis shaping of general geometrically uniform codes imposes an additional constraint on the set of mappings $\mathbf{\Phi}$ of the trellis code Ω_s as defined in Section 2.2. Namely, for all $\boldsymbol{x} \in \Omega_c$ and all $\boldsymbol{\phi} \in \mathbf{\Phi}$, $\boldsymbol{y} = \boldsymbol{\phi}(\boldsymbol{x}) \in \Omega_c$. Mappings which do not preserve the channel code Ω_c cannot be used for shaping. Recall that the set $\mathbf{\Phi}$ must contain exactly one mapping $\boldsymbol{\phi}_{\boldsymbol{x}\mapsto\boldsymbol{x}'}$ for each pair $\boldsymbol{x}, \boldsymbol{x}' \in \Omega_s$.

A specific mapping sequence which preserves the channel code can be constructed by mapping each component of the sequence independently. The componentwise mapping ϕ_i must be restricted to affect the "uncoded bits" of the trellis code. In other words, if $x \in \Lambda'_c + a$ for some $a \in [\Lambda_c/\Lambda'_c]$ and $x' = \phi_{x\mapsto x'}(x)$, then $x' \in \Lambda'_c + a$. Since the lattices are nested, $\Lambda_s \subseteq \Lambda_c$, the resulting point is still a valid channel symbol from Λ_c. In general, finding a mapping function which works for each element of a sequence will ensure the sequence also maps properly.

Let $\boldsymbol{e}$ be a codeword from Ω_c and call $\boldsymbol{e}$ the base sequence. Any codeword $\boldsymbol{c} \in \Omega_c$ can be decomposed into two components as follows. For every codeword $\boldsymbol{c} \in \Omega_c$, there exists a unique $\boldsymbol{c}_s \in \Omega_s$ and a unique point $\boldsymbol{c}_c \in \boldsymbol{\mathcal{R}}_V^{\Omega_s}(\boldsymbol{e})$ such that $\boldsymbol{c} = \boldsymbol{\phi}_{\boldsymbol{e}\mapsto\boldsymbol{c}_s}(\boldsymbol{c}_c)$. Similarly, an inverse mapping also exists such that $\boldsymbol{\phi}^{-1}_{\boldsymbol{e}\mapsto\boldsymbol{c}_s}(\boldsymbol{c}) = \boldsymbol{\phi}_{\boldsymbol{c}_s\mapsto\boldsymbol{e}}(\boldsymbol{c}) = \boldsymbol{c}_c$. By construction, every codeword consists of a component $\boldsymbol{c}_c$ which is not and a component $\boldsymbol{c}_s$ which *is* affected by the mappings. The input bits encoded to the codeword $\boldsymbol{c}$ must select a unique component $\boldsymbol{c}_c$, otherwise, shaping the codeword $\boldsymbol{c}$ will result in a non-unique mapping of input bits to shaped codewords. Stated differently, the component $\boldsymbol{c}_c$ of $\boldsymbol{c}$ should be unique for every $\boldsymbol{c} \in \Omega_c$.

Forney assumes the decomposition of $\boldsymbol{c} \in \Omega_c$ is unique in both components $\boldsymbol{c}_s$ and $\boldsymbol{c}_c$. However, two different codewords $\boldsymbol{c}^{(1)} \in \Omega_c$ and $\boldsymbol{c}^{(2)} \in \Omega_c$, when decomposed, might agree in the components $\boldsymbol{c}_c^{(1)}$ and $\boldsymbol{c}_c^{(2)}$. Luckily, for all good trellis codes, the probability of sequences which have non-uniquely specified components $\boldsymbol{c}_c$ approaches zero as the length of the sequences approaches infinity.

For the shaping codes used by Forney [3], the mappings $\{\phi_i\}$ are translations and

the codes are linear. Thus, any codeword $\boldsymbol{c}$ of Ω_c can be decomposed into

$$\boldsymbol{c} = \boldsymbol{c}_c + \boldsymbol{c}_s.$$

Shaping of $\boldsymbol{c}$ is performed by applying a translation of $\boldsymbol{c}$ by $\boldsymbol{c}_s$ so that the final codeword is

$$\boldsymbol{c}_c = \boldsymbol{c} - \boldsymbol{c}_s.$$

Because $\Lambda_s \subseteq \Lambda_c$, shaping only affects "uncoded" bits and $\boldsymbol{c}_c$ can be formed by applying componentwise translations.

Chapter 3

A Trellis Coded Unequal Error Protection Scheme

Unequal error protection (UEP) codes such as those devised by Wei [10] and Calderbank [1] suffer from at least three shortcomings: their performance is difficult to predict, there is no general design algorithm, and the codes do not reach the qtheoretical limits on performance. Unlike Wei and Calderbank's methods, the unequal error protection scheme described in this chapter was devised by considering the interaction between the inner and outer codes in high dimensional spaces. The proposed coding scheme is a structured method for creating UEP codes using trellis coding and trellis shaping. The performance of these codes is easily predicted. This scheme improves performance by using the outer code for two purposes: encoding the more important bits and shaping the inner code (used for the less important bits) by restricting the inner code points to the Voronoi regions of the outer code.

The goal of unequal error protection is to encode two separate streams of data with different levels of error protection. Two intermixed codes are used to transmit the data streams. Since each stream is encoded using a separate code, the streams may have different rates. To first order, the error probabilities of the codes are dependent on their minimum distances. Thus, to use a single codebook to transmit data with two different error rates, a codebook must exhibit a dual $d_{\min}$ property. This dual $d_{\min}$ property is suggested by Cover's description of broadcast channels [2]. In this

description, the codebook consists of clouds of points with intercloud distance d_c and intracloud distance d_f. The centers of the clouds can be considered "superpoints." These superpoints form the code Ω_c which is referred to as the coarse code since it encodes the more important data. The intracloud points form the code Ω_f which is called the fine code since it encodes the less important data. From the superpoint and cloud description, one visualizes the more important bits being encoded using the cloud locations. The less important data is encoded using the intracloud points. When $d_f < d_c$, the important data has a lower bit error rate than the less important bits. Noise which causes decoding errors within a cloud affects only the less important data. A larger amount of noise can be tolerated with the more important data.

Extending the dual $d_{\min}$ idea to sequences using trellis coding is the mechanism of the algorithm described in this chapter. Trellis codes combine two-dimensional symbols into a sequence which forms an infinite dimensional codeword. Wei's [10] and Calderbank's [1] codes make no effort to efficiently pack the codewords of Ω_f into the Voronoi region surrounding a superpoint sequence. By molding the fine point cloud to match the Voronoi regions of Ω_c, higher rates can be achieved by the fine code while maintaining the desired $d_{\min}$ pair. Section 3.1 begins with a simple example of an unequal error protection scheme. A generalization of the trellis coding UEP algorithm is given in Section 3.2. Section 3.3 provides insight and interprets the UEP algorithm and its performance. Section 3.4 discusses performance bounds for this algorithm. Section 3.5 presents a more complicated example using the trellis coding algorithm.

3.1 A Simple Example of UEP

In this section, some general notation used for the UEP codes is presented first. Next, a simple example is presented which will serve as our baseline for performance comparisons. In this example, the more important data stream is $\boldsymbol{x}^{(c)} = \{x_i^{(c)}\}$ and the less important data stream is $\boldsymbol{x}^{(f)} = \{x_i^{(f)}\}$. The two data streams are encoded separately and then combined into a single output stream in a final stage.

Each input at time i for the fine code consists of k_f bits. The sequence $\boldsymbol{x}^{(f)}$ is encoded using a trellis code Ω_f and a bounded subset of the lattice Λ_f or a translate of Λ_f. At each time i, the code Ω_f selects a point from Λ_f as a symbol. The minimum distance of Ω_f is d_{Ω_f}. The code Ω_f will be referred to as the fine code and the points chosen from Λ_f will be the fine code symbols. The finite set of points selected from Λ_f is called a constellation, $\mathcal{C}_f$.

Similarly, each input at time i for the coarse code consists of k_c bits. The sequence $\boldsymbol{x}^{(c)}$ is encoded using a trellis code Ω_c and a bounded subset of the lattice Λ_c or a translate of Λ_c. At each time i, the code Ω_c selects a point from Λ_c as a symbol. The minimum distance of Ω_c is d_{Ω_c}. The code Ω_c is referred to as the coarse code and the points selected from Λ_c are called superpoint symbols or coarse code symbols. The finite set of points selected from Λ_c forms a constellation, $\mathcal{C}_c$. The overall UEP code, Ω_{UEP}, has a constellation in R^N, $\mathcal{C}_{\text{UEP}}$. The code Ω_{UEP} has two minimum distances: the intracloud distance $d_f \leq d_{\Omega_f}$ and the intercloud distance $d_c \leq d_{\Omega_c}$. The distances d_c and d_f are derived from d_{Ω_c} and d_{Ω_f} (see Section 3.4.1).

In this example, each $x_i^{(c)}$ and $x_i^{(f)}$ consists of two bits ($k_c = 2, k_f = 2$). No coding is performed on either $\boldsymbol{x}^{(c)}$ or $\boldsymbol{x}^{(f)}$. Therefore, the constellations $\mathcal{C}_f$ and $\mathcal{C}_c$ each have 4 points. The fine lattice Λ_f is the lattice $Z^2 + (\frac{1}{2}, \frac{1}{2})$ while the coarse lattice Λ_c is $4Z^2$. Notice that Λ_c is a sublattice of Λ_f. The distances of each lattice are $d^2_{\Lambda_f} = 1$ for the fine lattice and $d^2_{\Lambda_c} = 4$ for the coarse lattice. The constellations $\mathcal{C}_f$ and $\mathcal{C}_c$ are shown in Figures 3-1 and 3-2. In addition to denoting the fine code constellation, $\mathcal{C}_f(\cdot)$ will also denote the mapping from input bits to symbols in the constellation. Similarly, $\mathcal{C}_c(\cdot)$ will also denote the mapping from input bits to the symbols in the coarse code constellation.

Generalizing the example to M-QAM constellations, the number of coarse bits k_c and fine bits k_f per symbol can be any positive even integer. The fine constellation $\mathcal{C}_f$ has $M^{(f)} = 2^{k_f}$ points and the coarse constellation $\mathcal{C}_c$ has $M^{(c)} = 2^{k_c}$ points. The fine lattice Λ_f remains $Z^2 + (\frac{1}{2}, \frac{1}{2})$, and Λ_c is still a sublattice of Λ_f, namely $2^{k_f}\Lambda_f$. Therefore,

$$\Lambda_c = 2^{k_f}\Lambda_f = 2^{k_f}Z^2 + (\frac{1}{2}, \frac{1}{2}).$$

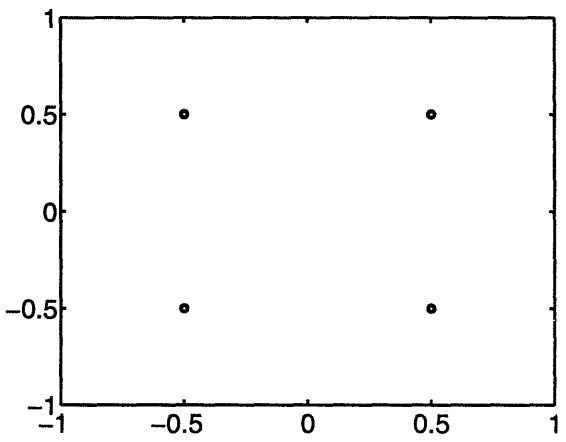


Figure 3-1: $\mathcal{C}_f$ for $k_f = 2$ — 4-QAM

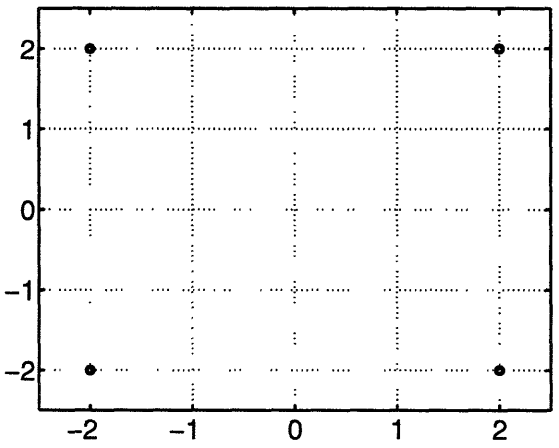


Figure 3-2: $\mathcal{C}_c$ for $k_c = 2$ — 4-QAM

The fine symbol lattice has minimum distance $d^2_{\Lambda_f} = 1$ and the coarse symbol lattice has minimum distance $d^2_{\Lambda_c} = 2^{k_f}$. Again, no coding is performed on either the fine or the coarse code sequences, hence $d^2_{\Omega_c} = d^2_{\Lambda_c}$ and $d^2_{\Omega_f} = d^2_{\Lambda_f}$.

To create a cloud of points around each of the superpoints, a scaled copy of $\mathcal{C}_f$ is centered at each point of $\mathcal{C}_c$. Associated with each point y of $\mathcal{C}_c$ is a distance preserving function $\lambda_y(\cdot)$ which maps the origin to y. The set $\lambda_y(\alpha\mathcal{C}_f)$ is a cloud of points around superpoint y. The factor $0 \leq \alpha \leq 1$ shrinks the fine constellation before it is moved. The intercloud distance is $d_f = \alpha d_{\Omega_f}$. The shrinking factor, α, also affects d_c. The mapping creates a new constellation $\mathcal{C}_{\text{UEP}} = \bigcup_{y\in\mathcal{C}_c} \lambda_y(\alpha\mathcal{C}_f)$ shown in Figure 3-3 for the $k_f = k_c = 2$ case.

The encoding process begins by mapping the important bits, $x_i^{(f)}$, at time i to points in the fine constellation $\mathcal{C}_f$ to form $y_i^{(f)} = \mathcal{C}_f(x_i^{(f)})$. Next, $x_i^{(c)}$ is encoded using $\mathcal{C}_c(\cdot)$ to form $y_i^{(c)}$. The final encoded point w_i is

$$w_i = \lambda_{y_i^{(c)}}(\alpha \cdot y_i^{(f)})$$

where the mapping function $\lambda_{y_i^{(c)}}(\cdot)$ is translation:

$$\lambda_{y_i^{(c)}}(x) = x + y_i^{(c)}. \tag{3.1}$$

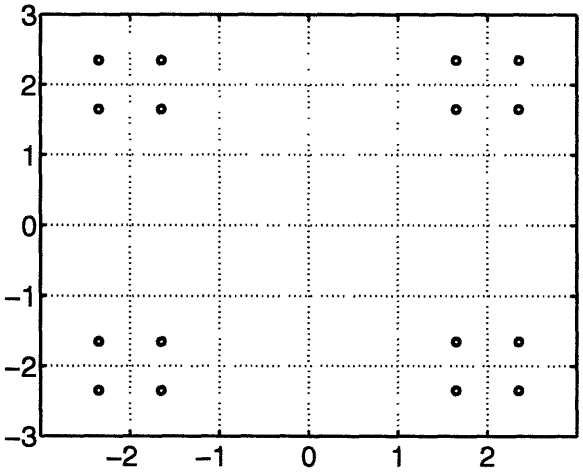


Figure 3-3: $\mathcal{C}$ constellation for $\alpha = 0.7$

Decoding the sequence $\{w_i\}$ can be done separately for the fine and coarse code[1]. Define $\tilde{\boldsymbol{w}}$ to be the received sequence

$$\tilde{w}_i = w_i + n_i,$$

where n_i is white Gaussian noise. Minimum distance decoding to the constellation $\mathcal{C}_c$ is used to recover the coarse symbol first. This decoding yields $\hat{y}_i^{(c)} = \mathcal{C}_c(\hat{x}_i^{(c)})$. Next, $\hat{y}_i^{(c)}$ is subtracted from $\tilde{w}_i$ to form a new point, $\tilde{w}_i' = \lambda_{\hat{y}_i^{(c)}}^{-1}(\tilde{w}_i)$. Minimum distance decoding of $\tilde{w}_i'$ to $\alpha\mathcal{C}_f$ gives the estimated fine code point, $\hat{y}_i^{(f)} = \mathcal{C}_f(\hat{x}_i^{(f)})$.

An alternative view of decoding is to recover $x_i^{(c)}$ and $x_i^{(f)}$ simultaneously by decoding $\tilde{w}_i$ to the nearest point in $s \in \mathcal{C}_{\text{UEP}}$. The specific $x_i^{(c)}$ and $x_i^{(f)}$ which yield s become the estimates $\hat{x}_i^{(c)}$ and $\hat{x}_i^{(f)}$. For the general M-QAM case, all $2^{(k_c+k_f)}$ inputs must be considered.

Performance measures for this code can be calculated directly for this scheme. The average power of any M-QAM constellation is

$$\overline{E}_{M-\text{QAM}} = \frac{2(M-1)d^2}{12} \tag{3.2}$$

[1]Independent fine and coarse decoding is optimal only when the decision region boundaries of the coarse code coincide with the decision region boundaries of the fine code. It appears that this occurs only in the special case of UEP QAM.

where d is the minimum distance of the M-QAM signal set. For the constellations used in this example, $M^{(c)} = 2^{k_c}$ and $M^{(f)} = 2^{k_f}$.

For the UEP QAM scheme, the peak energy of $\mathcal{C}_{\text{UEP}}$ is

$$\frac{1}{2} \cdot [(2^{\frac{k_c}{2}} - 1)d_{\Lambda_c} + (2^{\frac{k_f}{2}} - 1)d_f]^2, \tag{3.3}$$

where

$$d_f^2 = \alpha^2 d_{\Omega_f}^2 = \alpha^2 d_{\Lambda_f}^2 = \alpha^2 \tag{3.4}$$

is the minimum distance of Ω_f after scaling.

The average energy of $\mathcal{C}_{\text{UEP}}$ is the sum of the coarse constellation energy and the scaled fine constellation energy. (See Appendix A.1.) Thus,

$$\overline{E}_{\text{UEP}} = \overline{E}_{\mathcal{C}_c} + \overline{E}_{\alpha\mathcal{C}_f}. \tag{3.5}$$

Using Equation (3.2), the average energy of $\mathcal{C}_c$ with $k_c = 2$ and minimum distance d_{Λ_c} is

$$\overline{E}_{\mathcal{C}_c} = \frac{d_{\Lambda_c}^2}{2}.$$

The average energy of $\alpha\mathcal{C}_f$ with $n_f = 2$ and minimum distance α is

$$\overline{E}_{\alpha\mathcal{C}_f} = \frac{\alpha^2}{2}.$$

Thus, the average energy for the UEP constellation used in this example is

$$\overline{E}_{\mathcal{C}} = \frac{d_{\Lambda_c}^2 + \alpha^2}{2},$$

which is 2.245 for $\alpha = 0.7$ ($d_{\Lambda_c}^2 = 2^{k_f}$ in this case). From Equation (3.3), the peak energy of $\mathcal{C}_{\text{UEP}}$ is

$$\frac{d_{\Lambda_c}^2 + 2\alpha d_{\Lambda_c} + \alpha^2}{2} = 3.645$$

and thus, the Peak-to-Average Ratio (PAR) is 1.624 for $\alpha = 0.7$.

For the general M-QAM UEP code, the average energy of $\mathcal{C}_c = M^{(c)}$-QAM is

$$\overline{E}_{\mathcal{C}_c} = \frac{(2^{k_c} - 1)d^2_{\Lambda_c}}{6}. \tag{3.6}$$

The average energy of $\alpha\mathcal{C}_f$ is

$$\overline{E}_{\alpha\mathcal{C}_f} = \alpha^2 \frac{(2^{k_f} - 1)d^2_{\Lambda_f}}{6}. \tag{3.7}$$

Because $\Lambda_c = 2^{k_f}\Lambda_f$, Equation (3.6) becomes

$$\overline{E}_{\mathcal{C}_c} = \frac{(2^{k_c+k_f} - 2^{k_f})d^2_{\Lambda_f}}{6}. \tag{3.8}$$

Using Equations (3.5), (3.8), and (3.7), the average energy of $\mathcal{C}_{\text{UEP}}$ is

$$\overline{E}_{\mathcal{C}_{\text{UEP}}} = \frac{(2^{k_c+k_f} - \alpha^2) - (1 - \alpha^2)2^{k_f}}{6} d^2_{\Lambda_f}. \tag{3.9}$$

The minimum distance d_c is the distance separating two clouds. Notice, this distance is not the distance from cloud centers. Instead, d_c depends on α and is the distance between the two closest points in neighboring clouds:

$$d_c^2 = (d_{\Omega_c} - (2^{\frac{k_f}{2}} - 1)\alpha)^2. \tag{3.10}$$

Since a total of 4 bits per symbol are used during the encoding, the coding gain of the UEP QAM code is compared to a 16-QAM constellation which selects from the lattice $Z^2 + (\frac{1}{2}, \frac{1}{2})$. The average energy and minimum distance 16-QAM are

$$\overline{E}_{\text{16-QAM}} = \frac{5}{2} \tag{3.11}$$

and

$$d^2_{\text{16-QAM}} = 1. \tag{3.12}$$

Since the coarse and find codes have different levels of error protection, two dif-

ferent coding gains are calculated. The coding gain of the important bits is

$$\gamma_c = \frac{d_c^2}{d_{16\text{-QAM}}^2} \cdot \frac{\overline{E}_{16\text{-QAM}}}{\overline{E}_C} = \frac{(d_{\Omega_c} - \alpha)^2}{1} \cdot \frac{5}{d_{\Lambda_c}^2 + \alpha^2}. \tag{3.13}$$

The coding gain of the less important bits is

$$\gamma_f = \frac{d_f^2}{d_{16\text{-QAM}}^2} \cdot \frac{\overline{E}_{16\text{-QAM}}}{\overline{E}_C} = \frac{\alpha^2}{1} \cdot \frac{5}{d_{\Lambda_c}^2 + \alpha^2}. \tag{3.14}$$

Thus, for $\alpha = 0.7$, the coding gains are $\gamma_c = 2.75$dB and $\gamma_f = -2.64$dB.

For the general M-QAM UEP case, the minimum distance of Λ_c is $d_{\Lambda_c}^2 = 2^{k_f} d_{\Lambda_f}^2$. There are total of $k_c + k_f$ bits per symbol, thus the base simple QAM system is a $2^{k_c+k_f}$-QAM constellation. Since Z^2 is used for the baseline QAM lattice ($d_{\text{M-QAM}}^2 = 1$), Z^2 is also used for Λ_f. Thus, $d_{\Omega_f}^2 = 1$ and $d_{\Omega_c}^2 = 2^{k_f}$. Combining Equations (3.10), (3.2), (3.9), the coding gain for the coarse code is

$$\begin{aligned}
\gamma_c &= \frac{d_c^2}{d_{M-\text{QAM}}^2} \cdot \frac{\overline{E}_{M-\text{QAM}}}{\overline{E}_C} && (3.15)\\
&= \frac{(d_{\Omega_c} - (2^{\frac{k_f}{2}} - 1)\alpha)^2}{1} \cdot \frac{2^{k_c+k_f} - 1}{(2^{k_c+k_f} - \alpha^2) - (1-\alpha^2)2^{k_f}} \\
&= \frac{(2^{\frac{k_f}{2}} - (2^{\frac{k_f}{2}} - 1)\alpha)^2}{1} \cdot \frac{2^{k_c+k_f} - 1}{(2^{k_c+k_f} - \alpha^2) - (1-\alpha^2)2^{k_f}} \\
&= \frac{\left[2^{\frac{k_f}{2}}(1-\alpha) + \alpha\right]^2}{1} \cdot \frac{2^{k_c+k_f} - 1}{(2^{k_c+k_f} - \alpha^2) - (1-\alpha^2)2^{k_f}}. && (3.16)
\end{aligned}$$

The fine coding gain is

$$\begin{aligned}
\gamma_f &= \frac{d_f^2}{d_{M-\text{QAM}}^2} \cdot \frac{\overline{E}_{M-\text{QAM}}}{\overline{E}_C} && (3.17)\\
&= \frac{\alpha^2}{1} \cdot \frac{2^{k_c+k_f} - 1}{(2^{k_c+k_f} - \alpha^2) - (1-\alpha^2)2^{k_f}}.
\end{aligned}$$

The coding gain, γ_c, for the coarse code depends almost entirely upon k_f and α, and is nearly independent of k_c. The coding gain γ_f is very nearly equal to α^2. When $\alpha = 1$, the UEP QAM reduces to simple QAM as is evidenced by $\gamma_c = \gamma_f = 1$.

When $\alpha = 0$, the coarse coding gain is $\frac{2^{k_c+k_f-1}}{2^{k_c}} \approx 2^{k_f} \approx 3k_f$dB while the fine coding gain $\gamma_f = 0 = -\infty$dB. This corresponds to using an $M^{(c)}$-QAM constellation for the coarse code and disregarding the fine bits.

Now, consider increasing k_f while hold the energy of the UEP QAM and simple QAM constellations constant. This corresponds to keeping the area of the constellation constant. If the number of fine bits (k_f) is doubled, the squared distance between fine points must decrease a factor of 2. The distance of the coarse code remains the same since k_c is not changed. For the simple QAM constellation, k_c+k_f has increased by 2, so the minimum distance of the simple QAM constellation has also decreased by a factor of 2. Thus, the coding gain for the fine code remains constant since both fine code and base QAM distances are scaled equally and all energies are constant. The coding gain for the coarse code, however, increases since the coarse distance does not change, but the base distance decreases while the ratio of the energies remains constant. This argument suggests that increasing the number of fine bits can increase the coarse coding gain while not effecting the fine coding gain. This relationship is confirmed from Equations (3.15) and (3.17).

Increasing the number of less important bits to infinity ($k_f \rightarrow \infty$), the coarse coding gain γ_c tends to infinity while γ_f remains nearly constant at α^2. The maximum achievable coarse coding gain is controlled almost entirely by the number of fine bits k_f. This behavior continues to hold for the trellis coded schemes described in Section 3.2, and appears to hold for almost all classes of good UEP codes.

3.2 A General Description of a Trellis Coded UEP Algorithm

Extending the ideas presented in Section 3.1 to trellis coded UEP requires the introduction of infinite sequences and their associated signal spaces. Begin by considering an infinite length input sequence. This sequence is separated into two infinite length sequences: the more important bits, $\boldsymbol{x}^{(c)} = \{x_i^{(c)}\}$, and the less important bits,

$\boldsymbol{x}^{(f)} = \{x_i^{(f)}\}$. The sequence $\boldsymbol{x}^{(c)}$ is composed of k_c bits and the sequence $\boldsymbol{x}^{(f)}$ is composed of k_f bits at every time i. The two sequences $\boldsymbol{x}^{(c)}$ and $\boldsymbol{x}^{(f)}$ are encoded using two trellis codes Ω_c and Ω_f and the associated lattice partition chain $\Lambda_f/\Lambda_f'/\Lambda_c/\Lambda_c'$ to be described in Section 3.2.1. The codewords used in trellis coding are infinite length sequences. Since the input sequence is infinite length, the encoded sequence is also infinite. The idea of forming a "cloud" of fine code points (less important data) around a coarse code point (more important data) is still the objective of this algorithm. However, codeword spaces and lattices become infinite dimensional.

3.2.1 A Complete Description

The encoder

The basic block structure for the encoder is shown in Figure 3-4. The less important bits are encoded using a trellis code Ω_f with an associated lattice $\Lambda_f \subseteq R^N$. At each time i, the fine code Ω_f takes $k_f = k_f' + k_f^u$ bits as input. The k_f' bits are encoded using the FSM and the k_f^u bits are uncoded. The k_f' bits are encoded into n_f' bits which choose a coset of Λ_f' in Λ_f. The lattice Λ_f' is a sublattice of Λ_f of order $2^{n_f'}$. Each coset is selected by choosing a coset representative from $[\Lambda_f/\Lambda_f']$. The uncoded less important bits n_f^u select a specific point in the coset. A total of $n_f = n_f' + k_f^u$ bits select points from Λ_f. At each time i, only 2^{n_f} points in Λ_f are used for encoding. These points form a constellation $\mathcal{C}_f \subseteq R^N$. The notation $\Omega_f(\cdot)$ will be used to denoted the encoding of an input sequence to produce a codeword in Ω_f.

The more important bits are encoded using a trellis code Ω_c with the lattice $\Lambda_c \subseteq R^N$. At each time i, the coarse code Ω_c takes $k_c = k_c' + k_c^u$ bits as input. The k_c' bits are encoded using the FSM and the k_c^u bits are uncoded. The k_c' bits are encoded into n_c' bits which select a coset of Λ_c' in Λ_c. The lattice Λ_c' is a sublattice of Λ_c of order $2^{n_c'}$. Each coset is selected by picking a coset representative from $[\Lambda_c/\Lambda_c']$. The uncoded more important bits bits n_c^u select a specific point in the coset. A total of $n_c = n_c' + k_c^u$ bits select points from Λ_c. At each time i, only 2^{n_c} points in Λ_c are used for encoding. These points form a constellation $\mathcal{C}_c \subseteq R^N$. Any codeword $\boldsymbol{z} \in \Omega_c$ has

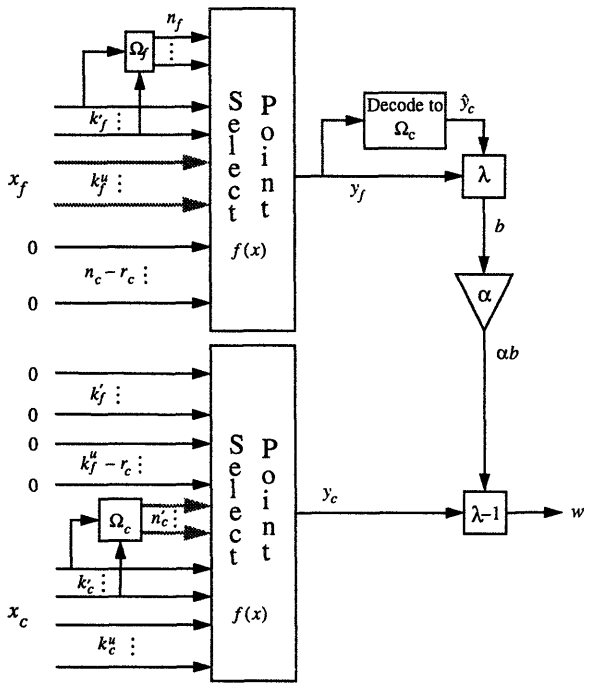


Figure 3-4: Block diagram of the UEP encoder

a Voronoi region in R^∞, denoted $\mathcal{R}_V^{\Omega_c}(\boldsymbol{z})$, centered at $\boldsymbol{z}$. The notation $\Omega_c(\cdot)$ will be used to denote the encoding of an input sequence to produce a codeword in Ω_c.

Typically, the lattices Λ_c and Λ_f are in R^2 or R^4. The codes Ω_c and Ω_f are assumed to be geometrically uniform. A codeword in Ω_f is a sequence of points from Λ_f and is referred to as a fine sequence. A codeword in Ω_c is a sequence of points from Λ_c and is referred to as a coarse sequence.

Scaled versions of the Voronoi regions of Ω_c are used as cloud regions (in R^∞) centered around every coarse sequence. Thus, the code Ω_c serves as both the coarse code for the more important bits and the shaping code for Ω_f. Shrinking the Voronoi regions of Ω_c creates a code with two different minimum distances, one for the important data (intercloud distance) and one for the less important data (intracloud distance).

Let $\boldsymbol{y}^{(f)} = \Omega_f(\boldsymbol{x}^{(f)})$ be the encoded sequence of $\boldsymbol{x}^{(f)}$. The point $\boldsymbol{y}^{(f)}$ lies in R^∞. The codewords of Ω_c also lie in R^∞. From Definition 5, $\mathcal{R}_V^{\Omega_c}(\cdot)$ completely tiles R^∞. Thus, $\boldsymbol{y}^{(f)} \in \mathcal{R}_V^{\Omega_c}(\hat{\boldsymbol{y}}^{(c)})$ for some unique $\hat{\boldsymbol{y}}^{(c)} \in \Omega_c$. The sequence $\hat{\boldsymbol{y}}^{(c)}$ is the coarse sequence whose Voronoi region contains $\boldsymbol{y}^{(f)}$. At every time i, the element $\hat{y}_i^{(c)}$ is a

member of a coset of Λ_c' in Λ_c identified by $a_i^{(c)} \in [\Lambda_c/\Lambda_c']$. The sequence $\boldsymbol{a}^{(c)} = \{a_i^{(c)}\}$ is the sequence of coset representatives associated with $\hat{\boldsymbol{y}}^{(c)}$.

The creation of a cloud structure begins by mapping $\boldsymbol{y}^{(f)}$ from $\mathcal{R}_V^{\Omega_c}(\hat{\boldsymbol{y}}^{(c)})$ to a base region centered at zero, $\mathcal{R}_V^{\Omega_c}(\mathbf{0})$, defined by some convenient base sequence $\boldsymbol{e}^0 \in \Omega_c$. Recall from Section 2.2 that the mapping $\phi_{\hat{\boldsymbol{y}}^{(c)} \mapsto \boldsymbol{e}^0}(\cdot)$ moves points from $\mathcal{R}_V^{\Omega_c}(\hat{\boldsymbol{y}}^{(c)})$ to $\mathcal{R}_V^{\Omega_c}(\boldsymbol{e}^0)$ while preserving Ω_c and Ω_f. Next, points in $\mathcal{R}_V^{\Omega_c}(\boldsymbol{e}^0)$ are moved to $\mathcal{R}_V^{\Omega_c}(\mathbf{0})$ using a translation of $\boldsymbol{e}^0$. Thus, the composite mapping $\boldsymbol{\lambda}(\cdot)$ which moves $\boldsymbol{y}^{(f)}$ from inside $\mathcal{R}_V^{\Omega_c}(\hat{\boldsymbol{y}}^{(c)})$ to a point $\boldsymbol{b}^{(f)}$ in the base region is defined by

$$\begin{aligned} \boldsymbol{b}^{(f)} &= \phi_{\hat{\boldsymbol{y}}^{(c)} \mapsto \boldsymbol{e}^0}(\boldsymbol{y}^{(f)}) - \boldsymbol{e}^0 \\ &= \boldsymbol{\lambda}_{\hat{\boldsymbol{y}}^{(c)}}(\boldsymbol{y}^{(f)}). \end{aligned}$$

The base region is then scaled by $0 \leq \alpha \leq 1$ and serves as the cloud region. The point $\boldsymbol{b}^{(f)} \in \mathcal{R}_V^{\Omega_c}(\mathbf{0})$ is scaled by α to yield $\alpha \boldsymbol{b}^{(f)}$.

Once the $\boldsymbol{y}^{(f)}$ is mapped to the base cloud region, a second mapping is applied to $\alpha \boldsymbol{b}^{(f)}$. This mapping moves points from the base region to the Voronoi region of $\boldsymbol{y}^{(c)} = \Omega_c(\boldsymbol{x}^{(c)})$. The sequence $\alpha \boldsymbol{b}^{(f)}$ is first moved to $\mathcal{R}_V^{\Omega_c}(\boldsymbol{e}^0)$ by a translate of $\boldsymbol{e}^0$. Then, the mapping $\phi_{\boldsymbol{e}^0 \mapsto \boldsymbol{y}^{(c)}}(\cdot)$ is used to move the scaled base sequence to a cloud region surrounding the coarse sequence $\boldsymbol{y}^{(c)}$. Thus, the output sequence $\boldsymbol{w}'$ formed from the composite mappings is

$$\begin{aligned} \boldsymbol{w}' &= \phi_{\boldsymbol{e}^0 \mapsto \boldsymbol{y}^{(c)}}(\alpha \boldsymbol{b}^{(f)} + \boldsymbol{e}^0) \\ &= \boldsymbol{\lambda}^{-1}_{\boldsymbol{y}^{(c)}}(\alpha \boldsymbol{b}^{(f)}). \end{aligned}$$

Define

$$\mathcal{R}_{V,\alpha}^{\Omega_c}(\boldsymbol{y}^{(c)}) = \phi_{\boldsymbol{e}^0 \mapsto \boldsymbol{y}^{(c)}}\left(\alpha \mathcal{R}_V^{\Omega_c}(\mathbf{0}) + \boldsymbol{e}^0\right)$$

to be the cloud region around the coarse sequence $\boldsymbol{y}^{(c)} \in \Omega_c$. By construction, the sequence $\boldsymbol{w}'$ lies in $\mathcal{R}_{V,\alpha}^{\Omega_c}(\boldsymbol{y}^{(c)}) \subseteq \mathcal{R}_V^{\Omega_c}(\boldsymbol{y}^{(c)})$.

The final step is to select a finite signal set. As with ordinary trellis codes, shaping and coding are decoupled. Simple finite dimensional shaping will be used here, but

more complex methods (*e.g.*, trellis shaping) can be used. Define a shaping region $\boldsymbol{\mathcal{R}}_s$ which will bound the constellation $\mathcal{C}_{\text{UEP}}$. The final output $\boldsymbol{w}$ is a sequence whose components $w_i \in \mathcal{C}_{\text{UEP}}$ are all contained within the shaping region. The shaping region $\boldsymbol{\mathcal{R}}_s$ must be a fundamental region of $\Lambda_c^{(n_c)}$, where $\Lambda_c^{(n_c)}$ is the sublattice of Λ_c used in the partition chain $\Lambda_f/\Lambda_f'/\Lambda_c/\Lambda_c'/\Lambda_c^{(n_c)}$. Furthermore, $\boldsymbol{\mathcal{R}}_s$ must be a finite union of fundamental regions of Λ_c. Choosing $\boldsymbol{\mathcal{R}}_s$ in this manner insures that translations by points in $\Lambda_c^{(n_c)}$ do not affect the code Ω_{UEP}. Because $\boldsymbol{\mathcal{R}}_s$ is a fundamental region, any point which lies outside $\boldsymbol{\mathcal{R}}_s$ can be moved inside by translating by a point in $\Lambda_c^{(n_c)}$. The translation can be implemented by a modulo operation on each sequence element, thus yielding the final sequence:

$$\boldsymbol{w} = \{w_i\} = \{w_i' \bmod \boldsymbol{\mathcal{R}}_s\}.$$

The Decoder

The encoded sequence $\boldsymbol{w}$ is transmitted across a channel. The channel is assumed to add white Gaussian noise. The received sequence is typically decoded using the Viterbi Algorithm (VA). Because the UEP trellis encoder is rather complex and has many states, a stack decoder is used instead. The stack decoder is a sequential decoder which encodes hypothesized input pairs $\hat{\boldsymbol{x}}^{(c)}$ and $\hat{\boldsymbol{x}}^{(f)}$ to yield codewords in Ω_{UEP}. Distance metrics between the received codeword and the hypothesized codewords are calculated. The minimum distance codeword is selected and the corresponding sequences $\hat{\boldsymbol{x}}^{(c)}$ and $\hat{\boldsymbol{x}}^{(f)}$ are the decoded input sequences.

The stack decoder is a metric-first search algorithm. For each iteration, the decoder has an ordered set of hypothesized input sequences of various lengths. Each sequence consists of inputs $\hat{x}_i^{(c)}$ and $\hat{x}_i^{(f)}$ up to some time t and the corresponding hypothesized codeword. The paths are kept in ascending order by a path metric. The best hypothesized input sequence is extended a single time unit to $t+1$. State information kept with each path allows this extension without the need to re-encode all past inputs (see Chapter 4).

For the stack algorithm, extending a path from time t to $t+1$ involves encoding

all $2^{k_c+k_f}$ pairs of $x_{i+1}^{(c)}$ and $x_{i+1}^{(f)}$ inputs. A metric is calculated for each extended sequence using

$$\text{metric} = \sum_{i=0}^{t+1}(e_i - R\sigma^2).$$

The term e_i is the metric between the encoder output and the received symbol at time i, R is the information rate per N dimensions, and σ^2 is a correction factor reflecting the expected metric gain per unit time [9]. For a Gaussian noise channel, σ^2 is the noise variance per N dimensions. The extended paths are added to the list of hypotheses. This process continues until the time of the top path is equal to the time of the last received sequence element.

The stack algorithm has a branching factor of $2^{k_c+k_f}$ at each step. If there are many input bits, the branching factor will be very large, thus requiring large amounts of storage and computation during decoding. To reduce storage at the expense of computation, a Fano algorithm could be used [7].

3.2.2 A Description of the Mapping Functions

The mapping functions from Section 3.2.1 are described in greater detail here. Specifically, the componentwise behavior with lattices is explored. Notice that the set of mapping functions $\{\boldsymbol{\lambda}\}$ is infinite since trellis codewords have infinite extent. In this section, it is shown that the mapping functions can be implemented using a finite set of mappings combined with translates, thus greatly simplifying the implementation of the maps.

Simplification of the mapping sequences begins by choosing a constant base sequence $\boldsymbol{e}^0 \in \Omega_c$ where the base region is region $\boldsymbol{\mathcal{R}}_V^{\Omega_c}(\boldsymbol{e}^0)$ translated to the origin. The sequence $\boldsymbol{e}^0 = \{\ldots, e^0, e^0, e^0, \ldots\} = \{e_i^0\}$ where each $e_i^0 = e^0 \in \Lambda_c$. Further restrict e^0 to be a member of $[\Lambda_c/\Lambda_c']$, *i.e.*, a coset representative. Therefore, e^0 identifies both the coset of Λ_c' and a point in Λ_c.

Define a partition-permuting mapping $\tilde{\phi}_{a\mapsto e^0}(\cdot)$ for each $a \in [\Lambda_c/\Lambda_c']$ which maps the point a to the point e^0 and permutes the cosets of Λ_c/Λ_c'. In particular, $\tilde{\phi}_{a\mapsto e^0}(\cdot)$ maps all points in the coset $\Lambda_c' + a$ to points in the coset $\Lambda_c' + e^0$. Since $[\Lambda_c/\Lambda_c']$ is a

finite set, the set of mappings is also finite.

Recall that $\boldsymbol{y}^{(f)} = \{y_i^{(f)}\}$ is decoded to some coarse sequence $\hat{\boldsymbol{y}}^{(c)} = \{y_i^{(c)}\}$ which has an associated coset representative sequence $\hat{\boldsymbol{a}}^{(c)} = \{\hat{a}_i^{(c)}\}$. Applying the mapping $\tilde{\phi}_{\hat{a}_i^{(c)} \mapsto e^0}(\cdot)$ to each component $\hat{y}_i^{(c)}$ results in a new coarse sequence $\tilde{\boldsymbol{y}} = \{\tilde{y}_i\}$ where $\tilde{y}_i \in \Lambda_c' + e^0$ but $\tilde{y}_i$ is not necessarily equal to e^0. A translation of $e^0 - \tilde{y}_i$ is needed on each component to make the two sequences identical.

Applying $\tilde{\phi}_{\hat{a}_i^{(c)} \mapsto e^0}(\cdot)$ to each $y_i^{(f)}$ moves $\boldsymbol{y}^{(f)}$ to the Voronoi region of $\tilde{\boldsymbol{y}}$. A translation of $\boldsymbol{e}^0 - \tilde{\boldsymbol{y}}$ is needed to move $\boldsymbol{y}^{(f)}$ to the Voronoi region of $\boldsymbol{e}^0$. Componentwise, this translation becomes

$$\boldsymbol{e}^0 - \tilde{\boldsymbol{y}} = \{e^0 - \tilde{\phi}_{\hat{a}_i^{(c)} \mapsto e^0}(\hat{y}_i^{(c)})\}.$$

The composite mapping function $\boldsymbol{\phi}_{\hat{\boldsymbol{y}}^{(c)} \mapsto \boldsymbol{e}^0}(\cdot)$ which moves the sequence $\hat{\boldsymbol{y}}^{(c)}$ to the sequence $\boldsymbol{e}^0$ becomes

$$\begin{aligned} \boldsymbol{\phi}_{\hat{\boldsymbol{y}}^{(c)} \mapsto \boldsymbol{e}^0}(\cdot) &= \{\phi_{\hat{y}_i^{(c)} \mapsto e^0}(\cdot)\} \\ &= \{\tilde{\phi}_{\hat{a}_i^{(c)} \mapsto e^0}(\cdot) + e^0 - \tilde{\phi}_{\hat{a}_i^{(c)} \mapsto e^0}(\hat{y}_i^{(c)})\}. \end{aligned}$$

The final forward mapping sequence $\boldsymbol{\lambda}_{\hat{\boldsymbol{y}}^{(c)}}(\cdot) = \boldsymbol{\phi}_{\hat{\boldsymbol{y}}^{(c)} \mapsto \boldsymbol{e}^0}(\cdot) - \boldsymbol{e}^0$ reduces to the componentwise mapping

$$\begin{aligned} \boldsymbol{\lambda}_{\hat{\boldsymbol{y}}^{(c)}}(\cdot) &= \{\lambda_{\hat{y}_i^{(c)}}(\cdot)\} \\ &= \{\phi_{\hat{y}_i^{(c)} \mapsto e^0}(\cdot) - e^0\} \\ &= \{\tilde{\phi}_{\hat{a}_i^{(c)} \mapsto e^0}(\cdot) - \tilde{\phi}_{\hat{a}_i^{(c)} \mapsto e^0}(\hat{y}_i^{(c)})\}. \end{aligned} \tag{3.18}$$

Encoding $\boldsymbol{x}^{(c)} = \{x_i^{(c)}\}$ yields $\boldsymbol{y}^{(c)} = \{y_i^{(c)}\}$ and the associated coset representative sequence $\boldsymbol{a}^{(c)} = \{a_i^{(c)}\}$. The second mapping $\boldsymbol{\lambda}^{-1}_{\boldsymbol{y}^{(c)}}(\cdot)$ which moves base sequences to the Voronoi region of $\boldsymbol{y}^{(c)}$ can also be constructed with the same finite set of componentwise mappings. The function $\tilde{\phi}_{e^0 \mapsto a_i^{(c)}}(\cdot) = \tilde{\phi}^{-1}_{a_i^{(c)} \mapsto e^0}(\cdot)$ moves the coset representative e^0 to the coset representative $a_i^{(c)}$, thus the map moves points from

$\Lambda_c' + e^0$ to $\Lambda_c' + a_i^{(c)}$. Again, the finite set $[\Lambda_c/\Lambda_c']$ guarantees a finite number of inverse mappings.

For any base sequence $\alpha\boldsymbol{b}$, the mapping becomes $\boldsymbol{\lambda}^{-1}_{\boldsymbol{y}^{(c)}}(\alpha\boldsymbol{b}) = \boldsymbol{\phi}_{\boldsymbol{e}^0 \mapsto \boldsymbol{y}^{(c)}}(\alpha\boldsymbol{b} + \boldsymbol{e}^0)$ where

$$(\alpha\boldsymbol{b} + \boldsymbol{e}^0) = \{\alpha b_i + e^0\}$$

and

$$\begin{aligned} \boldsymbol{\phi}_{\boldsymbol{e}^0 \mapsto \boldsymbol{y}^{(c)}}(\cdot) &= \{\phi_{e^0 \mapsto y_i^{(c)}}(\cdot)\} \\ &= \{\tilde{\phi}_{e^0 \mapsto a_i^{(c)}}(\cdot) + y_i^{(c)} - \tilde{\phi}_{e^0 \mapsto a_i^{(c)}}(e^0)\}. \end{aligned}$$

Thus, the second (inverse) mapping is

$$\begin{aligned} \boldsymbol{\lambda}^{-1}_{\boldsymbol{y}^{(c)}}(\alpha\boldsymbol{b}) &= \boldsymbol{\phi}_{\boldsymbol{e}^0 \mapsto \boldsymbol{y}^{(c)}}(\alpha\boldsymbol{b} + \boldsymbol{e}^0) \\ &= \{\phi_{e^0 \mapsto y_i^{(c)}}(\alpha b_i + e^0)\} \\ &= \{\tilde{\phi}_{e^0 \mapsto a_i^{(c)}}(\alpha b_i + e^0) + y_i^{(c)} - \tilde{\phi}_{e^0 \mapsto a_i^{(c)}}(e^0)\}. \end{aligned}$$

Thus, both the forward and inverse mappings can be constructed using translations and a finite set of componentwise mappings which move between cosets. Translations within a coset of Λ_c' does not destroy either code and, thus, the composite mapping properly moves sequences to other sequences.

3.2.3 A Description of the UEP Constellation

Since every codeword $\boldsymbol{w}' = \{w_i'\}$ of Ω_{UEP} is a sequence of points in R^N, the code Ω_{UEP} must be contained in the Cartesian product of a constellation $\mathcal{C}_{\text{UEP}} \subseteq R^N$ from which each w_i' is chosen, *i.e.*, $\Omega_{\text{UEP}} = (\cdots \mathcal{C}_{\text{UEP}} \times \mathcal{C}_{\text{UEP}} \times \mathcal{C}_{\text{UEP}} \cdots)$. Shaping $\mathcal{C}_f$ with the coarse code Ω_c and then scaling by α forms a base cloud constellation $\mathcal{C}_f^\alpha$. The shaped constellation $\mathcal{C}_f^\alpha$ generally contains more points than the original unshaped constellation $\mathcal{C}_f$. However, for the UEP QAM example of Section 3.1, no shaping is present so the base constellation is merely $\alpha\mathcal{C}_f$.

The constellation $\mathcal{C}_{\text{UEP}}$ is formed by mapping points in the base cloud constellation $\mathcal{C}_f^{\alpha} \subseteq R^N$ around every superpoint in $\mathcal{C}_c$. Using the results from Section 2.3, $\mathcal{C}_f$ must lie within the basic region $\boldsymbol{\mathcal{R}}_{\Lambda_0}$ of Λ_0 of Ω_c. Given Ω_f and Ω_c, the following algorithm will produce the constellation $\mathcal{C}_f^{\alpha}$.

1. Choose an $s \in [\Lambda_c/\Lambda_c']$, to identify a coset of Λ_c' in Λ_c.
2. Apply the mapping $\lambda_s(\cdot)$ (see Equation (3.18)) to each point in $\mathcal{C}_f$. This mapping moves points from the Voronoi region of coarse points to the base Voronoi region, *i.e.*, the mapping performs componentwise shaping on the fine code.
3. Scale the mapped points by α to form a new set of points $\mathcal{C}_f^s$.
4. Repeat steps 1 through 3 until all elements in $[\Lambda_c/\Lambda_c']$ have been used only once.
5. $\mathcal{C}_f^{\alpha} = \bigcup_{s \in [\Lambda_c/\Lambda_c']} \mathcal{C}_f^s$

The PMF on $\mathcal{C}_f^{\alpha}$ and on $\mathcal{C}_{\text{UEP}}$ is not uniform. The true PMF can be difficult to compute. The probability of a point x_0 in $\mathcal{C}_f^{\alpha}$ is

$$p_x(x_0) = \sum_{s_0 \in [\Lambda_c/\Lambda_c']} p_{x|s}(x_0|s_0) \cdot p_s(s_0). \tag{3.19}$$

Given a distribution on the set of coset representatives $[\Lambda_c/\Lambda_c']$, knowing $p_{x|s}(x_0|s_0)$ and $p_s(s_0)$ allows one to compute the probability mass function (PMF) for the points in $\mathcal{C}_f^{\alpha}$.

Taking $\mathcal{C}_f^{\alpha}$ and centering at each coarse point $z \in \mathcal{C}_c$, the constellation $\mathcal{C}_{\text{UEP}}$ is obtained. Let $\tilde{x}_0 \in \mathcal{C}_{\text{UEP}}$ be the mapped point of $x_0 \in \mathcal{C}_f^{\alpha}$ in the region surrounding the coarse symbol $z \in \mathcal{C}_c$, *i.e.*, $\tilde{x}_0 = \lambda_z^{-1}(x_0)$. The probability of the point $\tilde{x}_0$ in $\mathcal{C}_{\text{UEP}}$ occuring is

$$p_{\tilde{x}}(\tilde{x}_0) \sum_{z_0 \in \mathcal{C}_c} p_{\tilde{x}|z}(\tilde{x}_0|z_0) \cdot p(z_0).$$

Now,

$$p_{\tilde{x}|z}(\tilde{x}_0|z_0) = \begin{cases} p_x(x_0) & \text{if } \tilde{x}_0 = \lambda_{z_0}(x_0), \\ 0 & \text{otherwise.} \end{cases}$$

Thus,

$$p_{\tilde{x}}(\tilde{x}_0) = \sum_{z_0 \in \mathcal{C}_c} p_x(x_0) \cdot p_z(z_0) \tag{3.20}$$

where x_0 is the point in $\mathcal{C}_f^a$ which yields $\tilde{x}_0$ when centered around z_0.

Since Ω_c shapes Ω_f, $\mathcal{C}_f$ must be contained in the region $\boldsymbol{\mathcal{R}}_{\Lambda_0}$ where Λ_0 is the time-zero lattice of Ω_c. As discussed in Section 2.3, the non-unique decomposition of some codewords in Ω_f creates two major problems.

The first problem caused by the non-uniqueness involves a possible infinite decoding delay. The decoding delay of a trellis code is the delay between the input sequence and the decoded output sequence. The shaping decoder used on the encoded fine point sequence has a possible infinite decoding delay. Certain fine constellation sequences can cause the decoder to alternate between a set of states which generate two or more paths with equal weights. Thus, the decoder cannot decide which path is best until it exits the set of "don't care" states. If the encoded fine point sequence continually force the shaping decoder to alternate between the "don't care" states, then the decoding delay becomes infinite. Luckily, the probability of such sequences occuring approaches zero as the input sequence length approaches infinity.

The number of points in $\mathcal{C}_f$ which can cause transitions to another "don't care" state or to stay in the present "don't care" state may be a non-negligible fraction of $|\mathcal{C}_f|$. However, the probability of seeing long sequences of "don't care symbols" of Ω_f is small (see Section 4.3 for a discussion of how this impacts decoding complexity).

The second problem caused by the non-uniqueness involves non-uniquely decodable sequences. Since the fundamental region of Ω_c is the infinite Cartesian product of $\boldsymbol{\mathcal{R}}_{\Lambda_0}$, it is possible to construct sequences of fine code symbols that coincide exactly with a valid coarse code sequence (shaping sequence). Applying the distance preserving mapping $\boldsymbol{\lambda}(\cdot)$ for each of the different shaping sequences, different fine code sequences can result in the same base region sequence (shaped sequence). Thus, the codeword in Ω_{UEP} generated by any of these fine code sequences is non-uniquely decodable. However, a sequence of fine symbols which corresponds exactly with the coarse symbols becomes less likely as the length of the sequence grows, *i.e.*, the prob-

ability of a sequence being non-uniquely decodable approaches zero as the length of the sequence approaches infinity. If such a sequence of symbols were to occur, then a decoding error could occur even in a noiseless environment.

3.3 Interpretation of the UEP Trellis Code

The novelty of this coding scheme lies how the fine code is shaped by the coarse code. Shaping involves changing the uncoded bits of the inner code to achieve a more Gaussian distribution on the codebook. In other words, the shaping code selects the uncoded bits for the inner code. In this UEP trellis coding scheme, the inner code is the fine code. The main difference between normal trellis shaping the shaping used in the UEP trellis code is that some of the shaped bits are reused by the coarse code. The effects of overlapping input bits and reusing the shaped bits are the main focus of this section. The effects of reusing shaped bits on code performance and the number of overlap bits to reuse are discussed below.

The reuse of the shaped bits would seem to defeat the purpose of using a shaping code if the effects of the shaping are immediately undone by the coarse code, *i.e.*, the bits which were set by the shaping code are selected to be entirely different values by the coarse code. In fact, the shaping has *no* effect on the overall UEP code if $\alpha = 1$. When $\alpha = 1$, the performance of the UEP code becomes the performance of just the fine code because the shaping bits are being ignored, *i.e.*, the values of the bits are being selected independently of the shaping code. Geometrically, since Λ_c is a sublattice of Λ_f and all mappings move points in Λ_c to points in Λ_c, when no scaling is applied to $\mathcal{C}_f \in \Lambda_f$ the mapping of $\mathcal{C}_f$ mod $\boldsymbol{\mathcal{R}}_s$ will produce another point in $\mathcal{C}_f$. Thus, the overall UEP code acts like the fine code alone.

However, if $\alpha < 1$, shaping *does* still impact the overall UEP code since a shrunken shaped code means each element in a codeword sequence does not even coincide with a lattice point in either Λ_c or Λ_s. Thus, shrinking the base Voronoi region after shaping by the coarse code means that the fine code points are shaped to the shrunken Voronoi region *and* the bits which were used to overlap the fine code for shaping can

be reused for encoding the coarse code. For the extreme case of $\alpha = 0$, the UEP trellis code approaches the performance of the coarse code alone since all fine sequences are mapped a Voronoi region and completely shrunk to the coarse sequence of the Voronoi region. Thus, the fine coding is entirely ignored.

As in normal trellis shaping, the shaped bits of the fine code must be the k_f^u uncoded bits. The bits used to shape the fine code are the $n_c' - k_c$ redundant bits of the coarse code. The number of overlapping bits which are reused is $r_c \leq \min(k_f^u, n_c' - k_c)$ and fixes $|\Lambda_f/\Lambda_c| = 2^{n_f - r_c}$. Reusing the overlapped bits allows a higher fine code rate without increasing the bounding region $\mathcal{R}_s$ by efficiently packing fine sequences into the Voronoi regions of the coarse sequences. The size of $\mathcal{C}_{\text{UEP}}$ is $2^{\frac{2(n_c+n_f-r_c)}{N}}$ per two dimensions. When $r_c = 0$, coding is applied independently to the fine constellation and the coarse constellation as in the schemes presented by Wei [10]. If independent coding is used with the same number of fine and coarse bits, the size of $\mathcal{C}_{\text{UEP}}$ would be $2^{\frac{2(n_c+n_f)}{N}}$ per two dimensions. Thus, the signal set of the UEP trellis coding system is smaller by a factor of $2^{\frac{2r_c}{N}}$.

Now, suppose the size of $\mathcal{C}_{\text{UEP}}$ is fixed, *i.e.*, $n_c + n_f - r_c$ is fixed, and suppose $r_c < \min(k_f^u, n_c' - k_c')$. The redundancy can be increased by reducing the rate of the FSM used in Ω_c (increasing n_c'). The immediate design question becomes, "What rate trellis code should be used?" Does increasing n_c' so that $n_c' - k_c' = k_f^u$ help the overall system performance? Since choosing n_c' for fixed k_c' is the same as choosing a rate for the trellis code Ω_c, the design problem becomes choosing a trellis code with maximum rate which achieves acceptable performance.

From the design of trellis codes, a rate $k/k+1$ code can achieve coding gains near 5-6 dB. For instance, a 128 state rate 3/4 trellis code can achieve a nominal gain of 6dB with squared minimum distance increased by 8. However, nominal coding gain does not completely describe the code's performance. The number of nearest neighbors, or *error coefficient*, must also be considered, thus the effective coding gain is the true parameter of interest. Increasing the number of nearest neighbors will decrease the effective coding gain from the nominal coding gain. Empirical results show that using a rate lower than $k/k+2$ does not achieve much additional coding

gain because the number of nearest neighbors begins to grow very large, thus the effects from an increased minimum distance are offset by the large error coefficient. In addition, complexity is increased slightly for lower rate codes. Thus, using 1-2 bits of overlap achieves most of the significant distance gain and coding gain, *i.e.*, choosing $r_c > 2$ does not serve much practical purpose.

3.4 Performance analysis of UEP Trellis Coding

The performance of the UEP trellis code can be compared to the uncoded QAM system which takes $k_c + k_f$ bits at each time i. In addition to the coding gains over an uncoded system, performance for the UEP trellis coding scheme can be compared to the baseline M-QAM UEP scheme presented in Section 3.1. Define the minimum distance between lattice points in Λ_c as d_{Λ_c} and the minimum distance between lattice points in Λ_f is d_{Λ_f}. Each code Ω_c and Ω_f also has a minimum distance between codewords. These distances are d_{Ω_c} and d_{Ω_f} respectively. Recall that for any trellis code Ω, the gain in squared minimum distance over the lattice minimum distance is $d^2_{\text{gain},\Omega}$.

3.4.1 Minimum Distances for the UEP Trellis Code

The performance of Ω_{UEP} is largely determined by the coarse and fine minimum distances d_c and d_f. Performance analysis begins with the derivation of these two quantities. The distance d_f is the minimum distance between mapped fine sequences in the constellation Ω_{UEP}, *i.e.*, the intracloud distance. The minimum distance between fine sequences is

$$d^2_{\Omega_f} = d^2_{\text{gain},\Omega_f} d^2_{\Lambda_f}. \tag{3.21}$$

Since mapping preserves Ω_f, the sequences in the shrunken base region have squared minimum distance

$$d^2_f = \alpha^2 d^2_{\Omega_f} = \alpha^2 d^2_{\text{gain},\Omega_f} d^2_{\Lambda_f}. \tag{3.22}$$

The distance d_c corresponds to the minimum distance between the clouds of fine

sequences in Ω_{UEP}. First, notice that the order of mapping and scaling can be interchanged. That is, the sequences in a base cloud region can be mapped to the Voronoi region of a coarse sequence first, then the distance between the mapped sequence and the coarse sequence can be scaled by α. The intercloud distance is governed by the two closest sequences which lie in different adjacent clouds. To find d_c, begin by considering the shaped fine code points which lie in the base region. When these points are mapped to each coarse sequence $\boldsymbol{z}$, the fine sequences can lie anywhere within the region $\boldsymbol{\mathcal{R}}_V^{\Omega_c}(\boldsymbol{z})$ or on the boundary of the region.

Since Λ_f/Λ_c has order $2^{n_f-r_c}$,

$$d_{\Lambda_c}^2 = 2^{n_f-r_c} d_{\Lambda_f}^2. \tag{3.23}$$

The trellis code Ω_c has a minimum distance

$$d_{\Omega_c}^2 = d_{\text{gain},\Omega_c}^2 d_{\Lambda_c}^2, \tag{3.24}$$

and the squared minimum distance between any two adjacent coarse sequences $\boldsymbol{A}$ and $\boldsymbol{B}$ is $d_{\Omega_c}^2$. Now, the code Ω_c is formed by mapping the base region points around all coarse sequences $\boldsymbol{z} \in \Omega_c$ and scaling the distance between the mapped sequence and $\boldsymbol{z}$ by α.

The minimum intercloud distance d_c^2 is

$$d_c^2 = \min_{\substack{\boldsymbol{x}_1 \in \boldsymbol{\mathcal{R}}_V^{\Omega_c}(\boldsymbol{A}) \cap \Omega_{\text{UEP}} \\ \boldsymbol{x}_2 \in \boldsymbol{\mathcal{R}}_V^{\Omega_c}(\boldsymbol{B}) \cap \Omega_{\text{UEP}}}} \|\boldsymbol{x}_1 - \boldsymbol{x}_2\|^2$$

where the minimum is taken over all $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \Omega_{\text{UEP}}$ that lie in distinct Voronoi regions $\boldsymbol{\mathcal{R}}_V^{\Omega_c}(\boldsymbol{A})$ and $\boldsymbol{\mathcal{R}}_V^{\Omega_c}(\boldsymbol{B})$. Notice that $d_c^2 \geq d_f^2$ with equality if $\alpha = 1$ (no scaling is performed).

Let $\boldsymbol{\mathcal{R}}_V^{\Omega_c}(\boldsymbol{A})$ and $\boldsymbol{\mathcal{R}}_V^{\Omega_c}(\boldsymbol{B})$ be adjacent Voronoi regions. The intercloud distance d_c can be upper and lower bounded as a function of α by considering the geometry

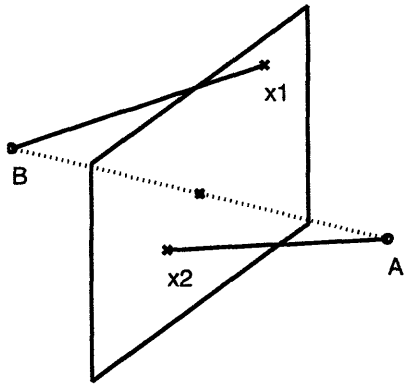


Figure 3-5: Worst case boundary condition for minimum distance of d_c

of the boundary between $\mathcal{R}_V^{\Omega_c}(\boldsymbol{A})$ and $\mathcal{R}_V^{\Omega_c}(\boldsymbol{B})$. Let $\boldsymbol{x}_1 \in \mathcal{R}_V^{\Omega_c}(\boldsymbol{A}) \cap \Omega_{\text{UEP}}$ and $\boldsymbol{x}_2 \in \mathcal{R}_V^{\Omega_c}(\boldsymbol{B}) \cap \Omega_{\text{UEP}}$ be codewords at minimum distance d_f^2. Notice that when $\alpha = 1$, the minimum distance d_c^2 will be equal to d_f^2. Now, one of the three following geometries will occur.

1. $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ lie strictly within $\mathcal{R}_V^{\Omega_c}(\boldsymbol{A})$ and $\mathcal{R}_V^{\Omega_c}(\boldsymbol{B})$ respectively

2. $\boldsymbol{x}_1$ lies within $\mathcal{R}_V^{\Omega_c}(\boldsymbol{A})$ and $\boldsymbol{x}_2$ lies on the boundary between $\mathcal{R}_V^{\Omega_c}(\boldsymbol{A})$ and $\mathcal{R}_V^{\Omega_c}(\boldsymbol{B})$

3. $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ both lie on the boundary between $\mathcal{R}_V^{\Omega_c}(\boldsymbol{A})$ and $\mathcal{R}_V^{\Omega_c}(\boldsymbol{B})$

The worst minimum distance will arise with geometry 3 whereas the most optimistic minimum distance will arise in geometry 1.

Using geometry 3 shown in Figure 3-5, a lower bound on the minimum distance can be determined. As both $\mathcal{R}_V^{\Omega_c}(\boldsymbol{A})$ and $\mathcal{R}_V^{\Omega_c}(\boldsymbol{B})$ are scaled by α, a point originally on the boundary will move toward the center of the region, thus moving away from the boundary. By performing a suitable change of basis followed by translation, there is no loss in generality in assuming

$$\boldsymbol{A} = \left(-\frac{d_{\Omega_c}}{2}, 0, 0, \ldots\right)$$
$$\boldsymbol{B} = \left(\frac{d_{\Omega_c}}{2}, 0, 0, \ldots\right)$$

Since both $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are on the boundary their components have the form

$$\boldsymbol{x}_1 = (0, x_{1_1}, x_{1_2}, x_{1_3}, \ldots)$$
$$\boldsymbol{x}_2 = (0, x_{2_1}, x_{2_2}, x_{2_3}, \ldots)$$

where

$$\sum_{i=1}^{\infty}(x_{1_i} - x_{2_i})^2 = d_{\Omega_f}^2. \tag{3.25}$$

Scaling by α brings the points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ closer to $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively. The scaled vectors become

$$\vec{\boldsymbol{x}}_{1,\alpha} = \left((1-\alpha)(-\frac{d_{\Omega_c}}{2}), \alpha x_{1_1}, \alpha x_{1_2}, \alpha x_{1_3}, \ldots\right)$$
$$\vec{\boldsymbol{x}}_{2,\alpha} = \left((1-\alpha)(\frac{d_{\Omega_c}}{2}), \alpha x_{2_1}, \alpha x_{2_2}, \alpha x_{2_3}, \ldots\right)$$

Let

$$\vec{\boldsymbol{x}} = \boldsymbol{x}_{1,\alpha} - \boldsymbol{x}_{2,\alpha}.$$

The squared distance between the two points is the l_2 norm squared of $\vec{\boldsymbol{x}}$:

$$\begin{aligned} d_{\text{c,worst}}^2 = \|\vec{\boldsymbol{x}}\|^2 &= (1-\alpha)^2 d_{\Omega_c}^2 + \sum_{i=1}^{\infty}(\alpha x_{1_i} - \alpha x_{2_i})^2 \\ &= (1-\alpha)^2 d_{\Omega_c}^2 + \alpha^2 d_{\Omega_f}^2 \\ &= (1-\alpha)^2 d_{\Omega_c}^2 + d_f^2. \end{aligned} \tag{3.26}$$

Notice that the coarse code squared distance has very little impact on the intercloud distance d_c^2 when α is close to one. As α approaches one, the reduction of the intercloud distance can be detrimental to the performance of the UEP code on more important data.

For an upper bound, consider geometry 1 shown in Figure 3-6. In this geometry, the points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ lie along the line connecting $\boldsymbol{A}$ to $\boldsymbol{B}$ and the full effect of the α scaling is observed. The points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are still separated by d_{Ω_f} before the α

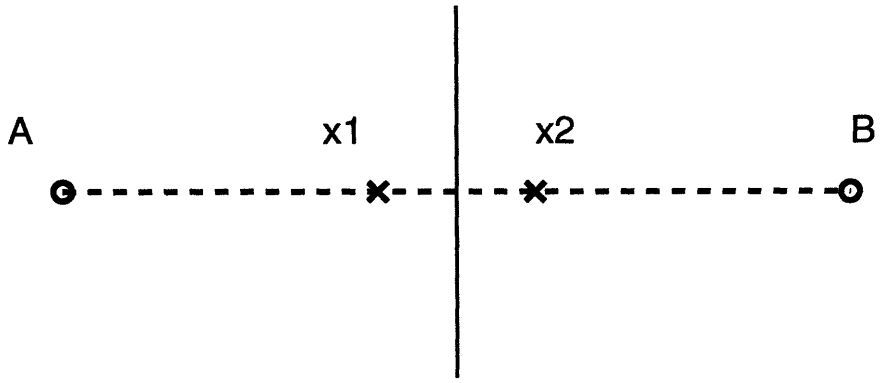


Figure 3-6: Best case boundary condition for minimum distance of d_c

scaling. Let

$$\vec{\boldsymbol{x}}_1 = \boldsymbol{x}_1 - \boldsymbol{A}$$

$$\vec{\boldsymbol{x}}_2 = \boldsymbol{x}_2 - \boldsymbol{B}$$

and define the first component of each vector to lie along the direction of the line joining $\boldsymbol{A}$ and $\boldsymbol{B}$. The vectors become

$$\vec{\boldsymbol{x}}_1 = (x_{1_0}, 0, 0, \ldots) \tag{3.27}$$

$$\vec{\boldsymbol{x}}_2 = (x_{2_0}, 0, 0, \ldots) \tag{3.28}$$

$$\tag{3.29}$$

where $(\boldsymbol{B} - \boldsymbol{A}) = (d_{\Omega_c}, 0, 0, \ldots)$ and

$$x_{1_0} = \frac{d_{\Omega_c} - d_{\Omega_f}}{2} \tag{3.30}$$

$$x_{2_0} = -\frac{d_{\Omega_c} - d_{\Omega_f}}{2}. \tag{3.31}$$

The scaled vectors are

$$\alpha\vec{\boldsymbol{x}}_1 = \left(\alpha\frac{d_{\Omega_c} - d_{\Omega_f}}{2}, 0, 0, \ldots\right) \tag{3.32}$$

$$\alpha\vec{\boldsymbol{x}}_2 = \left(-\alpha\frac{d_{\Omega_c} - d_{\Omega_f}}{2}, 0, 0, \ldots\right) \tag{3.33}$$

$$\vec{x} = \alpha\vec{x}_1 - (\alpha\vec{x}_2 + (\boldsymbol{B} - \boldsymbol{A})) = \alpha\vec{x}_1 - \alpha\vec{x}_2 - (\boldsymbol{B} - \boldsymbol{A}) \tag{3.34}$$

From Equations (3.34), (3.27), (3.28), (3.32), and (3.33), the minimum distance d_c becomes

$$\begin{aligned} d^2_{c,\text{best}} &= \|\vec{x}\|^2 \\ &= \left(\alpha\frac{d_{\Omega_c} - d_{\Omega_f}}{2} + \alpha\frac{d_{\Omega_c} - d_{\Omega_f}}{2} - d_{\Omega_c}\right)^2 + \sum_{i=1}^{\infty}(x_{1_i} - x_{2_i})^2 \\ &= (\alpha d_{\Omega_c} - \alpha d_{\Omega_f} - d_{\Omega_c})^2 + 0 \\ &= \left((\alpha - 1)d_{\Omega_c} - \alpha d_{\Omega_f}\right)^2 \\ &= \left((1 - \alpha)d_{\Omega_c} + d_f\right)^2 . \end{aligned} \tag{3.35}$$

From Equations (3.26) and (3.35), the difference between the worst case estimate and the best case estimate for the squared minimum distance d_c^2 is $2(1-\alpha)d_{\Omega_c}\alpha d_{\Omega_f}$. The difference between the best and worst case minimum distances can be quite significant. For the example in Section 3.5, the difference is close to 1dB for $\alpha \approx 0.8$. If the geometry of the code could be arranged so that geometry 1 was always true, then the coarse code would achieve higher effective coding gains. In fact, the UEP QAM example in Section 3.1 and the codes developed by Wei [10] manage to always remain in the best case geometry. However, Wei's codes suffer from other problems as will be discussed in Chapter 5.

3.4.2 Constellation Energies

The average energy of the constellation $\mathcal{C}_{\text{UEP}}$ is needed to calculate the coding gain of the UEP trellis code versus standard two dimensional M-QAM.

The expected average energy of a constellation $\mathcal{C}_{\text{UEP}}$ is

$$\overline{E}_{\mathcal{C}_{\text{UEP}}} = \sum_{x_0 \in \mathcal{C}_{\text{UEP}}} \|x_0\|^2 \cdot p_x(x_0). \tag{3.36}$$

Since the distribution $p_x(x_0)$ depends on $\mathcal{C}_{\text{UEP}}$, Equation (3.36) depends on the choice

of Ω_c and Ω_s. Finding the exact $\mathcal{C}_{\text{UEP}}$ may not be feasible, thus approximations of $\overline{E}_{\mathcal{C}_{\text{UEP}}}$ are used.

The continuous approximation [5] can be used to estimate the average energy of $\mathcal{C}_{\text{UEP}}$. The continuous approximation assumes a continuum of points over $\boldsymbol{\mathcal{R}}_s$, the shaping region of $\mathcal{C}_{\text{UEP}}$. Each point is equally likely, thus forming a uniform distribution, $f_x(x_0) = 1/V(\boldsymbol{\mathcal{R}}_s)$, over $\boldsymbol{\mathcal{R}}_s$ where $V(\boldsymbol{\mathcal{R}}_s)$ is the volume of the bounding region. The estimated expected energy in $\boldsymbol{\mathcal{R}}_s$ is

$$\begin{aligned} \overline{E}_{\mathcal{C}_{\text{UEP}}} &\approx \overline{E}(\boldsymbol{\mathcal{R}}_s) = \int_{\boldsymbol{x}_0 \in \boldsymbol{\mathcal{R}}_s} \|\boldsymbol{x}_0\|^2 f_x(\boldsymbol{x}_0)\, d\boldsymbol{x}_0 \\ &= \frac{1}{V(\boldsymbol{\mathcal{R}}_s)} \int_{\boldsymbol{x}_0 \in \boldsymbol{\mathcal{R}}_s} \|\boldsymbol{x}_0\|^2 \, d\boldsymbol{x}_0. \end{aligned} \tag{3.37}$$

To compare to the two dimensional M-QAM constellation, the energy must be normalized to energy per two dimensions:

$$pE_{\mathcal{C}_{\text{UEP}}} \approx \frac{2\overline{E}(\boldsymbol{\mathcal{R}}_s)}{N}. \tag{3.38}$$

Define γ_s to be the shaping gain which is the inverse of the energy gain of the region $\boldsymbol{\mathcal{R}}_s \in R^2$ over a square region S_2 in R^2 with sides of length $2r$. Thus, in two dimensions,

$$(2r)^2 = [V(\boldsymbol{\mathcal{R}}_s)]^{2/N}$$

and

$$r^2 = \frac{[V(\boldsymbol{\mathcal{R}}_s)]^{2/N}}{4}. \tag{3.39}$$

Using the continuous approximation over the square region, the average energy of S_2 is

$$E(S_2) = \frac{2r^2}{3}.$$

From Equations (3.38) and (3.39), the shaping gain becomes

$$\gamma_s = \frac{2r^2/3}{E_{\mathcal{C}_{\text{UEP}}}} = \frac{[V(\boldsymbol{\mathcal{R}}_s)]^{2/N}}{6E_{\mathcal{C}_{\text{UEP}}}}. \tag{3.40}$$

For an M-QAM constellation with $M = 2^{\frac{2(k_f+k_c)}{N}}$ points occupying the region S_2 the squared minimum distance between points is

$$d^2_{\text{QAM}} = \frac{r^2}{2^{\frac{2(k_f+k_c)}{N}}} = \frac{[V(\mathcal{R}_s)]^{2/N}}{4 \cdot 2^{\frac{2(k_c+k_f)}{N}}} \tag{3.41}$$

and the average energy of the M-QAM constellation is

$$E(S_2) = \frac{6}{[V(\mathcal{R}_s)]^{2/N}}. \tag{3.42}$$

For the coarse trellis code, the FSM introduces redundancy in the form of extra bits selecting points at each time i from Λ_c (*i.e.*, $n'_c > k'_c$). Thus, there is an expansion in the signal set and either the bounding region must be increased in energy or the points in the bounding region are moved closer together. Define the normalized redundancy per two dimensions as [5, p. 687]

$$\rho(\Omega_c) = \frac{n'_c - k'_c}{N/2}. \tag{3.43}$$

The energy increase per two dimensions is approximately $2^{\rho(\Omega_c)}$.

The FSM of Ω_f also introduces redundancy in the form of extra bits selecting points at each time i from Λ_f ($n'_f > k'_f$). The normalized redundancy per two dimensions for the fine code is

$$\rho(\Omega_f) = \frac{n'_f - k'_f}{N/2}. \tag{3.44}$$

The energy increase per two dimensions is approximately $2^{\rho(\Omega_f)}$.

The total redundancy for the UEP code is

$$n_c + n_f - r_c - k_c - k_f = n'_c - k'_c + n'_f - k'_f - r_c$$

bits. The reduction of r_c bits is due to the overlap described in Section 3.3. The

normalized redundancy per two dimensions for the UEP trellis code is

$$\rho(\Omega_{\text{UEP}}) = \frac{n_c' - k_c' + n_f' - k_f' - r_c}{\frac{N}{2}} = \rho(\Omega_c) + \rho(\Omega_f) - \frac{2r_c}{N}$$

The energy increase due to redundancy is approximately

$$2^{\rho(\Omega_{\text{UEP}})} = 2^{\rho(\Omega_c)} 2^{\rho(\Omega_f)} 2^{-\frac{2r_c}{N}}$$

and define the coding gain due to the overlap as

$$\gamma_{\text{UEP}} = 2^{\frac{2r_c}{N}}.$$

3.4.3 Coding Gains

The coding gain for the coarse and fine codes can now be derived. Since the exact minimum distance of the coarse code is not known, only a lower bound on the coarse coding gain is derived. The coding gain for the fine code γ_f is defined as the ratio of the energy normalized d_f^2 to the energy-normalized d_{QAM}^2. Thus,

$$\begin{aligned}
\gamma_f &\approx \frac{d_f^2}{d_{\text{QAM}}^2} \cdot \frac{E_{\text{QAM}}}{E_{C_{\text{UEP}}}} \\
&= \frac{4\alpha^2 d_{\text{gain},\Omega_f}^2 d_{\Lambda_f}^2 2^{\frac{2(k_c+k_f)}{N}}}{[V(\boldsymbol{\mathcal{R}}_s)]^{2/N}} \cdot \frac{E(S_2)}{2^{\rho(\Omega_{\text{UEP}})} E_{C_{\text{UEP}}}} \\
&= \frac{4\alpha^2 d_{\text{gain},\Omega_f}^2 d_{\Lambda_f}^2 2^{\frac{2(k_c+k_f)}{N}}}{[V(\boldsymbol{\mathcal{R}}_s)]^{2/N}} \cdot \frac{E(S_2)\gamma_s}{2^{\rho(\Omega_{\text{UEP}})} E(S_2)} \\
&= \frac{4\alpha^2 d_{\text{gain},\Omega_f}^2 d_{\Lambda_f}^2 2^{\frac{2(k_c+k_f)}{N}}}{[V(\boldsymbol{\mathcal{R}}_s)]^{2/N}} \cdot \frac{\gamma_s}{2^{\rho(\Omega_{\text{UEP}})}} \\
&= 4\alpha^2 \frac{d_{\Lambda_f}^2 2^{\frac{2(k_c+k_f)}{N}}}{[V(\boldsymbol{\mathcal{R}}_s)]^{2/N}} \cdot \frac{d_{\text{gain},\Omega_f}^2}{2^{\rho(\Omega_f)}} \cdot \frac{1}{2^{\rho(\Omega_c)}} \cdot \frac{1}{2^{-\frac{2r_c}{N}}} \cdot \gamma_s \\
&= \alpha^2 \gamma_{\Lambda_f} \cdot \gamma_{\Omega_f} \cdot \gamma_s \cdot \gamma_{\text{UEP}} \cdot 2^{-\rho(\Omega_c)}
\end{aligned} \tag{3.45}$$

where the gain from the lattice is

$$\begin{aligned}\gamma_{\Lambda_f} &= \frac{d^2_{\Lambda_f}}{d^2_{\text{QAM}}} \\ &= \frac{4d^2_{\Lambda_f} 2^{\frac{2(k_c+k_f)}{N}}}{[V(\boldsymbol{\mathcal{R}}_s)]^{2/N}}\end{aligned}$$

and the gain from the trellis code is

$$\gamma_{\Omega_f} = \frac{d^2_{\text{gain},\Omega_f}}{2^{\rho(\Omega_f)}}.$$

The coding gain can be decomposed into contributions from the lattice, the FSM with constellation expansion, the shaping performed by the bounding region, and the overlap between coarse and fine bits. Notice, the expansion of the constellation by the redundancy of the coarse bits is also a factor in the fine coding gain. The coding gain for the UEP trellis code is very similar to the coding gain expression of a normal trellis codes except for the factors γ_{UEP} and $2^{-\rho(\Omega_c)}$. Increasing the redundancy of the coarse code penalizes the coding gain through the term $2^{-\rho(\Omega_c)}$. However, increasing the redundancy of the coarse code means that the overlap can be increased, thus the overlap coding gain γ_{UEP} will offset the penalty from the increased coarse code redundancy. As discussed in Section 3.3, the number overlap bits does not need to be more than roughly 2 bits to obtain most of the effective coding gain.

The lower bound on the coding gain for the coarse code, γ_c, is defined as the ratio of the energy-normalized $d^2_c = d^2_{c,\text{worst}}$ to the energy-normalized d^2_{QAM}. Thus,

$$\begin{aligned}\gamma_c &\approx \frac{d^2_c}{d^2_{\text{QAM}}} \cdot \frac{E_{\text{QAM}}}{Ec_{\text{UEP}}} \\ &= \frac{4\left((1-\alpha)^2 d^2_{\Omega_c} + d^2_f\right) 2^{\frac{2(k_c+k_f)}{N}}}{[V(\boldsymbol{\mathcal{R}}_s)]^{2/N}} \cdot \frac{E(S_2)}{2^{\rho(\Omega_{\text{UEP}})} Ec_{\text{UEP}}}\end{aligned}$$

$$
\begin{aligned}
&= \frac{4\left((1-\alpha)^2 d^2_{\text{gain},\Omega_c} d^2_{\Lambda_c} + d^2_f\right) 2^{\frac{2(k_c+k_f)}{N}}}{[V(\boldsymbol{\mathcal{R}}_s)]^{2/N}} \cdot \frac{E(S_2)\gamma_s}{2^{\rho(\Omega_{\text{UEP}})} E(S_2)} \\
&= \frac{4\left((1-\alpha)^2 d^2_{\text{gain},\Omega_c} d^2_{\Lambda_c} + d^2_f\right) 2^{\frac{2(k_c+k_f)}{N}}}{[V(\boldsymbol{\mathcal{R}}_s)]^{2/N}} \cdot \frac{\gamma_s}{2^{\rho(\Omega_{\text{UEP}})}} \\
&= \frac{4(1-\alpha)^2 d^2_{\text{gain},\Omega_c} d^2_{\Lambda_c} 2^{\frac{2(k_c+k_f)}{N}}}{[V(\boldsymbol{\mathcal{R}}_s)]^{2/N}} \cdot \frac{\gamma_s}{2^{\rho(\Omega_{\text{UEP}})}} + \frac{4d^2_f 2^{\frac{2(k_c+k_f)}{N}}}{[V(\boldsymbol{\mathcal{R}}_s)]^{2/N}} \cdot \frac{\gamma_s}{2^{\rho(\Omega_{\text{UEP}})}} \\
&= (1-\alpha)^2 \cdot \frac{4d^2_{\Lambda_c} 2^{\frac{2(k_c+k_f)}{N}}}{[V(\boldsymbol{\mathcal{R}}_s)]^{2/N}} \cdot \frac{d^2_{\text{gain},\Omega_c}}{2^{\rho(\Omega_c)}} \cdot \frac{1}{2^{\rho(\Omega_f)}} \cdot \frac{1}{2^{-\frac{2r_c}{N}}} \cdot \gamma_s + \gamma_f \\
&= (1-\alpha)^2 \cdot \gamma_{\Lambda_c} \cdot \gamma_{\Omega_c} \cdot \gamma_{\text{UEP}} \cdot \gamma_s \cdot 2^{-\rho(\Omega_f)} + \gamma_f \qquad (3.46)
\end{aligned}
$$

where the coding gain from the coarse lattice is

$$
\begin{aligned}
\gamma_{\Lambda_c} &= \frac{d^2_{\Lambda_c}}{d^2_{\text{QAM}}} && (3.47) \\
&= \frac{4d^2_{\Lambda_c} 2^{\frac{2(k_c+k_f)}{N}}}{[V(\boldsymbol{\mathcal{R}}_s)]^{2/N}} && (3.48) \\
& && (3.49)
\end{aligned}
$$

and the coding gain from the coarse trellis code is

$$
\begin{aligned}
\gamma_{\Omega_c} &= \frac{d^2_{\text{gain},\Omega_c}}{2^{\rho(\Omega_c)}}. && (3.50) \\
& && (3.51)
\end{aligned}
$$

Using Equation (3.23) in Equation (3.47), the coding gain of Λ_c can be expressed in terms of the coding gain of the fine code lattice, Λ_f. In Equation (3.23), symbols were assumed to come from the space of R^N. Since all coding gains are compared to the two dimensional QAM constellation, the number of bits per 2-D must be used.

$$
\begin{aligned}
\gamma_{\Lambda_c} &= 2^{\frac{2(n_f-r_c)}{N}} \cdot \frac{4d^2_{\Lambda_f} 2^{\frac{2(k_c+k_f)}{N}}}{[V(\boldsymbol{\mathcal{R}}_s)]^{2/N}} \\
&= 2^{\frac{2(n_f-r_c)}{N}} \gamma_{\Lambda_f}. && (3.52)
\end{aligned}
$$

Using Equations (3.52) and (3.46), the coarse coding gain becomes

$$\begin{aligned}\gamma_c &= (1-\alpha)^2 \cdot 2^{\frac{2(n_f-r_c)}{N}-\rho(\Omega_f)}\gamma_{\Lambda_f} \cdot \gamma_{\Omega_c} \cdot \gamma_{\text{UEP}} \cdot \gamma_s + \gamma_f \\ &= (1-\alpha)^2 \cdot 2^{\frac{2(k_f-r_c)}{N}}\gamma_{\Lambda_f} \cdot \gamma_{\Omega_c} \cdot \gamma_{\text{UEP}} \cdot \gamma_s + \gamma_f \\ &= (1-\alpha)^2 \cdot 2^{\frac{2k_f}{N}} \cdot \gamma_{\Lambda_f} \cdot \gamma_{\Omega_c} \cdot \gamma_s + \gamma_f. \end{aligned} \tag{3.53}$$

The coarse coding gain is dependent on the absolute number of fine bits used, not the ratio of the number of fine bits to the number of coarse bits. Wei and Calderbank both use the ratio of fine to coarse bits as their parameter for code design. As can be seen from the fine coding gain in Equation (3.45), the energy expansion from the increased normalized redundancy of the coarse code can be offset by increasing the overlap bits. Thus, only the number of fine bits should effect the coding gain performance and the effect is only seen in the fine code as evident in Equation (3.53).

3.5 An Example of UEP Using Trellis Coding

In this example, every input consists of sequences of four bits, two more important bits ($k_c = 2$) for $\boldsymbol{x}^{(c)} = \{x_i^{(c)}\}$ and two less important bits ($k_f = 2$) for $\boldsymbol{x}^{(f)} = \{x_i^{(f)}\}$. The lattice Λ_f is the shifted integer lattice

$$\Lambda_f = Z^2 + (\frac{1}{2}, \frac{1}{2}).$$

Since $\Lambda_f \in R^2$, $N = 2$. The code Ω_f is uncoded 4-QAM, thus $d^2_{\Omega_f} = d^2_{\Lambda_f} = 1$, $k_f = 2, k_f^u = 2, k_f' = 0, n_f' = 0$, and $n_f = 2$. Thus, $|\Lambda_f/\Lambda_f'| = 2^{n_f'} = 1$, so $\Lambda_f' = Z^2 + (\frac{1}{2}, \frac{1}{2}) = \Lambda_f$.

An overlap of $r_c = 1$ bit is chosen between the coarse code encoded bits and the uncoded fine code bits. Since $n_f = 2$ and $r_c = 1$, $|\Lambda_f/\Lambda_c| = 2$ and $\Lambda_c = 2Z^2 + (\frac{1}{2}, \frac{1}{2})$. The coarse code Ω_c is a rate-1/2 trellis code shown in Figure 3-7(a). Since $k_c = 2, k_c' = 1, k_c^u = 1, n_c' = 2$, and $n_c = 3$, $|\Lambda_c/\Lambda_c'| = 2^{n_c'} = 4$ and $\Lambda_c' = 4\Lambda_c = 8Z^2 + (\frac{1}{2}, \frac{1}{2})$. The output of the convolution coder (FSM) for Ω_c selects a coset representative at

Label	Coset Rep.	Output of Conv. Coder
E	$(1/2, 3/2)$	00
F	$(3/2, 1/2)$	10
G	$(1/2, -1/2)$	11
H	$(-1/2, 1/2)$	01

Table 3.1: Label assignments for the conv. coder output of Ω_c

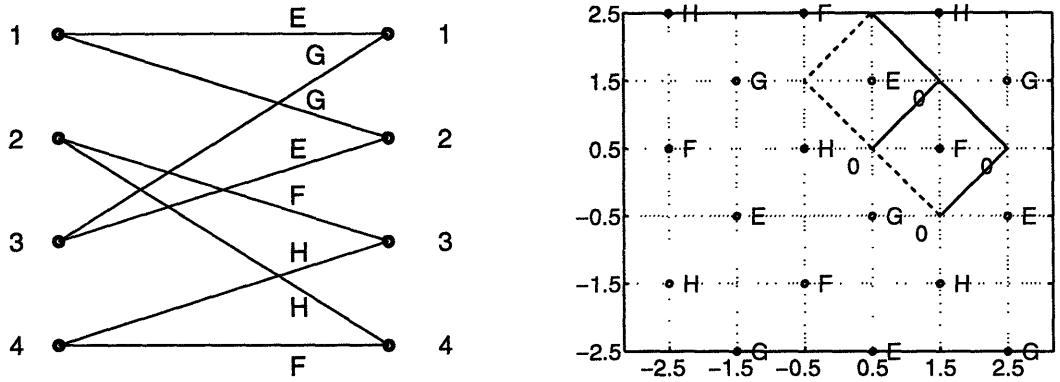


Figure 3-7: (a) FSM diagram for Ω_c (b) Λ_c and the labeled cosets

each time i. The correspondence between labels use in the trellis of Figure 3-7(a) and the coset representatives of Λ_c' in Λ_c is given in Table 3.1. Notice each coset representative is a member of Λ_c as required. The lattice Λ_c and its labeled cosets are shown in Figure 3-7(b). Since $k_c^u = 1$, two points are selected from each coset. A total of $2^{n_c} = 8$ points are chosen. The eight points chosen are the points which fall into $\mathcal{R}_s \cap \Lambda_c$ where $\mathcal{R}_s$ is a 4×4 square centered at the origin as shown in Figure 3-9 by the dashed box. The eight points form which form the coarse symbol constellation $\mathcal{C}_c$ and are: $(\frac{1}{2}, \frac{3}{2}), (-\frac{1}{2}, \frac{1}{2}), (\frac{3}{2}, \frac{1}{2}), (\frac{1}{2}, -\frac{1}{2}), (-\frac{3}{2}, -\frac{1}{2}), (\frac{3}{2}, -\frac{3}{2}), (-\frac{1}{2}, -\frac{3}{2}), (-\frac{3}{2}, \frac{3}{2})$.

The region $\mathcal{R}_{\Lambda_0}$ of Ω_c, is the union of two fundamental lattice regions associated with points from the cosets E, F or G, H. The particular choice for $\mathcal{R}_{\Lambda_0}$ is shown in Figure 3-7(b) by the two boxes surrounding E_0 and F_0. Given $\mathcal{R}_{\Lambda_0}$, the 2^{k_f} fine code points are selected by $\Lambda_f \cap \mathcal{R}_{\Lambda_0}$. The fine symbol constellation $\mathcal{C}_f$ is shown in Figure 3-8. These four points also form the code Ω_f. The labels $\{A, B, C, D\}$ correspond to the inputs $x_i^{(f)} = \{00, 01, 10, 11\}$.

A block diagram of this encoder is shown in Figure 3-4. The selection function

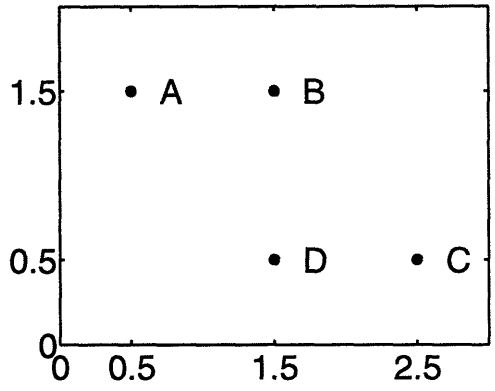


Figure 3-8: 4 points used to encode uncoded less important bits

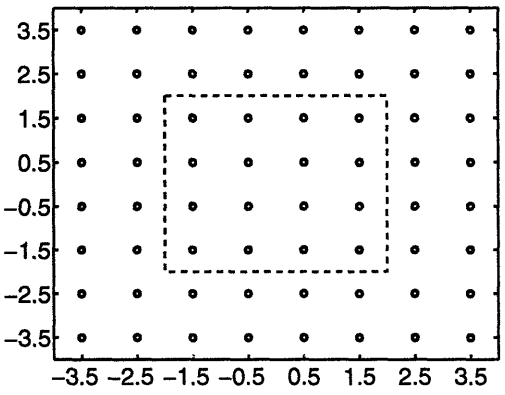


Figure 3-9: $\mathcal{R}_s \cap Z^2 + (\frac{1}{2}, \frac{1}{2})$ with $\mathcal{R}_s$ denoted by dashes

$f(x)$ chooses from the fine code points when the 2 most significant bits are zero. When the least significant bit is zero, the coarse points are selected. One choice for $f(x)$ is

$$f(x) = \begin{cases} (\frac{1}{2}, \frac{3}{2}) & \text{if } x = 0000, \\ (\frac{3}{2}, \frac{3}{2}) & \text{if } x = 0001, \\ (\frac{5}{2}, \frac{1}{2}) & \text{if } x = 0010, \\ (\frac{3}{2}, \frac{1}{2}) & \text{if } x = 0011, \\ (\frac{3}{2}, \frac{1}{2}) & \text{if } x = 0100, \\ (\frac{1}{2}, -\frac{1}{2}) & \text{if } x = 0110, \\ (-\frac{3}{2}, -\frac{1}{2}) & \text{if } x = 1000, \\ (\frac{3}{2}, -\frac{3}{2}) & \text{if } x = 1010, \\ (-\frac{1}{2}, -\frac{3}{2}) & \text{if } x = 1100, \\ (-\frac{3}{2}, \frac{3}{2}) & \text{if } x = 1110. \end{cases} \tag{3.54}$$

The base sequence is chosen to be the all $e^0 = (\frac{1}{2}, \frac{3}{2})$ sequence, $\boldsymbol{e}^0 = \{\ldots, e^0, e^0, \ldots\}$. The mapping function described in Section 3.2.2, $\lambda_{\hat{y}_i^{(c)}}(\cdot)$, where $\hat{y}_i^{(c)} \in \Lambda_c' + \hat{a}_i^{(c)}$ and $\hat{a}_i^{(c)} \in [\Lambda_c/\Lambda_c']$, moves any point in the coset $\Lambda_c' + \hat{a}_i^{(c)}$ to the base region associated with $\boldsymbol{e}^0$ and is given by:

$$\lambda_{\hat{y}_i^{(c)}}(x_i) = \tilde{\phi}_{\hat{y}_i^{(c)} \mapsto e_i^0}(x_i) - \tilde{\phi}_{\hat{y}_i^{(c)} \mapsto e_i^0}(\hat{y}_i^{(c)})$$

where $\tilde{\phi}_{\hat{y}_i^{(c)} \mapsto e_i^0}(\cdot)$ is

$$\tilde{\phi}_{\hat{y}_i^{(c)} \mapsto e_i^0}(x_i) = \begin{cases} x_i & \text{if } \hat{y}_i^{(c)} \in \Lambda_c' + E, \\ x_i + (-1, 1) & \text{if } \hat{y}_i^{(c)} \in \Lambda_c' + F, \\ x_i + (0, 2) & \text{if } \hat{y}_i^{(c)} \in \Lambda_c' + G, \\ x_i + (1, 1) & \text{if } \hat{y}_i^{(c)} \in \Lambda_c' + H. \end{cases} . \tag{3.55}$$

Also from Section 3.2.2, the inverse mapping function, $\lambda^{-1}_{y_i^{(c)}}(\cdot)$, moves points from the base region to the coset $\Lambda_c' + a_i^{(c)}$ where $a_i^{(c)}$ is the coset representative which identifies the coset of Λ_c' of which $y_i^{(c)} = \mathcal{C}_c(x_i^{(c)})$ is a member. The mapping from the

base region to the encoded superpoint is

$$\lambda^{-1}_{a_i^{(c)}}(x_i) = \tilde{\phi}^{-1}_{y_i^{(c)} \mapsto e_i^0}(x_i + e_i^0) + y_i^{(c)} - \tilde{\phi}_{y_i^{(c)} \mapsto e_i^0}(e_i^0).$$

3.5.1 Constellations of the UEP Trellis Code Example

The constellation $\mathcal{C}_{\text{UEP}}$ is formed from the base constellation $\mathcal{C}_f^\alpha$. Thus, $\mathcal{C}_f^\alpha$ is first derived, then $\mathcal{C}_{\text{UEP}}$ is formed by mapping $\mathcal{C}_f^\alpha$ around each coarse code symbol in $\mathcal{C}_c$. From Equations (3.19) and (3.20), both constellations $\mathcal{C}_{\text{UEP}}$ and $\mathcal{C}_f^\alpha$ depend on $p_{x|s}(x_0|s_0)$ where $x_0 \in \mathcal{C}_f^\alpha$ and $s_0 \in [\Lambda_c/\Lambda_c']$.

An efficient method for finding the average energy of $\mathcal{C}_f^\alpha$ was developed for this simple example UEP code. At any time i, the Viterbi Algorithm (VA) keeps a set of survivor paths, each with an accumulated squared distance metric. The number of possible paths at any time i is the same as the number of states in the trellis, which is four in this example. The path metrics are represented by a 4-tuple. These metrics are normalized by subtracting the least weight metric at time i (called the *normalization value* at time i) so that the minimum value in the 4-tuple is zero. Notice that the 4 points in $\mathcal{C}_f$ have squared metric values of 0, 1, 2, or 4 from any coarse lattice point. After normalization, each path turns out to have a weight of either 0, 1, 2, 4, or 6 and only 33 distinct 4-tuples are used by the VA. Thus, a 33 state FSM, referred to as the VA FSM, can be used to compute the energy and PMF of $\mathcal{C}_f^{\alpha=1}$ (the constellation which arises from shaping $\mathcal{C}_f$).

The unnormalized metric at time i is the sum of the current normalized state metric and the past normalization values. Because the path metrics increase by only 0, 1, 2, 4, or 6 each time, the metric of any path at time i satisfies

$$\begin{aligned} \sum_{k=0}^{i-1} [\text{normalization value at time } k] \;&\leq\; \text{path metric} \\ &\leq\; 6 + \sum_{k=0}^{i-1} [\text{normalization value at time } k]\,. \end{aligned}$$

Taking the expected value of this expression yields bounds on the expected metric:

$$\begin{aligned}\sum_{k=0}^{i-1} E[\text{normalization at time } k] &\leq E[\text{path metric}] \\ &\leq 6 + \sum_{k=0}^{i-1} E[\text{normalization value at time } k.]\end{aligned}$$

Now, the expected average energy is the expected metric of a decoded path divided by the length of the path. Using the bounds on the expected metric yields

$$\begin{aligned}\frac{1}{i}\sum_{k=0}^{i-1} E[\text{normalization value at time } k] &\leq E[\text{Average Energy}] \\ &\leq \frac{6}{i} + \frac{1}{i}\sum_{k=0}^{i-1} E[\text{normalization value at time } k].\end{aligned}$$

Taking the limit as $i \to \infty$, the bounds converge. Thus, the expected average energy of $\mathcal{C}_f^{\alpha=1}$ is computed using the steady state probabilities at each state and the normalization value used in each transition out of every state.

Using the VA FSM, the steady state probabilities can be found for each state assuming all inputs are equally likely, *i.e.*,

$$p(x_k^{(f)}) = \frac{1}{4}.$$

Assuming all inputs are equally likely implies that all transition are equally likely and allow the steady state probabilities $p_\sigma(\sigma_0)$ to be calculated. These steady state probabilities were computed rather easily. For compactness, the full 33 state VA FSM and the steady state probabilities associated with all 33 states is will not be explicitly given here.

Finding $p_{x|s}(x_0|s_0)$ is a difficult problem. However, the constellation $\mathcal{C}_f^{\alpha=1}$ can be found using the method described in Section 3.2.3. The constellation $\mathcal{C}_f^{\alpha=1}$ is the union of the four constellations $\mathcal{C}_f^{s_0}$ where $s_0 \in [\Lambda_c/\Lambda_c']$ shown in Figure 3-10. Labeled next to each point is the input which generated the point in $\mathcal{C}_f^{s_0}$. The complete, unscaled base constellation $\mathcal{C}_f^{\alpha=1}$ is shown in Figure 3-11.

Because both $p_s(s_0)$ and $p_x(x_0)$, where $x_0 \in \mathcal{C}_f^{\alpha=1}$, are difficult to compute, em-

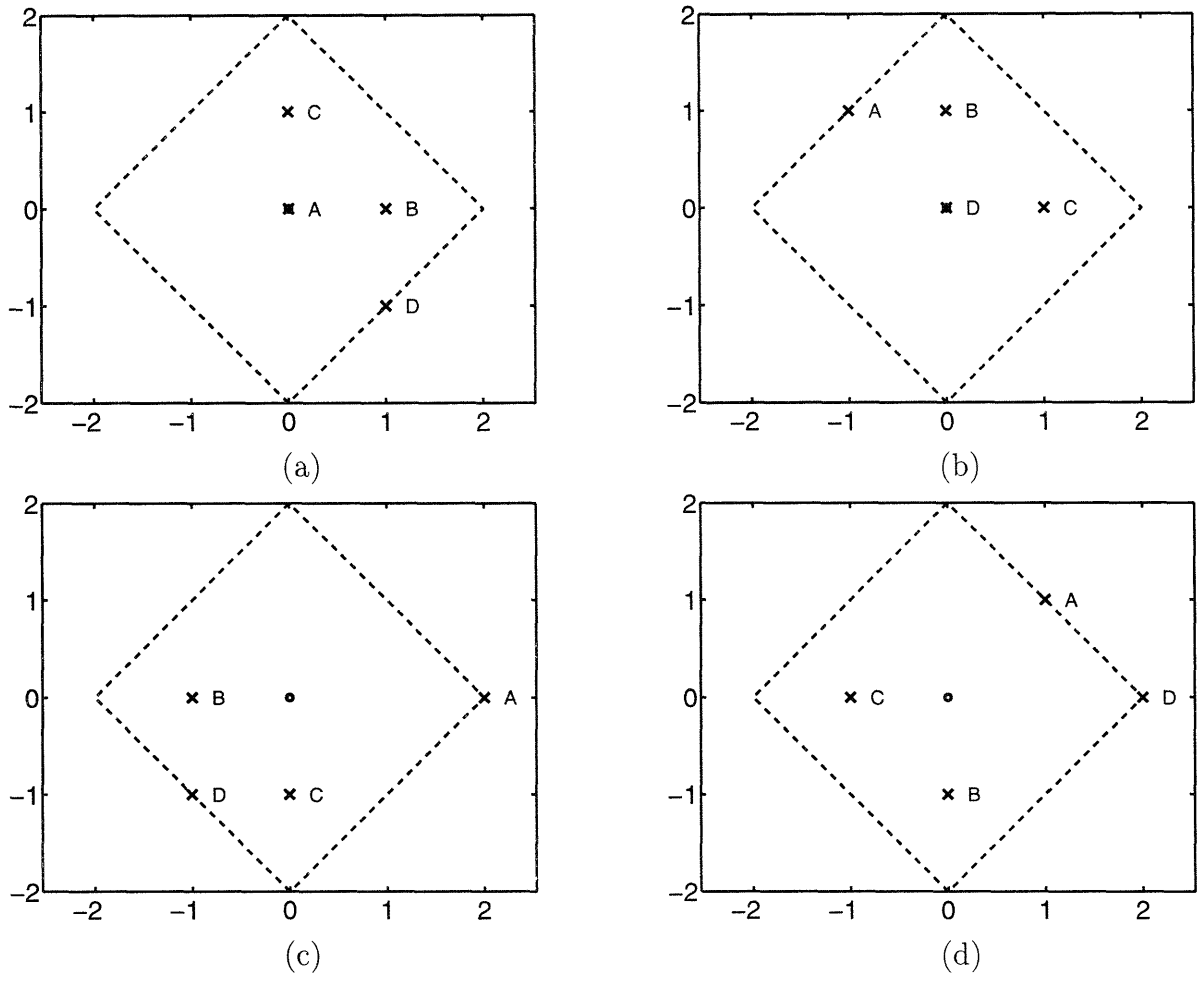


Figure 3-10: (a) $\mathcal{C}_f^E$ (b) $\mathcal{C}_f^F$ (c) $\mathcal{C}_f^G$ (d) $\mathcal{C}_f^H$

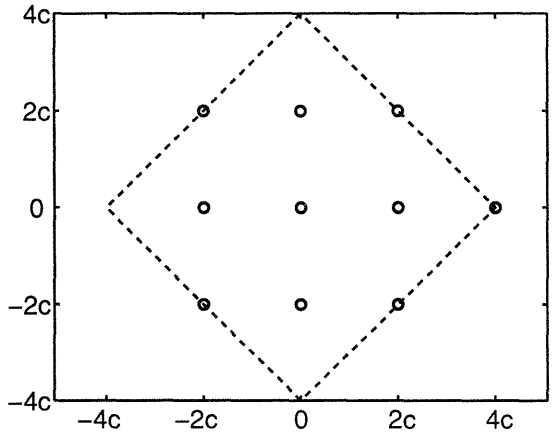


Figure 3-11: $\mathcal{C}_f^{\alpha=1}$ constellation

perical methods were used to determine these distributions. The values of $p_s(s_0)$ were

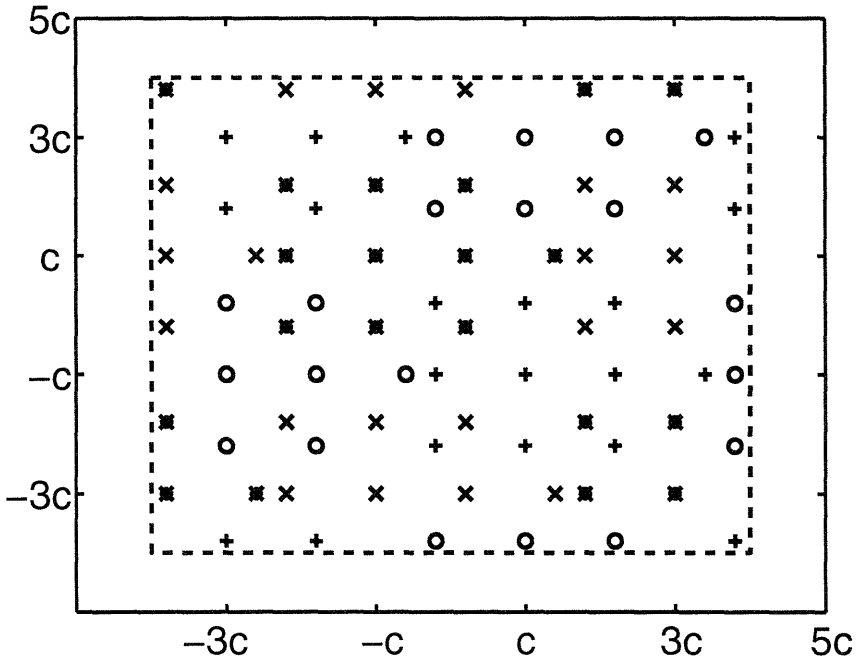
$$p_s(s_0) = \begin{cases} 0.27 & \text{if } s_0 = E, \\ 0.53 & \text{if } s_0 = F, \\ 0.10 & \text{if } s_0 = G, \\ 0.10 & \text{if } s_0 = H, \end{cases}$$

and the values for $p_x(x_0)$ were determined to be:

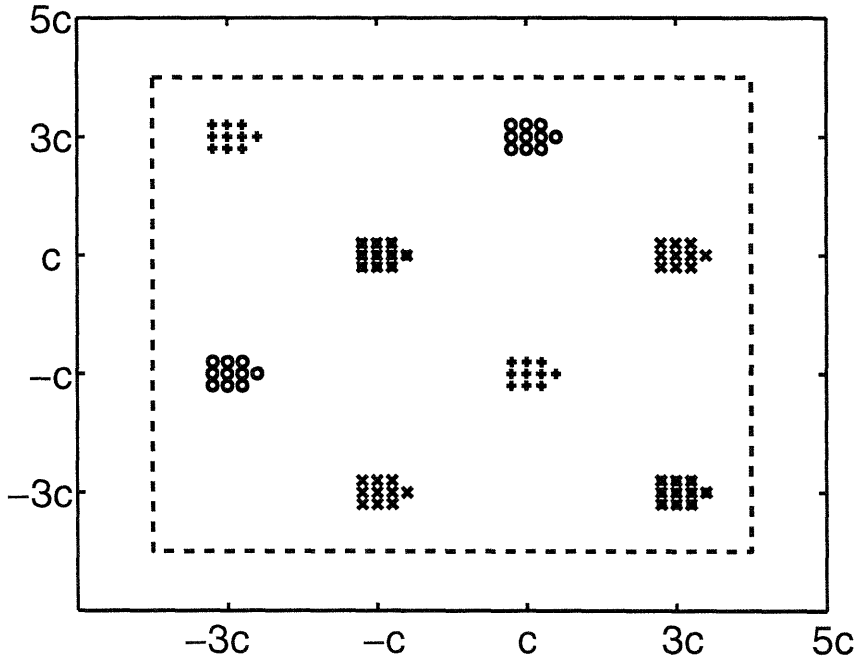
$$p_x(x_0) = \begin{cases} 0.60 & \text{if } x_0 = (0,0), \\ 6.3 \times 10^{-2} & \text{if } x_0 = (1,0), \\ 1.3 \times 10^{-2} & \text{if } x_0 = (1,-1), \\ 6.9 \times 10^{-2} & \text{if } x_0 = (0,-1), \\ 4.4 \times 10^{-2} & \text{if } x_0 = (-1,-1), \\ 6.2 \times 10^{-2} & \text{if } x_0 = (-1,0), \\ 2.1 \times 10^{-2} & \text{if } x_0 = (-1,1), \\ 7.0 \times 10^{-2} & \text{if } x_0 = (0,1), \\ 5.1 \times 10^{-2} & \text{if } x_0 = (1,1), \\ 7.0 \times 10^{-3} & \text{if } x_0 = (2,0). \end{cases} \tag{3.56}$$

Assuming each point in $\mathcal{C}_c$ is equally likely ($p_{y^{(c)}}(y_i^{(c)}) = \frac{1}{8}$), Equation (3.20) is used to yields the PMF for the unscaled version of $\mathcal{C}_{\text{UEP}}$.

Scaling the constellation $\mathcal{C}_f^{\alpha=1}$ shown in Figure 3-11 by α yields $\mathcal{C}_f^{\alpha}$ which is then centered around each $y_i^{(c)} \in \mathcal{C}_c$. Finally, all points which are mapped outside of $\boldsymbol{\mathcal{R}}_s$ are moved into the region by applying the modulo operation, thus yielding $\mathcal{C}_{\text{UEP}}$. The constellations corresponding to $\alpha = 0.6$ and $\alpha = 0.1$ are shown in Figure 3-12.



(a)



(b)

Figure 3-12: Constellations $\mathcal{C}_{\text{UEP}}$ for (a) $\alpha = 0.6$ (b) $\alpha = 0.1$

3.5.2 Performance Analysis for Example Code

From Section 3.4.2, the following quantities are computed for the 4×4 shaping region $\mathcal{R}_s$ centered at the origin:

$$V(\mathcal{R}_s) = 16 \tag{3.57}$$

$$E_{\mathcal{C}_{\text{UEP}}} = \frac{1}{16} \int_{\boldsymbol{x} \in \mathcal{R}_s} \|\boldsymbol{x}\|^2 d\boldsymbol{x} = \frac{8}{3}. \tag{3.58}$$

As expected, $\gamma_s = 1$ since there is no gain due to the choice of the shaping region $\mathcal{R}_s$. Instead of using the continuous approximation which works well on large signal sets, a second approximation is used. This approximation assumes

$$E_{\mathcal{C}_{\text{UEP}}} = E_{\mathcal{C}_c} + E_{\mathcal{C}_f^\alpha}$$

where $E_{\mathcal{C}_c}$ is the average energy of the coarse constellation and $E_{\mathcal{C}_f^\alpha}$ is the average energy of the fine cloud constellation centered at the origin. From $\mathcal{C}_c$ in Figure 3-7(b), the average energy is

$$E_{\mathcal{C}_c} = 10c^2 = \frac{5}{2}.$$

For $\mathcal{C}_f^\alpha$, the average energy calculation comes from the VA FSM. Thus,

$$E_{\mathcal{C}_f^\alpha} = \alpha^2 E_{\mathcal{C}_f^{\alpha=1}}.$$

Thus, as a second approximation,

$$E_{\mathcal{C}_{\text{UEP}}} \approx \frac{5}{2} + \alpha^2 E_{\mathcal{C}_f^{\alpha=1}}.$$

For the continuous approximation, there is no dependence on α since it is assumed that points are uniformly distributed throughout $\mathcal{R}_s$. Table 3.2 summarizes the estimated average energies versus the simulated or exact constellation energies for $\alpha = 0.8$.

The Peak-to-Average ratios (PAR) of the UEP constellation require the maximum

Constellation	Continuous Approx.	Second Approx.	Exact/Simulated Energies
$\mathcal{C}_f^{\alpha=1}$	0.6667	—	0.6132
$\mathcal{C}_c$	2.6667	—	2.5
$\mathcal{C}_{\text{UEP}}$	2.6667	2.89	2.52

Table 3.2: Comparison of true and estimated constellation energies ($\alpha = 0.8$)

	Method of Average Energy Calculations			
Constellation	Continuous Approx.	Second Approx.	Exact Energies	Exact PAR
PAR($\mathcal{C}_f^{\alpha}$)	7.78dB	—	8.14dB	8.14dB
PAR($\mathcal{C}_{\text{UEP}}$)	4.77dB	4.42dB	4.99dB	3.64dB

Table 3.3: Comparisons of PAR's for $\mathcal{C}_f^{\alpha}$ and $\mathcal{C}_{\text{UEP}}$ ($\alpha = 0.8$)

energy point of $\mathcal{C}_{\text{UEP}}$:

$$E_{\max}(\mathcal{C}_{\text{UEP}}) = \max_{x \in \mathcal{C}_{\text{UEP}}} \|x\|^2.$$

Finding the peak energy for $\mathcal{C}_{\text{UEP}}$ involves modulo $\boldsymbol{\mathcal{R}}_s$ mappings of each scaled $\mathcal{C}_f^{s_0}$ around every superpoint. The modulo operation confines all points to lie within $\boldsymbol{\mathcal{R}}_s$. Thus, $E_{\max}(\mathcal{C}_{\text{UEP}}) \leq E_{\max}(\boldsymbol{\mathcal{R}}_s)$ where equality holds if $\alpha = 1$ for the constellations in this example. Using $E_{\max}(\boldsymbol{\mathcal{R}}_s)$ to estimate the peak power,

$$E_{\max}(\mathcal{C}_{\text{UEP}}) = 8. \tag{3.59}$$

Using both the true peak value and the estimated peak value from Equation (3.59) yields Table 3.3 for the average energies in Table 3.2.

Since Ω_f is 4-QAM, $d^2_{\text{gain},\Omega_f} = 1$. From Equation (3.21),

$$d^2_{\Omega_f} = d^2_{\Lambda_f} = 1,$$

and from Equation (3.22), the fine code distance is

$$d_f^2 = \alpha^2. \tag{3.60}$$

For Ω_c, $d^2_{\text{gain},\Omega_c} = 4$ and from Equation (3.24),

$$d^2_{\Omega_c} = 4d^2_{\Lambda_c} = 8.$$

Assuming the worst case geometry for the intercloud minimum distance, Equation (3.26) yields

$$d_c^2 = 8(1-\alpha)^2 + \alpha^2. \tag{3.61}$$

The coding gains for the fine and coarse code are calculated with the 16-QAM as the base constellation. For the 16-QAM constellation, the squared minimum distance and average energy are

$$\begin{aligned} d^2_{16\text{-QAM}} &= 1, \\ E_{16\text{-QAM}} &= \frac{5}{2}. \end{aligned}$$

Thus, the coding gains for the fine and coarse code are

$$\begin{aligned} \gamma_f &= \frac{d_f^2}{d^2_{16\text{-QAM}}} \cdot \frac{E_{16\text{-QAM}}}{Ec_{\text{UEP}}}, \\ \gamma_c &= \frac{d_c^2}{d^2_{16\text{-QAM}}} \cdot \frac{E_{16\text{-QAM}}}{Ec_{\text{UEP}}}. \end{aligned}$$

The coding gains are summarized in Table 3.4 for each of the average energy estimates and for $\alpha = 0.8$ and $\alpha = 0.3$. A more extensive analysis of these results is given in Chapter 5.

		Method of Average Energy Calculations		
Coding Gain	α	Continuous Approx.	Second Approx.	Exact Energies
γ_f	0.8	-2.22dB	-2.57dB	-2.01dB
γ_c	0.8	-0.46dB	-0.81dB	-0.25dB
γ_f	0.3	-10.74dB	-11.09dB	-10.53dB
γ_c	0.3	5.75dB	5.40dB	5.97dB

Table 3.4: Coding Gains for $\alpha = 0.8$ and $\alpha = 0.3$

Chapter 4

Implementation

The complexity of the UEP code described in Chapter 3 creates several implementation issues. The Viterbi Algorithm (VA) is generally used for maximum likelihood sequence decoding (MLSD). However, for the simple example of Section 3.5, the VA FSM described in Section 3.5.1 with minor modifications can be used instead of the VA in both the encoder and decoder for the UEP code. The VA FSM used in the encoder is discussed in further detail in Section 4.1. The encoder implementation is discussed in Section 4.2. The stack algorithm used to decode the UEP trellis code and the implementation of the stack algorithm are described in Section 4.3. Finally, extensions to more complicated codes are described in Section 4.4.

4.1 The Viterbi Algorithm Decoder FSM

The shaping of the fine code by the coarse code creates the need for maximum likelihood sequence decoding of the fine code using the coarse code trellis code. Typically, the Viterbi Algorithm (VA) is used to perform the sequence decoding. For a code with N states, the Viterbi Algorithm requires N floating point weights and N output symbols to be stored at every time i. For the special case where the number of coarse code states N is small and there is a finite set of squared distances between points in $\mathcal{C}_f$ and $\mathcal{C}_c$, the shaping decoder can be implemented using an FSM, called the VA FSM described briefly in Section 3.5.1. The advantages of using a VA FSM are the

reduction in storage needed during decoding and the improvement of decoding speed.

The VA FSM decodes the maximum likelihood sequence of Ω_c given an input sequence from Ω_f. At each time i, the VA FSM takes a symbol from $\mathcal{C}_f$ and updates a list of paths and their associated weights. When the last input symbol is reached, the path with the minimum weight is selected as the maximum likelihood sequence and the list of output symbols associated with that path is the decoded sequence.

Each state of the VA FSM is partially identified by an N-tuple of normalized path weights. A set of N lists of previously decoded coarse symbols (paths) is also maintained. The $N-tuple$ and the associated N lists specify the state of the VA FSM. An infinite number of states exists because of the N lists of arbitrary length associated with each N-tuple. Ignore these lists creates a finite number of states, thus the term VA FSM is used. The component of the N-tuple equal to zero identifies the current minimum weight path. Decoding the received symbol at time $i+1$ involves updating the weights and paths of the VA FSM. At time $i+1$, a transition metric is calculated for each transition into the coarse code FSM state σ_j $(1 \leq j \leq N)$. This transition metric is the squared distance between the input symbol and the transition symbol. Each transition metric is added to the metric of the state from which the transition originated. The minimum transition metric sum is selected and the associated transition symbol is placed onto the list corresponding to state σ_j. The transition metric sum is added to the N-tuple entry corresponding to σ_j. This process is repeated until all transitions have been encoded and tested, *i.e.*, all N states of the coarse code trellis have been entered. Then, the N-tuple is renormalized so that the minimum entry is zero. The renormalized N-tuple identifies a new state in the VA FSM. A one-to-one correspondence between the N-tuples and VA FSM state numbers can be generated. This correspondence list grows rapidly as the number of coarse code trellis states, input bits, or the fine constellation size increases.

Since there are N states in the coarse trellis, only N survivor paths need to be kept at any time. Each path represents a hypothesized codeword from Ω_c used for shaping Ω_f. A transition into σ_j from σ_l generates a symbol in $\mathcal{C}_c$ and appends the symbol the path associated with σ_l. This new extended path becomes the path for

σ_j at time $i+1$. If $l \neq j$, then the path of σ_l up to time i becomes the path of σ_j up to time i. This path switch can be implemented by either copying the entire path of length i over the current path of σ_j or by using a pointer in σ_j's path indicating the parent path (σ_l's path). Obviously, the pointers save large amounts of computation over copying i elements during each step, but the pointers require extra storage since every entry of a path consists of both a symbol and a pointer to its parent path.

The VA FSM allows the encoder described in the next section to keep track of the fine code shaping by keeping only the VA FSM state number and the shaping codeword paths. For the UEP trellis code example in Section 3.5, the coarse code trellis has 4 states, thus a 4-tuple identifies each VA FSM state. Four separate paths are maintained at each time, one for each trellis state. The VA FSM has 33 distinct 4-tuples, thus, 33 states and requires only 6 bits for the VA FSM state number. If a normal VA were used, each state would be represented by 4 double precision values (representing the unnormalized path weights) and would require 32 bytes per step. Thus, the storage required for each stack entry becomes 3 bits instead of 32 bytes, a rather large reduction in storage when encoding long sequences.

As the number coarse code FSM states increases, the number of VA FSM states increases rapidly and this scheme becomes infeasible. For instance, an 8 state trellis requires 683 VA FSM while a 16 state trellis requires 31607 states. A 16 state trellis requires roughly 316K bytes to store just the one-to-one correspondence between 16-tuples and state numbers and then an additional 2 bytes per step. Another drawback caused by a larger number of VA FSM states is the search time required to map an N-tuple to a VA FSM state number. As the number of states increase, the search time also increases.

4.2 The Encoder

The UEP trellis encoder consists of two finite state machines (one for Ω_c and one for Ω_f) and a shaping decoder which decodes the sequence from Ω_f to the trellis of Ω_c. At each time i, the encoder for Ω_f takes inputs $x_i^{(f)}$ and selects a symbol from

$\mathcal{C}_f$. Given the state of Ω_f's FSM at time $i-1$, the $x_i^{(f)}$th transition leaving the state is selected. A similar process is followed when encoding $x_i^{(c)}$ with the FSM for Ω_c. Thus, the states of the FSM's for Ω_f, Ω_c, and the shaping decoder contain all the information needed to encode future inputs and must be updated at each time i.

The encoded symbol $y_i^{(f)} = \mathcal{C}_f(x_i^{(f)})$ is decoded using a shaping decoder. This shaping decoder is implemented using the VA FSM described in Section 4.1. At time i, there are N paths which the VA FSM keeps as hypothesized shaped codewords $(\hat{\boldsymbol{y}}^{(c)})$. Together with $y_i^{(c)} = \mathcal{C}_c(x_i^{(c)})$, each hypothesized $\hat{y}_i^{(c)}$ specifies the mapping pair $\lambda_{\hat{y}_i^{(c)}}$ and $\lambda^{-1}_{y_i^{(c)}}$ that move the point to the base region and then to the Voronoi region of the coarse code. However, any one of the N hypothesized $\hat{y}_i^{(c)}$'s could be the correct one, thus no definite mapping can be selected yet. As the encoding continues, the hypothesized paths should merge (just like the VA) at some past time. If all N paths have merged at some time $i-\delta$, then the mappings for the symbols previous to time $i-\delta$ are fixed regardless of which of the N paths at time i is the correct path. Thus, the final encoded symbols for times prior to $i-\delta$ can be generated. The delay between the encoder inputs and the encoder outputs is δ. Implementing the VA FSM with pointers allows a simple search from time i backwards until all paths point to the same parent path. The first time when all parent paths agree is the time $i-\delta$.

4.3 The Stack Decoder

Due to the complexity of the UEP trellis code, a sequential decoder is used. The advantage of a stack decoder over the Viterbi Algorithm (VA) is the reduction in storage requirements for codes with a large number of states. Even for a simple system like the one presented in Section 3.5, the composite UEP code has an infinite number of total states as previously discussed in Section 4.1. The symbols in the N paths from time $i-\delta$ to i are undetermined and thus represent a potentially unbounded number of states. The Viterbi Algorithm requires a constant amount of computation per input symbol, but requires all hypothesized paths to be stored, thus an exponential growth of memory with an increasing number of states. The stack

decoder trades off computation for storage, thus the storage grows only slowly with the number of states, but the computation is no longer constant per input symbol.

The stack algorithm was briefly described in Section 3.2.1 and will be described in further detail here. The stack decoder operates by extending a single hypothesized codeword (represented as a path) until either the received sequence is decoded or the path becomes inferior to the other available paths. The list of available paths is called the stack and every entry in the stack represents a different path. Each entry has a decoding metric and the stack is kept in ascending order by the metrics. The top entry of the stack has the lowest metric while the bottom entry has the largest metric. Removing the top entry from the stack is called popping while placing an element onto the stack is called pushing. Every entry in the stack may have a different length.

The stack decoder operates by popping the stack to get a parent path of length i and extending the parent path to length $i+1$. The extension is performed by encoding each of the $2^{k_c+k_f}$ inputs $\{x^{(c)}\}$ and $\{x^{(f)}\}$. The extension can be viewed as creating $2^{k_c+k_f}$ child paths from the parent path. Each child path involves encoding a specific $x_i^{(c)}$ and $x_i^{(f)}$ given the state of the parent path. From Section 4.2, the state of the parent path is summarized by the state number of the FSM's used for Ω_c and Ω_f, the state number of the VA FSM, the N hypothesized shaping codewords (VA FSM paths), and N decoding metrics described below. Encoding $x_i^{(f)}$ and $x_i^{(c)}$ results in N possible output UEP symbols generated from mappings selected by the ith component of the N hypothesized shaping codewords.

Associated with each of the N VA FSM paths is a decoding metric. This decoding metric represents the squared distance between a hypothesized UEP codeword and the actual received sequence. At each step, the squared distance between the N possible UEP symbols and the received symbol is computed and added to the decoding metric of the VA FSM path which selected the mapping function used to generate the UEP symbol. Notice that the decoding metric is different from the metric used in the VA FSM. This metric measures the distance of the *final* codeword from the received codeword and is dependent on channel noise. The metric used in the VA FSM measures the distance between the fine code symbols and the coarse code symbols in

the absence of noise.

After computing the decoding metrics, the stack decoder must push the new child path onto the stack. An entry metric must be selected so the child path can be ordered properly in the stack. The entry metric should indicate the path's likelihood to be the correct final sequence. Due to the delay in the encoding, the decision of a correct path is impossible. However, a tentative decision can be made by using the minimum decoding metric as the entry metric. Thus, the stack algorithm implemented here uses the minimum decoding metric over the N metrics of the survivor paths to select the path to grow at every time. This metric is not guaranteed to be the metric associated with the eventual correct path at any given time i. The hope is that if the stack decoder pursues an incorrect path, the minimum decoding metric of the incorrect path will eventually exceed the minimum decoding metric of the correct path and the stack decoder will start to grow the correct path.

The decoding metrics must be compensated for the path length since longer paths will tend have a higher metric than shorter paths, even if the longer path is correct. The constant correction factor used is the same correction factor used in the Fano metric [9]. The correction factor used at each step is $R\sigma^2$, where R is the information rate per N dimensions and σ^2 is the noise variance per N dimensions. The correction factor is subtracted from the path metric at each step.

Notice, the stack algorithm could potentially require large amounts of storage as the length of the input sequence grows large. However, the entries at the bottom of the stack are less likely to be the correct path. Truncating the stack to a fixed depth is reasonable if the stack capacity is relatively large, thus insuring that entries at the bottom have a very low probability of being the correct path. Problems arise when the branching factor at each step is large and the stack decoder pursues an incorrect path which closely resembles the correct path during a segment. In such a situation, the correct path may be purged from the stack if the branching factor causes too many entries to be generated before the incorrect path metric grows larger than the metric for the correct path. The minimum stack size should be large enough to accommodate at least $\delta \cdot 2^{k_c+k_f}$ entries where δ is the average decoding delay of the

shaping decoder. A stack size of 10,000 entries was used throughout the simulations of the UEP trellis code scheme presented in Section 3.5.

One problem with decoding the UEP trellis codes is the possibility of long error sequences. In particular, there exists long sequences of "don't care" symbols. These sequences will have a path weight which is very close to the correct path's weight. Thus, the decoder could continue to pursue the long "don't care" sequences. These sequences can increase the decoding time since the decoder will basically perform a exhaustive search over all branches for each symbol in the don't care sequence. The problem of long don't care sequences arises because some symbols lie exactly on the boundary of a Voronoi region. To solve the problem of long don't care sequences, the encoder and decoder can dither the fine code constellation by a pseudonoise sequence. Dithering involves translating the origin of the fine code sequence by a vector determined from the pseudonoise sequence.

Another computational difficulty with the stack algorithm is the sorting of the stack nodes. As the stack size grows, the sort algorithm becomes slower and slower. One solution to this problem is to divide the stack into fixed size "bins." Each bin contains all the entries which have a metric between the bin weight and the bin weight plus bin width. If the bins are arranged in an array, they can be indexed directly knowing the weight of the entry to be inserted and the minimum bin weight. The number of bins in the array is fixed, thus a maximum acceptable weight for the stack is set to the minimum bin weight plus the number of bins times the bin width. The bins must also be kept in sorted order. Notice that for every pop of stack, there are $2^{k_c+k_f}$ pushes. Therefore, sorting the bins after each push requires $2^{k_c+k_f}$ times more effort than sorting before each pop.

Notice, path weights can become negative due to the constant correction factor. Thus, new bins are created if the weight of the entry to be added exceeds the current bottom bin weight *or* is smaller than the current top bin weight. Bins can also be deleted if a bin becomes empty or if the maximum stack size is exceeded. The maximum stack size is exceeded when the total number of entries in all bins exceeds the stack size. In this case, the worst weight bin (bottom bin) is deleted and the

stack size is checked again. If the stack size is still too large, then the next worst bin is removed. This process continues until the stack size is less than the maximum allowable size. Empty bins are generally removed from the top of the stack as the bins are emptied by continual popping from one bin and pushing into other bins. Thus, the weight of the top bin and bottom bin are dynamic. The maximum number of bins should be selected provide enough dynamic range in the best and worst bin weights. The maximum number of bins used in the simulations was 100,000.

The bin width is also a critical parameter. If the bin width is too large compared to the noise variance, then all the stack entries may fall into one bin since their weights are not variable enough to require another bin. If the maximum stack size is exceeded, the single bin which contains *all* the entries will be deleted, leaving no entries in the stack to decode. This situation should be avoided. Depending on the noise variance and minimum distances of the codes Ω_c and Ω_f, bin widths of 0.1, 0.05, and 0.02 were used in the simulations.

Notice that the algorithm is only concerned with the *best* node. Therefore, if the maximum number of bins and bin width are selected properly and if the top bin is kept sorted, then the bin solution is essentially the same as sorting the entire stack but will be much faster. By selecting small bin widths, each bin tends to contain fewer elements and sorting the top bin becomes easier. However, the small bin width implies a large number of bins to maintain a large overall dynamic range of metrics in the stack.

The final parameter of the stack algorithm is the correction factor used in computing the decoding metrics. Intuitively, this factor represents the expected rate of increase of the correct path metric. By using a factor of $R\sigma^2$, the expected metric of the correct path is zero. If the correction factor is set slightly higher than the minimal value, $R\sigma^2$, then the metric of the correct path tends to become negative, thus becoming more distinguishable from the incorrect path metrics. Larger factors are considered more aggressive because the stack decoder will tend to grow a single path longer rather than switching to another path. However, if the factor is too aggressive, an incorrect path can be completely decoded and the incorrect path metric

will still be less than the metric of the correct path which was never grown. A careful choice of the correction factor should be made to avoid decoding errors cause by the implementation of the stack decoder.

Throughout the simulations, the aggressiveness of the correction factor drastically affected the computation required to decode the UEP trellis code. For an input of length L, a minimum of L steps are needed to decode the input if the correct path is selected at first and is the only path grown. With the correction factor set to 10% above the minimum, the average number of steps taken by the stack decoder was $20L$. By increasing the correction factor to roughly 40% above the minimum, the average number of steps taken was only $3.4L$. However, the probability of bit errors increased as the correction factor became more aggressive.

4.4 Future Implementations of Sequential Decoders

Using more complicated codes for Ω_c and Ω_f does not drastically affect the operation or implementation of the encoder. The main difference would be the inability to exploit the VA FSM. The VA FSM becomes infeasible if the number of VA FSM states grows too large, or if the symbols in $\mathcal{C}_f$ do not have a finite set of distances from all the symbols in $\mathcal{C}_c$. The latter case arises if the fine code constellation is dithered by a pseudonoise sequence. The dithering creates a time varying constellation which will not have a finite set of distances with a fixed $\mathcal{C}_c$ over all time. Thus, the encoder must keep the actual N unnormalized metrics in the shaping decoder. This is the only modification of the encoder which must be made for more complex codes.

Although the stack algorithm works well with the simple example in Section 3.5, the stack decoder is inherently limited by the branching factor at each step, $2^{k_c+k_f}$, and the decoding delay of the shaping code, δ. For a large number of input bits and a complex shaping code, the quantity $\delta \cdot 2^{k_c+k_f}$ will become very large, thus requiring too much storage. Again, more complex codes or a lack of a finite set of distances from the coarse symbols renders the VA FSM unusable. Thus, the storage per entry of the stack would further increase.

Instead of using a stack algorithm, a constant storage algorithm such as the Fano algorithm could be used to decode the UEP trellis code. The Fano algorithm further trades off computation for storage. With the Fano algorithm, only a single path of variable length needs to be stored at any time. The Fano algorithm pursues a path until the path metric exceeds some threshold. At that time, the Fano algorithm backs up on the path and pursues a different branch. This process continues until the path which is being extended has the same length as the received sequence.

Chapter 5

Results

Both Wei and Calderbank have proposed various schemes for unequal error protection. All their schemes deal with two separate levels of data, more important data and less important data. In Section 5.1, the unequal error protection schemes presented by Wei in [10] are discussed and analyzed. Section 5.2 gives an example of Wei's code which is the parallel of the UEP trellis code from Section 3.5. In Section 5.3, the schemes presented in [1] by Calderbank are discussed. In Section 5.4, the results of simulations for the UEP trellis code are compared to results of Wei's and Calderbank's codes.

5.1 Wei's Coding Schemes

Wei presents a series of coding schemes all based on independent coding of the coarse and fine bits. The fine bits and coarse bits are encoded separately using trellis codes. The output of the two encoders are non-overlapping and select points from a constellation. The constellation consists of symmetric clouds of points centered at each of the coarse code points. Wei uses the constellation twice for every input, thus the final output symbol is in four dimensions.

The structure of Wei's code can be described in the framework of the UEP trellis code from Chapter 3. The fine code, Ω_f, takes k_f bits in and produces n_f bits which select points from a lattice Λ_f. The points selected by Ω_f form a fine constellation

$\mathcal{C}_f$. The fine code encodes the sequence $\boldsymbol{x}^{(f)} = \{x_i^{(f)}\}$ to form the sequence $\boldsymbol{y}^{(f)} = \Omega_f(\boldsymbol{x}^{(f)})$, *i.e.*, $y_i^{(f)} = \mathcal{C}_f(x_i^{(f)})$. A coarse code, Ω_c, takes k_c bits in and produces n_c bits that select points from the lattice Λ_c, a sublattice of Λ_f. The points selected by Ω_c form the coarse constellation $\mathcal{C}_c$. The coarse code encodes the sequence $\boldsymbol{x}^{(c)} = \{x_i^{(c)}\}$ to produce the sequence $\boldsymbol{y}^{(c)} = \Omega_c(\boldsymbol{x}^{(c)})$, *i.e.*, $y_i^{(c)} = \mathcal{C}_c(x_i^{(c)})$.

The fine code is not shaped by the coarse code ($r_c = 0$) and therefore, the fine code does not need to be decoded to the coarse trellis. The mapping $\boldsymbol{\lambda}_{\hat{\boldsymbol{y}}^{(c)}}(\cdot)$ is just the identity. The inverse mapping $\boldsymbol{\lambda}^{-1}_{\boldsymbol{y}^{(c)}}(\cdot)$ is just a translation from the origin to the sequence $\boldsymbol{y}^{(c)}$, *i.e.*, $\lambda^{-1}_{y_i^{(c)}}(x) = x + y_i^{(c)}$. Wei maximizes the coarse code distance by arranging $\mathcal{C}_f$ and $\mathcal{C}_c$ so that the geometry between points is always the best case geometry of Section 3.4. That is, the points are always perpendicular to the boundaries of the lattice Voronoi regions. To achieve this property, the coarse code lattice Λ_c must have a cubic Voronoi region and the fine code lattice, Λ_f, must be a partition of the integer lattice Z^N. A specific choice of Λ_c which yields a cubic Voronoi region is the scaled integer lattice. Therefore, the partition chain used for selecting the coarse and find lattices must be $Z^N/\cdots/\Lambda_f/\cdots/2^l Z^N$ where $l \in Z$.

Results from the UEP trellis code (Section 3.4) with $r_c = 0$ can be applied to Wei's codes. One major difference between the analyses is that the coarse code distance d_{Ω_c} of Wei's code can be calculated exactly. Since the codes are linear and decoupled and the fine constellation is not shaped by the coarse constellation, the minimum distance gained along an error path $\boldsymbol{p}_e$ can be computed directly from the constellations. The last difference between Wei's code and the UEP trellis code of Section 3.2 is the interpretation of the scaling factor α. Since no overlap exists in Wei's code, Wei's code does not reduce to the fine code constellation at $\alpha = 1$. Wei's scaling factor can conceptually be greater than 1, representing an expansion of each cloud around the coarse points. For the sake of comparison, if Ω_c and Ω_f are identical for both an overlapping UEP trellis code in R^2 and Wei's UEP code in R^4, the relation between Wei's alpha (α_{Wei}) and the UEP overlap alpha (α) is $\alpha_{\text{Wei}} = 2^{\frac{r_c}{2}}\alpha$. With this normalization, both codes have the same fine coding gain.

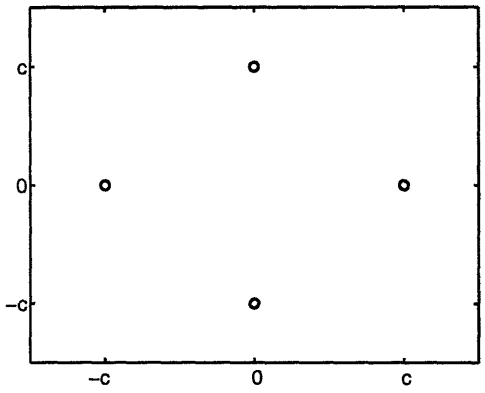


Figure 5-1: The constellation $\mathcal{C}_f$ for Wei's example code

5.2 An Example of Wei's Code

Using the same codes for Ω_f and Ω_c as in the UEP trellis code example of Section 3.5, Wei's code can be constructed and its performance compared to the UEP trellis code. The code Ω_f is still the uncoded 4-QAM which takes $k_f = 2$ bits per symbol. The lattice Λ_f is the checkerboard lattice with $d^2_{\Lambda_f} = 1$ and since $k_f = 2$, the coarse lattice is $2^{k_f - r_c}\Lambda_f = 4\Lambda_f$ which is also a checkerboard lattice with $d^2_{\Lambda_c} = 4$. The four points taken from Λ_f form the constellation $\mathcal{C}_f$ shown in Figure 5-1. The coarse code is a sublattice of Z^2, thus satisfying the perpendicular boundary condition. The coarse constellation is formed from the $2^{n_c} = 8$ points of Λ_c which are selected from $\Lambda_c \bigcap \mathcal{R}_s$ where $\mathcal{R}_s$ is a square shaping region. Since the checkerboard lattice is used instead of the simple integer lattice, the bounding region $\mathcal{R}_s$ must be $\sqrt{2}$ times larger for this example[1]. The constellation $\mathcal{C}_f$ is mapped around every point in $\mathcal{C}_c$ modulo $\mathcal{R}_s$ to form the constellation $\mathcal{C}_{\text{UEP}}$ shown in Figure 5-2 for $\alpha_{\text{Wei}} = 1.0$ and $\alpha_{\text{Wei}} = 0.5$.

The minimum distance of the coarse code can be computed directly from the trellis in Figure 3-7(a) and the constellation $\mathcal{C}_{\text{UEP}}$ for Wei's code shown in Figure 5-2. Since Ω_c is linear, the minimum distance path can be computed by comparing the distance gain of the $G \rightarrow F \rightarrow G$ error path which splits from the $E \rightarrow E \rightarrow E$ path. The minimum parallel transition distance must also be considered since $2^{k_c^u} = 2$ points are chosen from every coset. The parallel transition error event is when the "wrong"

[1]In general, the bounding box and energy would have been held constant by choosing the distance of the checkerboard lattice so that $d^2_{\Lambda_f} = \frac{1}{\sqrt{2}}$.

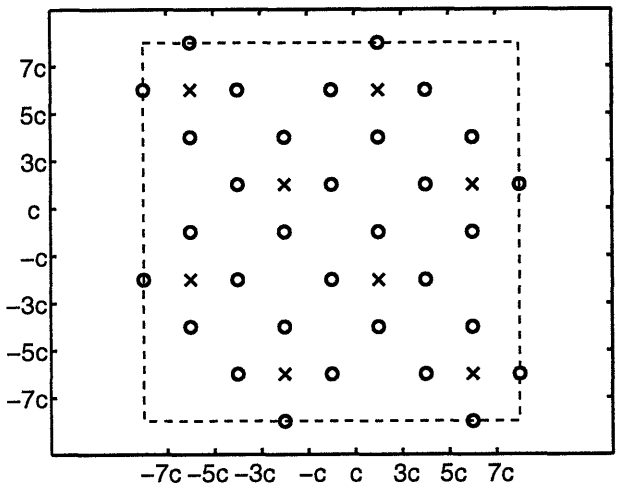


Figure 5-2: Wei's UEP constellation with $\alpha_{\text{Wei}} = 1.0$

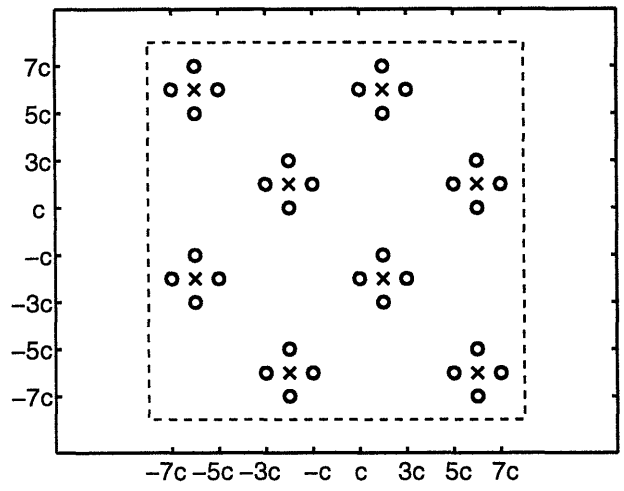


Figure 5-3: Wei's UEP constellation with $\alpha_{\text{Wei}} = 0.5$

point from the correct coset is chosen.

First analyzing the $G \rightarrow F \rightarrow G$ error path, the minimum distance can be found by comparing any two minimally spaced points centered around the coarse points labeled E, G, and F. For the distance gained from decoding a G instead of an E, the geometry of case 1 with $\alpha = 1$ from Section 3.4 applies and the gain is

$$d^2_{E,G} = 2d^2_{\Lambda_f} = 16c^2. \tag{5.1}$$

The distance gain when decoding F instead of E also uses the geometry of case 1 with $\alpha = 1$ and is

$$d^2_{E,F} = d^2_{\Lambda_f} = 8c^2. \tag{5.2}$$

Now, using Equations (5.1) and (5.2), the minimum distance of the error path $G \rightarrow F \rightarrow G$ is $40c^2$.

The parallel transition distance is the minimum distance between two points from the clouds of two different coarse points in the same coset, *i.e.*, the two points labeled E, E or G, G, etc. The minimum parallel transition distance is

$$d^2_{\|} = 4d^2_{\Lambda_f} = 32c^2.$$

Thus, the minimum distance error event of the trellis code is the parallel transition error, and the minimum distance of Ω_c becomes

$$d^2_{\Omega_c} = d^2_{\|} = 32c^2. \tag{5.3}$$

For the minimum distance d_c of Wei's UEP code, the $G \rightarrow F \rightarrow G$ and the parallel transitions can both be analyzed using geometry 1. The following distances are computed:

$$\begin{aligned} (d^{\alpha_{\text{Wei}}}_{E,G})^2 &= 2d^2_{\Lambda_f}(2 - \alpha_{\text{Wei}})^2 \\ &= 16c^2(2 - \alpha_{\text{Wei}})^2 \end{aligned} \tag{5.4}$$

$$
\begin{aligned}
(d_{E,F}^{\alpha_{\text{Wei}}})^2 &= d_{\Lambda_f}^2(2-\alpha_{\text{Wei}})^2 \\
&= 8c^2(2-\alpha_{\text{Wei}})^2 \qquad (5.5) \\
(d_{G\to F\to G}^{\alpha_{\text{Wei}}})^2 &= 5d_{\Lambda_f}^2(2-\alpha_{\text{Wei}})^2 \\
&= 40c_{\Lambda_f}^2(2-\alpha_{\text{Wei}})^2 \qquad (5.6) \\
(d_{\|}^{\alpha_{\text{Wei}}})^2 &= 4d_{\Lambda_f}^2(2-\frac{\alpha_{\text{Wei}}}{2})^2 \\
&= 32c^2(2-\frac{\alpha_{\text{Wei}}}{2})^2. \qquad (5.7)
\end{aligned}
$$

The minimum distance d_c^2 is the minimum of the two distances given in Equations (5.6) and (5.7) yielding

$$
\begin{aligned}
d_c^2 &= \min(5d_{\Lambda_f}^2(2-\alpha_{\text{Wei}})^2, d_{\Lambda_f}^2(2-\frac{\alpha_{\text{Wei}}}{2})^2) \\
&= \min(40c^2(2-\alpha_{\text{Wei}})^2, 32c^2(2-\frac{\alpha_{\text{Wei}}}{2})^2). \qquad (5.8)
\end{aligned}
$$

For the lattices Λ_f and Λ_c in this example $c = \frac{1}{2\sqrt{2}}$. Notice that the minimum distance d_c reduces to the minimum distance d_{Ω_c} when $\alpha_{\text{Wei}} = 0$ but when $\alpha_{\text{Wei}} = 1$, the fine code is *not* the 16-QAM constellation.

Since the constellations are modulo $\mathcal{R}_s$, the average energies of the constellations were calculated numerically for each α_{Wei}. With the average energy of Wei's constellation per 2 dimension and minimum distance per 2 dimensions, the coding gains for the fine and coarse code can be calculated. As a final comment, one of Wei's main design parameters is the ratio of fine to coarse bits. This ratio, however, has been shown to be the wrong design parameter. In Section 3.4.3, the dependency was shown to lie almost entirely with the *absolute* number of fine bits, k_f. Since Wei's codes are a subset of the UEP trellis codes, the proper design parameter for his codes is also k_f.

5.3 Calderbank's UEP codes

The unequal error protection codes devised by Calderbank in [1] do not fit well into the framework of the UEP code presented here. Most of Calderbank's constellations

are rather difficult to analyze and their performance varies in a somewhat unpredictable manner. A unifying design scheme does not exist for Calderbank's codes and thus, general results are rather difficult to find. The nominal coding gains given by Calderbank also do not always reflect the code's actual performance. The difference in effective and nominal coding gains is sometimes quite large (1-2 dB at times). Like Wei, Calderbank uses the ratio of fine to coarse bits as his choice of parameters. Due to these drawbacks and the ad hoc nature of Calderbank's schemes, the UEP codes he presents are difficult to understand. Their performance may or may not be better than the UEP trellis code presented here. Since Calderbank's codes do not fit into the framework of the UEP trellis code, these codes will not be discussed. Readers interested in the coding schemes and results derived by Calderbank are referred to [1].

5.4 Results of the UEP trellis code

Some initial results such as average energies and nominal coding gains were presented for specific values of α in Section 3.5.2. In this section, a more complete treatment of the average energies, nominal coding gains, probability of error, and effective coding gain is presented with simulation results where appropriate. The nominal coding gains and average energies are also compared to Wei's and the differences are analyzed. Trellis codes are designed for infinite length sequences but these sequences are rather difficult to simulate. Blocks of length 1,100 symbols were used in the simulations where each symbol represents 2 more important bits and 2 less important bits. The last 100 symbols were discarded under the assumption that the merge depth is less than 100 symbols for the VA used to decode the UEP code. Since both the encoder and decoder were forced to a common state at the start of a block, the initial 100 symbols were not discarded. Thus, the effective block length was 1,000 symbols (4,000 bits).

The average energy of the UEP trellis code is a function of α. The average energy is rather complex to calculate and was only computable in Section 3.5.2 because the UEP code was relatively simple. As the codes Ω_c and Ω_f become larger and more

complicated, the exact energy calculations are not feasible. However, the continuous approximation becomes more accurate with large fine codes (large n_f). The average energy for the UEP trellis code from Section 3.5 is plotted as a function of α in Figure 5-4. Also shown in Figure 5-4 is the average energy for Wei's equivalent code. Notice, the two average energies have almost identical dependencies on α which is expected. The difference in average energies is a constant scale of 2 which appears because Λ_f for Wei's code was the unit distance checkerboard lattice instead of the Z^2 lattice used for the UEP trellis code.

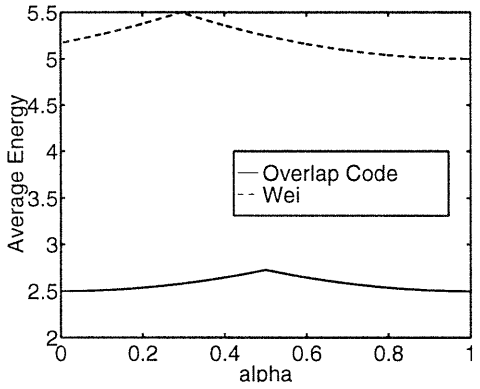


Figure 5-4: Average Energies of the UEP overlap code and Wei's code versus α

Several simulations were run to test the minimum distance of the UEP trellis code for both the fine and coarse code. Each experiment forces an error to occur in the first symbol of the sequence decoding. The length of the error path before it remerges with the correct path is the error path length while the weight accumulated during the error path is the error distance. The intracloud minimum squared distance (d_f^2) was confirmed to be 0.64, 0.25, and 0.09 for $\alpha = 0.8, 0.5$, and 0.3. The distribution of the lengths of the minimum distance error paths is shown in Figure 5-5. Notice, most of the error paths are relatively short, with the longest error event found in 1,000 blocks of 200 symbols per block being 42 symbols.

The lower bound on the intercloud distance d_c^2 was also tested with simulations of 10,000 blocks of 200 symbols. The lower bound on d_c^2 was confirmed to be 0.96, 2.25 and 4.01 for $\alpha = 0.8, 0.5$ and 0.3. Very few minimum distance error paths were found, and when a minimum error path occurred, the path length was relatively short. The

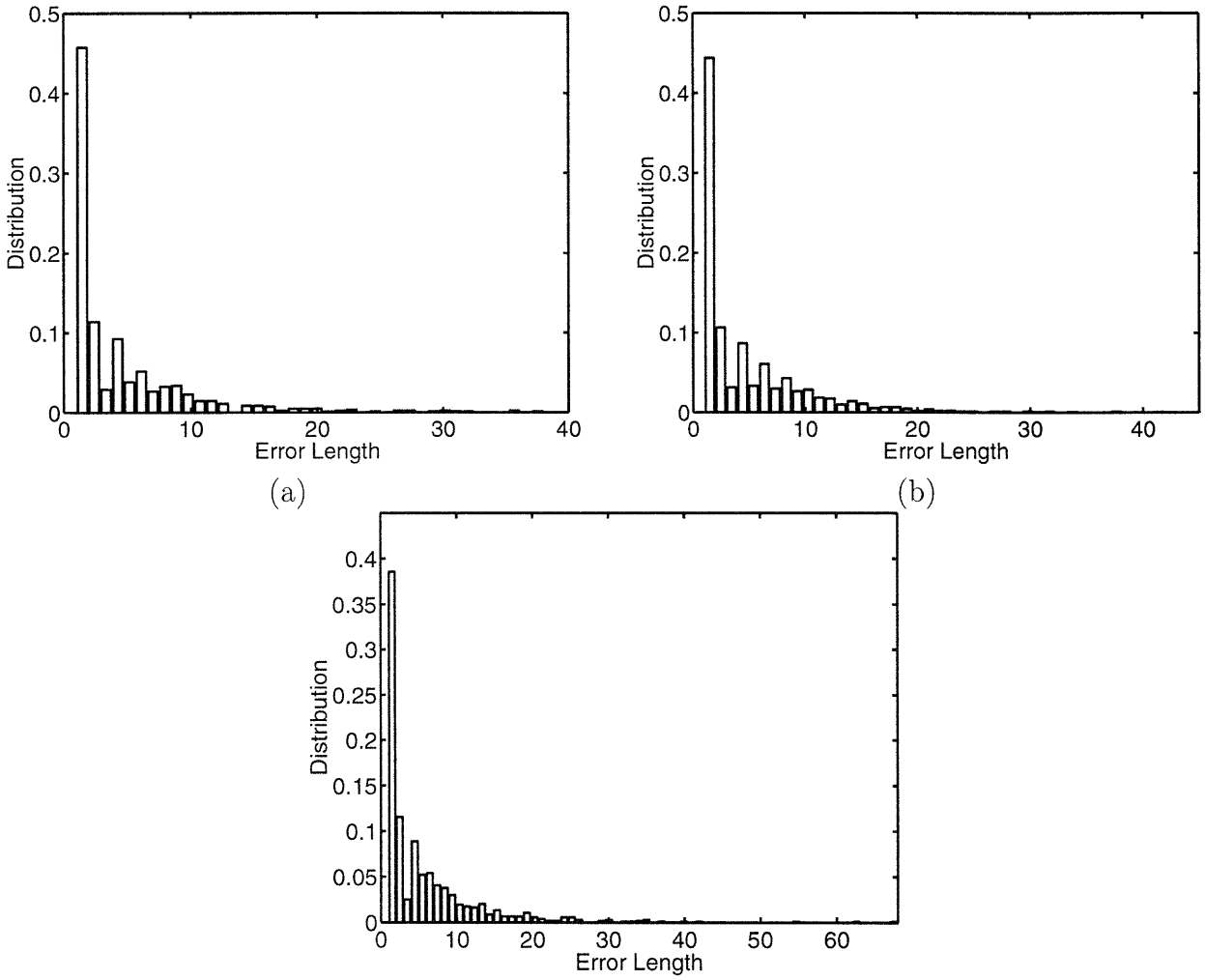


Figure 5-5: Distribution of lengths of minimum distance error events for Ω_f with (a) $\alpha = 0.8$ (b) $\alpha = 0.5$ (c) $\alpha = 0.3$

α	Min. d_c^2	Error Lengths in symbols
0.96	0.64	4, 10
0.5	2.25	4
0.3	4.01	4, 6, 12

Table 5.1: Minimum distance error events for Ω_c of the overlap UEP code

results for the coarse code minimum distance error events are summarized in Table 5.1 for $\alpha = 0.8, 0.5$, and 0.3.

With the average energies and the minimum distances given in Section 3.5.2, the fine coding gain, γ_f, and the bounds on the coarse coding gain, γ_c, can be computed. These coding gains are the nominal coding gains since the effect of the number of nearest neighbors is not taken into account. The plot of γ_c as a function of α is shown in Figure 5-6. Also shown in Figure 5-6 is Wei's nominal coarse coding gain. Notice that Wei does better than the worst case coding gain for almost all values of α. This occurs because the points in Wei's code are always perpendicular to the boundary of their Voronoi region. Wei's coding gain also exceeds the best case coding gain for a range of α. This increased coding gain comes from the construction of Wei's constellations. All symbols of the fine code lie inside a Voronoi region of Λ_c, in particular, none of the points lie *on* a lattice Voronoi region boundary. However, the fine constellation used in the overlapping UEP code *does* have points on the Voronoi region boundary, thus the best case for the UEP overlapping code can fall short of Wei's coarse code performance.

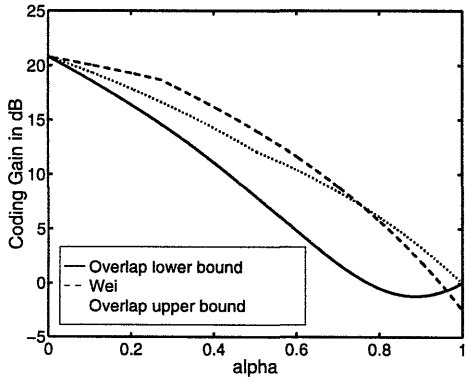


Figure 5-6: Upper and lower bounds for γ_c and Wei's γ_c

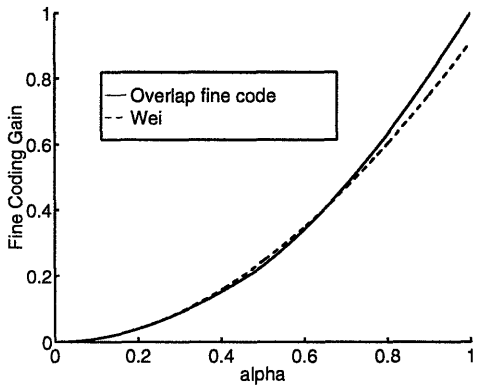


Figure 5-7: γ_f for the Overlapping UEP code and Wei's code

The fine code coding gain γ_f for both the UEP overlapping code and Wei's code are plotted in Figure 5-7 as a function of α. Even though $d_f^2 = \alpha^2$ for both codes and $\alpha_{\text{Wei}} = \sqrt{2}$ whereas $\alpha = 1$, both coding gains are approximately equal since the average energy for Wei's code also increases with α^2 and is roughly 2 times larger than the average energy of the UEP code. The average energy of the UEP overlapping code is approximately $2^{\frac{2r_c}{N}}$ times smaller due to the overlap of r_c bits. As the number of fine bits increases and the continuous approximation becomes more accurate, and the difference between the overlapping code and Wei's code should approach 1.5dB at $\alpha = 1(\alpha_{\text{Wei}} = \sqrt{2})$. The 1.5dB difference can also be seen from Equation (3.45). The term $\gamma_{\text{UEP}} = 2^{\frac{2r_c}{N}}$ is 1 for Wei's code since $r_c = 0$ bits in 4 dimensions. The overlapping code has a $r_c = 1$ bit overlap, thus with both codes selecting from symbols from R^4, $\frac{2r_c}{N} = \frac{1}{2}$. Even with only a 32 point constellation, a small difference in fine coding gains at $\alpha = 1$ and $\alpha_{\text{Wei}} = \sqrt{2}$ can be seen.

The error performance of the UEP trellis code example was also tested with simulations. All signal to noise ratios are divided by $2^R - 1$ where R is the bit rate per two dimensions. The normalized signal-to-noise ratios (SNR_{norm}) achieves the Shannon capacity for a Gaussian channel when SNR_{norm}=1, thus results are easier to interpret with SNR_{norm}. Several simulations for various values of α were run and the probability of bit error was computed. Each experiment consisted of 1,000 blocks. The fine code probability of bit error is plotted in Figure 5-8 as a function of SNR_{norm} along with the predicted probability of error curves for both the UEP trellis code and the

α	SNR_{norm} (dB)	Pred. $P_f(\epsilon)$	Sim. $P_f(\epsilon)$	Pred. Worst $P_c(\epsilon)$	Pred. Best $P_c(\epsilon)$	Sim $P_c(\epsilon)$
0.8	12.81	1.00×10^{-5}	3.0×10^{-6}	8.83×10^{-8}	1.68×10^{-13}	0
0.8	11.62	1.00×10^{-4}	1.95×10^{-5}	2.64×10^{-6}	1.09×10^{-10}	0
0.8	10.01	1.00×10^{-3}	2.8×10^{-4}	7.71×10^{-5}	6.65×10^{-8}	1.00×10^{-6}
0.8	7.54	1.00×10^{-2}	1.08×10^{-2}	2.19×10^{-3}	3.58×10^{-5}	1.08×10^{-3}
0.8	11.05	2.49×10^{-4}	6.35×10^{-5}	1.00×10^{-5}	1.40×10^{-9}	0
0.8	9.86	1.20×10^{-3}	4.70×10^{-4}	1.00×10^{-4}	1.09×10^{-7}	2.5×10^{-6}
0.8	8.25	5.82×10^{-3}	3.73×10^{-3}	1.00×10^{-3}	8.29×10^{-6}	2.04×10^{-4}
0.5	17.08	1.00×10^{-5}	2.50×10^{-6}	8.97×10^{-38}	3.24×10^{-60}	0
0.5	15.89	1.00×10^{-4}	1.09×10^{-4}	3.32×10^{-29}	2.67×10^{-46}	0
0.5	14.29	1.00×10^{-3}	3.96×10^{-4}	9.26×10^{-21}	1.37×10^{-32}	0
0.5	7.54	7.76×10^{-2}	2.48×10^{-1}	1.00×10^{-5}	2.63×10^{-8}	0
0.5	6.35	1.08×10^{-1}	3.36×10^{-1}	1.00×10^{-4}	1.04×10^{-6}	6.5×10^{-6}
0.5	4.74	1.52×10^{-1}	4.15×10^{-1}	1.00×10^{-3}	4.02×10^{-5}	1.37×10^{-4}
0.5	2.27	2.19×10^{-1}	4.70×10^{-1}	1.00×10^{-2}	1.50×10^{-3}	6.12×10^{-3}
0.3	21.38	1.00×10^{-5}	5.50×10^{-6}	1.46×10^{-178}	9.41×10^{-231}	0
0.3	20.19	1.00×10^{-4}	4.00×10^{-5}	2.55×10^{-136}	5.17×10^{-176}	0
0.3	18.58	1.00×10^{-3}	4.64×10^{-4}	8.44×10^{-94}	3.23×10^{-122}	0

Table 5.2: Probability of bit error from simulations of UEP trellis code with $\alpha = 0.8, 0.5$ and 0.3

16-QAM code. The coarse code probability of bit error is plotted in Figure 5-9 along with the upper and lower bounds for the coarse error probabilities and the 16-QAM bit error probabilities. The results are also tabulated in Table 5.2 where $P_f(\epsilon)$ and $P_c(\epsilon)$ denote the probability of a fine and coarse bit error respectively.

Another measure of performance is the *effective coding gain* which represents the actual reduction in SNR over an uncoded QAM system which achieves the 10^{-5} bit error probability. Effective coding gains can also be computed at different probability of error levels, but for higher probability of errors, the SNR gap to Shannon capacity becomes less, thus, there is less available SNR to "gain." The 10^{-5} mark was chosen over 10^{-6} error probability to reduce the simulation time. The effective coding gains at specific values of α for the fine code ($\gamma_{f,\text{eff}}$) are plotted in Figure 5-10 versus the nominal coding gain curves.

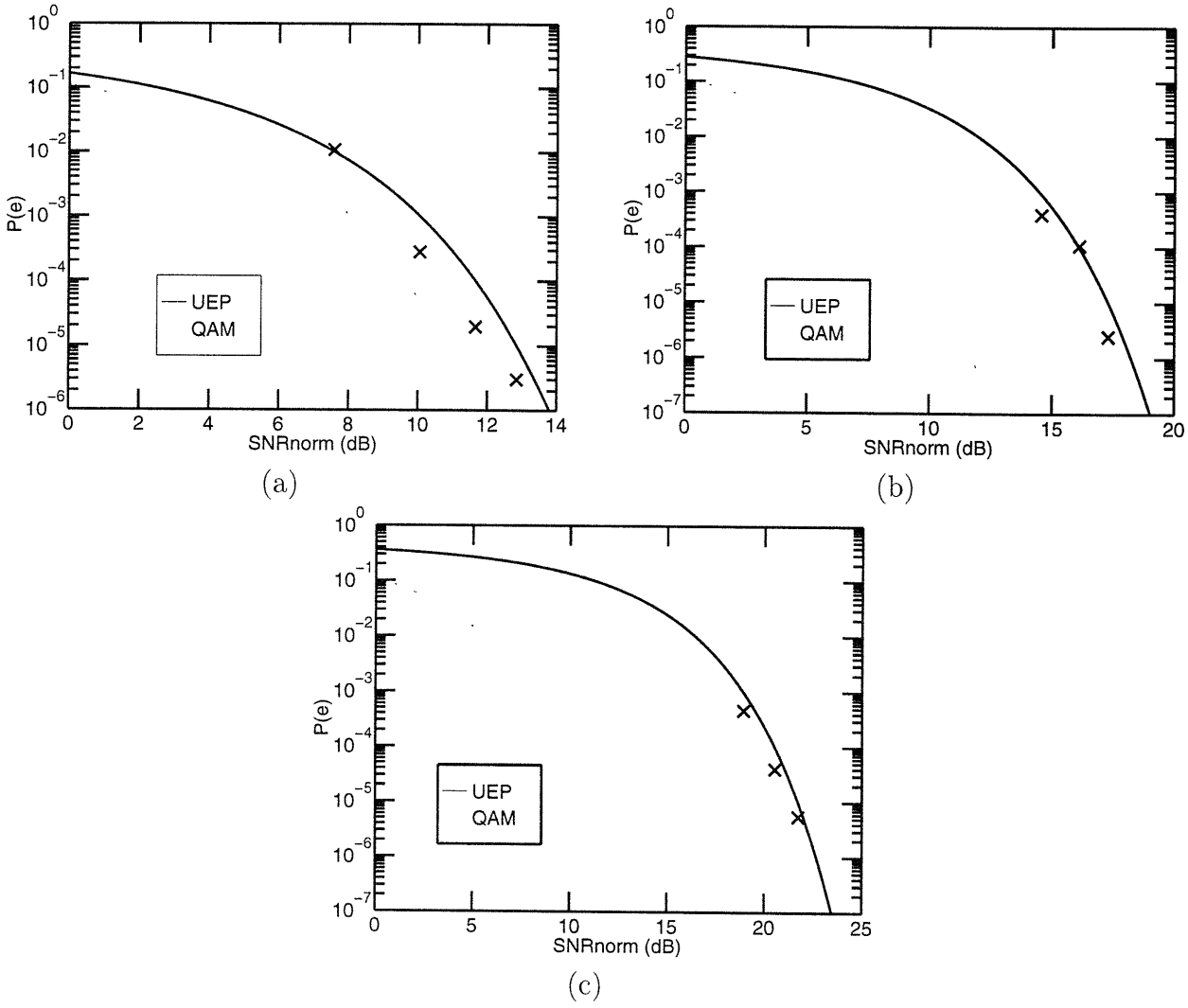


Figure 5-8: Probability of fine bit error for (a) $\alpha = 0.8$ (b) $\alpha = 0.5$ (c) $\alpha = 0.3$

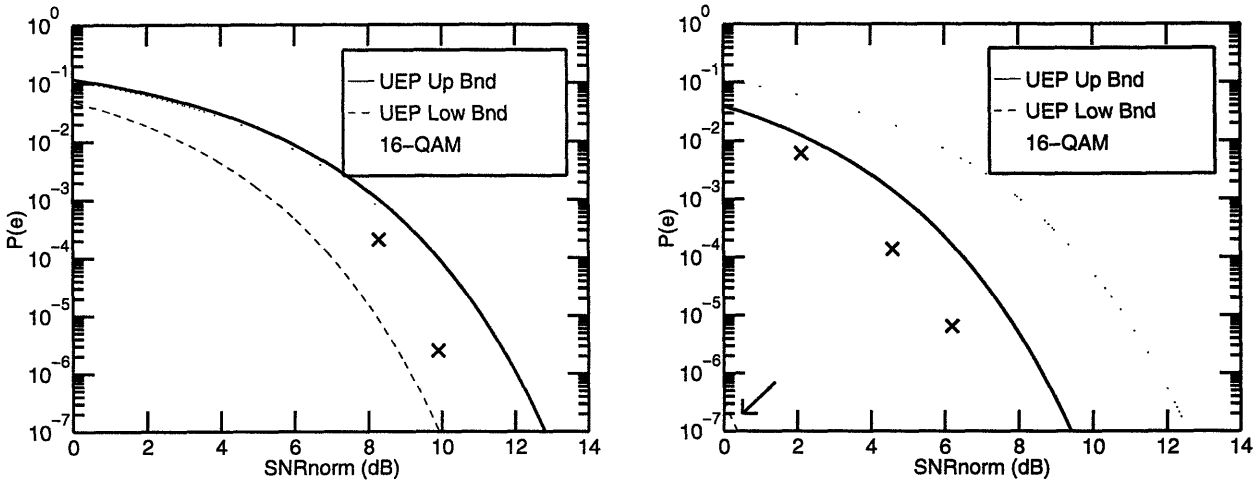


Figure 5-9: Probability of coarse bit error for (a) $\alpha = 0.8$ (b) $\alpha = 0.5$

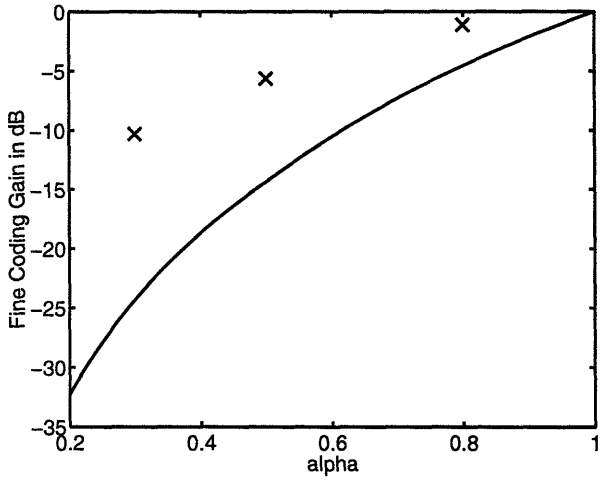


Figure 5-10: Simulation results for the effective fine coding gain vs. the predicted nominal coding gain

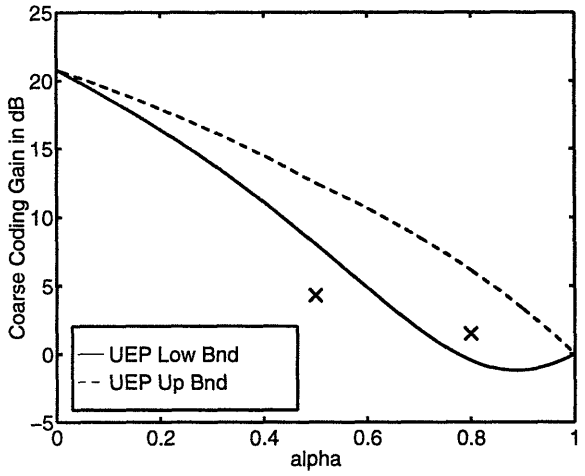


Figure 5-11: Simulation results for the effective coarse coding gain vs. the predicted nominal coding gain

α	$\gamma_{f,\text{eff}}$	$\gamma_{c,\text{eff}}$
0.8	-1.10dB	1.50dB
0.5	-5.65dB	4.31dB
0.3	-10.3dB	—

Table 5.3: Effective fine and coarse coding gains

The effective coarse coding gains ($\gamma_{c,\text{eff}}$) at the 10^{-5} probability of error level are plotted along with the nominal coding gain bounds in Figure 5-11. Table 5.3 summarizes the effective coding gains and the predicted nominal coding gains for the values of α which were simulated.

Notice that the effective coding gain for $\alpha = 0.5$ was much lower than expected. As the probability of coarse error increases, the sensitivity to the correction factor used in the stack algorithm increases. The increase in sensitivity is caused by the increase in the minimum value $R\sigma^2$ for the correction factor as σ^2 increases. A sensitivity test was run to determine the dependence of the error probability on the correction factor. The $\alpha = 0.8$ and SNR_{norm}=8.29dB code was run with six different values for the correction factor. Each value increased by 20% of the minimum value, thus the values used were $1.4R\sigma^2$, $1.6R\sigma^2$, $1.8R\sigma^2$, $2.0R\sigma^2$, $2.2R\sigma^2$. The results are plotted in Figure 5-12 and the numerical values are given in Table 5.4.

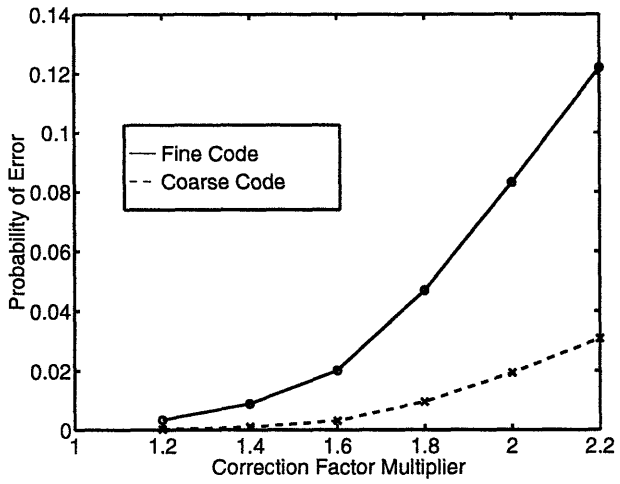


Figure 5-12: Sensitivity of probability of error to increases in the correction factor

Increase in Corr. Factor	$P_f(\epsilon)$	$P_c(\epsilon)$
1.2	3.39×10^{-3}	2.15×10^{-4}
1.4	8.89×10^{-3}	9.48×10^{-4}
1.6	2.01×10^{-2}	3.01×10^{-3}
1.8	4.69×10^{-2}	9.58×10^{-3}
2.0	8.43×10^{-2}	1.93×10^{-2}
2.2	1.22×10^{-1}	3.08×10^{-2}

Table 5.4: Sensitivity of the probability of bit error to the correction factor

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The unequal error protection code developed in this thesis achieves superior fine code performance compared to the codes developed by Wei. The superior performance comes from the trellis shaping of the fine code by the coarse code. Reusing the shaped bits allows the UEP trellis code to have a higher fine code rate. The expected increase in the fine coding gain over the Wei's codes approaches 3dB per overlap bit per two dimensions as the number of fine codewords increases.

The UEP trellis code, as do most UEP codes, creates a "cloud" of fine codewords around every coarse codeword. The cloud region used here is the Voronoi region of the coarse codeword and the mechanism used to create the clouds is trellis shaping. The Voronoi region is first shrunk before the shaping bits are reused. The shrinking is critical in order to preserve the effects of shaping. When no shrinking occurs, the coarse code does not have any effect and the UEP trellis code reduces to the fine code. When shrinking is complete, *i.e.*, the Voronoi region is completely shrunk down to the coarse codeword, the UEP trellis code reduces to the coarse code.

Detailed analysis of the UEP trellis code shows that the coding gain of both the coarse and fine code are most directly dependent on the *absolute number* of fine bits, not the ratio of fine to coarse bits. Both Wei and Calderbank use the ratio as their design parameter. Calderbank's codes do not seem to fit into the framework of

the UEP trellis code, thus the ratio of fine to coarse bits may be the correct design parameter for his codes. However, Wei's codes are a subset of the UEP trellis code and therefore, follow the behavior predicted for the UEP trellis code. Thus, the ratio of fine to coarse bits is clearly the wrong design parameter.

The UEP trellis code does not perform as well as Wei's codes for the coarse coding gain. The degradation in performance stems from the special arrangements which Wei selects for his two dimensional constellations. Sequences of symbols chosen from these two dimensional constellations lie well within a Voronoi region. Furthermore, any two adjacent codewords which lie in adjacent Voronoi regions lie along a perpendicular line to the Voronoi boundary and are separated by at least the minimum fine code distance. This arrangement ensures that when the Voronoi region is shrunk, maximum separation of the two codewords results. Thus, the intercloud distance which governs the coarse coding gain increases faster as the Voronoi region is shrunk.

The UEP trellis code, however, does not restrict codewords to lie entirely within a Voronoi region nor are adjacent codewords aligned in a perpendicular fashion across Voronoi region boundaries. Two adjacent points may lie on the Voronoi boundary and be separated by the minimum fine code distance along the Voronoi boundary. Thus, as the region is shrunk, the separation between the two points does not increase as quickly as Wei's codes. Simulations indicate that the geometry just described does occur, but not very frequently. The actual performance of the UEP trellis code suggests that most minimum distance codewords are arranged somewhere between the best and worst case geometries.

The overlapping bits and gain in fine code performance come at the cost of complexity and decoding delay. The shaping code creates a decoding delay for the shaping code. This decoding delay can be arbitrarily large and thus, the number of states in the UEP code is very large. The Viterbi Algorithm cannot be used to decode the UEP trellis code. Instead, a modified stack decoder is used. The modified stack decoder can only make tentative decisions on which path to grow due to the coding delay of the shaping code. Several design parameters for the stack decoder affect the decoding speed and accuracy. The main parameter which affects decoding is the

correction factor used to adjust the expected growth rate of the stack decoder. As the correction factor is made more aggressive, the decoding time decreases as does the accuracy. Thus, a tradeoff between decoding time and decoding error rate must be made.

Even though the UEP trellis code is rather complex as compared to Wei's UEP codes, the UEP trellis code is preferable if increased fine code performance is desired. The advantage of the UEP code is the increased fine code rate, thus the typical ranges of operation are when the scaling factor α is close to 1. As α decreases, the system's performance begins to resemble the coarse code more. Thus, for smaller α, this UEP code may not perform as well as other codes such as Wei's codes. Faster and more efficient methods for decoding also need to be researched for the UEP trellis code to be a practical code.

6.2 Future Work

More research must be done to determine the exact behavior of the UEP trellis code. Several aspects of the UEP code have not yet been resolved. The dependence on the correction factor needs to be quantified and the coarse code experiments which resulted in lower than expected coding gains need to be confirmed. Another aspect of the current implementation which needs to be explored is the effect of the "don't care" states and the effects non-uniquely decodable sequences. The time spent decoding long sequences of "don't care" states should be quantified and if, as suspected, the majority of the time decoding is spent searching a wrong path, solutions to the problem must be pursued.

One possible solution to the "don't care" sequences is the dithering approach. However, once the fine code is dithered, the VA FSM can no longer be used. Thus the stack algorithm may become infeasible due to large storage requirements and long computation time. A Fano algorithm should be implemented to facilitate the dithering approach and to allow larger, more complicated codes to be used in the UEP trellis code. One design goal is to create a UEP trellis code which achieves 6dB

of coding gain on the coarse code and 3dB of coding gain on the fine code. Initial calculations indicate that a 128 state rate-3/4 fine trellis code and a 64-state rate-3/4 coarse trellis code can be used to achieve the 6dB, 3dB mark.

Future work must be done describing relation between the unequal error protection problem and the Cover's degraded broadcast channel model. Quantifying the performance of UEP codes with the coding gain over an uncoded QAM system without unequal error protection does not seem like the proper baseline. The UEP QAM system described here was an attempt at creating an appropriate baseline system. However, if the connection between UEP codes and Cover's degraded broadcast channel can be understood, then the performance of the code can be described using results from the degraded broadcast model.

Appendix A

Derivation of Average Energy Sum

A.1 Average Energy of $\mathcal{C}_{\text{UEP}}$

For the $M^{(c)}$-QAM and $M^{(f)}$-QAM schemes described in section 3.1, the result for the average energy of the UEP constellation, $\mathcal{C}_{\text{UEP}}$ is derived in this section. Remember, $\mathcal{C}_f^{\alpha}$ is the scaled constellation which is mapped around each superpoint symbol. The constellation $\mathcal{C}_f^{\alpha}$ is centered around the origin and has two-fold symmetry. Thus, if $c = (c_x, c_y) \in \mathcal{C}_f^{\alpha}$, then there is a point $c' \in \mathcal{C}_f^{\alpha}$ such that $c_x = -c_x'$, $c_y = -c_y'$ or $c = -c'$.

The superpoint symbols are taken from the constellation $\mathcal{C}_c$. Therefore, after mapping each point in $\mathcal{C}_f$ to a superpoint $s = (s_x, s_y)$, each mapped constellation has two-fold symmetry around the superpoint. Let $|X|$ denote the number of elements in a constellation X. Notice, $|\mathcal{C}_{\text{UEP}}| = |\mathcal{C}_c| \cdot |\mathcal{C}_f^{\alpha}|$. Now, define the average energy of $\mathcal{C}_{\text{UEP}}$ as

$$\begin{aligned} \overline{E}_{\text{UEP}} &= \frac{1}{|\mathcal{C}_{\text{UEP}}|} \sum_{c \in \mathcal{C}_{\text{UEP}}} \|c\|^2 \\ &= \frac{1}{|\mathcal{C}_{\text{UEP}}|} \sum_{c \in \mathcal{C}_{\text{UEP}}} (c_x^2 + c_y^2) \\ &= \frac{1}{|\mathcal{C}_{\text{UEP}}|} \sum_{s \in \mathcal{C}_c} \left[\sum_{c \in \mathcal{C}_f^{\alpha}} (c_x^2 + c_y^2) \right]. \end{aligned} \tag{A.1}$$

Now, every point in a cloud region has the coordinates $(s_x + c_x, s_y + c_y)$. Every point of $\mathcal{C}_f^\alpha$ lies in one of the four quadrants of the 2 dimensional plane. The sets of points in which lie in the four quadrants will be denoted Q_1, Q_2, Q_3, and Q_4. Each set has $\frac{|\mathcal{C}_f^\alpha|}{4}$ points in the set. Thus, using Equation A.1,

$$\begin{aligned}\overline{E}_{\text{UEP}} &= \frac{1}{|\mathcal{C}_{\text{UEP}}|}\sum_{s\in\mathcal{C}_c}\left\{\sum_{c\in Q_1}\left[(s_x+c_x)^2+(s_y+c_y)^2\right]\right. \\ &+ \sum_{c\in Q_2}\left[(s_x+c_x)^2+(s_y+c_y)^2\right]+\sum_{c\in Q3}\left[(s_x+c_x)^2+(s_y+c_y)^2\right] \\ &+ \left.\sum_{c\in Q4}\left[(s_x+c_x)^2+(s_y+c_y)^2\right]\right\}.\end{aligned} \tag{A.2}$$

Now, using the two-fold symmetry of $\mathcal{C}_f^\alpha$, the x-coordinates of points in quadrants 1,2 and quadrants 4,3 can be grouped together. The same is done for the y-coordinates with quadrants 1,4 and quadrants 2,3. Thus,

$$\begin{aligned}\overline{E}_{\text{UEP}} &= \frac{1}{|\mathcal{C}_{\text{UEP}}|}\sum_{s\in\mathcal{C}_c}\left\{\sum_{c\in Q_1}\left[(s_x+c_x)^2+(s_x-c_x)^2\right]\right. \\ &+ \sum_{c\in Q_4}\left[(s_x+c_x)^2+(s_x-c_x)^2\right]+\sum_{c\in Q_1}\left[(s_y+c_y)^2+(s_y-c_y)^2\right] \\ &+ \left.\sum_{c\in Q_2}\left[(s_y+c_y)^2+(s_y-c_y)^2\right]\right\}.\end{aligned} \tag{A.3}$$

Notice, in quadrants 1 and 4, the x-coordinates are equal. Also, in quadrants 1 and 2, the y-coordinates are equal. Thus, combining terms,

$$\begin{aligned}\overline{E}_{\text{UEP}} &= \frac{1}{|\mathcal{C}_{\text{UEP}}|}\sum_{s\in\mathcal{C}_c}\left\{2\sum_{c\in Q_1}(2s_x^2+2c_x^2)+2\sum_{c\in Q_1}(s_y^2+c_y^2)\right\} \\ &= \frac{4}{|\mathcal{C}_{\text{UEP}}|}\sum_{s\in\mathcal{C}_c}\sum_{c\in Q_1}(s_x^2+s_y^2+c_x^2+c_y^2) \\ &= \frac{4}{|\mathcal{C}_{\text{UEP}}|}\frac{|\mathcal{C}_f^\alpha|}{4}\sum_{s\in\mathcal{C}_c}\|s\|^2+\frac{4}{|\mathcal{C}_{\text{UEP}}|}|\mathcal{C}_s|\sum_{c\in Q_1}\|c\|^2 \\ &= \frac{1}{|\mathcal{C}_c|}\sum_{s\in\mathcal{C}_c}\|s\|^2+\frac{4}{|\mathcal{C}_f^\alpha|}\sum_{c\in Q_1}\|c\|^2 \\ &= \overline{E}_{\mathcal{C}_c}+\overline{E}_{\mathcal{C}_f^\alpha}.\end{aligned} \tag{A.4}$$

Thus, the average energy of $\mathcal{C}_{\text{UEP}}$ is the sum of the average energies of the coarse constellation and the scaled, fine constellation.

Bibliography

[1] A. R. Calderbank and N. Seshadri, "Multilevel codes for unequal error protection," *IEEE Transactions on Information Theory*, vol. 39, pp. 1234–1248, July 1993.

[2] T. M. Cover and J. A. Thomas, *Elements of Information Theory.* New York: Wiley, 1991.

[3] G. D. Forney, Jr., "Trellis shaping," *IEEE Transactions on Information Theory*, vol. 38, pp. 281–300, Mar. 1992.

[4] R. G. Gallager, *Information Theory and Reliable Communication.* New York: Wiley, 1968.

[5] E. A. Lee and D. G. Messerschmitt, *Digital Communications.* Massachusetts: Kluwer, second ed., 1994.

[6] M.-C. Lin, C.-C. Lin, and S. Lin, "Computer search for binary cyclic uep codes of odd length up to 65," *IEEE Transactions on Information Theory*, vol. 36, pp. 924–935, July 1990.

[7] S. Lin and D. J. Constello, Jr, *Error Control Coding: Fundamentals and Applications.* New Jersey: Prentice-Hall, 1983.

[8] B. Masnick and J. Wolf, "On linear unequal error protection codes," *IEEE Transactions on Information Theory*, vol. 3, pp. 600–607, Oct. 1967.

[9] J. L. Massey, "Variable-length codes and the fano metric," *IEEE Transactions on Information Theory*, vol. 18, pp. 196–198, Jan. 1972.

[10] L. F. Wei, "Coded modulation with unequal error protection," *IEEE Transactions on Communications*, vol. 41, pp. 1439–1449, Oct. 1993.