Sequential Decoding of Trellis Codes through ISI Channels

by

Patrick M. Maurer

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

June 1996

© Massachusetts Institute of Technology 1996. All rights reserved.

Author
Department of Electrical Engineering and Computer Science
May 28, 1996

Certified by
Mitchell D. Trott
Assistant Professor
Thesis Supervisor

Accepted by
F. R. Morgenthaler
Chairman, Departmental Committee on Graduate Students
Sequential Decoding of Trellis Codes through ISI Channels

by

Patrick M. Maurer

Submitted to the Department of Electrical Engineering and Computer Science on May 28, 1996, in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering and Computer Science

Abstract

The Fano algorithm performance is well understood for the standard additive white Gaussian noise (AWGN) channel, and has been shown to be a feasible decoding method for complex codes [12, 13]. This work aims to determine whether the Fano algorithm performance degrades relative to the Viterbi algorithm when the channel has intersymbol interference.

We compare Fano and Viterbi decoding by implementing a short constraint-length code. The performance of the code over an ISI channel using Tomlinson-Harashima precoding is compared to the baseline case of a non-ISI channel with the same effective SNR. A long constraint-length code is also evaluated over both the ISI and non-ISI channels, using only Fano decoding (as Viterbi decoding is computationally infeasible).

No significant difference was found between performance over the ISI and non-ISI channel. Thus, if sequential decoding can be used beneficially with high constraint length codes over AWGN channels, then it should be equally beneficial over ISI channels.

Thesis Supervisor: Mitchell D. Trott
Title: Assistant Professor
Acknowledgments

To Professor Mitchell Trott I owe my deepest gratitude. He provided me with countless hours of his time, invaluable teachings, and unwavering support. But of all he has given me, I value his friendship the most.

I also wish to thank Dr. G. David Forney for providing me with knowledge and opportunities instrumental to my understanding of communication systems.

I acknowledge and thank Dr. Vedat Eyuboglu, Marc Ibanez, Aradhana Narula, James Sarvis, Brett Schein, Donald Wilcoxson and Kevin Yu for their contributions. I also wish to thank Michael Schmitt. I can always count on Mike to make me laugh, put life in perspective, and

The single largest constant in my life has been my family. When my life was turned upside down, I was able to count on my family most of all, even when I offered nothing in return. For their unconditional love, I thank my parents: Robert and Marylou Maurer, and Sharad and Shobha Rastogi. I also thank all my brothers and sisters. I also acknowledge my new inspirations, Cameron and Katherine Maurer, and Brianna Barros.

Most of all, I thank my wife Mala for her patience and love. And for missing me while I’m here writing my thesis.

This work was supported by a grant from the Motorola’s University Partnerships in Research (UPR) program, and by National Science Foundation grant NCR-9457509.
In loving memory of
my Mother,
Marion Elizabeth Maurer
## Contents

1 Introduction 10  

2 Background 13  
   2.1 Trellis Codes ................................................. 13  
      2.1.1 Trellis Diagrams ....................................... 18  
      2.1.2 Metrics and Soft-Decision Decoding ............... 20  
      2.1.3 Decoding Trellis Coded Sequences ................. 22  
   2.2 Equalization .................................................. 28  
      2.2.1 Linear Equalization .................................... 28  
      2.2.2 Precursor versus Postcursor ISI .................. 29  
      2.2.3 Decision-Feedback Equalization .................... 30  
   2.3 Precoding ..................................................... 32  

3 The System Models 38  
   3.1 Data Source and Sink ......................................... 38  
   3.2 The Encoding Scheme ......................................... 39  
      3.2.1 Convolutional Encoders ................................ 39  
      3.2.2 Mapping .................................................. 41  
      3.2.3 Precoder ................................................ 42  
   3.3 The Channel ................................................... 42  
   3.4 The Decoding Scheme ......................................... 44  

4 Simulation Results 46
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Sequential versus Viterbi Decoding</td>
<td>46</td>
</tr>
<tr>
<td>4.2 Long Constraint Length Code</td>
<td>49</td>
</tr>
<tr>
<td>4.3 Computational Distributions</td>
<td>51</td>
</tr>
<tr>
<td>5 Conclusions</td>
<td>54</td>
</tr>
</tbody>
</table>
List of Figures

1-1 Block diagram overview of communication system. .................. 11

2-1 Basic structure of a trellis encoder. .............................. 13
2-2 16-QAM constellation and a single subset. ........................ 14
2-3 Ungerboeck’s four state encoder. ................................. 15
2-4 Discrete time model of a standard (AWGN) channel. ............... 17
2-5 Diagram of interaction between signal and noise. .................. 17
2-6 Decision regions for (a) 8-QAM and (b) 16-QAM signal structures. 18
2-7 State-space diagram for Ungerboeck’s encoder. ..................... 19
2-8 Trellis diagram for Ungerboeck’s code. ............................ 20
2-9 Event sequence for Viterbi decoder. ............................... 24
2-10 Path metric progression for Fano decoder. .......................... 25
2-11 Event sequence for Fano decoder. ................................. 26
2-12 Discrete time model of an ISI channel. ............................ 28
2-13 Linear equalization filters. ....................................... 29
2-14 Linear versus decision feedback equalizers. ....................... 31
2-15 Trellis encoder and linear precoder in transmitter. ............... 34
2-16 Extension of 16-QAM to infinite lattice. .......................... 36
2-17 Trellis encoder and Tomlinson-Harashima precoder. ............... 37

3-1 64 state encoder from the V.34 standard. .......................... 40
3-2 Decision feedback equalizer. ....................................... 42
3-3 ISI channel frequency characteristic. ............................ 43
List of Tables

3.1 Polynomial coefficients for the long constraint-length code. . . . . . . 41
Chapter 1

Introduction

The availability and popularity of distributed network services and similar products has created a demand for increased data communication rates over the existing wire-line (telephone) network. However, the existing network, originally designed to handle analog voice signals, is not well suited for fast access to massive amounts of information. We are thus faced with the problem of transmitting reliably at as high a rate as possible, given a reasonable complexity.

Our communication system model is shown in Figure 1-1. At its highest level, the system consists of a data source, such as a facsimile machine or a computer terminal, a transmit modem, a telephone channel, a receive modem, and a data sink, which might also be a facsimile machine or computer terminal. The objective is for the data source to reliably send information as fast as possible to the data sink. The telephone channel limits the negligible-error transmission speed because it adds noise to the transmitted information while constraining the allowable transmit power. Further, the telephone channel is a bandlimited, intersymbol interference (ISI) channel. ISI channels will be discussed in detail in Chapter 2.

Since the data source and sink by assumption require a transparent communication path, the modems are fully responsible for preparing the information from the data source so it can pass through the telephone channel at a high rate and reliability. There are several components to this process, such as D/A conversion, modulation, channel coding, and equalization. We shall limit our discussion to a form of combined
channel coding and modulation known as *trellis coded modulation* (TCM) and how it interacts with a specific equalization structure known as *precoding*.

![Block diagram overview of communication system.](image)

The Viterbi algorithm is currently the most popular method for decoding trellis coded information. However, this method is limited to fairly simple codes with modest coding gains. More complicated codes provide more substantial gains, and hence higher communication rates, but these codes can only be decoded in practice using sequential decoding algorithms such as the stack algorithm or the Fano algorithm. While these algorithms allow the use of complex codes, these decoders are not guaranteed to find the most likely transmitted sequence. Also, the delay in decoding a particular symbol varies randomly. Randomly varying decoding delay creates a need for large buffers, and even then there exists a significant probability of buffer overflow. Recent work by Wang and Costello [12] suggests that the buffer overflow problem can be addressed, making possible the use of more complicated codes.

The current technology modems [1] confirm that Viterbi decoding can be used in conjunction with precoding. Our aim is to show that precoding does not adversely affect the performance of sequential decoding, even when complicated trellis codes are used. This will be done by comparing the performance of the Viterbi algorithm to that of the Fano algorithm with and without the use of precoding.

We start with a description of the underlying theory in Chapter 2. Trellis coding and decoding are described in general, and the behavior of the Viterbi algorithm and the Fano algorithm is laid out. Intersymbol interference is defined, and zero-forcing equalization described. Precoding is then introduced, along with the changes it causes in the definition of the distance metric. An simple example is used throughout the chapter to illustrate the operation of encoders, decoders, precoders, and various concepts relevant to the remainder of the paper.

Chapter 3 describes the simulated system in detail. Simulation results are pre-
presented in Chapter 4. Sequential decoding performance is compared against that of the Viterbi decoder over both an ISI and non-ISI channel. Performance is measured using waterfall curves and plots of computational distributions. Conclusions are drawn from these results in Chapter 5.
Chapter 2

Background

To better understand the transmission scheme used in this paper, it is helpful to have some background in trellis coded modulation and equalization.

2.1 Trellis Codes

Trellis coded modulation uses binary channel coding jointly designed with multilevel modulation to maximize the minimum Euclidean distance between coded signal sequences [11]. As shown in Figure 2-1, the trellis encoder consists of a binary linear convolutional encoder and signal mapper.

![Figure 2-1: Basic structure of a trellis encoder.](image)

The convolutional encoder accepts \( k \) information bits \( x_{0..k-1} \) every symbol, and
sends out $n > k$ bits $y_{0..n-1}$ to the signal mapper. We consider $x$ and $y$ to be row vectors. The convolutional encoder can be completely characterized by its transfer function matrix $G(D)$. The transfer function matrix is a $k \times n$ matrix of polynomials in $D$, the delay operator. The operator $D$ can (but need not be) interpreted as a discrete-time transform operator, related to the $z$-transform by $D = z^{-1}$. Each element $g_{ij}(D)$ of $G$ is then a transfer function relating $x_i(D)$ to $y_j(D)$. If we define the encoder input sequence to be the length $k$ vector $x(D)$, then the length $n$ encoder output sequence $y(D)$ is

$$y(D) = x(D)G(D).$$

The signal mapper receives $m$ uncoded bits $u_{0..m-1}$ every symbol in addition to the coded bits $y_{0..n-1}$. The uncoded bits choose one of $2^m$ points in a reduced signal constellation called a subset. The coded bits determine which of the $2^n$ subsets the uncoded bits will actually choose from. Thus there are a total of $2^{m+n}$ points in the full signal constellation.

![Figure 2-2: 16-QAM constellation and a single subset.](image)

For example, suppose $m = n = 2$, and we are using 16-QAM as the constellation, as shown in Figure 2-2. The two coded bits choose among subsets 0, 1, 2, and 3. Suppose subset 0 is chosen. The two uncoded bits then determine which quadrant the point in subset 0 will come from. The coded bits represent a 2-way partitioning of the constellation according to Ungerboeck's method described in [11]. We see that the uncoded bits choose among points spaced far apart. The coded bits then make
the finer-grained decisions, where errors are much more likely. Ideally, the code is strong enough to prevent these errors.

Returning to Figure 2-1, the output of the signal mapper is a discrete-time signal $v[k]$. A single component of this output waveform is a symbol, and for QAM it represents the complex amplitude of some continuous-time pulse. When the message is decoded at the receiver, it is this symbol which we are concerned with accurately determining. The message bits are then completely determined by the symbol.

An example is probably best for explaining the operation and benefits of trellis coding. Toward this end, a convolutional encoder used as an example by Ungerboeck [11], shown in Figure 2-3, will be used in conjunction with a signal mapper resulting in the 16-QAM signal space structure of Figure 2-2.

In Figure 2-3, the trellis encoder input consists of three information bits: $u_0[k]$, $u_1[k]$, and $x_0[k]$. The bits $u_0$ and $u_1$ remain uncoded, while $x_0$ is convolutionally encoded resulting in the two coded bits $y_0$ and $y_1$. As described in the last section, these four bits determine which of the 16 symbols are transmitted. In this example we denote each symbol by the equivalent representation $u_0u_1y_0y_1$. For example, 1011 binary equals 11 decimal, and represents point 3 in quadrant 2 in Figure 2-2.

![Figure 2-3: Ungerboeck's four state encoder.](image)

An obvious question arises: Why use trellis coded modulation at all? After all, it would appear we are sending 4 bits through the channel for every 3 bits of information.
To answer this, we first need to understand the noise characteristics of the channel. In QAM transmission, information is carried in the in-phase and quadrature amplitudes of the continuous-time pulses, not in the pulse shape. If we assume a standard continuous-time bandlimited additive white Gaussian noise (AWGN) channel, then the equivalent discrete-time baseband channel adds a complex noise component $z[k]$ to the transmitted signal $v[k]$ resulting in the received vector $r[k] = v[k] + z[k]$. This is depicted in Figure 2-4. The decoder must determine $v[k]$ from the noise corrupted sequence $r[k]$. Figure 2-5 shows the behavior of the signal and noise for a single symbol, from the viewpoint of the signal constellation.

If an uncoded transmission scheme was chosen, then the decoding rule that minimizes probability of error is symbol by symbol minimum distance decoding. A received symbol is compared with each point in the constellation, and the point closest to the received value is assumed to be the correct symbol. Clearly, if constellation points can be placed farther apart, fewer errors will occur. A rule of thumb for the symbol error probability $P_{\text{symbol error}}$ is

$$P_{\text{symbol error}} \propto Q\left(\frac{d_{\text{min}}}{2\sigma}\right)$$  \hspace{1cm} (2.1)

where $d_{\text{min}}$ is the distance between the closest two points in the constellation, $\sigma^2$ is the noise variance (energy) per dimension, and the function $Q$, the tail probability of a univariate Gaussian, is defined by [8, 11]

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-\frac{t^2}{2}} dt.$$  \hspace{1cm} (2.2)

Equations 2.1 and 2.2 show that increasing $d_{\text{min}}$ decreases the symbol error probability for a fixed noise energy $\sigma^2$.

In our example, we transmit 3 information bits, $u_0$, $u_1$, and $x_0$, per symbol. Uncoded transmission requires 8 points in the constellation, as in Figure 2-6a. With our 4-bit coded scheme, a 16 point constellation such as that in Figures 2-2 and 2-6b is required. Suppose we have an average power constraint of 5 per dimension, i.e., the average energy over the entire constellation is 10. In this case, the 16-QAM structure
of Figure 2-6b has a minimum distance of $d_{\text{min}} = 2$, while the 8-QAM structure of Figure 2-6a has a minimum distance of $d'_{\text{min}} = 2\sqrt{2} = \sqrt{2}d_{\text{min}}$.

But constellation $d_{\text{min}}$ is not the correct measure of performance for a coded system; to compare coded 16-QAM to uncoded 8-QAM, we must use the effective minimum distance $d_{\text{free}}$ for the coded system. If $d_{\text{free}}$ is greater than $\sqrt{2}d_{\text{min}}$, the coded system will be an improvement on 8-QAM.

To determine $d_{\text{free}}$, we must examine how redundancy is added to the transmitted sequence by the convolutional encoder. A good way to visualize this is through the use of trellis diagrams.
Figure 2-6: Decision regions for (a) 8-QAM and (b) 16-QAM signal structures.

2.1.1 Trellis Diagrams

As with all finite state machines, convolutional encoders and trellis encoders can be described using a state diagram. The state diagram for Ungerboeck’s encoder is shown in Figure 2-7. A transition in the state diagram corresponds to a single output symbol. Sequences of symbols are more easily visualized when the state diagram is redrawn in the form of a trellis, as in Figure 2-8. Trellis diagrams show the possible progression of states as a function of time [5, 8]. The symbol sequence is shown as the transitions between states, tracing out a path through the trellis. Note that the uncoded bits $u_0u_1$ are not represented in the state or trellis diagrams. These simply correspond to parallel transitions, and could be shown by replacing each single arrow in Figures 2-7 and 2-8 by four parallel arrows. A particular path is shown in Figure 2-8 by the dark arrows. This path, extended infinitely in both directions, is a single codeword.

For codes with a modest number of states, we can use the trellis diagram to easily determine $d_{free}$. Since the Ungerboeck code is linear, we can simply look for the minimum distance path deviation from the all-zero path. This deviation is the path
shown in Figure 2-8, and corresponds to the input stream \( x_0 = \ldots, 0, 1, 0, 1, 0, \ldots \). This produces the output stream \( y_0 y_1 = \ldots, 00, 10, 01, 10, 00, \ldots \). To compute the free distance of this code, we add the squared Euclidean distances between subsets from the time where we first leave the all-zero state until the time we rejoin. Distances between subsets are computed in the following manner. Because there are uncoded bits, the different path branches refer to different subsets (0, 1, 2, and 3 in our example). The distance between subsets is the minimum distance between points in one subset and points in the other. Since the two paths agree everywhere else, the distance outside the path deviation is zero. Using Figure 2-2 as a guide, the squared distances are:

\[
\begin{align*}
    d_{\text{free}}^2 &= d^2(00, 10) + d^2(00, 01) + d^2(00, 10) \\
                      &= d^2(0, 2) + d^2(0, 1) + d^2(0, 2) \\
                      &= 2d_{\text{min}}^2 + d_{\text{min}}^2 + 2d_{\text{min}}^2 \\
                      &= 5d_{\text{min}}^2
\end{align*}
\]
Comparing this to $d_{\text{free}}^2 = 2d_{\text{min}}^2$, we see that $d_{\text{free}} > d_{\text{min}}^\prime$, and so we expect the error probability for the coded system to be lower.

### 2.1.2 Metrics and Soft-Decision Decoding

The trellis encoded sequence is received after passing through an AWGN channel. To understand the methods of decoding at the receiver, it is important first to understand our coding paradigm. We have already seen the benefit of TCM as an increase in the effective distance between constellation points, and we have a hint about the nature of decoding from the trellis diagram in Figure 2-8.

Soft decision decoding is a process where the information bits are determined using the unprocessed received symbols, without making intermediate decisions at the symbol level [8]. In the channel model of our example, the received sequence $r[k]$ consists of complex-valued symbols. The information bits are determined using either the Fano algorithm or the Viterbi algorithm, both of which are soft-decision decoders. The decoders compute subset likelihoods based on the entire received sequence $r[k]$.

A **metric** is a performance measure by which two (or more) alternatives can be compared. Each of our decoding methods use two metrics: a **branch metric** and a **path metric**. Suppose we have received a sequence of $l$ symbols, and know a possible
transmitted codeword for this sequence. In Viterbi decoding, the codeword is represented as a path of $l$ branches connected end-to-end through the trellis. The path metric is computed by summing the $l$ branch metrics

$$M_{VA,p}(v[0..l-1], r[0..l-1]) = \sum_{i=0}^{l-1} M_{VA,b}(v[i], r[i])$$ (2.3)$$

where for Gaussian noise channels, the branch metric is simply the squared Euclidean subset distance

$$M_{VA,b}(v[i], r[i]) = |v[i] - r[i]|^2.$$ (2.4)

Squared Euclidean distance is the log-likelihood measure on this channel. The codeword with the smallest path metric is the one closest to the received sequence, and is the one most likely to have been transmitted.

In determining the maximum likelihood codeword, the Viterbi algorithm always compares codewords of equal length. The Fano algorithm compares codewords of different lengths. If we used the path metric above, longer paths would appear worse simply because their path metric is the sum of more terms. To compensate for this effect, the Fano metric is used. Following Massey [9], Wang showed [13] that the direct trellis-coding analog of the Fano metric is a good choice for sequential decoding of trellis codes. The path metric for the Fano algorithm is

$$M_{SD,p}(v[0..l-1], r[0..l-1]) = \sum_{i=0}^{l-1} M_{SD,b}(v[i], r[i])$$ (2.5)$$

where the branch metric is

$$M_{SD,b}(v[i], r[i]) = -\log_2 \frac{P(r[i]|v[i])}{\sum_{j=0}^{2^m-1} P(r[i]|v_j)} - (n - k).$$ (2.6)$$

Here, $m$ is the number of uncoded information bits, $2^m$ is the number of subsets in the constellation, $v_j$ is the closest point in subset $j$ to $r[i]$, $n$ is the number of coded bits, and $k$ is the number of information bits sent through the convolutional encoder.
For our example, this branch metric reduces to

\[ M_{SD,b}(v[i], r[i]) = \log_2 e^{-\frac{|r[i] - v[i]|^2}{2\sigma^2}} + \log_2 \sum_{j=0}^{2^2-1} e^{-\frac{|r[i] - v[j]|^2}{2\sigma^2}} - 1 \]

\[ \propto |r[i] - v[i]|^2 + 2\sigma^2 \ln \sum_{j=0}^{3} e^{-\frac{|r[i] - v[j]|^2}{2\sigma^2}} - 2\ln(2)\sigma^2 \]

\[ = M_{VA,b}(v[i], r[i]) + 2\sigma^2 L_r[i] - 2\ln(2)\sigma^2 \]

where \( L_r[i] \) is the a-priori log-likelihood of \( r[i] \) occurring. This can be computed and stored using pre-decoder logic. In practice, \( L_r[i] \) is often just approximated using a lookup table. The two last terms act to balance the comparison between paths of different lengths.

It is important to note that the Fano algorithm requires knowledge of the channel noise variance, whereas the Viterbi algorithm does not.

### 2.1.3 Decoding Trellis Coded Sequences

There are a variety of techniques for decoding the received sequence, two of the most popular being sequential decoding and maximum-likelihood decoding. Each method has benefits and drawbacks. The Fano algorithm [15] is a type of sequential decoder; the Viterbi algorithm is a maximum-likelihood decoder. In this section, we describe each decoding method, and compare their behaviors.

The Viterbi decoder is a breadth-first tree searching algorithm which is guaranteed to find the maximum-likelihood transmitted sequence. Breadth-first decoders search every trellis state at each “depth” before moving to the next depth. The Viterbi algorithm performs a set number of computations per state per symbol (subset).

Perhaps the best way to understand the operation of the Viterbi algorithm is to continue our example for a hypothetical received sequence. Suppose the first 5 received symbols \( r[0..4] \) are the sequence \( \{0.5 + i, 1.2 - 2i, 4.2 + 2.2i, 0 - 0.5i, 1 + i\} \). Figure 2-9 shows the sequence of events for the Viterbi algorithm decoding of this sequence. During the first two steps \( (k = 0 \text{ and } k = 1) \), path metrics are computed, but no comparison is made. An equivalent startup sets the initial path metric of the
zero state to zero and the other three to \( \infty \). At the end of startup, there exists one current path metric for each state of the trellis, shown as the rightmost numbers in each stage. Also, one path history is stored for each state, shown as dark arrows from the initial state tracing to one of the four current states.

When using the Viterbi algorithm, the trellis depth corresponds directly to the time interval. At time \( k = 2 \), the decoder computes every branch metric for \( r[2] \) (the numbers on the arrows) according to Equation 2.4. These are added to the path metric of the branch origin state (shown as the numbers inside ovals), and the lowest path metric entering a state defines the winning path for that state. For example, looking at state 00, there are branches coming from 00 with a total metric of \( 1.3 + 11.68 = 13.0 \) and from 10 with a total metric of \( 10.5 + 2.08 = 12.6 \). Since the branch from 10 has a lower overall metric, it wins. Losing paths are shown as dotted arrows. Total best-path metrics for each state are stored, and the previous values (and previous branch metric values) are discarded.

When all branches emanating from a state lose, the path entering that state also loses. This occurs for state 11 (shown as an oval with path metric value 7.3) at \( k = 2 \). Both paths leaving this state lose, and so the one entering it also loses. All are shown as dotted lines. Eventually, all states but one at each trellis depth will lose. When this happens to depth \( \ell \), all subsets up to depth \( \ell \) have been decoded. An example of this is time \( k = 3 \). When state 00 loses, a path going back to the initial state is eliminated, and the symbol \( r[0] \) is decoded to subset 0 (Figure 2-2). Although not guaranteed, a good rule of thumb is that a trellis depth of \( \ell \) is very likely to be decoded the time the Viterbi algorithm reaches depth \( \ell + 5\nu \) [10]. This results in a decoding delay of \( 5\nu T \), where \( \nu \) is the constraint length and \( T \) is one symbol period.

The Fano algorithm [15] is a fast, simple tree-searching algorithm. It is not guaranteed to find the maximum-likelihood sequence, but its performance is close to that of the Viterbi algorithm. This sequential decoder indirectly compares paths against others previously visited. By comparing the current path metric against a dynamic threshold set to keep track of the lowest Fano path metric, the Fano algorithm can judge the codeword without keeping track of every path encountered. Wozencraft
Figure 2-9: Event sequence for Viterbi decoder.
and Jacobs discuss the Fano algorithm in detail in [15]. The behavior and operation of the Fano algorithm is the same as in [15] with the exception of the metric used, and will be discussed here without full detail.

The Fano algorithm is a forward looking, best-first decoder. "Forward looking" means the decoder looks forward from its current state (the Viterbi algorithm looks back from a future state). "Best-first" means the Fano algorithm follows the best path emanating from the current node before considering others. This results in the decoder having to back up occasionally, often by several symbols. There is no longer a deterministic relationship between time and trellis depth. The decoding delay varies randomly as a result. The path metric progression is graphed in Figure 2-10, and the decoding event sequence for $r[0..4]$ using the Fano algorithm is shown in Figure 2-11.

![Path Metric: Fano Decoding Example](image)

Figure 2-10: Path metric progression for Fano decoder.

The Fano algorithm moves either forward or backward one step every time interval, and never visits the same node twice with the same threshold value. This prevents the decoder from becoming trapped in an endless loop, without having to
Figure 2-11: Event sequence for Fano decoder.
record all previously visited nodes. Remarkably, the storage required by the decoder consists only of the current path, threshold and metric, and a single binary variable \( \theta \). (See Wozencraft and Jacobs [15] for a full description. We know of no more recent reference that gives sufficient detail to allow the construction of a memory-efficient implementation.) The variable \( \theta \) is set whenever the current threshold \( M_T \) is violated (i.e., when \( M_{SD,p} > M_T \)) and is reset only when a move leaves or enters a node violating a tighter threshold \( M_T - \Delta \), where \( \Delta \) is the “threshold increment”.

Following our example, the Fano decoder starts at a trellis depth of 0 with a threshold value of \( M_T = 0 \). The decoder moves forward in the trellis from (A) to (E) by following the branches with the lowest metrics. At (E), the decoder is halted when both paths violate the threshold, and \( \theta \) is set. The decoder backs up, successively checking previous nodes for a “next-best” branch that satisfies the threshold. As can be seen in Figure 2-10, none of the next-best thresholds satisfy the threshold \( M_T = 0 \), and the decoder backs all the way up to (A). At this point, the threshold is raised by the threshold increment \( \Delta = 2 \) to \( M_T = 2 \). The decoder then follows all the steps from (A) to (H). Upon reaching (E) the second time, both paths violate the new threshold, and the decoder backs up to (F). At this point, the decoder does find a next-best path that satisfies \( M_T \). However, node (G) violates \( M_T - \Delta \), so \( \theta \) is reset, indicating the decoder is on a previously unexplored path. The decoder moves to (G) and continues decoding.

The decoder required exactly one “computation” to go from a trellis depth of 0 to 1, 1 to 2, 2 to 3, and 3 to 4, where we define a computation to be one “forward look”. However, it took eleven such computations to reach a depth of 5 from depth 4. This results in rather nasty behavior for computational load and decoding delay. The “buffer overflow” probability decreases linearly (and somewhat slowly) with buffer length.

In this example, we chose the threshold increment, \( \Delta \) to be 2. If we had chosen \( \Delta = 4 \), the decoder might have gone through (E) without ever returning. This might result in a decoding error. On the other hand, if we had chosen \( \Delta = 0.5 \), the decoder would travel from (A) to (E) and back at thresholds of \( M_T = 0, 0.5, 1, 1.5 \) before
finally reaching (G). This would result in a much larger computational load and a much higher decoding delay.

2.2 Equalization

An intersymbol interference (ISI) channel, depicted in Figure 2-12, consists of a linear channel filter \( g[k] \) followed by the standard additive white gaussian noise (AWGN) channel. If we send the signal \( v[k] \) through the channel, the channel output is \( r[k] = (v*g)[k] + z[k] \), where * denotes convolution. The convolution sum \( v[k]*g[k] \) represents the spreading of input energy at time \( k \) into other times.

2.2.1 Linear Equalization

Suppose the receiver knows \( g[k] \). If \( g[k] \) has an inverse \( g^{-1}[k] \), then we can remove the ISI by convolving \( r[k] \) with \( g^{-1}[k] \). This results in a signal \( r'[k] = v[k] + g^{-1}[k] * z[k] \). We see that the ISI has been removed at the cost of coloring the noise \( z[k] \) by \( g^{-1}[k] \). This is known as noise enhancement. This simple equalization technique is known as zero-forcing linear equalization (LEZF) because it uses a linear filter to meet the zero-forcing criterion [8].

Two implementations of the LEZF are shown in Figure 2-13. The first diagram shows a direct implementation of the LEZF. The second diagram is equivalent to the
first, as we can see in the frequency domain:

\[ R'(z) = R(z) + [1 - G(z)]R'(z) \]
\[ \Rightarrow G(z)R'(z) = R(z) \]
\[ \Rightarrow R'(z) = G^{-1}(z)R(z) \]

Thus, both equalizer structures send \( r'[k] \) to the decoder. The two filters are equivalent, but the second structure uses the direct channel response, which does not require inversion of the channel.

2.2.2 Precursor versus Postcursor ISI

Symbols are usually thought of as occupying orthogonal dimensions. For example, symbols might be transmitted serially (i.e., one by one) through a channel. Each symbol is received in its own segment of time, or time slot. These time slots are orthogonal from all others, simply because they do not overlap. An ISI channel is one in which the received data are smeared to the point where orthogonality is lost. Following our example, energy from one time slot is smeared into other time slots,
causing symbols to interfere with one another.

Intersymbol interference may be separated into two types: precursor ISI and postcursor ISI. If we define the current symbol as the cursor, then precursor ISI is interference from future symbols (time slots), and postcursor ISI is interference from previous symbols. The value of a symbol in its own slot at the receiver is $g[0]$ times the value at the transmitter. The precursor ISI is represented by $g[k]$ for $k < 0$. The postcursor ISI is represented by $g[k]$ for $k > 0$. For example, $g[1]$ is the fraction of the current symbol which will land in the next symbol to come. Alternately, $g[1]$ is also the fraction of the previous symbol which landed in the current symbol’s time slot.

### 2.2.3 Decision-Feedback Equalization

Looking now at Figure 2-14, we see a modification of the linear equalizer structure. The feedback path has been extended around the decoder. The new structure is known as a decision feedback equalizer.

To understand the operation of the DFE, it is first necessary to examine the operation of the LE filter as configured in the top of Figure 2-14. As discussed earlier, the received sequence of symbols is $r[k] = v[k] * g[k] + z[k]$. The received sequence $r[k]$ is sent through the inverse channel, resulting in $r'[k]$, which removes ISI at the cost of noise enhancement. We see that the expected value of each symbol $r'[k]$ is

$$E\{r'[k]\} = E\{v[k] + g^{-1}[k] * z[k]\}$$

$$= E\{v[k]\} + E\{g^{-1}[k] * z[k]\}$$

$$= v[k] + g^{-1}[k] * E\{z[k]\}$$

$$= v[k]$$

because $z[k]$ is zero-mean AWGN. Hence the benefit of LE can be viewed as a “re-
centering” of the sequence so that standard decoding methods can be used. However, if the channel is not fairly flat, the noise is significantly enhanced by the inverse filter \( g^{-1}[k] \), and can be shown [8] to strictly lower the signal to noise ratio. It is desirable to remove the ISI without enhancing the noise.

One method that attempts to remove ISI without enhancing the noise is decision feedback equalization (DFE). The DFE structure is shown in the bottom of Figure 2-14. The feedback path for the channel inverse filter is extended around the decoder. We assume for now that the decoder is merely a slicer, making hard decisions on each symbol individually. We also assume only postcursor ISI; more specifically, let us assume the channel impulse response is zero except for \( g[0] \), \( g[1] \), and \( g[2] \). Finally, assume for now that the decoder always decodes correctly.

![Figure 2-14: Linear versus decision feedback equalizers.](image)

Based on these assumptions, we can describe chronologically the operation of the DFE. We can, without loss of generality, take \( g[0] = 1 \). This amounts to a scaling of the received sequence by \( 1/g[0] \). Since the signal and noise are scaled equally, the SNR is unchanged. The feedback filter is then

\[
1 - G(z) = 1 - g[0] - g[1]z^{-1} - g[2]z^{-2}
\]

\[
= 1 - 1 - g[1]z^{-1} - g[2]z^{-2}
\]
\[ = -g[1]z^{-1} - g[2]z^{-2} \]

which results in an impulse response of

\[
\delta[k] - g[k] = -g[1]\delta[k-1] - g[2]\delta[k-2].
\]

Before reception begins \((k < 0)\), the symbols are all zeros. Thus, the output of the feedback filter is zero. At time \(k = 0\), the first symbol is received, and has the value \(r[0] = v[0] + z[0]\) (since \(g[0] = 1\)). The decoder outputs \(v[0]\), and the feedback output remains zero. At time \(k = 1\), symbol \(r[1] = v[1] + g[1]v[0] + z[1]\) is received. The output of the feedback filter at this time is \(-g[1]v[0]\). Thus, the signal fed into the decoder is

\[
\]

\[
= v[1] + z[1]
\]

and the decoder outputs \(v[1]\). At time \(k = 2\), symbol \(r[2]\) is received. Adding the output of the feedback filter, the decoder input at this time is

\[
\]

\[
\]

and the decoder outputs \(v[2]\). We see that, at least under our idealizing assumptions, the DFE cancels the ISI without enhancing the noise. The decoder input at any time \(k\) is \(v[k] + z[k]\), rather than \(v[k] + g^{-1}[k] * z[k]\).

### 2.3 Precoding

Decision feedback equalization is a strong tool for combating ISI, but it has drawbacks. The most obvious is the assumption that the decoder never makes errors. Most of the symbol error protection is across symbols, and the advantage of trellis coding can
only be gained by decoding many symbols together. Realistically, our slicer is likely to make decoding errors. When this occurs, incorrect information is sent through the feedback path, making future symbols appear less like their true values. This is known as error propagation, and is a significant problem with the DFE.

A potential solution to this is to use a soft-decision decoder in place of the slicer, with a hard decoder further on. This creates a trade off between error propagation and noise enhancement. The softer a decision we make, the more noise energy gets sent back through the feedback filter. In the extreme case, the decision is so soft that there is no decision at all, and the result is the original LE filter.

Yet another possible solution is to put the entire trellis decoder within the feedback loop. This creates a different type of problem. Trellis decoders require a sizeable delay between input and output. The DFE, on the other hand, requires nearly immediate decoder decisions to remove the ISI from the sequence. Thus, it is not feasible to place the entire trellis decoder in the DFE feedback path.

A feasible solution does exist in our situation. The DFE can be effectively moved from the receiver to the transmitter. This practice, known as precoding, has significant benefits, but requires changes to the decoder and added handshaking between the receiver and transmitter.

Normally, the DFE filter is trained using an adaptive algorithm with a known test sequence. This is also done with precoding. Once the filter coefficients or taps are established, a precoding system sends these tap values back to the transmitter, where the DFE-like filter is set up after the trellis encoder. Unlike the LE filter of Figure 2-13, there is no noise sent through the feedback path, and hence no noise enhancement. This linear filter, one form of precoding, is shown in Figure 2-15.

Using the linear precoding filter has two advantages. It creates no noise enhancement, and so has the performance of the DFE. Also, since it works in the transmitter, where the transmitted sequence is known exactly, there is no chance of error propagation. This second benefit can be viewed from the receiver point of view, in that the pressure of making quick decisions is removed from the trellis decoder.

There is still a major problem with using the linear precoding filter. This has to
do with an increase in the average transmit power and in the peak-to-average ratio (PAR), which is the ratio of the maximum possible instantaneous transmit power to the average transmit power.

To understand the importance of the PAR, we must examine our constraints. In designing a trellis code, the focus is on maximizing the effective Euclidean distance between codewords. This clearly is proportional to the minimum distance ($d_{\text{min}}$) of the signal constellation.

Returning to our example, we can compute the PAR of the 16-QAM constellation in the following way. The average power of a constellation is given by

$$P_{\text{ave}} = \sum_{x \in C} ||x||^2 P(X = x)$$  \hspace{1cm} (2.7)

where $C$ is the set of points in the constellation, and $P(X = x)$ is the probability that a particular point $x$ is used. In our example, each constellation point is used equally often ($\frac{1}{16}$ of the time). Furthermore, since power is not related to the phase of the constellation point, we can use the symmetry of the 16-QAM structure by averaging over a single quadrant. The average power equation reduces to

$$P_{\text{ave}} = 4\left(\frac{1}{16} \sum_{x \in C'} ||x||^2\right)$$  \hspace{1cm} (2.8)

where $C'$ is the set of constellation points in the first quadrant of the signal space:

$$C' = \frac{d_{\text{min}}}{2}\{1 + i, 1 + 3i, 3 + i, 3 + 3i\}.$$  \hspace{1cm} (2.9)
Finally, the average power for the 16-QAM constellation is $\frac{5}{2}d_{\text{min}}^2$. Comparing this with the peak power $P_{\text{peak}} = \frac{9}{2}d_{\text{min}}^2$, we find that

$$\text{PAR} = \frac{P_{\text{peak}}}{P_{\text{ave}}} = \frac{9d_{\text{min}}^2}{5d_{\text{min}}^2} = 1.8$$

which clearly is independent of $d_{\text{min}}$. So if we design our system to have an average power of $P_{\text{ave}}$, and we use this constellation, we must be allowed a peak power constraint greater than $1.8P_{\text{ave}}$.

We can now examine the effect of precoding on the peak value of the transmitted signal. Suppose a test sequence has been sent, and the channel is determined to be

$$G(z) = 1 - \frac{1}{2}z^{-1}.$$  

(2.11)

Given this channel, the precoder has a frequency response

$$G^{-1}(z) = 1 + \frac{1}{2}z^{-1} + \frac{1}{4}z^{-2} + \frac{1}{8}z^{-3} + \frac{1}{16}z^{-4} + \cdots.$$  

(2.12)

The symbol stream $v[k]$ is colored by this filter, and consequently $v'[k]$ can have large energy spikes. In fact, if a run of symbols $(3 + 3i)^{d_{\text{min}}}$ occurs, then the precoder output symbols can grow as high as $2(3 + 3i)^{\frac{d_{\text{min}}}{2}} = (3 + 3i)d_{\text{min}}$, resulting in a peak power of $18d_{\text{min}}^2$.

The overall system, originally designed with a PAR of 1.8 in mind, has an effective PAR of 7.2 as a result of precoding. To compensate for the increase in the PAR, $d_{\text{min}}$ must be scaled down by a factor of 2. This brings the peak power back down to $\frac{9}{2}d_{\text{min}}^2$, but the decrease of $d_{\text{min}}$ considerably increases error probability, as shown by Equation 2.1.

Tomlinson-Harashima precoding [8] solves this problem by using a “mod” operation to force the PAR down to an acceptable level. In effect, the finite constellation is extended to an infinite lattice. This operation can be viewed in the following way. We first superimpose a scaled, translated version of the $\mathbb{Z}^2$ lattice ($\mathbb{Z} \times \mathbb{Z}$, where $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$) on the complex plane. We then partition the lattice
into regular sets, as in Figure 2-16. Continuing the Ungerboeck example, the lattice
in Figure 2-16 is partitioned into 16-QAM constellations. When the feedback filter in
the precoder forces the proposed transmit symbol outside the borders of the center
constellation, the symbol necessarily lands within the borders of an image constella-
tion (but not necessarily on $\mathbb{Z}^2$). The origin is then considered to be the center of
the image constellation, and the amplitude of the transmit symbol is taken from the
new origin.

Figure 2-16: Extension of 16-QAM to infinite lattice.

Referring again to our example, having found the channel to be as in Equation
2.11, we implement a precoder without the mod operation (Figure 2-15). We wish
to examine the effect of the precoder and channel in the absence of noise. Letting
$d_{\text{min}} = 2$, suppose our trellis encoder output starts with a run of symbols $3 + 3i, 3 +
3i, 3 + 3i, \ldots$ The linear precoder output is then $3 + 3i, 4.5 + 4.5i, 5.25 + 5.25i, \ldots,$
approaching $6 + 6i$. The symbols pass through the channel, ending up as the original
encoder output sequence $3 + 3i, 3 + 3i, 3 + 3i, \ldots$. Compare this with the operation of the Tomlinson-Harashima precoder of Figure 2-17. The same encoder output sequence is fed into the precoder with the mod operation, with the following results. The first symbol passes through unchanged as $3 + 3i$. The second symbol has half of the first symbol added to it, resulting in a value of $4.5 + 4.5i$ going into the mod operator. This is outside the center constellation. The origin is shifted to $8 + 8i$, and the symbol output from the mod operator is $(4.5 + 4.5i) - (8 + 8i) = -3.5 - 3.5i$. The third symbol has half the modded second symbol added to it, resulting in a value of $1.25 + 1.25i$ moving unchanged through the mod operator. This iterates through the entire transmission. The same three symbols pass through the channel, becoming $3 + 3i, -5 - 5i, 3 + 3i, \ldots$ at the receiver. At the receiver, the symbols encounter another mod operator, producing a final output sequence the same as the original sequence $3 + 3i, 3 + 3i, 3 + 3i, \ldots$.

![Figure 2-17: Trellis encoder and Tomlinson-Harashima precoder.](image)

When noise is added before the receiver, interesting problems arise. Suppose a symbol, after passing through the channel filter, is equal to $3.9 + 2.3i$, and noise having value $0.2 - 0.1i$ is added to it. The noise forces the symbol into a different constellation region. Its new value, $4.1 + 2.2i$, gets translated by the mod operator to $-3.9 + 2.2i$. Using our standard distance metric, the effective noise component would be $-7.8 - 0.1i$, vastly higher than the channel noise value. This is dealt with using a mod-distance metric. To compute the mod-distance from a received symbol $r$ to a point $v$ in 16-QAM, we find the nearest point to $r$ in the lattice translate $v + 8\mathbb{Z}^2$. In other words, we find the closest distance from $r$ to $v$ relative to each of the image constellations, and select the smallest.
Chapter 3

The System Models

The simulated system is described in this chapter. Using the ideas discussed in Chapter 2, we can now assemble our model for testing the compatibility of sequential decoding of trellis codes with the precoding filters used to mitigate the effects of the ISI channel.

The simulated system includes in some sense every component of the overall system shown in Figure 1-1. The data source and sink are represented as generally as possible. The channel is a simple ISI filter and AWGN generator. The two modems are simulated in more detail. Where it made sense, we tried to follow the methods used in the current technology [1].

3.1 Data Source and Sink

When examining performance of trellis codes, the all-zero bit sequence is often the only one used for testing. We chose the data source to be a Bernoulli$(\frac{1}{2})$ i.i.d. binary random process. This process is maximum entropy, which means the process contains no redundancy, which in turn implies no error protection is provided by the source itself.

We are also interested in examining the performance of nonlinear codes. When linear codes are tested, one can make use of the symmetry which linearity implies to simplify the testing procedure. Let us assume our random process chooses $l$ symbols
of $k$ bits each. Our random process ultimately is choosing a single codeword of a possible $2^k$ in an $nl$ dimensional space (for a rate $k/n$ code). Since the code is linear, we can shift our origin to be the codeword we have chosen, with no change in the properties of the code. This property, known as geometric uniformity, allows us to consider only the all-zero codeword. Any errors which occur in decoding a randomly selected codeword would occur in an equivalent way with the all-zero codeword.

Nonlinear codes do not exhibit geometric uniformity, and a nonlinear code could protect one codeword better than another. Thus, it does not make sense to use the all-zero codeword. Further, we are precoding our sequence. Using the all-zero codeword would result in a repeated symbol pattern, which would not accurately describe the performance of the precoding scheme.

3.2 The Encoding Scheme

The encoding scheme consists of a trellis encoder and a precoder. Since our model never deals with continuous time, there is no need to go beyond the creation of the sequence of complex amplitudes. Both the V.34 encoder [1] and the more complex encoder [13] are systematic, rate 4/5 encoders.

The two encoding schemes differ only in the convolutional code used. The same QAM signal constellation is used, and the same precoding filter. Section 3.2.1 discusses the encoders for the two codes, and the differences between them. Section 3.2.2 describes the signal constellation and modulation scheme. The precoder operation is discussed in Section 3.2.3.

3.2.1 Convolutional Encoders

We will refer to the code taken from the V.34 standard [1] as the short code, in reference to its constraint length $\nu = 6$. The code from Wang’s paper [13] will be referred to as the long ($\nu = 16$) code.

The encoder for the short code is shown in Figure 3-1. While linear convolutional codes are easily characterized by their transfer function matrix, the short code is non-
linear, and thus difficult to characterize simply. Despite this, most of the information about the short code can be gathered from the encoder diagram.

We see from Figure 3-1 that the V.34 encoder has 6 memory elements. This indicates the constraint length of $\nu = 6$, or equivalently, that the trellis has 64 states. The presence of the AND gates indicates this code is nonlinear. The encoder takes in 4 bits and puts out 5 bits every time increment. This shows that the code has rate 4/5.

The short code is also systematic. This means that the encoder input bits can be seen plainly at the output. The output bits $y_{0..3}$ are independent of all other bit streams, while $y_4$ depends on all these as well as itself. Even though the bit streams $y_{0..3}$ are unchanged, they are still protected by the code. Thus all 5 bits $y_{0..4}$ are coded bits.

![Figure 3-1: 64 state encoder from the V.34 standard.](image)

The long code [13] is linear, and can therefore be represented by a transfer function matrix. The transfer function matrix for the long code is as follows:
Table 3.1: Polynomial coefficients for the long constraint-length code.

<table>
<thead>
<tr>
<th>j</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>( h_0(j) )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( h_1(j) )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( h_2(j) )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( h_3(j) )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( h_4(j) )</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3.1: Polynomial coefficients for the long constraint-length code.

\[
G = \begin{bmatrix} I_4 \mid \frac{1}{p_4(D)} \end{bmatrix} \begin{pmatrix} p_0(D) \\ p_1(D) \\ p_2(D) \\ p_3(D) \end{pmatrix}
\]

(3.1)

The component \( I_4 \) is a \( 4 \times 4 \) identity matrix, which shows the systematic nature of the encoder, i.e., \( y_{0..3} = x_{0..3} \). The terms \( p_{0..3}(D) \) are the feedforward polynomial terms relating each of the inputs \( x_{0..3} \) to the output \( y_4 \). The denominator term \( p_4(D) \) is the \( y_4 \) feedback polynomial term. Each polynomial term is given by the equation

\[
p_i(D) = \sum_{j=0}^{\nu} h_i(j)D^j
\]

(3.2)

where each term \( h_i(j) \) is listed in Table 3.1. Note again that \( \nu \) is the constraint length of the code. Also, since \( p_0(D) \) is the feedback polynomial, the component \( D^0 = 1 \) is always present in this polynomial. Table 3.1 may also be used to see the encoder tap locations on a systematic encoder template as in [13].

### 3.2.2 Mapping

Both encoders have rate \( \frac{4}{5} \). In addition to the 5 coded bits, 11 uncoded bits are mapped in a fashion similar to the example in Section 2.1. These 16 bits choose one of \( 2^{16} \) points in a \( 16 \times 16 \times 16 \times 16 \) four dimensional QAM constellation. The 5 coded bits are mapped into one of 32 4-dimensional subsets using the specific method described in [14]. The uncoded bits then choose which particular point in the subset is selected.
3.2.3 Precoder

The simulated precoder was based on the finite impulse response minimum mean-squared error decision feedback equalizer (FIR MMSE-DFE) described in [2]. This DFE is an extension of the results in [4] to the more practical situation of tap-length constrained filters. Given a fixed filter complexity and exact knowledge of the channel, this method finds the FIR filter $\tilde{G}(z)$ that minimizes the mean-squared error between the input $\bar{s}[k]$ and output $\bar{v}[k]$ of the decoder as in Figure 3-2, assuming $\bar{v}[k]$ is never in error. In practice, an iterative LMS scheme based on training data is used to find the filter, but we wish to isolate the effect of training error from other effects in the simulation.

![Figure 3-2: Decision feedback equalizer.](image)

As discussed in Chapter 2, the precoder performs in the same way as a DFE but never has to deal with incorrect decisions, hence the same filter $\tilde{G}(z)$ can also be used for precoding. The receiver determines the tap coefficients for the FIR MMSE-DFE using the method described in [2] and sends the coefficients back to the transmitter.

A four tap precoder was used for the ISI AWGN channel. No precoding was performed for the standard AWGN channel.

3.3 The Channel

Two channels were simulated; a memoryless AWGN channel and an AWGN inter-symbol interference channel. The standard AWGN channel is of the form in Figure 2-4. As in Chapter 2, the simulated AWGN channel simply adds a complex noise
sequence $z[k]$ to the transmitted sequence $v[k]$, resulting in a received sequence $r[k]$. The signal-to-noise ratio of $r[k]$ was controlled by adjusting the variance of the noise $z[k]$.

The AWGN intersymbol interference channel shown in Figure 2-12 is similar to the standard AWGN channel, but shapes the spectrum $V(z)$ of the transmitted sequence $v[k]$ with the channel characteristic

$$G(z) = 1 + \frac{z^{-1}}{2} + \frac{z^{-2}}{4} + \frac{z^{-3}}{8} + \frac{z^{-4}}{16}$$

normalized to have unit gain. The frequency characteristics of the simulated ISI channel are shown in Figure 3-3.

Figure 3-3: ISI channel frequency characteristic.
3.4 The Decoding Scheme

The decoding scheme consists of a twenty tap linear equalizer, a trellis decoder, and an uncoded bit decoder. The optimal FIR equalizer [2] incorporates the predictor (feedforward) portion of the precoding equalizer in the linear equalizer at the receiver. The linear equalizer tap coefficients are determined at the same time as the precoder tap coefficients. Once established, the linear equalizer can compensate for time variation in the channel using a least mean-square adaptation method [3, 10].

In simulation, the equalizer remained constant after its initial setting. Once the symbols were filtered by the linear equalizer, they moved into the trellis decoder. Two decoding methods were simulated; the Fano algorithm and the Viterbi algorithm. The trellis decoder operation is the same as described in Chapter 2. A few aspects relevant to the simulated decoders remain to be discussed.

An important parameter in discussion of the simulation results is the number of decoder computations per symbol. A computation is a unit of work performed by a decoder. A reasonable definition of this unit is the amount of work required of the decoder to decide its next state given its current state. In the Fano algorithm, one computation equates to one forward look, and the next state either adds or subtracts a symbol from the decoded path. The Viterbi algorithm always adds a symbol to a decoded path, but it performs a computation for every possible state. If the Fano algorithm ran without ever having to back up, the decoder would perform one computation per symbol. This is the minimum possible computation rate.

A computation is not a direct measure of the number of operations required, such as floating point or integer operations. The number of operations comprising a computation varies greatly with the code structure and complexity. The example used in Chapter 2 is a rate $\frac{1}{2}$ code, and the single input bit means only two path metrics are calculated per computation. The simulated codes are of rate $\frac{4}{5}$, and thus sixteen path metrics are calculated per computation. Since both simulated codes have the same rate, it is fair to use this definition of computations as a unit of comparison.

As mentioned in Chapter 1, use of the Fano algorithm requires buffer management
An infinite buffer length was assumed during simulation, and so the problem of buffer overflow was not addressed.

The trellis decoder only determines the coded bits. As mentioned in Section 3.2.2, there are 11 uncoded bits per 4D symbol. The 5 coded bits determine which of 32 subsets the received symbol belongs to. There are $2^{11} = 2048$ symbols in each subset. Each symbol in the subset is spaced by $2\sqrt{2}d_{\text{min}}$. The uncoded bits are determined simply by quantizing the received symbol to the closest valid subset point using the mod-distance metric.

At the start of the Fano algorithm, a mark is set at zero symbol depth. Each time a depth of 100 4D symbols beyond the current mark is reached, the marker is moved to that depth. During simulation, a count is kept of the number of computations required to go from one marker to the next. In this way, the computational distribution can be determined.

After each simulation run, the position of each decoding error was recorded. This allowed viewing of path error lengths. The number of bit errors and block errors were computed. These values are defined and discussed in Chapter 4.
Chapter 4

Simulation Results

In this chapter we examine and interpret the results of simulation of the channels, codes, and decoding methods described in Chapters 2 and 3.

We simulated the decoding of two trellis codes: a “short” code (64 states) and a “long” code ($2^{16}$ or 65536 states). Both codes were decoded using sequential decoding. Additionally, the short code was decoded using Viterbi decoding. Because Viterbi decoding cannot be used with long trellis codes, we could not examine the Viterbi algorithm performance with the long code. All simulations were performed for both the standard (non-ISI) AWGN channel, and the ISI channel (mitigated by Tomlinson precoding) described in Chapter 3.

We start by comparing the behavior of our two decoding techniques using the 64 state V.34 code, for both the ISI and non-ISI channel. We then examine the effect of ISI on the long code performance relative to the short code performance.

4.1 Sequential versus Viterbi Decoding

As discussed in Chapter 2, sequential decoding is a suboptimal decoding technique. Although many aspects of its behavior are understood, we must still answer the question: How much better is the Viterbi algorithm under equivalent circumstances? For complex codes, this question is best answered through simulation. Using the methods described in Chapter 3, we simulated the 64 state V.34 trellis code over each
channel for each decoding method. The bit error probability was recorded for several signal-to-noise ratios, resulting in Figure 4-1.

Figure 4-1: Waterfall curves for various decoding techniques: AWGN channel.

Figure 4-1 is a waterfall plot, named for the appearance of the curves plotted. This particular graph plots the Monte Carlo estimates of the bit error probability $P_{bit}$ as a function of normalized signal to noise ratio (SNR$_{norm}$). The estimate of the bit error probability was computed by comparing the encoder input $x[k], u[k]$ with the decoder output $\hat{x}[k], \hat{u}[k]$, finding the number of incorrect information bits, and dividing by the total number of information bits sent.

To better understand waterfall plots, a few characteristics should be noted. Waterfall plots describe the signal strength necessary for achieving a certain error probability with a particular coding scheme. Suppose we decide to communicate at a rate of $R$ bits per channel use. Using the information theoretic capacity equation for our particular channel, we assign $R$ as our capacity and solve backwards for SNR$_{min}$, the theoretic minimum SNR for error free communication. Assuming fixed noise, error
free communication is not possible with transmit power lower than that dictated by \( \text{SNR}_{\text{min}} \). The SNR for any practical communication scheme is normalized by \( \text{SNR}_{\text{min}} \) as follows:

\[
\text{SNR}_{\text{norm}} = \frac{\text{SNR}}{\text{SNR}_{\text{min}}}
\]  

(4.1)

So, capacity is represented on the waterfall plot by unit \( \text{SNR}_{\text{norm}} \) or zero decibels, as in Figure 4-1.

All curves on the waterfall plot represent communication at the same rate. Communication schemes are judged by their proximity to the capacity line at a particular error probability. Strong schemes are further to the left of the graph than weaker ones, because they require less signal power to achieve the same error probability.

Looking at Figure 4-1, we see curves for the sequentially decoded V.34 code (labeled “SD, Short”), and the same code decoded with the Viterbi algorithm (labeled “VA”). The capacity line (0 dB) and uncoded QAM transmission (labeled “2 × 256 QAM”) are shown for scale reference. The waterfall plot of the sequentially decoded long code simulation (labeled “SD, Long”) is also shown. This curve will be discussed later in this chapter. Figure 4-2 plots the same codes and decoding algorithms for the ISI channel.

For the short code, through our simulated range, maximum likelihood decoding performs an average of about 0.1 dB better than sequential decoding. This is true for both channels.

Figure 4-3 is a waterfall curve that depicts the block error probability as a function of \( \text{SNR}_{\text{norm}} \) over the AWGN channel. Also shown are the error bars representing a 95% confidence interval for each point [7]. The upper and lower error bar limits are marked by an “x”. A block consists of 100 4D symbols, corresponding to 1500 information bits. The block error probability was computed by comparing an input block with the corresponding output block. If any bit was different, then that block was in error. The number of blocks in error was then divided by the total number of blocks sent, resulting in the block error probability estimate. The error bars are approximate because a Gaussian distribution was used rather than a binomial, and
because the correlation of burst errors that span two blocks was neglected. Both effects are minor.

Block error probability is often more relevant in practice than bit error probability, particularly when the entire block is retransmitted if any part of that block is in error. Figure 4-3 confirms that the Fano algorithm (SD, Short) performs only slightly worse than the Viterbi algorithm (VA) with respect to block error probability.

### 4.2 Long Constraint Length Code

The long constraint length code was also simulated for each channel. Figure 4-1 confirms results obtained by Wang and Costello in [13], showing that the sequentially decoded long code achieves a coding gain of about 4.9 dB over uncoded 2×256 QAM at a bit error rate of 10⁻⁵. A more relevant comparison for our discussion is that the coding gain achieved is about 1.1 dB higher than that of the Viterbi decoded short code at a bit error rate of 3×10⁻⁵. It is worth noting that this is, in a sense,
Figure 4-3: Block error waterfall curves: AWGN channel.

comparing apples and oranges. The normal failure mode for the Fano algorithm is buffer overflow, which is not considered here beyond the examination of the computational distribution later in the chapter. Buffer overflow can be dealt with using a “Buffer Looking Algorithm” [12], or by framing the data. Either method degrades performance. These methods, however, are not examined here.

Figure 4-2 shows that again, the sequentially decoded long code performs about 1.1 dB better than the Viterbi decoded short code. This shows that the differences in sequential decoding performance are negligible between the ISI and non-ISI channels. This fact is shown strikingly in Figure 4-4, which displays the waterfall curves for the sequential decoding simulations. Results of both channels are superimposed. With both codes, the AWGN channel curves (shown as solid lines) and the ISI channel curves (shown as dotted lines) lie very close to one another.
4.3 Computational Distributions

The computational load of sequential decoding is random. As mentioned in Chapter 2, the Viterbi algorithm performs a fixed number of computations per symbol. This is acceptable for short constraint length codes, but is paralyzing for the high constraint length codes that provide high coding gain.

Figures 4-5 and 4-6 plot computational distribution results for the sequentially decoded long code. These graphs show the estimated probability $P(N_c \geq N)$ of requiring more than $N$ computations to advance the length of the hypothesized decoded sequence by one symbol, as a function of $N$. Every advance requires by definition at least one computation, so

$$P(N_c \geq 1) = 1 \quad (4.2)$$

where $N_c$ is a random variable representing the number of computations. This distribution is well approximated by the Pareto distribution [12, 15]. The distributions
plotted in these two figures were computed by recording the number of computations required to move a depth of 100 symbols forward, as described in Section 3.4. The values recorded were then divided by 100 to normalize $N_c$ to one symbol, and a histogram was made, resulting in the plotted curves.

![Computational Distribution for Sequential Decoding](image)

Figure 4-5: Comparing computational distributions for each channel.

Figure 4-5 superimposes the computational distribution plots for the AWGN channel (plotted as “o”s) and the ISI channel (plotted as “x”s) for two values of SNR$_{norm}$. The computational distribution does not change noticeably between the AWGN and ISI channels. Together, Figures 4-4 and 4-5 show that sequential decoding of long codes is as feasible on ISI channels as it is on AWGN channels.

Figure 4-6 shows the computational distribution for decoding of the long code over the ISI channel. The distributions for the Fano algorithm, labeled “SD”, are displayed for SNR$_{norm}$ equal to 3dB, 3.1dB, 3.3dB, and 3.5dB. As a (possibly unfair) comparison, the distribution for the Viterbi algorithm, labeled “VA” is shown. As expected, as SNR$_{norm}$ increases, the downward slope of the sequential decoding computational
distributions becomes steeper. This is evidence that the sequential decoder rapidly becomes well-behaved as $\text{SNR}_{\text{norm}}$ moves away from the computational cutoff rate [6, 12], which, based on our simulations, was found to be slightly less than 3 dB.

Figure 4-6: Long code computational distributions for the ISI channel.
Chapter 5

Conclusions

The results of Chapter 4 show that sequential decoding performance is essentially unaffected by precoding. Hence, the gains of high constraint-length trellis codes can be realized using sequential decoding whether or not the channel introduces intersymbol interference. At a bit error probability of $3 \times 10^{-5}$, a coding gain of 1.1 dB over the V.34 code is realized using a $2^{16}$ state code. Higher constraint length codes should do slightly better at the cost of increased buffer overflow probability and resynchronization delays. By bringing together precoding, sequential decoding, and shell mapping (which achieves about 1 dB of the ultimate shaping gain of 1.53 dB), it is possible to signal at rates close to the computational cutoff rate of an ISI Gaussian channel (about 2 dB from capacity).

An improvement in coding gain can be interpreted in at least three ways: as a decrease in the required signal-to-noise ratio to achieve a fixed error probability, as a decrease in the error probability at a fixed SNR, or as an increase in the communication rate for a fixed error probability and SNR. For a voiceband modem, the average SNR is predefined and the error probability is a fixed design parameter based on external requirements. Thus the last of these interpretations is most appropriate. A 1.1 dB improvement adds roughly 1.17 kb/sec at a bandwidth of 3200 Hz, the typical bandwidth for a modem following the V.34 recommendation. At the V.34 maximum rate of 28.8 kb/sec, this represents a four percent increase in transmission rate.

Asymmetric digital subscriber line modems, currently undergoing standardiza-
tion, are a potential application for sequential decoding. The high bit rate makes computation costly. Sequential decoding can increase performance while requiring less computational effort than a comparable Viterbi decoder. Turbo codes and iterative decoding are an alternative way to approach capacity, but they require a substantial decoding delay.

Many practical issues still must be addressed. Buffer overflow and resynchronization remain crucial to practical application. Framing is a simple solution, but probably not the most efficient. An analytical approach to this problem, possibly via large deviation theory, may be a useful direction for future work. Efficient hardware and software implementations of sequential decoding must also be considered. In hardware, the Viterbi algorithm can be to some extent parallelized. But for a software implementation (which is generally cheaper to design and maintain), the Fano algorithm might be a better solution than the Viterbi algorithm. Sequential decoding should become more prevalent as software modems proliferate.
Bibliography


