Palo Alto Research Center

Trellis Data Compression

by Lawrence Colm Stewart

XEROX

Trellis Data Compression

by Lawrence Colm Stewart

CSL-81-7 JUNE 1981

Abstract: See page iii.

This report reproduces a dissertation submitted to the Department of Electrical Engineering and the Committee on Graduate Studies of Stanford University in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

CR Categories: 3.24, 3.81, 5.6.

Key words and phrases: trellis encoding, tree encoding, information theory, data compression, speech compression.

XEROX
PALO ALTO RESEARCH CENTER
3333 Coyote Hill Road / Palo Alto / California 94304

© Copyright 1981

by

Lawrence Colm Stewart

ABSTRACT

Tree and trellis data compression systems have traditionally been designed by using a tree or trellis search algorithm to improve the performance of traditional coding systems such as adaptive delta modulation or predictive quantization. Recent work in the area of vector quantization has suggested the possibility of designing new tree and trellis codes which are well matched to particular sources. The main design procedure iterates on a long training sequence to improve the performance of an initial trellis decoder. An additional procedure, given a trellis decoder, can produce a decoder of longer constraint length which performs at least as well. Combined, these algorithms provide a complete design procedure for trellis encoding data compression systems.

For random sources, many existing data compression systems can be readily improved and performance close to the rate-distortion bound can be obtained. In the applications area of speech compression, tree and trellis codes designed with these algorithms permit the construction of low rate speech waveform coders, low rate residual excited linear predictive coders (RELP), and a new kind of hybrid tree coder which provides good quality speech at rates below 7000 bits per second.

ACKNOWLEDGEMENTS

I wish to thank my advisor, Professor Robert M. Gray, for his direction and excellent guidance and the faculty and students of the Stanford Information Systems Laboratory for their encouragement and interest. My thanks also go to my parents for their many years of patience and support. My special thanks go to the staff of the Xerox Palo Alto Research Center, whose wonderful tools made this work possible.

Support for this work was provided by the National Science Foundation graduate fellowship program, by Joint Services Electronics Program contract number DAAG-0047, by U.S. Army Research Office contract number DAAG29-80-K-0073, and by the Xerox Corporation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Data compression, or source coding, systems provide mechanisms for transferring data from an information source to an information receiver in the presence of constraints on the amount of information which may be transferred or on the rate at which it may be transmitted. As might be expected, such systems have a large variety of applications, from communicating speech over low rate digital channels to storing computer data files on a disk memory of limited size.

The overall problem of communications can be broken into two parts, as illustrated by the traditional communications system model of Figure 1.1 [13]: what aspects of the information coming from the source should be transmitted, and how should they be transmitted? The theory of channel coding, encompassing the channel itself as well as the channel encoder and decoder, addresses the latter problem, while data compression addresses the former. Whereas channel coding seeks to transmit information in an error free manner over a noisy channel, source coding assumes an error free channel and seeks to select appropriate information for transmission by removing redundant or less important aspects of the information from the source.

1.1 Discrete Time Source Coding

This thesis is primarily concerned with a particular problem in the area of data compression: the encoding of discrete time waveforms – sampled data sources – for transmission in order to achieve the maximum possible fidelity of reproduction subject to a limited rate of information transfer. More specifically, this thesis describes and applies new algorithms for the design of tree and trellis encoding data compression systems.

Discrete time source coding systems are sometimes divided into two categories: block codes and sliding block codes. A block code accepts a fixed number of samples, or symbols, from the source, and releases a fixed number of symbols to the communications channel. The encoding process proceeds independently on each block. A sliding block code works in an incremental fashion. For each step, the encoder accepts a single symbol from the source and releases a single symbol to the channel. The sliding block encoder may retain some internal state information; thus the encoding may not be independent from symbol to symbol. The existence of good block codes has been known since the 1950's [75],[76], but the existence of good sliding block codes was not fully established until the 1970's [20],[38],[45].

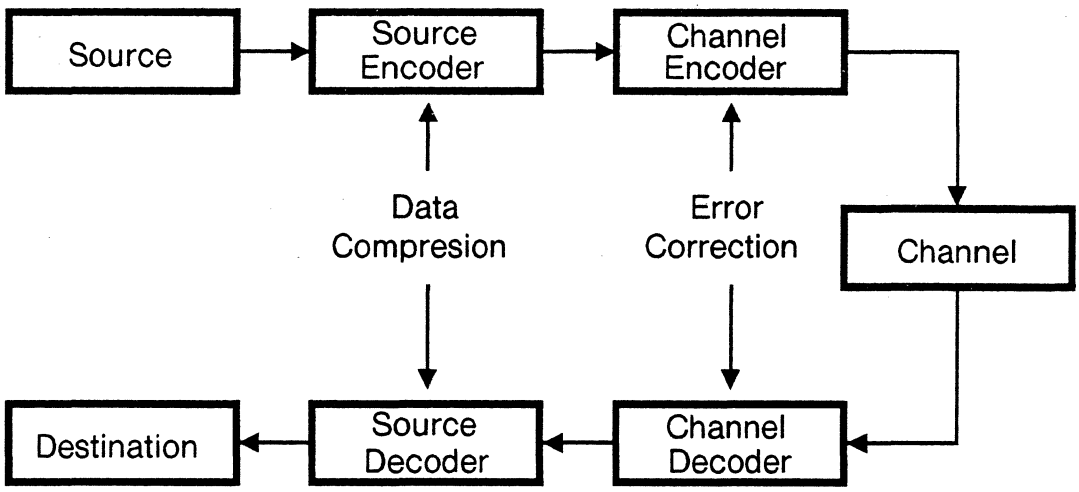


Figure 1.1 Communications system

Unfortunately these papers offer only existence proofs without describing ways of actually constructing good codes.

For sliding block codes, it is an oversimplification to state that the encoder accepts and releases a single symbol at each step. A sliding block code may deal with larger fixed blocks or with variable length groups of symbols. The key difference between the block and sliding block concepts is that the block encoding proceeds independently from block to block, while the sliding block encoding does not. A sliding block system which accepts multiple symbols may be considered to operate on single symbols from a vector valued source (a supersource). Further, a sliding block system that considers the indefinite (possibly infinite) past sequence of source symbols is well defined.

The tree and trellis codes discussed in this thesis are members of the sliding block class.

1.2 Tree and Trellis Systems

Consider the problem of encoding a source sequence $\{x_j\}$ into a channel sequence $\{u_j\}$ with the ultimate aim of producing a reproduction sequence $\{y_j\}$ at the decoder output. Viewed from the decoder, the particular output sequence generated is a consequence of receiving a particular one of all the possible channel sequences. The particular channel sequence received up to a time j can only be extended in a certain number of ways, depending on the number of possible values of a channel symbol. Similarly, a given decoder output sequence up to time j can only be extended in a

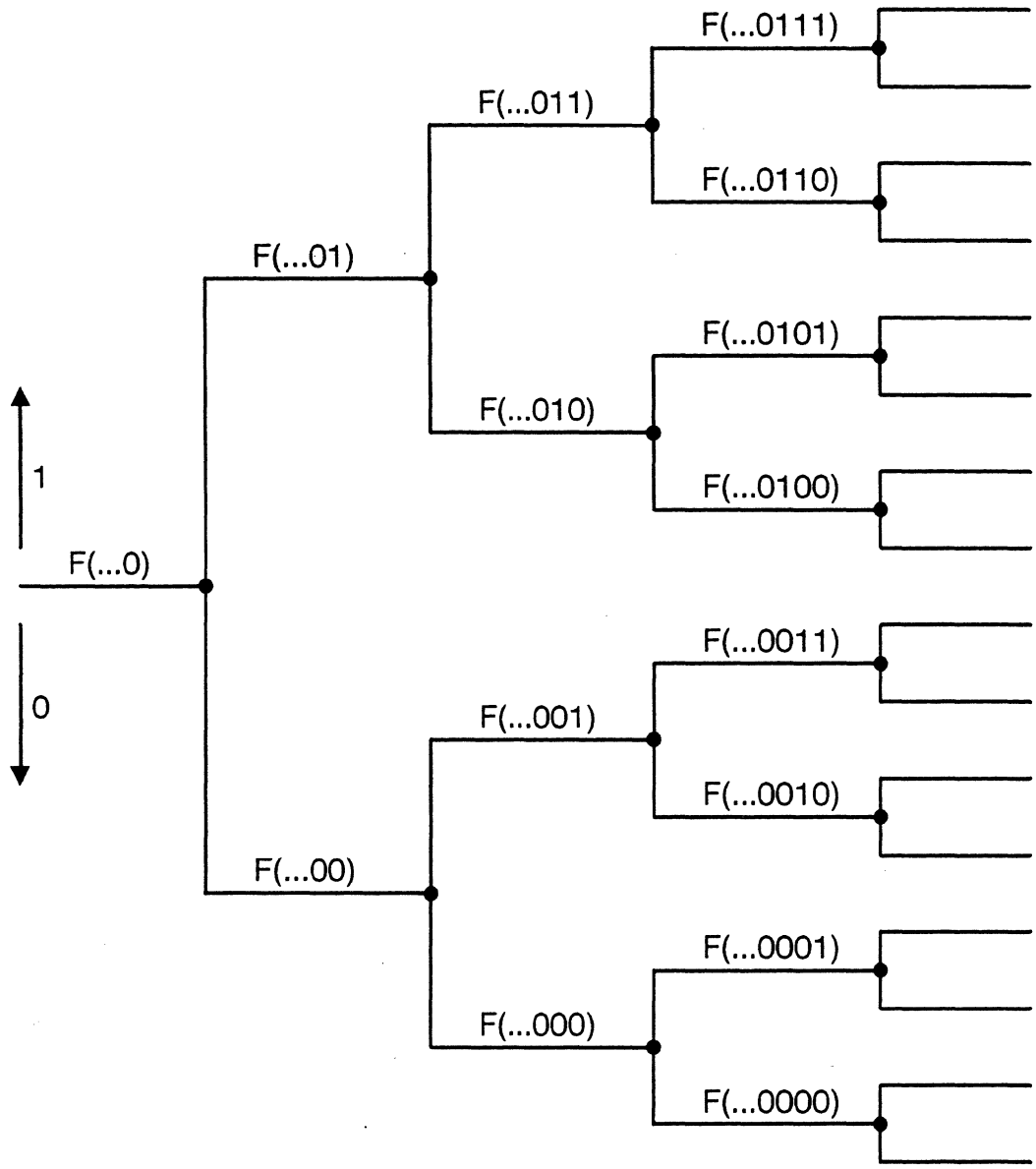


Figure 1.2 *Tree diagram for a one bit per symbol sliding block code. Because the channel symbols have two possible values, the tree branches two ways whenever a symbol is received from the channel. The arguments of the decoder function F represent the past sequence of channel symbols.*

fixed number of ways. This association of output sequences with channel sequences gives rise to a tree structure (Figure 1.2) in which the levels of the tree – points of successive branching – occur at each time a symbol arrives from the channel. The various alternatives represented by the different branches from a node are associated with the particular value of the symbol received. The decoder outputs are simply the labels on the tree branches. The sequence of channel symbols, or the *path*

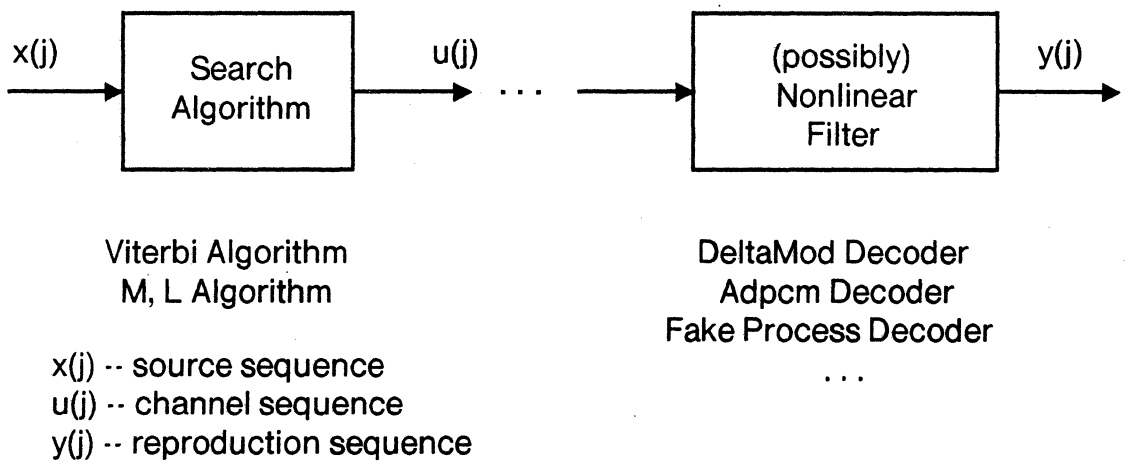


Figure 1.3 Data Compression System

map, steers the decoder through the tree, producing an output sequence. If the decoder is deterministic, initialized in a known state, then the encoder can predict what the decoder will do for each channel sequence the encoder might select. The tree structure of the decoder, combined with its predictable behavior, leads to a description of tree coding systems as shown in Figure 1.3. A tree system just consists of a linear or non-linear filter as the decoder, matched to a tree search algorithm as the encoder.

The decoder is an arbitrary device for transforming the channel sequence $\{u_j\}$ into the reproduction sequence $\{y_j\}$. Depending on the alphabet of the channel symbols, the decoder might be implemented as a linear time-invariant digital filter, but usually at least some nonlinear components will be necessary to translate the channel symbols into a suitable alphabet. For example, a delta modulation decoder usually consists of a transformation from a binary channel alphabet to the numbers $+1$ and -1 followed by a simple digital filter.

The process of encoding is accomplished by the tree search algorithm. By experimenting with a local copy of the decoder, the search algorithm can examine the consequences of transmitting various channel sequences and select the best available encoding. In such systems, "best" is defined by a single-symbol (additive) distortion measure. The overall distortion between the source and reproduction sequences is the sum of the distortions between the individual source and reproduction symbols. The tree encoder seeks to transmit the channel sequence which will produce the lowest possible overall distortion.

When the decoder of a tree coding system contains a finite amount of state

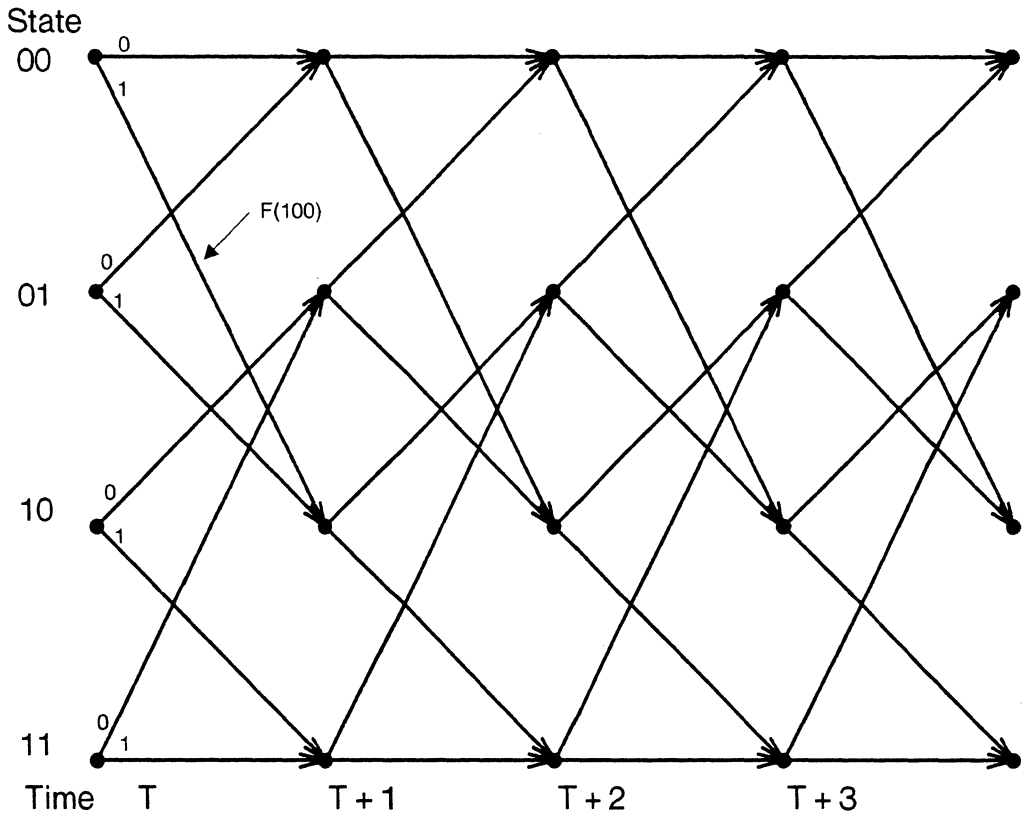


Figure 1.4 *Trellis for a one bit per symbol length 3 decoder*

information, the code tree formed by mapping the input-output behavior of the decoder assumes a more compact form. Since the decoder can be in only one of a finite number of states, the possible decoder outputs are only those which would result from receipt of each of the possible channel symbols while the decoder is in each of its possible states. This interpretation gives the code tree a folded tree, or trellis, structure in which the tree paths are now drawn as transitions between decoder states. The paths leaving a particular state may later merge as they reach an identical decoder state by different routes. In fact, digital decoder implementations always contain only a finite amount of state information, but even as few as 20 bits of internal memory would give the decoder over a million states! For this reason, a trellis decoder will usually be so identified only when the number of decoder states is fairly small – perhaps a few thousand.

A typical code trellis is shown in Figure 1.4. The word trellis is used because the merged code structure closely resembles a garden trellis. Only four sections of the trellis are shown; the structure is actually replicated indefinitely to the right and to the left. The trellis of Figure 1.4 is generated by the one bit per symbol constraint-length 3 decoder shown in Figure 1.5. (Constraint length is an historical

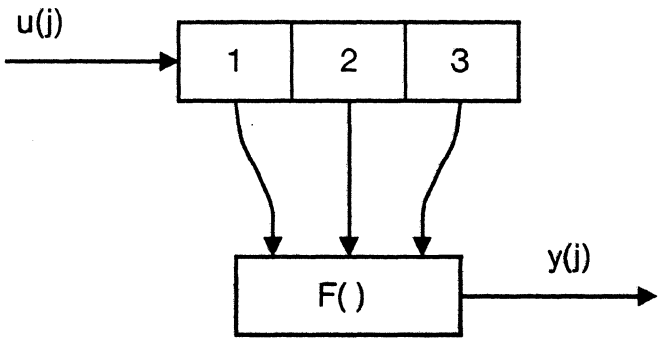


Figure 1.5 Constraint-length 3 trellis decoder

term from convolutional channel coding. It refers to the length of a shift register such as the one in the figure.) During operation of this decoder, bits from the channel are entered into a three bit shift register. As each bit is entered, the decoder output is generated by a function of the current and two previous channel bits – in the most general case, this function might be a table of eight values. This trellis has four states because only the contents of the two leftmost shift register stages are involved in determining the action of the decoder for the next input. The rightmost bit is shifted out when the new channel bit arrives. As an example, if the trellis decoder is in state 00 at time T and a 1 bit arrives from the channel, the decoder would move to state 10 and produce the output $F(100)$.

The encoder of a tree or trellis source encoding system may be implemented by a search algorithm. Because many effective encoding algorithms are known, the more difficult part of the design of a tree or trellis coding system lies in the design of the decoder. (We will often speak of a "tree code" or a "trellis code" when, in fact, a tree or trellis *decoder* is meant.) Decoders may be drawn from any of the traditional waveform coding techniques, such as delta modulation or predictive quantization. Systems using such *plagiarized decoders* achieve improved performance solely by use of a search algorithm rather than their traditional encoders. In the literature, various other methods for selecting decoders have been proposed. For speech sources, for example, attempts have been made at the design of tree decoders based on predictive quantizers which match the average correlation properties of speech [17]. Variational methods have been proposed to improve decoders based on predictive quantizers and Linde and Gray have used the notion of a fake process and theoretical ideas based on a measure of the similarity of random processes to suggest decoders [10],[55]. While most of the codes based on traditional

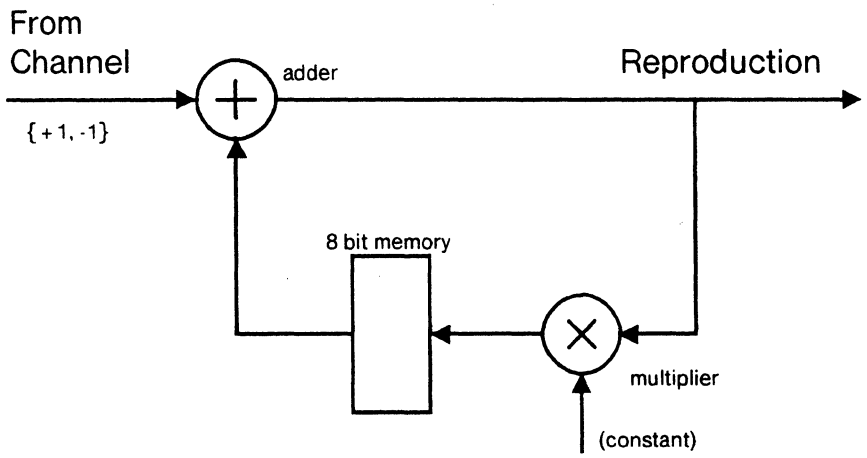


Figure 1.6 First order autoregressive filter decoder utilizing 8 bit arithmetic. (This decoder might be used in a predictive quantization system.)

systems incorporate structures similar to recursive digital filters and thus have such a large number of states that they generate tree codes, several investigators have used decoders incorporating either transversal digital filter approximations of recursive filters or decoders originally based on finite impulse response models [8],[52],[82]. These decoders have a relatively small number of states and therefore a trellis structure.

Even when a decoder incorporates feedback, such as when a recursive filter is used, if the internal structure of the decoder is purely digital (and sufficiently small), the code may still be considered to have a trellis structure, with the logarithm of the number of states equal to the sum of the number of bits of storage in the decoder. Thus a first order autoregressive filter using a delay element with eight bits of precision can have only 256 states (Figure 1.6). In a one bit per symbol system, an eighth order (constraint-length 9) transversal filter decoder also has only 256 states. (Again, one of the bits is shifted out before the new channel symbol is shifted in, so it need not be actually stored.)

1.3 Tree and Trellis Encoding

A rich variety of algorithms are known that address the problem of tree and trellis encoding, or tree and trellis search. The simplest approach is to select the current channel symbol solely on the basis of the current source symbol. This technique is known as *single path search* and is represented by the traditional encoders of systems like delta modulation – if the input is rising send a 1, otherwise send a 0. More complicated encoding algorithms, *multiple path search*, delay the sequence

of source symbols in order to "look into the future" and examine the longer range consequences of various encodings. In these encoders a succession of source symbols is matched against several possible channel sequences in a search for the encoding with minimum average distortion as determined by the given distortion measure.

The study of tree search algorithms has been of interest since the earliest work on convolutional channel coding. The Viterbi Algorithm [25],[80] is optimum for searching the trellis structures associated with finite-state decoders, but has computational cost exponential in the constraint length of the code. This large expense stems from the fact that the Viterbi Algorithm performs an exhaustive search of the code trellis. Of the non-optimal algorithms, some, such as the Fano [44, sec. 10.4] and stack [6] algorithms, are derived from problems of decoding error correcting codes. Other algorithms have been designed specifically for the tree source encoding problem. Of this group, the M,L Algorithm [46] is a breadth-first tree search. Others, such as the single stack algorithm [26] and the 2-cycle algorithm [9], are classed as depth-first or metric-first (distortion-first) types [4],[5].

The key difficulty with tree encoding systems is that the number of potential paths grows exponentially as the search algorithm delays for a longer time in order to look deeper into the code tree. In one way or another, the various tree search algorithms seek to overcome this difficulty. They either give up performance in trade for efficient operation or take special advantage of particular decoder (tree) structures.

Two search algorithms are used in this thesis, the Viterbi Algorithm and the M,L algorithm; their operation is briefly discussed in Appendix B. Both algorithms have fixed encoding delay and bounded search effort. Some other available algorithms run faster on the average, but require a variable amount of encoding delay or have a variable running time – with consequent buffering problems. The Viterbi Algorithm is optimal, but is suitable only for trellis codes because it takes advantage of their finite-state structure. The M,L Algorithm, although not optimal, can be used both for tree codes and for trellis codes.

1.4 Decoder Implementation

The decoder examples of section 1.2 are instructive. Because we are considering a digital communications channel, and the channel symbols take on only a few values – two, in the case of one bit per symbol systems – the decoders for quite complex tree or trellis codes can have very simple structures. A first order predictive quantization decoder, such as the recursive filter described earlier, might be implemented by

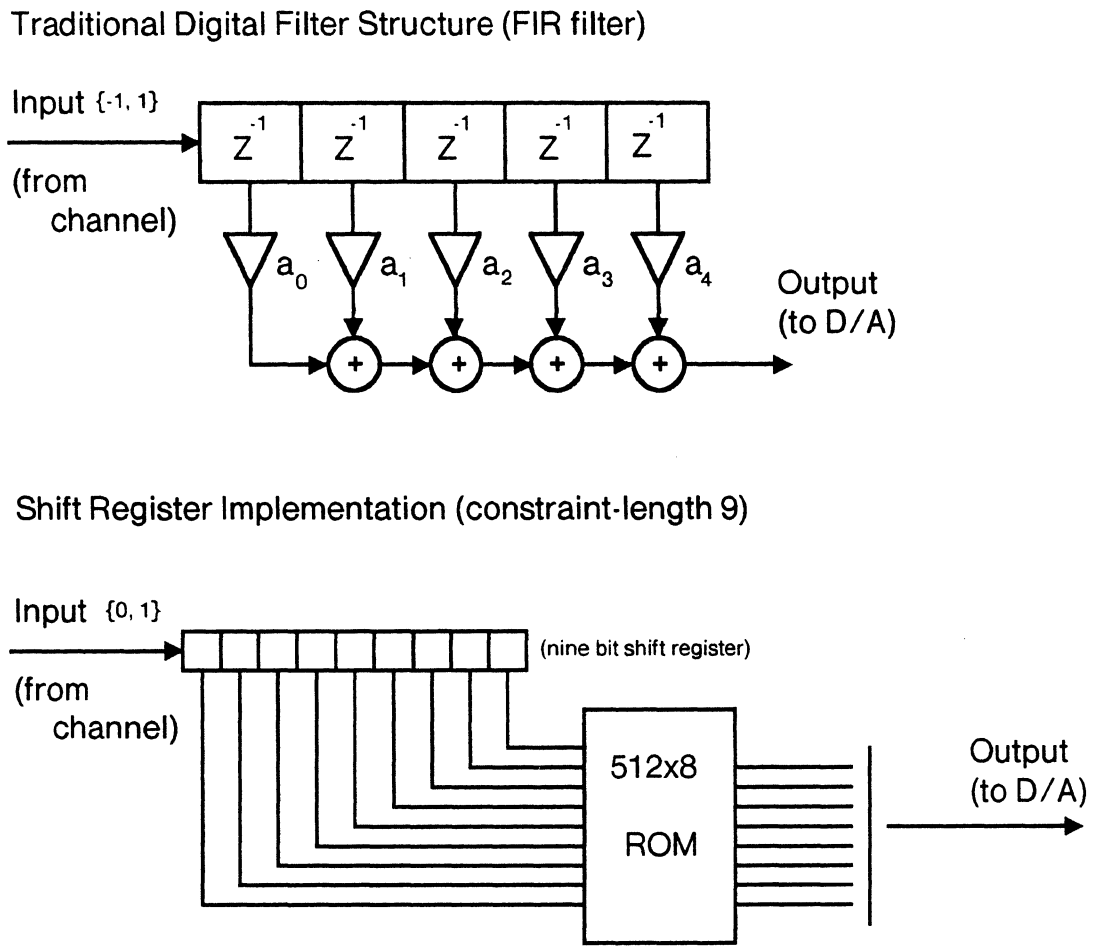


Figure 1.7 *Trellis decoder implementation*

a multiplier, a register, and an adder. In the case of the constraint-length nine transversal filter, rather than use any of the traditional transversal linear digital filter structures, the decoder may be simply implemented by a 9 bit shift register driving a 512 word read-only memory (Figure 1.7). All the complexities of the filter tap weights, multipliers, and adders are replaced by the read-only memory acting as a lookup table or codebook. The codebook output might be used to directly drive a digital to analog converter. In the case of a 256 state decoder, the most general trellis decoder structure possible could be implemented, as in Figure 1.8, by using two read-only memories. One of the memories provides the decoder output function, while the other provides a next-state function, implementing a general finite-state machine. The first order recursive filter of Figure 1.6, for example, could utilize such a structure in place of the multiplier, register, and adder.

Although most of the work reported by this thesis uses trellis decoders imple-

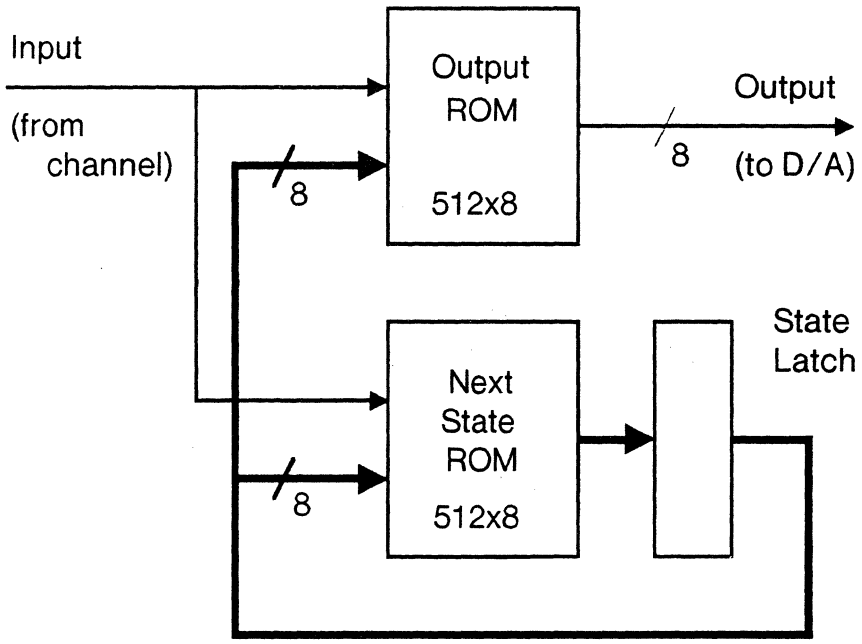


Figure 1.8 *General finite-state machine decoder*

mented by a shift register driving a codebook memory, section 3.4 will return to the question of alternative decoder structures.

1.5 Thesis Outline

In Chapter 1, we have described the general idea of source coding, or data compression, and its specialization in the form of tree and trellis codes. Chapter 2 presents the central algorithms and ideas of the thesis, including an iterative fixed point algorithm for improving a given trellis decoder and an algorithm for extending the constraint length of a given shift register trellis decoder. In chapter 3, these algorithms are tested against the standard Gaussian and Gauss-Markov sources and the results compared against both traditional trellis coding schemes and against the theoretical rate-distortion performance bounds. Additional material is introduced to explore alternative decoder structures and ways of providing the "given" initial decoder required by the design algorithm. Chapter 4 turns to the problem of speech coding. The design algorithms developed in the previous chapters are used to construct trellis codes for the original speech waveform, for residual excited linear predictive coding systems, and for a new kind of hybrid tree code which incorporates an internal trellis code. Appendices present some additional information on notation, on the Viterbi and M,L encoding algorithms, and on the production of random numbers for computer aided code design.

2. CODE DESIGN ALGORITHMS

This chapter describes new algorithms for the design of trellis source codes. The first section introduces the subject by describing an associated algorithm, originally due to Lloyd in 1957, for the design of block source codes. The application of this original idea to trellis codes is due to a suggestion by Linde and Gray in 1979. Section 2.2 describes the trellis code design algorithm which, given an initial code and a training sequence, returns an improved code. Finally, section 2.3 adds to the facilities for a complete design system for trellis codes by describing an algorithm for extending the constraint length of a given trellis code. This chapter touches only briefly on the initial conditions required for the use of these algorithms. These operational considerations, as well as questions of performance, are deferred until Chapter 3. For reference purposes, Appendix A contains a compilation of all the notation used in this chapter.

2.1 Block Code Design Algorithm

As a first step towards the description of the trellis code design method, we first establish some terminology and basic ideas by describing an algorithm for the design of block codes, or vector quantizers, based on a long training sequence of symbols from a source. This algorithm is a multi-dimensional version of a quantizer design method of Lloyd [56] and is more completely reported by Linde et al. [53].

An N-level k-dimensional quantizer is a function, f, that assigns to each input vector, $\vec{x} = (x_0, \ldots, x_{k-1})$, a reproduction vector, $\vec{x}'$, drawn from a finite reproduction alphabet, $A = \{\vec{y}_i, i = 0, \ldots, N-1\}$. The quantizer is completely described by the reproduction alphabet together with a partition, $S = \{S_i, i = 0, \ldots, N-1\}$, of the input vector space into the disjoint sets $S_i = \{\vec{x}_j \mid f(\vec{x}_j) = \vec{y}_i\}$ of input vectors mapping into the i^{th} reproduction vector. From this description and from the discussion of block codes in Chapter 1, we see that a vector quantizer is just a block source code. The encoding operation corresponds to the translation of the source vector $\vec{x}$ into the index of the reproduction vector - $encode(\vec{x}) = i$ - while decoding corresponds to looking up and producing the reproduction vector associated with that index - $decode(i) = \vec{y}_i$.

Usually the fidelity of reproduction, and hence the desirability of selecting a particular reproduction vector, is governed by a non-negative distortion measure $d(\vec{x}, \vec{x}')$. Many distortion measures have been proposed for various applications, but

perhaps the best known is the squared-error measure

$$d(\vec{x}, \vec{x}') = \sum_{i=0}^{k-1} (x_i - x_i')^2.$$

Given a distortion measure, a reasonable approach to the selection of a partition is to map each source vector into the reproduction vector giving minimum distortion. Thus

$$S_i = \{ \vec{x} \mid d(\vec{x}, \vec{y}_i) \leq d(\vec{x}, \vec{y}_j), \text{for all } j \}$$

with some tie breaking rule. As ties are generally low probability events, nearly any rule will do, such as assigning the sample to the partition cell with the lower index.

The initial conditions for the design algorithm are N, the desired number of reproduction vectors, A^0, an initial quantizer, and $\{\vec{x}_j : j = 0, \ldots, n-1\}$, a long training sequence of symbols from the source. The algorithm consists of the iterative execution of two steps: finding the best encoding of the training sequence for a given set of reproduction vectors, and finding the best set of reproduction vectors for the given encoding. In order to avoid some set-theoretic difficulties, in what follows we change the definition of a partition slightly so that S_i contains the *time indices* of those elements of the training sequence which are encoded by reproduction vector y_i, rather than the source vectors themselves. S is now a partition of $\{j : j = 0, \ldots, n-1\}$ and the encoding function f returns the index of the minimum distortion reproduction vector $f(\vec{x}) = i$.

Find Partition: Given A^m, the reproduction alphabet of generation m, find the minimum distortion partition $S^m = \mathcal{P}(A^m)$, by assigning each element of the training sequence to the minimum distortion reproduction vector:

$$f(\vec{x}) = i : d(\vec{x}, \vec{y}_i) \leq d(\vec{x}, \vec{y}_j), \text{for all } j$$

with some tie breaking rule.

Find Reproduction Alphabet: Given a partition $S^m = \mathcal{P}(A^m)$, find the minimum distortion reproduction alphabet for generation $m+1$

$$A^{m+1} = \mathcal{A}(S^m) = \mathcal{A}(\mathcal{P}(A^m)),$$

by setting $\vec{y}_i^{m+1}$, the i^{th} reproduction vector of the new alphabet, to the value giving the minimum average distortion over the training sequence vectors indexed by elements of S_i^m. This value will be the generalized centroid, or center of gravity

under the distortion measure, of those training sequence values which were quantized to the value $\vec{y}_i^m$.

In the case of the mean-square-error distortion measure, this center of gravity calculation is just the sample average over a partition:

$$\vec{y}_i^{m+1} = \frac{1}{\|S_i^m\|} \sum_{j \in S_i^m} \vec{x}_j.$$

In Euclidian space, the average of a set of vectors is just the term by term average.

Repeated application of these steps must result in decreasing sample average distortion over the training sequence. Since the average distortion is non-negative and decreasing, it must eventually reach a limit [50]. Although the limit may not be the global optimum, it is at least a local optimum. The algorithm may be stopped either when a fixed point is reached (when no changes occur in the reproduction vectors or partitions), or when the reduction in average distortion per iteration falls below some threshold.

If the vector source $\{\vec{x}_j\}$ is ergodic and stationary, the sample average distortion over the training sequence will be a valid estimate of the expected distortion over data from outside the training sequence. In this case, a quantizer designed by this algorithm will also work well on data from outside the training sequence. However, convergence of the design algorithm itself does not depend on either stationarity or ergodicity.

2.2 Trellis Code Design Algorithm

As will become clear, the trellis code design algorithm depends only on a table-lookup implementation of the trellis decoder. In spite of this fact, we present here a detailed description of a shift register decoder. Nearly all of the algorithm applications discussed later use the shift register and its description here may provide a useful focal point for the design algorithm description below.

A shift register trellis code of constraint-length (register length) k for a q-ary channel has $N = q^k$ reproduction symbols (codewords) and $R = q^{(k-1)}$ states. We number the channel symbols from 0 to $q-1$, the codewords from 0 to $N-1$, and the states from 0 to $R-1$. We adopt the conventions that channel symbols are shifted into the least significant end of the shift register and that the contents of the register are interpreted as the index of the output codeword. In this model, the codewords are associated with transitions between decoder states rather than with

the states themselves – they are branch labels of the code trellis. Suppose symbol u arrives at the decoder input from the communications channel when the decoder is in state r. The output of the decoder will be codeword $qr + u$ and the decoder will move to state $qr + u \pmod{q^{k-1}}$. The modulus operation strips off the 'oldest' symbol in the register. With these procedures for operating the decoder, a shift trellis code is completely described by q, the cardinality of the channel, and by the contents of the codebook.

The trellis code design algorithm seeks to fill in the contents of the codebook.

2.2.1 The basic algorithm

Suppose that we have a trellis decoder, a single symbol additive distortion measure, and an optimal encoding (search) algorithm such as the Viterbi Algorithm which will find the trellis encoding of a source sequence which gives the lowest sample average distortion. The trellis decoder itself will consist of a k symbol shift register whose contents are interpreted as an index into a table of reproduction symbols (the codebook). This is just the organization described above and shown in Figure 1.6. Each time a symbol arrives from the channel and is shifted into the register, the new register contents are used to select a codeword from the table. The codeword selected becomes the output of the decoder. The symbols of the channel sequence, or path map, chosen by the encoding algorithm are thus associated first with the sequence of source symbols, then with a sequence of decoder states, and finally with the reproduction sequence of codewords. It is this final association that is of the most interest; it can be used as the partition function in a trellis adaptation of the block quantizer design algorithm.

The initial conditions for the design algorithm are a table driven finite-state decoder with initial codebook C^0 (such as the shift register decoder described above), a single symbol additive distortion measure (such as squared-error), and $\{x_j\}$, a long training sequence of symbols from some source. The algorithm consists of two key steps: finding the best encoding of the training sequence for a given codebook, and finding the best codebook for the given encoding. Executed alternately, these steps provide an iterative design algorithm for improving the initial trellis code.

Find Encoding: Given C^m, the trellis codebook for generation m, find the minimum average distortion encoding of the source sequence $\{x_j\}$ by using an optimal encoding algorithm. This encoding induces the partition $S^m = P(C^m)$. The elements of cell S_i^m, the partition cell corresponding to codeword y_i^m, are the

time indices of those elements of the training sequence which were reproduced by codeword y_i^m.

Find Codebook: Given the partition for generation m, $S^m = P(C^m)$, find the minimum distortion codebook for generation $m+1$:

$$C^{m+1} = C(S^m) = C(P(C^m)),$$

where y_i^{m+1}, the i^{th} reproduction symbol for generation $m+1$, is the value giving the minimum average distortion over those elements of the training sequence indexed by S_i^m. This value will be the center of gravity, or centroid, under the distortion measure of those training sequence values which were encoded by codeword y_i^m. The encoding function – the trellis search – does not necessarily map a source symbol into the minimum distortion codeword, but maps the entire training sequence into the minimum average distortion *sequence* of codewords.

Each iteration of this procedure can only result in decreasing, or at least non-increasing, average distortion. Since the encoder is free to choose the same path sequence on successive iterations, a bound is placed on the distortion resulting from encoding with the updated codebook. Because the encoding algorithm is optimal, the new encoding can result in distortion no worse than the distortion due to using the old path with the new codewords – which, in turn, is at least as good as the distortion for the previous iteration. (In fact, if the same path is chosen twice, the algorithm has reached a fixed point and will proceed no further. The reader is referred to [57] for a general discussion of fixed point optimization algorithms.)

In the block quantizer algorithm, the partition function we chose was the minimum-distortion function, which mapped each training sequence sample into the reproduction vector giving the lowest distortion for that source vector. This function was applied independently to each block (vector) from the source so that source vectors with the same value would always be mapped to the same partition cell. In the trellis case, the partition function does not necessarily map each training sequence sample to the lowest distortion codeword, rather it maps the entire training sequence to the reproduction sequence giving the lowest *average* distortion subject to the constraints of the trellis and codebook. This function is no less well defined, but at different times it may map the same *value* source symbol into different partition cells.

2.2.2 Meeting the initial conditions

The initial requirements of the trellis code algorithm are a suitable distortion measure, a training sequence, and an initial decoder. Selecting an appropriate distortion

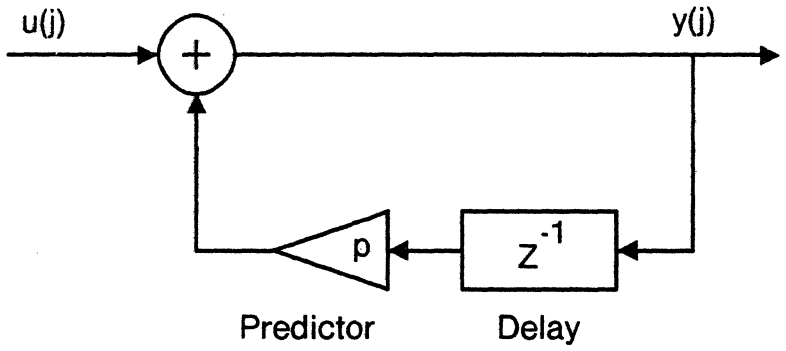


Figure 2.1 *Predictive quantization decoder*

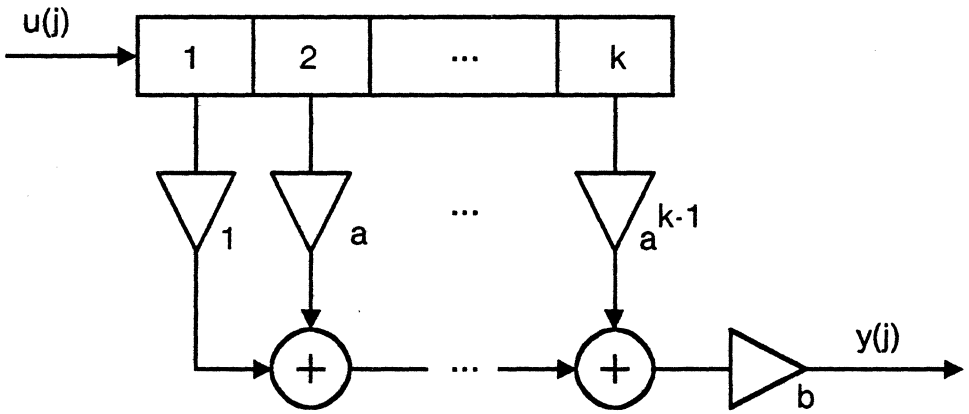


Figure 2.2 *Truncated predictive quantization decoder*

measure depends on the ultimate application of the data compression system and is beyond the scope of the design algorithm. The required characteristics of the training sequence are of some interest and will be addressed in section 3.1. The most important initial requirement is for a table-lookup decoder with some initial codebook C^0. We mention here several ways in which initial decoders may be obtained. The selection of initial decoders will be more thoroughly explored in section 2.3 and in Chapter 3.

Plagiarized decoders: One of the most appealing ways to choose an initial decoder is simply to use the decoder from a good existing data compression system such as predictive quantization, but to implement it by a table-lookup decoder and to use it in a trellis coding context. Since the decoder for the usual predictive quantizer (Figure 2.1) involves a recursive filter and thus has a tree, rather than a trellis, structure we must first transform the recursive filter into a transversal equivalent. Of the several methods for accomplishing this transformation, the

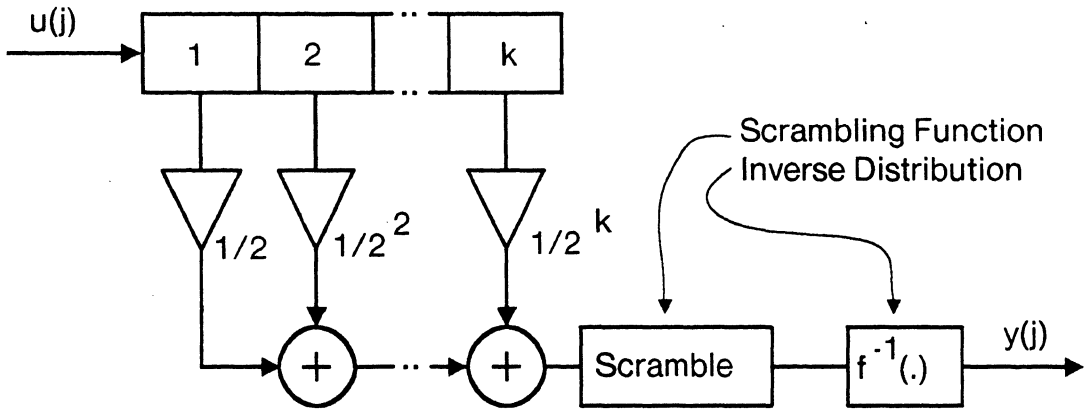


Figure 2.3 *Inverse distribution fake process decoder. The shift register contents are interpreted as a fraction in [0,1). This value drives a* $[0,1) \mapsto [0,1)$ *scrambling function, which in turn drives the inverse distribution function* f^{-1}.

simplest is to truncate the impulse response, which gives the *truncated predictive quantization decoder* of Figure 2.2. Once a finite-state decoder is obtained, it is a simple matter to implement it by a table of codewords indexed by the contents of the shift register. Other traditional data compression systems have similar kinds of transformations into the trellis code framework. The use of plagiarized decoders to generate initial codebooks for the design algorithm is discussed in section 3.2.2.

Fake process decoders: Linde and Gray [52],[54],[55] have proposed that when a trellis decoder is driven by a random channel sequence (e.g. coin tosses), the statistics of the decoder output should be close, in a certain sense, to the statistics of the source under consideration. They propose the generalized Ornstein distance [42] for this purpose. In the case of known source statistics, this approach leads to (among others) the so-called Inverse-Distribution decoder of Figure 2.3. Once again, it is straightforward to implement the scrambling function and the function f^{-1} by a table of codewords. The hybrid tree codes discussed in Section 4.5 are essentially fake process decoders.

Random decoders: The theoretical approaches to tree and trellis source coding systems have used random coding arguments, especially in the case of memoryless sources [38],[45]. While these arguments, that a randomly selected code will tend to be a good one, apply, at least in theory, only to long constraint lengths, it is not unreasonable to suspect that an initial codebook whose entries are randomly selected according to an appropriate distribution might work fairly well. Initial codes of this type are used several times in Chapter 3.

2.2.3 Termination conditions

One difficulty with iterative algorithms is deciding when to stop – or when to give up. As part of the encoding portion of the design algorithm, a sample average distortion may be obtained for each iteration m

$$\Delta^m = n^{-1} \sum_{j=0}^{n-1} d(x_j, \hat{x}_j),$$

where $\hat{x}_j$ is the reproduction of x_j. This is simply the average per-symbol distortion over the training sequence. We know that this distortion is non-negative and non-increasing with each iteration, so it is a natural value to use in a termination test. One approach is to stop when the distortion reaches a predetermined threshold, but there is a danger that the algorithm may reach a fixed point before attaining a particular distortion. A better approach is to set a threshold on the *change* in the sample average distortion, that is, terminate the algorithm when the change in distortion falls below a certain threshold

$$\left(\frac{\Delta^m - \Delta^{m+1}}{\Delta^m} \right) \leq \epsilon.$$

The selected value of ϵ may have any non-negative value, including 0, since the algorithm will reach a fixed point within a finite number of iterations. (A proof appears later in this section.)

2.2.4 Detailed trellis code design algorithm

(0) Initialization: Given a distortion threshold $\epsilon \geq 0$, a q-ary noiseless channel, an R state decoder, an initial codebook C^0 with cardinality $||C^0|| = N = qR$, and a training sequence $\{x_j\} = \{x_j : j = 0, \ldots, n-1\}$, set $m = 0$.

(1) Given $C^m = \{y_i^m : i = 0, \ldots, N-1\}$, the codebook for generation m, find the minimum distortion trellis encoding $\{\hat{x}_j : j = 0, \ldots, n-1\}$ of the training sequence $\{x_j\}$. This encoding induces a partition on the training sequence $\mathcal{P}(C^m, \{x_j\}) = \{S_i^m : i = 0, \ldots, N-1\}$ with $S_i^m = \{j : \hat{x}_j = y_i^m\}$. Each set S_i^m contains the time indices of those elements of the training sequence which were encoded by codeword y_i^m.

(2) Compute the average distortion

$$\Delta^m = n^{-1} \sum_{j=0}^{n-1} d(x_j, \hat{x}_j) = \sum_{i=0}^{N-1} \frac{1}{||S_i^m||} \sum_{j \in S_i^m} d(x_j, y_i^m).$$

The last expression causes difficulties if there are any empty partition cells. If $||S_i^m|| = 0$, then that partition cell simply does not contribute to the distortion.

(3) If at least one full iteration has been completed ($m > 0$) and the decrease in distortion has fallen below the threshold ϵ,

$$\left(\frac{\Delta^m - \Delta^{m-1}}{\Delta^{m-1}}\right) \leq \epsilon,$$

then halt with C^m as the final codebook. Otherwise continue with step (4).

(4) Find the optimal codebook C^{m+1} for generation $m+1$ as

$$C^{m+1} = \mathcal{C}(\mathcal{P}(C^m, \{x_j\}))$$

with

$$C^{m+1} = \{y_i^{m+1} : i = 0, \ldots, N-1\}$$

and

$$y_i^{m+1} = y : \sum_{j \in S_i^m} d(x_j, y) \leq \sum_{j \in S_i^m} d(x_j, y'), \text{for all } y'.$$

Each new codeword has the value which minimizes the average distortion over its partition cell. Each codeword in generation $m+1$ will be the centroid of those elements of the training sequence which were encoded by the corresponding codeword of generation m. If a particular partition cell is empty, the last equation is indeterminate and some rule must be adopted to cover this eventuality. Section 2.2.4.1 contains some approaches to handling empty partition cells.

(5) Replace m by $m+1$ and go to step (1).

2.2.4.1 Codebook update for mean-square-error

If the distortion measure is squared-error, $d(x, x') = (x - x')^2$, then the step (3) centroid computation for the updated codewords is particularly simple:

$$y_i^{m+1} = \frac{1}{||S_i^m||} \sum_{j \in S_i^m} x_j.$$

This expression does run into difficulties when a partition cell is empty. One way to handle this situation is simply to retain the old codeword, $y_i^{m+1} = y_i^m$, but if it is a poor value, it may *never* be used and the associated partition cell may remain empty on the next iteration. Another approach is $y_i^{m+1} = \sum_{j=0}^{n-1} x_j$ - set the codeword to the centroid of the entire training sequence. The goal of the method selected must be to select a value that will be *used* by the next pass of the encoder. Once the partition cell is no longer empty, the design algorithm will take care of adjusting the exact value of the associated codeword.

2.2.4.2 Distortion computation

As noted in step (2), there are alternative formulations of the sample average distortion. One,

$$\Delta^m = n^{-1} \sum_{j=0}^{n-1} d(x_j, \hat{x}_j)$$

may be computed "on the fly" as each encoded symbol is released to the channel by the encoding algorithm. The second form

$$\Delta^m = \sum_{i=0}^{N-1} \frac{1}{\|S_i^m\|} \sum_{j \in S_i^m} d(x_j, y_i^m)$$

operates in a batch fashion by separately considering each partition of the training sequence and computing that portion of the total distortion resulting from the use of codeword y_i^m. A variation of this formula:

$$\Delta^{m\prime} = \sum_{i=0}^{N-1} \frac{1}{\|S_i^m\|} \sum_{j \in S_i^m} d(x_j, y_i^{m+1}),$$

in which y_i^m has been replaced by y_i^{m+1}, may also be of interest. This last value is the distortion arising from use of the path sequence from generation m with the codebook from generation $m+1$ – the old encoding with the new codebook. This figure is an upper bound for Δ^{m+1}:

$$\Delta^m \geq \Delta^{m\prime} \geq \Delta^{m+1}.$$

These alternative computations each have a place. The first form occurs naturally as part of encoding algorithms such as the M,L Algorithm, which maintain several possible paths and select the one with minimum average distortion. The second form occurs naturally as part of the computation of the new codebook C^{m+1} in step (4), and the third form provides additional information about the progress of the overall design algorithm. The distortion arising from use of the old path with the new codebook should be below that for the old codebook, but not as low as that obtained for a new encoding with the new codebook.

2.2.4.3 Proof of algorithm termination

Let us consider the consequences of setting $\epsilon = 0$ in the termination test. Can the design algorithm iterate forever? Such an event would require an infinite sequence of

codebooks whose associated optimal encodings give constantly decreasing distortion. Since we assume a finite training sequence (of length n), a finite codebook, $||C|| = N$, and a finite q-ary channel, there can only Nq^n ways to encode the training sequence. (The number may actually be smaller than this if we do not permit an arbitrary encoding of the first source symbol.) If, as is usually the case, the centroid computation proceeds in a fashion independent of the time ordering of the samples, then there can only be Nq^n possible codebooks. An infinite sequence cannot happen. This argument remains valid in the presence of empty partition cells provided a deterministic rule is applied in such cases.

2.3 Extension

Earlier, we described several methods for generating an initial decoder for the trellis code design algorithm. In this section, we describe a method, given a shift register decoder of constraint-length k, of constructing a decoder based on a shift register of length $k+1$. Our method will have the advantage of constructing a decoder with sample average distortion over the training sequence at least as low as that of the starting (shorter) decoder. We call this method *extension*, because the general idea is to create the new decoder, with a longer constraint length, by extending the shift register of the starting decoder by adding an additional stage.

2.3.1 Notation

It will be helpful if we adopt some additional notation for the special case of shift register decoders. The most important fact about shift register decoders is that at any time j, the contents of the decoder shift register form a base q representation of the index in the decoder codebook of the output codeword. In particular, if $\hat{x}_j = y_i$ then

$$i = \sum_{p=0}^{k-1} u_{j-p} q^p.$$

Let α represent a string of base-q digits of length $|\alpha|$. Concatenation will be represented by juxtaposition – 0α is a string with a leading 0 and length $|\alpha|+1$. If the length of a string is important, it may be specified by superscript: $|\alpha^k| = k$. Let $(\alpha)_q$ be the digit string α interpreted as an integer in base q. Usually, the radix will be clear from context and we will write simply (α). Finally, if $\ldots v_{-1}v_0v_1\ldots$ is a sequence of digits, let v_j^k be the digit string composed of the k digits beginning with $v_{j-(k-1)}$ and ending with v_j. In this final case, since the q-ary channel sequence $\{u_j\}$ may be thought of as being composed of base-q digits, we have $u_j^k =$

$u_{j-(k-1)} \ldots u_{j-1} u_j$ as the decoder shift register contents at time j and $(u_j^k)_q$ as the codebook index of the decoder output at time j.

Since we will be considering codebooks for shift register decoders of different constraint lengths, it will be convenient both to index codewords by base-q numbers (strings) representing the contents of the decoder shift register and to index codebooks by their constraint lengths:

$$C_k = \{y_\alpha : |\alpha| = k\}.$$

2.3.2 Initial decoder

Given a q-ary channel, and a shift register decoder (refer to Figures 1.5 and 1.6) of length k and codebook C_k, we have $\|C\| = q^k$. We begin with codebook C_k, containing codewords $\{y_\alpha : |\alpha| = k\}$ and we wish to select codewords $\{y'_\beta : |\beta| = k+1\}$ for the extended codebook C_{k+1}.

2.3.3 Extension algorithm

Set

$$y'_{0\alpha} = y'_{1\alpha} = \cdots = y'_{(q-1)\alpha} = y_\alpha : \quad |\alpha| = k.$$

This method produces the codebook C_{k+1} by duplicating the contents of codebook C_k q times. The new decoder has an additional shift register stage added at the left end of the original register, represented by the leading digit of the codeword index β. Because of the way that the codebook is duplicated, the symbol held in the new register stage is a "don't care" symbol and has no effect on the decoder output. The path sequence chosen by the encoder for the codebook C_k is still available for the new codebook C_{k+1}, and, if used, would produce exactly the same distortion with the new codebook as with the old. The encoding of a source sequence with the old codebook therefore represents an upper bound on the distortion generated by an optimal encoding of the same sequence with the new codebook.

In fact, the situation is not that optimistic. Since the old and new decoders produce identical reproduction sequences when driven by identical path sequences, the old and new codebooks are in effect identical and their distortions will be exactly the same – the distortion of the old codebook is not only an upper bound to the distortion of the new, it is a lower bound as well. The benefit of the extension method lies not in encoding with the new codebook C_{k+1} itself, but in using C_{k+1} as an initial codebook for the trellis code improvement algorithm. To explore this idea,

let us first examine the encoding and update steps (1) and (4) of the improvement algorithm in the context of a shift register decoder.

(1) Given $C_k^m = \{y_\alpha^m : |\alpha| = k\}$, the constraint-length k codebook for generation m, find the minimum distortion trellis encoding $\{\hat{x}_j : j = 0, \ldots, n-1\}$ of the training sequence.

The encoding of the training sequence was described earlier by the sequence $\{\hat{x}_j\}$ of decoder outputs, but it can also be described by the sequence $\{u_j\}$ of channel symbols – the path sequence. Ignoring end effects, $\{\hat{x}_j\}$ can certainly be generated from $\{u_j\}$ – this is just the job of the decoder.

Given the codebook, this encoding induces a partition on the training sequence $\mathcal{P}(C_k^m, \{x_j\})$ consisting of the partition cells (sets) $\{S_\alpha^m : |\alpha| = k\}$. We have $S_\alpha^m = \{j : u_j^k = \alpha\}$. The index-set S_α^m contains the time indices of those moments when the contents of the decoder shift register was α, or alternatively, the time indices of those elements of the training sequence which were encoded by codeword y_α^m.

Suppose that the optimal encoding of the training sequence using the codebook C_k has resulted in the path sequence $\{u_j\}$. If we now extend C_k, producing C_{k+1}^0 as a new initial decoder for the improvement algorithm, the optimal encoding in step (1) of the algorithm will once again be $\{u_j\}$.

Now examine the relationship of the cells of the old partition $\mathcal{P}(C_k^m, \{x_j\})$ generated by encoding $\{x_j\}$ with decoder C_k^m to the cells of the new partition $\mathcal{P}(C_{k+1}^0, \{x_j\})$ generated by encoding $\{x_j\}$ with decoder C_{k+1}^0. The path sequence $\{u_j\}$ will be the same in each case, so we have

$$\bigcup_{v=0}^{q-1} S_{v\alpha}^0 = S_\alpha^m.$$

The partition cells of $\mathcal{P}(C_{k+1}^0, \{x_j\})$ are non-overlapping subsets of the partition cells of $\mathcal{P}(C_k^m, \{x_j\})$. $\mathcal{P}(C_{k+1}^0, \{x_j\})$ is a *refinement* of the original partition.

Now let us turn to the update step of the trellis code algorithm, specialized to the first iteration after the code has been extended.

(4) Find the extended codebook C_{k+1}^1 for generation 1 as

$$C_{k+1}^1 = \mathcal{C}(\mathcal{P}(C_{k+1}^0, \{x_j\}))$$

with

$$y_\beta^1 = y : \sum_{j \in S_\beta^0} d(x_j, y) \leq \sum_{j \in S_\beta^0} d(x_j, y'), \text{for all } y'.$$

Each new codeword y_β^1 is the centroid of those elements x_j of the training sequence such that $u_j^{k+1} = \beta = v\alpha$. The elements of $S_{v\alpha}^1$ are just the values j such that $j \in S_\alpha^m$ and $u_{j-k} = v$.

2.3.4 Combined extension and update

As we have just seen, the initial encoding of the training sequence with the extended codebook C_{k+1}^0 is the same as the final encoding with codebook C_k^m. We can now take advantage of this fact by modifying the extension method to omit the initial encoding with the extended codebook:

$$C_{k+1}^0 = \{y_\beta^0 : |\beta| = k+1, \sum_{u_j^{k+1}=\beta} d(x_j, y_\beta^0) \leq \sum_{u_j^{k+1}=\beta} d(x_j, y'), \text{for all } y'\}.$$

2.3.5 Code design using extension

The theory of trellis codes includes a proof that good codes exist – at least in the limit of long constraint length. Unfortunately the theory does not say how to build such codes. The extension idea offers a method of constructing long trellis codes which perform at least as well as their shorter ancestors. The shorter codes referred to might be generated in a number of ways, but the extension idea offers a method of constructing a long constraint-length trellis code entirely from scratch:

(1) Design an initial codebook C_1^0 by finding a good q-level quantizer for the training sequence. This may be accomplished by use of the block code design algorithm, as this initial codebook is just a block-length 1 block code with q codewords. Parenthetically, we note that for constraint-length 1, the block and trellis design algorithms are identical.

(2) Given C_k^0, an initial code of a certain constraint length, use the trellis code design algorithm to improve it. The extent of this procedure will be governed by the value of ϵ in the trellis code algorithm.

(3) If the constraint-length k is sufficiently large, or the distortion provided by C_k^m sufficiently low, then halt.

(4) Use the extension algorithm to produce the codebook C_{k+1}^0 from C_k^m and then return to step (2).

2.3.6 Example

The above procedure can be easily illustrated by considering its application to a hypothetical one bit per symbol ($q = 2$) system using squared error distortion.

Let the initial codebook for the constraint-length 1 decoder include codewords -1 and $+1$. The trellis code design algorithm for this constraint length reduces to the block code design algorithm and simply selects a locally optimal two level scalar quantizer for the training sequence. Assuming for the moment that the mean of the training data is 0, the optimum partition for the final constraint-length 1 decoder collects all the samples in the training sequence with positive values in one partition cell and all the samples with negative values in the other. In keeping with earlier notation we have

$$S_0 = \{j \mid x_j \geq 0\}$$

and

$$S_1 = \{j \mid x_j < 0\},$$

where we have chosen to break ties by assigning the associated sample to the first partition. The two codeword values will be the means, respectively, of the positive and negative training sequence samples. The first extension of this decoder – to constraint-length 2 – contains four codewords. The original partition is refined in a very intuitive way: the partition cell containing the positive valued samples of the training data is divided into two cells containing respectively the positive samples which were preceded in the training data by other positive samples, and those which were not.

$$S_{00} = \{j \mid x_j \geq 0, x_{j-1} \geq 0\}$$

and

$$S_{10} = \{j \mid x_j \geq 0, x_{j-1} < 0\}$$

are the two subcells created from S_0 where we have used binary notation for the partition cell indices.

As the decoder is further extended, it is no longer quite so easy to describe the contents of a particular cell, but in each case the extended codebook contains pairs of partition cells, each pair of which form a two cell refinement of one of the partition cells of the shorter code.

3. CODING RANDOM SOURCES

Source coding of truly random sources is perhaps less interesting for its practical applications than for the insights and understanding that may be gained. Well developed theoretical bounds on performance of codes for random sources and the existence of "competition" in the form of traditional coding systems give the problem of the data compression of random sources great value during the development of new coding systems. In the sections below, random sources are used to obtain an initial evaluation of the code design algorithms of the previous chapter, to study the problem of seeding the algorithm with an initial decoder, to compare the performance of the new codes with more traditional systems, and to investigate the effects of alternative trellis decoder structures.

3.1 Operational Considerations

The basic trellis code design algorithm guarantees only two things: it will eventually halt, and the average distortion obtained over the training data will be non-increasing with each successive iteration. These guarantees are of considerable theoretical interest, but the algorithm would be of little practical value if it either usually failed to produce significantly improved codes or converged very slowly. (A reasonable interpretation of slow convergence would be that the algorithm is simply unable to obtain much improvement.) In practice, it turns out that convergence is almost always extremely rapid – a few iterations usually suffice – and significant code performance improvements are usually obtained. (This sort of behavior is quite typical of fixed point algorithms [53].)

The three sections below address several practical concerns: code performance improvement, encoding algorithm and training sequence requirements, applicability of codes designed using a particular training sequence to other sequences, and algorithm rate of convergence.

3.1.1 Effects of the encoding algorithm

In discussing the history of tree and trellis coding, we mentioned that a wide variety of encoding algorithms – tree and trellis search algorithms – are known. Chapter 2, in discussing the details of the trellis code design algorithm, stated that an *optimal* encoding algorithm must be used in order to ensure that the training algorithm will converge. While such an algorithm exists – the Viterbi Algorithm – both the space and time costs of using it grow linearly with decoder codebook size or exponentially with the code constraint length. Since, for some sources, trellis

Table 3.1
Viterbi Algorithm design

iteration	1	2	3	4	5	6
encoding mse	0.317	0.308	0.303	0.299	0.296	0.294
new codebook mse	0.311	0.305	0.301	0.297	0.294	0.293

codes of long constraint length may be required to achieve performance close to the theoretical limits, it is important to discover whether, in practice, the use of an optimal encoding algorithm is necessary. Use of a faster but suboptimal encoding algorithm would offer substantial savings both in the code design phase and in later operation of the code.

The experiments described below employ both the optimal Viterbi Algorithm and the suboptimal M,L Algorithm in the trellis encoding phase of the code improvement algorithm. The operation of these search algorithms is described in Appendix B. Briefly, the Viterbi Algorithm takes advantage of the trellis structure of a finite-state decoder to produce an optimal encoding. It is essentially a form of dynamic programming. The M,L algorithm is a breadth first tree search, suitable for use as an encoder either for tree codes or for trellis codes. The M,L name is used because the algorithm maintains a fixed encoding delay of L symbols and simultaneously considers M potential encoding paths.

The Viterbi Algorithm and M,L Algorithms were each used to design constraint-length 6, one bit per symbol trellis codes for the memoryless Gaussian source with squared error distortion. The initial codebook for each experiment was populated with values taken from the $\mathcal{N}(0, 0.75)$ source – Gaussian, with 0 mean and 0.75 variance [13, eq. 4.3.35]. (Methods for the selection of random initial decoders will be discussed in section 3.2.1.) The training sequence consisted of 12,000 $\mathcal{N}(0, 1)$ samples.

Table 3.1 presents the per symbol squared error as a function of the training algorithm iteration when using the Viterbi optimal encoding algorithm. In addition, the per symbol squared errors resulting from using the "old" encodings with the updated codebooks are shown ($\Delta^{m\prime}$ from section 2.3.4.2). As expected, use of the new codebook with the old encoding (old path sequence) provides improved performance over the old codebook, but on the next iteration, a new encoding is able to do still better. On a different sequence of 12,000 $\mathcal{N}(0, 1)$ samples, outside the training sequence, the codebook from the sixth iteration achieved 0.305 average squared error.

Table 3.2 presents the per symbol squared error as a function of the training

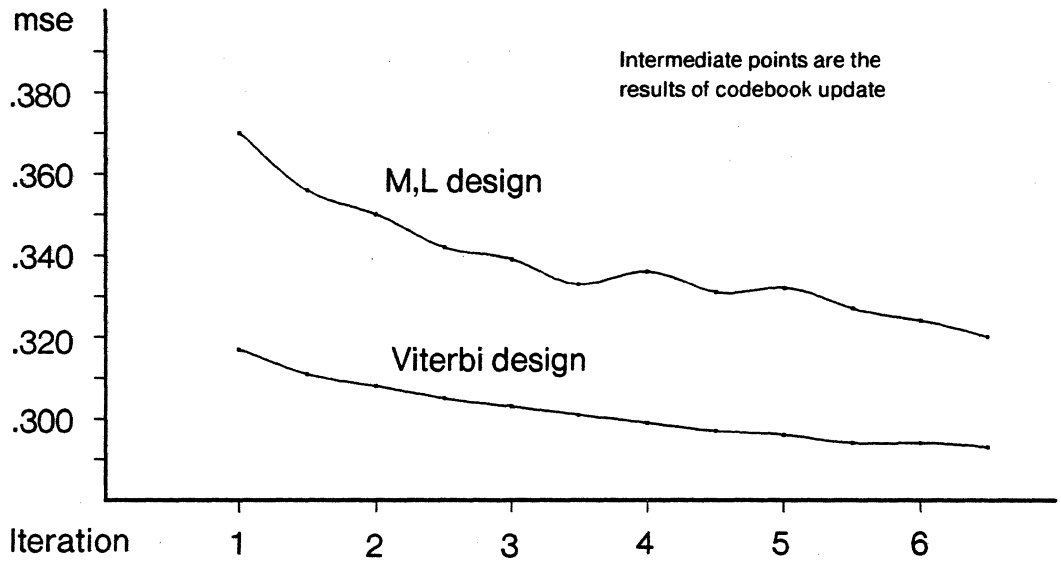


Figure 3.1 *M,L Algorithm vs Viterbi Algorithm code design*

Table 3.2
M,L Algorithm design with $M = 8$ and $L = 15$

iteration	1	2	3	4	5	6
encoding mse	0.370	0.350	0.339	0.336	0.332	0.324
new codebook mse	0.356	0.342	0.333	0.331	0.327	0.320

Table 3.3
M,L Algorithm encoding outside training sequence

L	15	15	15	15	31	31	31	31
M	4	8	12	16	4	8	12	16
M,L codebook mse	0.368	0.337	0.331	0.328	0.366	0.329	0.319	0.314
VA codebook mse	0.402	0.353	0.340	0.336	0.400	0.342	0.328	0.319

algorithm iteration when using the M,L Algorithm. As before, the average squared error resulting from using the old path sequences with the updated codebooks are also shown. For the design portion of the experiment, the encoding algorithm was set up with $M = 8$ and $L = 15$. One of the most interesting results of this experiment is that the average squared error of the new encoding of the fourth iteration actually increased from that of the updated codebook of the third iteration: $\Delta^4 > \Delta^{3\prime}$. The suboptimal encoding algorithm was unable to find the best path through the trellis, but nevertheless, the code performance continues to improve.

Figure 3.1 combines the data contained in the two tables. Comparing the results of these two code design experiments, the key observations are that both

encoding algorithms obtained most of their improvement in three to four iterations, but that the Viterbi Algorithm achieved better performance. In compensation, the M,L Algorithm runs much faster than the Viterbi Algorithm.

Table 3.3 outlines the performance of the M,L Algorithm on data outside the training sequence for various values of M and L. As expected, greater search effort (M) and greater search depth (L) contribute to improved performance. In addition, Table 3.3 presents performance results for the M,L Algorithm used for encoding with the decoder designed with the Viterbi Algorithm; the results are uniformly inferior. The code design algorithm has adapted the decoder codebook not only for the characteristics of the source, but for the characteristics of the encoding algorithm as well. One interpretation of these results is that the same encoding algorithm must be used during the design of a trellis decoder as will later be used in actual operation.

3.1.2 Training sequence length requirements

In this section, we investigate the connection between code performance on data inside the training sequence and code performance on data outside of the training sequence. The practicability of the code design methods hinges on the assumption that a code which works well on the sequence used for its design (the training sequence) will also work well on other sequences from the same source. This should be true if the source is *ergodic*, – if the time average statistics of a particular sequence are similar to the ensemble average statistics of the source. The training sequence must be *typical* of sequences from the source.

In addition to the typicality requirement, we should expect a connection between the length of the training sequence and the relative performance of the code on other sequences. If the training sequence is short we should expect that code performance inside the training sequence will be considerably better than performance outside. The design algorithm will fit the code to the short-term random characteristics of the training sequence itself rather than to the statistical characteristics of the source. As the length of the training sequence grows, code performance inside the training sequence should drop, while performance outside rises. In each case, we expect the performance to converge to some value characteristic of the source, rather than characteristic of a particular sequence from the source. Quite precise results along these lines are known for the block code design algorithm [40] but not, as yet, for the trellis case.

Table 3.4 presents some results of code designs for a one bit per symbol constraint-length 6 shift register decoder. Figure 3.2 displays the same data in graphical

Table 3.4
Effect of training sequence length
Table entries are mean squared error

Initial codebook distortion 0.353

Length	Inside	Outside
1,000	0.280	0.330
2,000	0.296	0.325
3,000	0.299	0.319
4,000	0.305	0.320
5,000	0.307	0.316
7,000	0.310	0.318
10,000	0.311	0.318
15,000	0.311	0.315
20,000	0.313	0.314
25,000	0.314	0.312
30,000	0.314	0.313
35,000	0.314	0.313
40,000	0.314	0.314

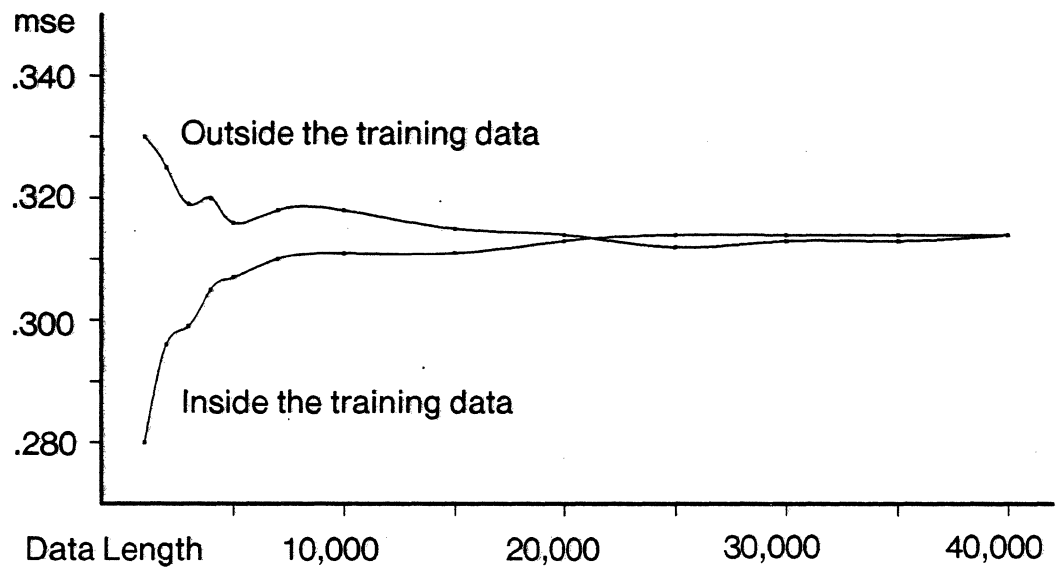


Figure 3.2 Code performance inside and outside the training sequence for a 64 codeword code as a function of training sequence length

form. The data summarizes thirteen experiments, with the training sequence varied in length from 1000 symbols up to 40,000 symbols. Each code design was tested on a separate sequence of 10,000 symbols drawn from the same source as the training sequence. The source was Gauss-Markov, or first order autoregressive,

with correlation coefficient 0.6. The Viterbi Algorithm was used as the encoder, operating on blocks of 1000 symbols and using the squared-error distortion measure. In each case the training algorithm was run for 5 iterations; in each case the change in distortion from the fourth to fifth iteration was less than one-half percent: $\epsilon = 0.005$. The initial codebook used was simply a table of 64 values consecutively drawn from the same Gauss-Markov source (but see section 3.2.1).

Table 3.4 and Figure 3.2 present average per-symbol squared error against the length of the training sequence. Although there are some non-monotonicities, the arguments of the previous paragraph are largely borne out. When the training sequence is too short, the code becomes overspecialized to the peculiarities of the training sequence – performance on the training sequence is better than it probably should be while performance outside the training sequence is worse than it should be. As the training sequence becomes longer, code performance on the training data becomes worse while performance outside improves, both approaching what one might call the *proper* performance for a code of this structure and size. Judging from these results for memoryless random sources, the training sequence must contain 70 – 100 times as many samples as the trellis code has codewords before the code performance outside the training sequence becomes close to code performance inside the training sequence. Similar results for the block code design case are shown in [53, Fig. 5].

In any of this discussion, we avoid the claim that the design algorithm is producing the best possible 64 codeword one bit per symbol code. However, the algorithm has produced codes with performance greatly improved over the 0.353 average distortion of the initial codebook, and as convergence to the vicinity of 0.314 average distortion indicates, the codes have been constructed to capitalize on *some* properties of the Gauss-Markov 0.6 source – although we may not be able to determine exactly which ones.

3.1.3 Segmentation

One of the major problems with iterative fixed point optimization procedures such as the trellis code design algorithm is that while convergence is guaranteed, that convergence is only to a *local* optimum [57]. Another problem with the trellis code design method is that nothing is said about the *rate* of convergence. In all cases tried, a few iterations seem sufficient, but this is little evidence that pathological cases of slow convergence do not exist.

By breaking the available training data into several smaller segments and training on them sequentially, it may be possible both to avoid local minima and to obtain

Table 3.5
Effect of segmenting the training sequence
Table entries are mean squared error

Initial codebook distortion: 0.328

Length	Segments	Inside	Outside
12,000	1	0.310	0.316
6,000	2	0.298	0.311
4,000	3	0.296	0.314
3,000	4	0.292	0.307
2,000	6	0.290	0.309
1,000	12	0.284	0.309

faster convergence. The *segmentation* procedure divides the training sequence into several shorter segments. The initial codebook is iteratively improved by using the design algorithm on the first training sequence segment. This improved codebook is then used as the initial codebook for the design algorithm on the second training sequence segment, and so on. There are several heuristic arguments why such a procedure might be effective. With a short training sequence, relatively few samples fall into each of the design algorithm's partition cells. Minor differences in the path sequence selected by the encoding algorithm, which result in the redistribution of a small number of training sequence samples among the partition cells, cause relatively larger movements of the codeword values because there are only a few participants in the centroid computations which produce the new codewords. These relatively larger movements of the codeword values may result in faster convergence of the codebook in a manner somewhat analagous to the use of a large adaptation constant in a gradient algorithm. In addition, since one would expect the centroid of a partition cell based on a short training sequence to have a higher variance than the centroid based on a longer training sequence, use of the short sequence results in noisy estimates of the "correct" values. These estimates may be sufficiently noisy to jog the algorithm loose from a local optimum and permit further improvement.

In order to test these ideas, several one bit per symbol code designs were completed using the same initial codebook, the same training sequence, and the same number of design algorithm iterations, but varying degrees of segmentation. In all, six designs were made, all using the same 12,000 sample sequence drawn from the independent $\mathcal{N}(0,1)$ Gaussian source. The initial decoder was a constraint-length 6 shift register decoder with the codebook populated with samples from the independent $\mathcal{N}(0,0.75)$ source. In all cases, the design algorithm was run for three iterations on each segment. Table 3.5 presents the results of these experiments.

The first column presents the length of a segment, the second presents the number of segments. In each case the product of the number of segments and the segment length is 12,000. The third column presents the per symbol average squared error of the final codebook on the last segment of the training sequence and the last column presents the per symbol squared error achieved using the final codebook on a separate test sequence of 6,000 $\mathcal{N}(0,1)$ samples.

As can be seen from these results, segmentation produces a slight improvement, provided that the segments are not too small. In the light of the previous section, this effect can be understood. As the segments become too short, the ratio of training sequence length to codebook size becomes too small and the designed code no longer fully applies to data outside the training sequence. These results suggest that an operational approach to using the design algorithm would be to first train on several fairly short sequences, and then refine the code on somewhat longer sequences. This technique has elements in common with the *k-means* technique of MacQueen [59].

3.2 Selection of Initial Decoders

We now return to the question of selecting a trellis decoder to use as an initial guess for the improvement algorithm. Section 2.2.2 described some alternative methods of constructing initial decoders. In each case – plagiarized decoders, fake process decoders, and random decoders – the question arises of whether the training algorithm can in fact improve on existing codes of a certain type and whether the improvements then apply to data outside the training sequence. Section 2.3 described a method of constructing a longer constraint length decoder from a shorter one by *extension*, but the method guarantees only the construction of a decoder *no worse* than its shorter ancestor. Lacking additional theory, in order to test the merit of the various methods of constructing initial decoders and in order to establish whether or not the extension technique actually produces *improved* decoders, we must turn to experiment.

3.2.1 Random decoders

The tests described in this section were designed to accomplish two goals. First, we wished to test the idea that one way to seed the training algorithm would be to populate an initial codebook with codewords drawn from an appropriate probability distribution – in other words, to select a *random* initial code. Second, we wished to look into the problem of local optima. Similar performance of codes derived from disparate initial codebooks would at least provide some confidence that the design

Table 3.6
Random initial codes for the Gaussian IID source
Table entries are mean squared error

length 2 (4 codewords)

initial	best	outside
0.475	0.361	0.363
0.576	0.363	0.365
0.480	0.361	0.363
0.506	0.362	0.364

length 3 (8 codewords)

initial	best	outside
0.442	0.340	0.341
0.435	0.335	0.336
0.412	0.320	0.321
0.541	0.349	0.352

length 4 (16 codewords)

initial	best	outside
0.421	0.343	0.348
0.369	0.323	0.319
0.382	0.337	0.334
0.367	0.315	0.315

length 5 (32 codewords)

initial	best	outside
0.383	0.322	0.322
0.388	0.321	0.320
0.364	0.320	0.322
0.356	0.318	0.320

length 6 (64 codewords)

initial	best	outside
0.343	0.308	0.310
0.349	0.307	0.309
0.362	0.309	0.312
0.342	0.307	0.312

length 7 (128 codewords)

initial	best	outside
0.328	0.298	0.308
0.325	0.298	0.307
0.344	0.299	0.302
0.325	0.300	0.305

length 8 (256 codewords)

initial	best	outside
0.314	0.289	0.303
0.325	0.292	0.303
0.320	0.292	0.302
0.322	0.289	0.300

length 9 (512 codewords)

initial	best	outside
0.310	0.281	0.296
0.309	0.278	0.294
0.308	0.279	0.297
0.310	0.278	0.297

length 10 (1024 codewords)

initial	best	outside
0.307	0.265	0.293
0.308	0.267	0.294

algorithm does not usually terminate in a local optimum which has significantly inferior performance. As detailed below, these tests provided some other information as well, including more evidence for the "ergodic assumption" of section 3.1.2.

From theory [13, Ch. 4] we know that the reproduction values of a data compression system should have a distribution yielding the appropriate distortion-rate function. In the case of the memoryless unit-variance Gaussian source encoded

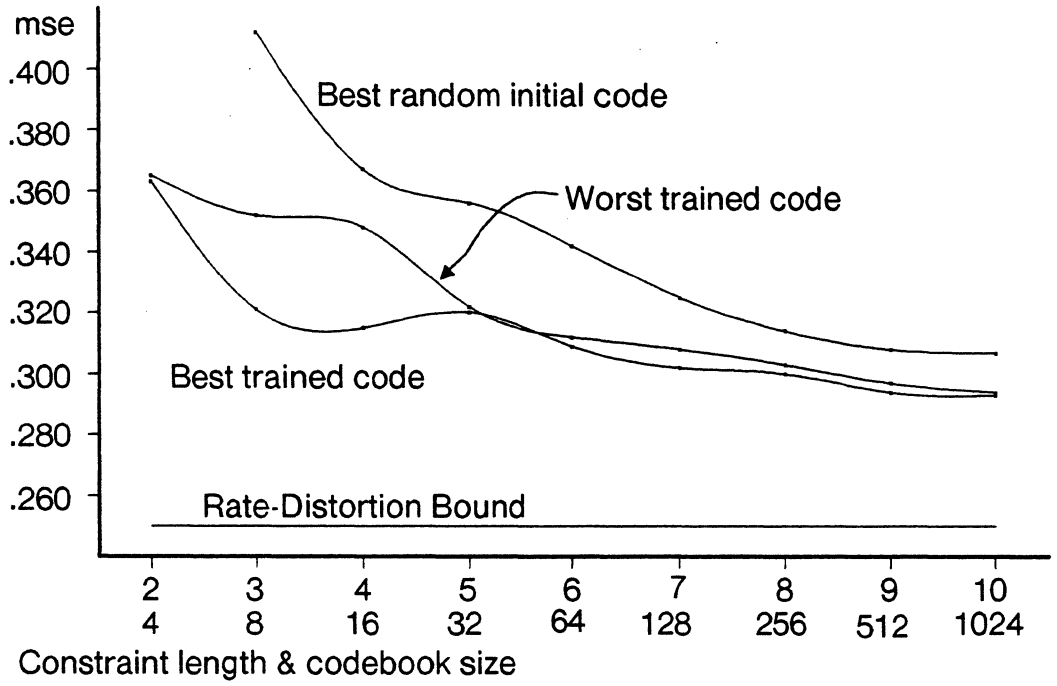


Figure 3.3 Random and trained codes for the Gaussian IID source

at rate 1 bit per symbol, this distribution is $\mathcal{N}(0, 0.75)$, as was used in section 3.1.1. Since we would eventually like to use the trellis code design algorithm for sources with an unknown distribution, we may not be able to calculate the best R-D distribution for the initial codebook. For this reason, we repeat the approach of [53] and use codewords drawn from the distribution of the source. When the source distribution is unknown, it may be estimated by a histogram of the training sequence or by selecting samples at random from the training sequence. If the distortion achieved is sufficiently low, the R-D distribution will approach the source distribution.

The random decoder experiments consisted of using the trellis design algorithm on shift register based initial decoders with constraint-lengths varying from two to ten – from four codewords to 1024 codewords – to design one bit per symbol trellis encoding systems for memoryless Gaussian noise. Except at constraint-length ten, four random initial codebooks were tested at each constraint length. The codewords for the initial codebooks were drawn from the memoryless $\mathcal{N}(0, 1)$ Gaussian source. The training algorithm used the Viterbi search algorithm with block length 100 and the squared-error distortion measure for five iterations on 200 blocks (20,000 symbols) from the source and then another five iterations on a different 20,000 symbol sequence from the same source – two segments of 20,000 symbols. The resulting codebooks were then tested on a third 20,000 symbol sequence.

All performance entries in Table 3.6 express per symbol average squared error. For comparison, the rate-distortion limit for one bit per symbol encoding of white gaussian noise is 0.25 average per-symbol squared error and the optimal one dimensional quantizer gives 0.36 average squared error. The columns labelled *Initial* present the average squared error of the initial random codebooks measured inside the training sequence. The columns labelled *Best* present the highest performance of the codebooks after training, but still measured inside the training sequence. The columns labelled *Outside* present the performance of the trained codebooks outside the training sequence. The same data are presented in Figure 3.3. For each constraint length, the performance outside the training sequence of the best and worst codes tested are shown, together with the performance inside the training sequence of the best random initial code.

Pearlman [23],[70] reports squared error performance of 0.303 and 0.301 for length 9 and 10 decoders, respectively, designed using a new theory for source coding with constrained alphabets. These results are very close to the results of Table 3.6. Pearlman's decoders use only 4 discrete codeword values, but a fairly complex function is used to map the decoder register contents to the four codewords.

For short constraint lengths, the different randomly selected initial codebooks shown in Table 3.6 provide substantially different performance, both before and after training. For longer constraint lengths, the effects of different initial codebooks on both initial and ultimate performance become slight. The latter effect provides some confidence that these codes are not often trapped in significantly inferior local optima, since one would expect the performance of distinct local optima to differ.

As the codebook size grows, the random initial codes work quite well by themselves, but the training procedure is still able to produce improved performance and that improvement applies outside the training sequence as well as inside. This effect argues that, at least for memoryless sources, random selection of initial codebooks is a worthwhile idea. The tendency for large random codebooks to work well without training is also in support of the theoretical result that, given a sufficiently large code, most random codebooks will be good ones [38]. We expect that in the limit of large constraint length, the training algorithm would not be able to obtain significant performance improvements over random codes.

As the ratio of training sequence length to codebook size declines, code performance inside the training sequence draws ahead of performance outside. The code apparently becomes specialized to the peculiarities of the training sequence rather than to the overall characteristics of the source. In these tests, the disparity in

performance inside and outside the training sequence becomes clear as the ratio of training sequence length to codebook size falls below about 200 to 1. This result is in good agreement with the results presented in section 3.1.2.

The random initial decoders described above are a special case of the Linde-Gray fake process decoders [52],[54],[55]. One of the fake process models (Figure 2.3) consists of interpreting the decoder shift register contents as a binary fraction in $[0, 1)$, which drives a scrambling function which in turn drives a function having the inverse distribution of the source. A little reflection shows that a table-lookup decoder populated by random values drawn from the distribution of the source is entirely equivalent.

3.2.3 Plagiarized decoders

Given any initial trellis decoder, the design algorithm will return one which performs at least as well; one appealing approach to the selection of initial decoders is to use trellis decoders from traditional systems. One example of a successful traditional coding system is the predictive quantizer, shown in Figure 2.1. The predictive quantizer is most often used with highly correlated sources. Since the predictive quantizer potentially contains a great deal of state information, it forms a tree code and is not directly suitable for use with the trellis code design algorithm. An approach taken by previous workers [8],[52] is to transform the infinite impulse response predictive quantization decoding filter into a finite impulse response filter by truncating the impulse response. This method produces the *Truncated Predictive Quantization Decoder* (TPQD) of Figure 2.2. When used with a trellis search algorithm as the encoder, the TPQD system can outperform ordinary predictive quantization. In this section, the trellis code design algorithm is used together with an initial TPQD decoder to produce an improved trellis code for sources with high correlation.

Using TPQD decoders as the initial guess decoders for the design algorithm, one bit per symbol codes of constraint-length 1 through 6 were designed for the Gauss-Markov source with correlation 0.9, using squared error distortion. Training and test sequences were each 12,000 sample long and the Viterbi Algorithm was used for encoding. Table 3.7 presents the average squared error performance of the original TPQD decoders and the trained decoders both inside and outside the training sequence for code constraint-lengths from 1 through 6. Figure 3.4 displays the results outside the training sequence only. In each case, the design algorithm was able to significantly improve the code although the improvement obtained is reduced at long constraint lengths.

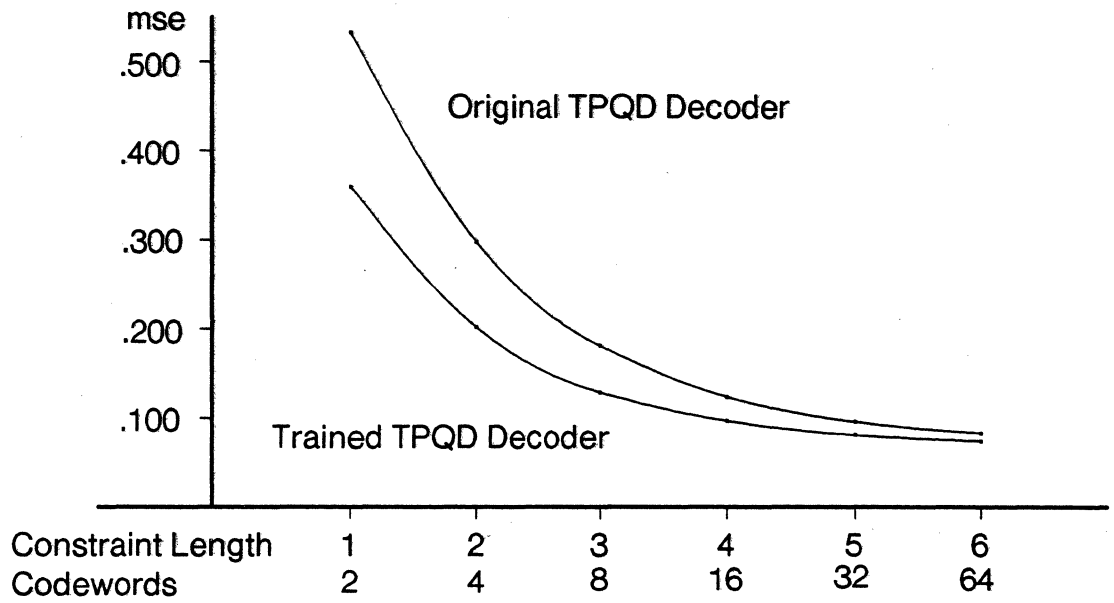


Figure 3.4 *Original and trained TPQD decoders for the 0.9 Gauss-Markov source*

Table 3.7
Truncated Predictive Quantization Decoder
Table entries are mean squared error

constraint length	1	2	3	4	5	6
number of codewords	2	4	8	16	32	64
TPQD inside	0.521	0.289	0.179	0.125	0.100	0.087
trained inside	0.364	0.207	0.135	0.102	0.086	0.078
TPQD outside	0.533	0.298	0.181	0.124	0.096	0.083
trained outside	0.360	0.202	0.129	0.097	0.081	0.074

3.2.3 Extension derived decoders

In section 2.3, the extension algorithm was introduced; by adding a stage to an existing shift register decoder, the extension method converts an existing decoder into a new decoder with constraint length one greater but that has performance at least as good as the shorter initial decoder. This section reports the successful use of extension to design decoders for a source with memory but the essential failure of the technique when used to design decoders for a memoryless source.

Two experiments were performed with the objective of designing one bit per symbol constraint-length 6 trellis decoders. The sources involved were the independent Gaussian source ($\mathcal{N}(0,1)$) and the first order Gauss-Markov source with correlation 0.9. The full extension design procedure was described in sections 2.3.5

Table 3.8
Extension design for the 0.9 autoregressive source

constraint length	1	2	3	4	5	6
number of codewords	2	4	8	16	32	64
mse inside	0.364	0.206	0.135	0.099	0.083	0.075
mse outside	0.360	0.203	0.129	0.094	0.079	0.071

Table 3.9
Extension design for the memoryless source

constraint length	1	2	3	4	5	6
number of codewords	2	4	8	16	32	64
mse inside	0.362	0.361	0.358	0.352	0.342	0.332
mse outside	0.366	0.365	0.362	0.356	0.353	0.347

and 2.3.6.

The same design procedure was followed for both sources. The training sequences for each source were 10,000 samples in length. The initial constraint-length 1 decoder contained codewords $+1$ and -1 – equivalent to a two level scalar quantizer. The trellis code design algorithm, employing the Viterbi Algorithm as the encoder, was used for five iterations to improve this initial code. The improved constraint-length 1 code was then extended and the the result used as an initial guess for the constraint-length 2 decoder. The design algorithm was used for five iterations at each constraint-length from 1 up to 6, producing a final decoder with 64 codewords. The end result of this procedure was the design of decoders of constraint-lengths 1 through 6 for each source. For each source, as a check, the final decoder of each constraint length was used to encode a separate test sequence of 10,000 symbols from the appropriate source

The results of the experiments are summarized in Tables 3.8 and 3.9. Each table contains a column for each constraint-length from 1 to 6 and rows for constraint length, number of codewords, performance of the final codebook inside the training sequence, and performance of the final codebook on the separate test sequence. The rate-distortion limits for one bit per symbol encoding are 0.25 for the memoryless Gaussian source and 0.047 for the 0.9 autoregressive Gauss-Markov source. All performance figures are average squared error.

Turning first to the autoregressive case, the extension method provided good results. With each increase in constraint length, the codes obtain significant improvement. Comparing Table 3.8 with Table 3.7, the extension method produced

codes which, for a given constraint length, outperform those derived from the truncated predictive quantization decoder – in an entirely automatic fashion. Unlike the plagiarized decoders, extension operates only on the training data supplied, using no additional information about the characteristics of the source. In the case of the random, plagiarized, and fake process decoders, the selection of the initial codebook must incorporate either external information about the source or the histogram of the training data must be used to estimate the source distribution.

In the case of memoryless sources, the situation is different. As can be seen from Table 3.9, the extension procedure produced codes of long constraint length which are little better than the shorter codes. The random initial decoders of section 3.2.1 are much better. However, this result should not be entirely unexpected. When a trellis decoder is extended, two codewords are formed where one existed before, and the values of the two codewords are the centroids of a refinement of the partition cell associated with the single original codeword. When the source is memoryless, the expected value of the centroid of an arbitrary subset of the training sequence samples in a particular partition cell is the same as the centroid of all the samples in that partition cell. The centroid conditioned on previous decoder states is the same as the unconditioned centroid. Statistical fluctuations in the centroids, which cause the two new codewords to differ, do result in some improvement with further training but, overall, the application of extension to memoryless sources must be judged a failure.

3.2.4 Conclusions

In view of the results of experiments on random initial decoders, on plagiarized decoders, and on extension derived decoders, we can now state a tentative operational procedure for selecting initial decoders for the improvement algorithm. If the source is thought to have at least some memory, extension should work quite well, if the source is memoryless, then random initial decoders will tend to be superior. In any event, use of a plagiarized initial decoder can provide a baseline performance figure that other methods must meet.

3.3 Performance on Random Sources

The Gaussian IID and Gauss-Markov (autoregressive) sources are the yardsticks of source coding. Because their rate-distortion functions are well known, these sources are often used for comparing the performance of new source coding systems both against the performance of existing systems and against the theoretical bounds.

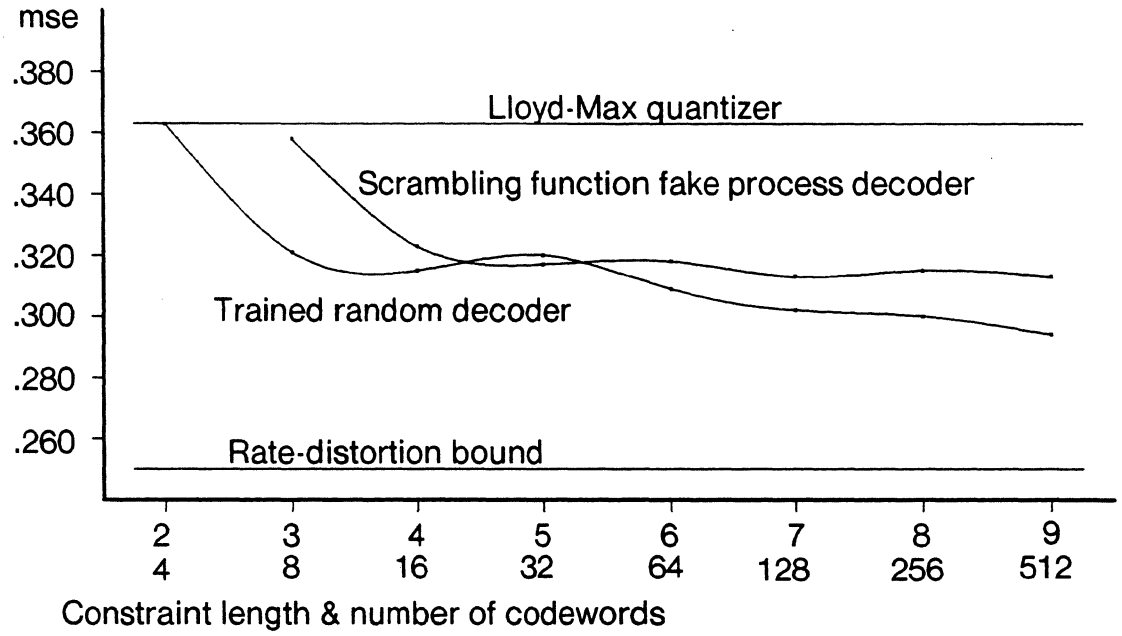


Figure 3.5 Performance of Fake Process codes and trained random codes on the memoryless Gaussian source

In this section, trellis codes designed by the iterative algorithm of Chapter 2 are compared with the earlier fake process trellis codes of Linde-Gray, with some recent results by other researchers, and with the rate distortion limits.

3.3.1 Rate distortion functions

A first order Gauss-Markov source $\{x_j\}$ is the output of a first order digital filter driven by Gaussian IID symbols:

$$x_j = ax_{j-1} + \omega_j, \qquad |a| < 1.$$

where ω is distributed $\mathcal{N}(0, \sigma^2)$. The Gaussian IID source is just the special case with $a = 0$. For $\sigma^2 = 1$ and sufficiently small distortion, the rate-distortion function for this source with squared error distortion is

$$R(D) = \frac{1}{2} \lg \frac{1-a^2}{D}, \qquad D \leq \frac{1-a}{1+a}$$

where D is the average squared error and R is the information rate in bits per symbol. The distortion constraint is met at rates of 1 bit per symbol and above for $a < 1$. The distortion-rate function is

$$D(R) = \frac{1-a^2}{2^{2R}}.$$

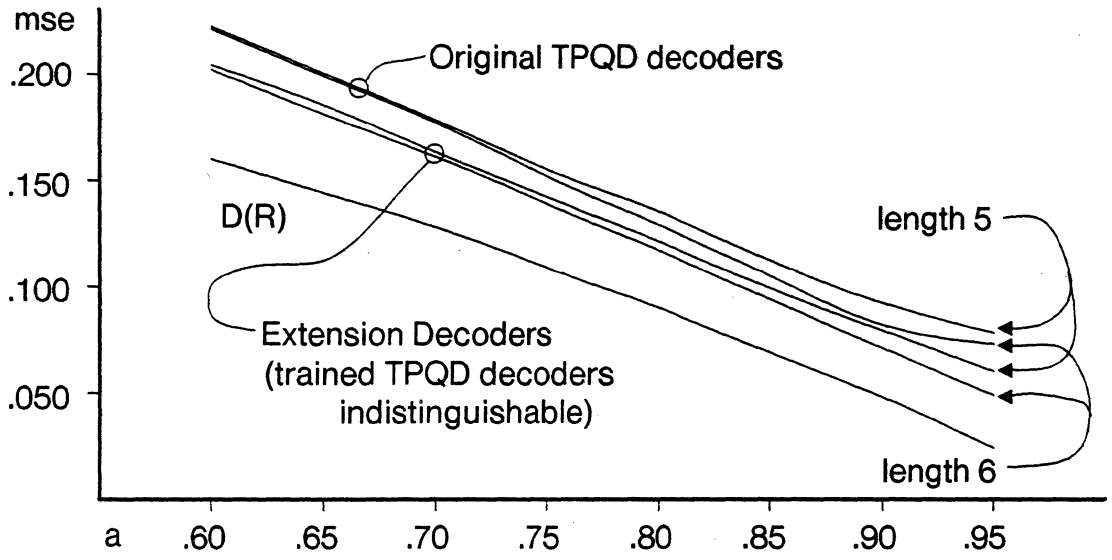


Figure 3.6 Performance of TPQD codes and Extension codes on first order Gauss-Markov sources

3.3.2 Memoryless Gaussian

Figure 3.5 displays the best results of the random initial decoder tests of section 3.2.1 together with the rate distortion bound, the best one bit scalar quantizer (the Lloyd-Max quantizer [66]), and the performance of the Linde-Gray scrambling function decoder of Figure 2.3. As noted earlier, Pearlman [70] reports mean squared error results of 0.303 for a length 9 decoder.

3.3.3 First order Gauss-Markov

The results presented here extend the results of sections 3.2.3 and 3.2.4 to cover the range of first order Gauss-Markov sources with correlation coefficients between 0.35 and 0.95. Tables 3.10 and 3.11 and Figure 3.6 contain performance data for one bit per symbol constraint-length 5 and 6 decoders of three varieties. The first group, the *Original TPQD* decoders, are truncated predictive quantization decoders designed according to [52] section 2.2.2. The second group, the *Trained TPQD* decoders, consist of the same collection of decoders, but iteratively improved by the trellis code design algorithm. The third group, the *Trained Extension* decoders, are new decoders designed by the extension method from an initial scalar quantizer with outputs $+1$ and -1. The columns b in Tables 3.10 and 3.11 present the scale factor constant shown in Figure 2.3, selected according to [52, Table 2.1]. All performance figures are per-symbol squared error. The results tend to confirm those of section

Table 3.10
Gauss Markov sources, Constraint Length 5
Table entries are mean squared error

a	b	Original TPQD	Trained TPQD	Trained Extension	$D(R)$
0.35	0.850	0.312	0.294	0.293	0.219
0.40	0.830	0.296	0.278	0.277	0.210
0.45	0.820	0.280	0.262	0.259	0.119
0.50	0.800	0.263	0.244	0.241	0.188
0.55	0.825	0.242	0.225	0.223	0.174
0.60	0.850	0.222	0.206	0.204	0.160
0.65	0.875	0.200	0.185	0.185	0.144
0.70	0.900	0.178	0.163	0.163	0.128
0.75	0.875	0.155	0.143	0.142	0.109
0.80	0.850	0.135	0.122	0.121	0.090
0.85	0.900	0.112	0.100	0.099	0.069
0.90	0.950	0.092	0.080	0.079	0.048
0.95	1.500	0.078	0.059	0.060	0.024

Table 3.11
Gauss Markov sources, Constraint Length 6
Table entries are mean squared error

a	b	Original TPQD	Trained TPQD	Trained Extension	$D(R)$
0.35	0.850	0.312	0.294	0.292	0.219
0.40	0.830	0.296	0.278	0.277	0.210
0.45	0.820	0.280	0.261	0.259	0.119
0.50	0.800	0.263	0.243	0.241	0.188
0.55	0.825	0.242	0.225	0.222	0.174
0.60	0.850	0.221	0.205	0.202	0.160
0.65	0.875	0.199	0.184	0.181	0.144
0.70	0.900	0.177	0.162	0.161	0.128
0.75	0.875	0.152	0.141	0.139	0.109
0.80	0.850	0.129	0.118	0.117	0.090
0.85	0.900	0.105	0.094	0.094	0.069
0.90	0.950	0.082	0.072	0.071	0.048
0.95	1.500	0.073	0.048	0.049	0.024

3.2.3 – the extension and the trained TPQD decoders work about the same and represent an improvement over the untrained TPQD decoders. Linde [52] notes that performance of the TPQD decoders tends to tail off for high correlations. We note that this effect seems less pronounced for the trained decoders. Current research

results of Gray for one bit per symbol block codes designed using the block code design algorithm yield results comparable to those reported here. For $a = 0.85$, a 128 codeword block code has been designed which yields an average distoriton of 0.101. For $a = 0.90$, an average distortion of 0.073 has been obtained [41].

3.4 The Effects of Code Structure

In Chapter 1, trellis decoders with a general finite-state machine structure were mentioned. One example was a finite precision arithmetic representation of a predictive quantization decoder. However, all example decoders we have actually examined have used the traditional shift register structure. While theoretical results for trellis codes [38] show that shift register decoders are *sufficient* to construct systems which operate close to the rate-distortion bounds, other code structures might offer better performance for equivalent complexity.

In this section, we begin by describing some classes of decoder structures, and by introducing some terminology from graph theory. A bound is developed on the number of possible decoder structures. These structures are categorized by their synchronization properties and by their response to channel errors. We then turn to the results of some tests on random sources which indicate that, at least for certain sources, alternative code structures perform better than shift registers. We do not specifically address the problem of finding the best finite-state code structure, but only attempt to demonstrate that the shift register may not always be best.

3.4.1 Alternative shift register structures

The mathematical description of the shift register trellis decoder in section 2.2 was generalized for a q-ary channel. In the case that $q = 2$, this is the familiar binary channel and binary shift register. The associated trellis was shown in Figure 1.4. It is equally possible to construct systems with $q \neq 2$. For example, Figure 3.7 shows a $q = 3$ trellis corresponding to a three stage ternary shift register.

However, q is not the only variable. It is also possible to interpret n-tuples from the source as single symbols. In the case of the binary channel, a system accepting symbol pairs from the source might be considered either as a rate 1/2 bit per symbol system or as a rate 1 system operating on a two dimensional vector source. This concept introduces systems which are in a sense intermediate between block and sliding block codes. Figure 3.8 a) shows a degenerate trellis structure with one decoder state and two codewords, a and b. This code is identical to a length 1 block code or to a scalar quantizer. Similarly, Figure 3.8 b) shows a 2

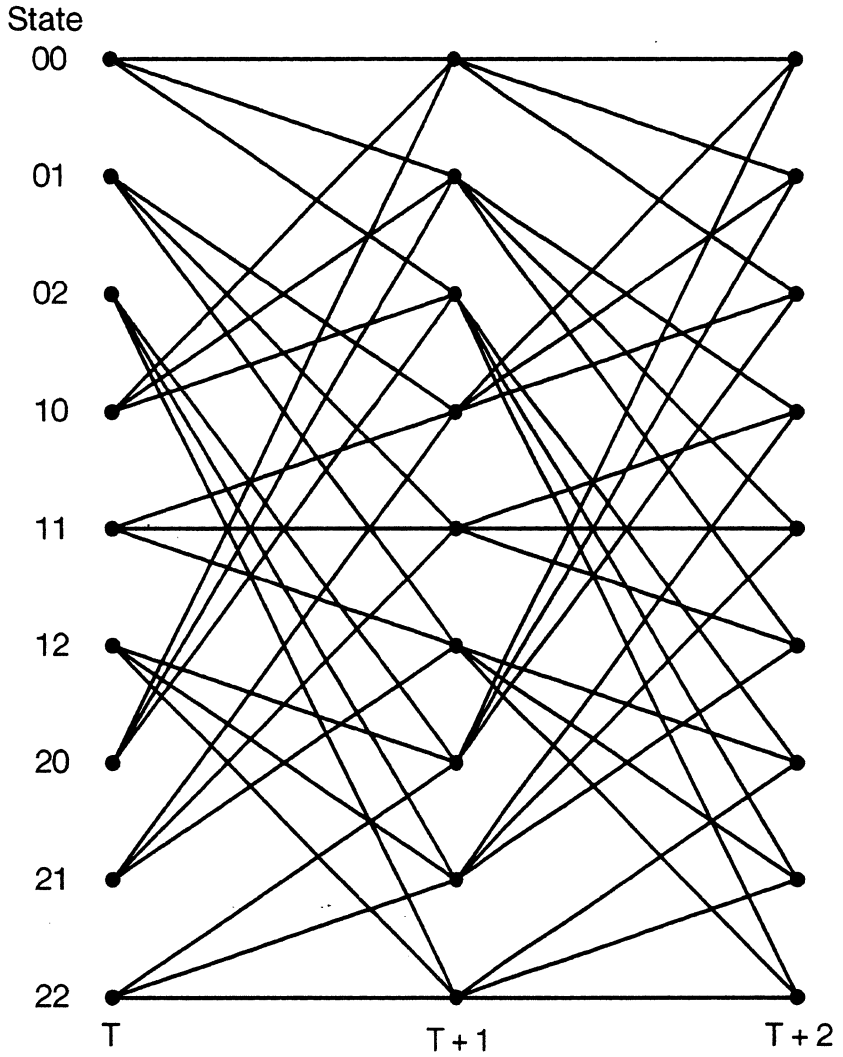


Figure 3.7 *Constraint-length 3 ternary trellis*

bit per symbol 1-state trellis code for the two dimensional vector source. The four codewords of this system, each a vector of length 2, are *aa*, *bb*, *cc*, and *dd*. This system is identical to a length 2 block code. In general, such codes consist of a collection of block codes connected by a trellis in such a way that the particular block code used to encode the source vector starting at a certain time is selected by the previous codeword sequence. We will describe such coding systems as "rate m/n", where m is the channel rate per source vector and n is the vector length. Interpreted as a fraction, m/n is also the channel rate per source symbol.

The trellis code design algorithm was used to construct a rate 2/2 code for the memoryless Gaussian source. The code operates at an overall rate of one bit per symbol but in fact encodes successive symbol pairs from the source and produces outputs for a $q = 4$ channel. The decoder contained 64 codeword pairs, for a

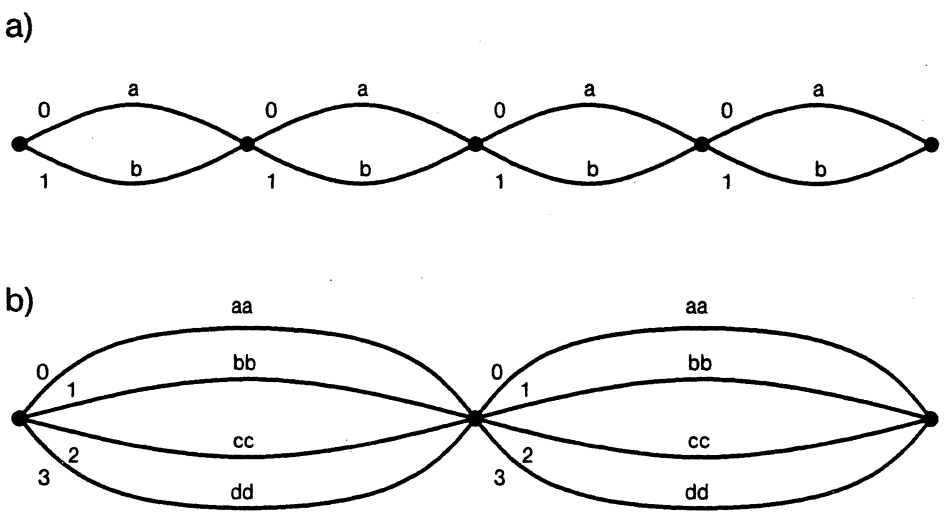


Figure 3.8 *Alternative one bit per symbol code structures*

Table 3.12
Rate 2/2 code with random initial codebooks
Table entries are mean squared error

length 3 (64 codeword pairs)

initial	best	outside
0.350	0.307	0.313
0.369	0.307	0.310
0.345	0.307	0.309
0.365	0.306	0.312

constraint length of three source vectors or channel symbols or six source samples. Except for these differences, the design procedure was identical to that for the codes described in section 3.2.1, so the performance results shown in Table 3.12 should be compared with those of Table 3.6.

Since symbol pairs from the source are encoded, the codebook for the rate 2/2 code is twice as large as that for the rate 1 code of equivalent constraint length. For this reason, comparisons are difficult, but the performance obtained is similar to the performance found for rate 1 codes of equivalent constraint length.

3.4.2 Graph representations of trellis decoders

In order to preserve generality, we can allow a trellis decoder to incorporate an arbitrary finite-state machine, using the following notation.

A finite-state machine is a five-tuple $[V, W, Z, \nu, \varsigma]$, where $V = \{v_i \mid i =$

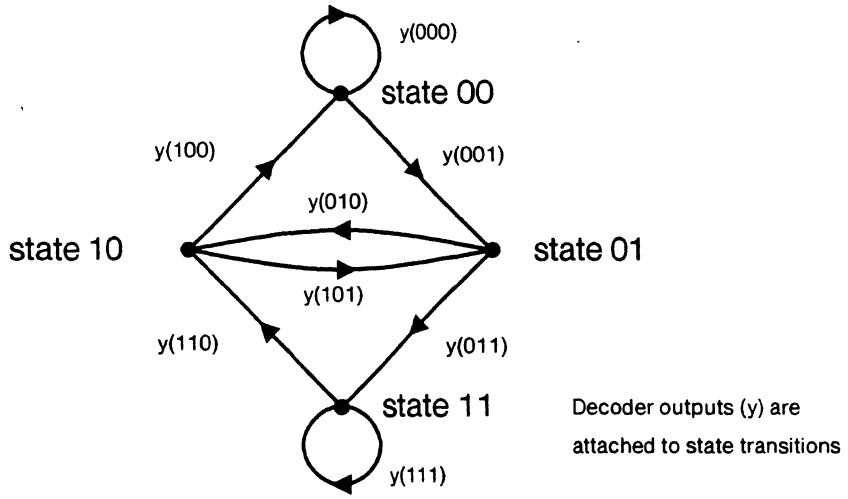


Figure 3.9 Graph for a constraint-length 3 one bit per symbol shift register decoder

$0, \ldots, q-1\}$ is the input (channel) alphabet, Z is the output alphabet, $W = \{w_i \mid i = 0, \ldots, R-1\}$ is the set of states, $\nu : W \times V \mapsto W$ is the *next-state function*, and $\varsigma : W \times V \mapsto Z$ is the *output function* [15, Ch. 3].

Using this notation, an R-state trellis decoder designed for a q-ary channel with channel sequence $\{u_j\}$ and output sequence $\{\hat{x}_j\}$ will consist of an R-state finite-state machine with state variable r, a time-invariant next-state function $r_{j+1} = \nu(r_j, u_j)$, and a time-invariant output function $\hat{x}_j = \varsigma(r_j, u_j)$. The function ν, given the current decoder state and the current channel symbol, produces the succeeding decoder state. The function ς, given the same information, produces the decoder output. Without loss of generality, we can use a table-lookup implementation of the output function ς consisting of a codebook $Y = \{y_i, i = 0, \ldots, N-1\}$ and a specialized version of the output function ς' which produces the codebook index of the decoder output rather than the codeword itself. The codebook must have cardinality $||Y|| = N = qR$. We will use $\varsigma(r_j, u_j) = \hat{x}_{(qr_j+u_j)}$ or $\varsigma'(r_j, u_j) = (qr_j + u_j)$ as the decoding function, although there are other candidate functions distinguished only up to a permutation of the codebook contents. Use of a lookup-table decoder implementation is required if the trellis code design algorithm is to be used.

A trellis decoder or a finite-state machine may be described by a directed graph. Figure 3.9 is a graph representation of a constraint-length three shift register decoder for a binary channel. The graph has exactly two arcs leaving each state, corresponding to the possible values of a binary channel symbol. Such graphs are called *uniform out-degree*. (The graph of Figure 3.9 is also *uniform in-degree* since

exactly two arcs enter each state.) When a channel symbol arrives, the decoder traverses an arc from its current state to a new state and produces the output symbol associated with that transition. In the general case of an R-state trellis decoder for a q-ary channel, the decoder may be represented as an R-node uniform out-degree q graph together with an output codebook and the fixed output index function ζ'.

It is also possible to construct a graph and then to interpret it as a trellis decoder, but some restrictions must be placed on the acceptable graphs. We have already established that a graph representing a decoder for a q-ary channel must be uniform out-degree q. The graph for a decoder should also be *strongly connected* – there must be at least one path through the graph from every node (state) to every other node [2]. If this is not the case, then either the decoder contains unattainable states and might as well not include them, or the decoder contains states that are possible to reach, but from which it is not possible to return. The uniform in-degree property possessed by shift register decoders is not itself a requirement for a general trellis decoder, but it makes implementation of the Viterbi Algorithm simpler. For that reason, all the graphs we will examine will possess this property.

Consider the R-state decoder for a q-ary channel mentioned above. The number of possible structures such a decoder could have is certainly bounded above by the number of uniform out-degree q graphs of R nodes. Since the destination of each arc in such a graph is unconstrained, this quantity is $(R^q)^R$. If we require trellis decoders to be uniform in-degree for easier implementation of the Viterbi encoding algorithm, then the bound reduces to $(R!)^q$. (A uniform in-degree 1, uniform out-degree 1 graph is just a permutation. $(R!)^q$ represents q independent overlaid permutations.) Since the entries in the decoder codebook may be permuted, the number of topologically distinct decoder structures is further reduced by a factor of $R!$ to $(R!)^{q-1}$. For binary channels, $q = 2$ and this value is just $R!$. Unfortunately, $R!$ is too large to consider an exhaustive search for the best decoder structure in other than trivial cases.

3.4.3 Channel errors and synchronization

Although source codes (data compression codes) are designed for use with noiseless channels, their behavior in the presence of channel errors is often of considerable interest. The literature on DPCM and the various sorts of adaptive PCM contain lengthly discussions of this topic [27],[78]. While we take the position that a noiseless channel can be provided through appropriate use of channel coding, trellis codes are certainly not exempt from noise considerations. The possible finite-state decoder

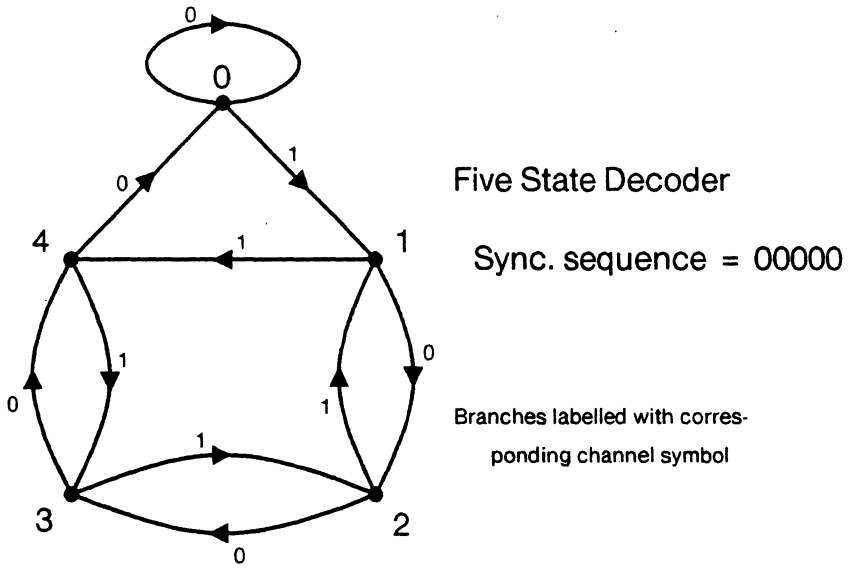


Figure 3.10 *Synchronizable five state decoder*

structures may be divided into three categories, based on their response to channel errors and on the ways in which the encoder and decoder can be resynchronized after an error.

The first category is represented by the shift register decoder. While an erroneous channel symbol may result in an error in the output of such a decoder, the decoder will return to the desired output sequence as soon as the channel error has been shifted out of the decoder register. The property of interest is that the state of the decoder depends only on a fixed, finite, number of previous channel symbols limited by the constraint length of the decoder. These codes are intuitively sliding block; their output sequences are generated by a finite-length sliding window on the channel sequence [38].

The second category of decoders, while lacking the finite window property, can still be resynchronized after an error. A code of this class is characterized by the existence of a *synchronization sequence*. The occurrence of a particular finite length channel sequence will place the decoder in a certain state, regardless of the previous state of the decoder. This class of structure is illustrated by the graph of Figure 3.10. While the state of this decoder (graph) may depend on the indefinite past of the channel sequence, after a sequence of five 0 symbols is received the decoder will be in state 0 – wherever it started.

The final category of decoder both lacks a synchronization sequence and depends on the indefinite past. Figure 3.11 contains an example. This class of decoder can still be synchronized, but only by adding an external device which independently

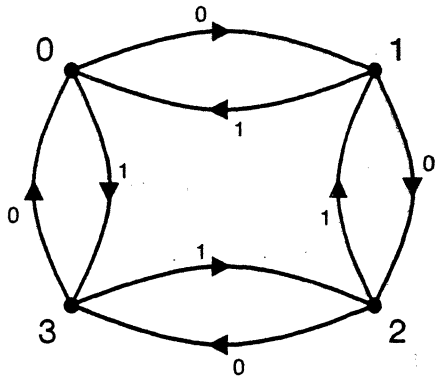


Figure 3.11 *Four state decoder for which no synchronization sequence exists*

watches the channel for a synchronization sequence. When the sequence is detected, the decoder is directly set to a certain state. Since the synchronization sequence may occur at random in the output of the trellis encoder, some method such as the *bit-stuffing* technique used by the SDLC protocol [21] must be used to eliminate unwanted occurrences of the synchronization sequence. Although bit-stuffing and similar procedures permit synchronization, they introduce a variable transmission rate by altering the channel sequence in a data dependent way.

3.4.4 Examples of finite-state decoders

This chapter concludes with the experimental evaluation of several non-shift register decoders. The first, a delta modulation decoder, offers improved performance for some sources, the others provide performance roughly equivalent to that of a shift register decoder.

3.4.4.1 Delta modulation decoders

Figure 3.12 shows both a trellis and a graph for a 6 state delta modulation decoder. The trellis and graph transitions are labelled with the output codewords. The "codewords" of a delta modulation decoder are equally spaced values within the range of the decoder. The structure of the delta modulation trellis differs from the usual shift register trellis (Figure 1.4), but its operation is easy to understand. The arrival of a 1 from the channel causes the decoder to "step-up", and the arrival of a 0 causes the decoder to "step-down". The top and bottom *trap states* act as limits, preventing the decoder output from leaving a certain range. Delta modulation decoders are not usually specified with these upper and lower limits, but analog implementations are in fact limited by available power supply voltages and digital

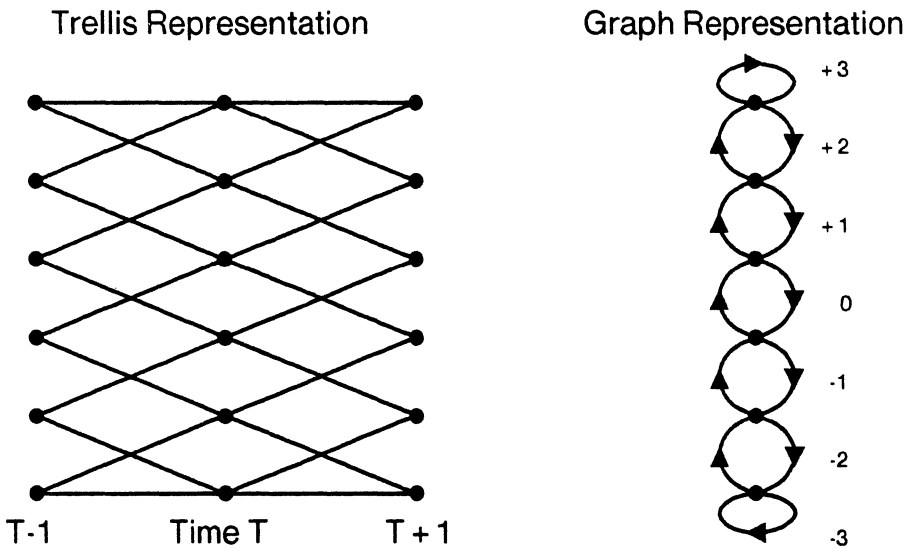


Figure 3.12 *Six state delta modulation decoder*

implementations are limited by the size of the state accumulator. Another viewpoint is that the inclusion of the trap states corresponds to the truncation applied to the TPQD decoders. The delta modulation structure is a member of the second category of decoders. While the state of the delta modulation decoder may depend on the indefinite past, sequences of six 1's or six 0's act as synchronization sequences by placing the decoder in, respectively, the top or bottom states.

Since the step-up, step-down nature of the delta modulation structure makes slowly changing sequences of outputs easier to produce than rapidly changing sequences of outputs, decoders with a delta modulation structure should be able to outperform decoders based on the more uniformly interconnected shift register structure when used to encode sources with high positive correlation. In order to investigate this idea, the Viterbi search algorithm was used to test both 32 codeword (constraint-length 5) shift register decoders and 32 codeword delta modulation decoders on several sequences of 10,000 samples from the first order Gauss-Markov source $x_j = 0.9x_{j-1} + \omega_j$, driven by $\mathcal{N}(0,1)$ noise. All tests were at rate one bit per symbol. The best step size for the initial delta modulation codebook was found by experiment to be about 0.9, giving a decoder with outputs ranging from -7 to +7. The shift register decoder was designed by the extension method.

With a single path search encoder (a traditional delta modulation encoder), the initial delta modulation code achieved an average squared error of 0.110. With a search algorithm for encoding, this initial code achieved performance in the range 0.076 - 0.072. After five training iterations, the delta modulation code performed

very sightly better, in the range 0.074 – 0.072 on data outside the training sequence. The shift register code, designed by extension, achieved performance in the range 0.081 – 0.079.

3.4.4.2 Random finite-state decoders

In the previous section, we argued that a decoder finite-state machine with good local connectivity would be well matched to the slowly changing sequences typical of highly correlated sources. Memoryless sources, such as the IID Gaussian source, lie at the opposite end of the spectrum from the highly correlated sources.

In the case of memoryless sources, there is no obvious way to assign connections within the decoder finite-state machine. In this section, we report the results of performance tests on decoders with *random structure* as well as random initial codebooks. All these tests encoded white gaussian noise at a rate of one bit per symbol, using 256 state (512 codeword) decoders. One group of decoders incorporated the shift register structure with initial random codebooks selected from the probability distribution $\mathcal{N}(0,1)$. (These initial codebooks are similar to those reported in sections 3.2.1 and 3.3.1.) The second group of decoders incorporated more general finite-state machines with random internal structure, together with the same set of initial codebooks. Each test consisted of training the initial codebook for three iterations on each of two sequences of 20,000 $\mathcal{N}(0,1)$ Gaussian samples using the M,L Algorithm with squared-error distortion, $M = 20$, and $L = 31$. The resulting trained codebooks were then tested on a third sequence of 20,000 samples.

In order to assure that the random finite-state machines were all strongly connected, a somewhat constrained structure was used. First, a connection was assigned from each state r to state $r+1 \pmod{256}$, linking all the states into a ring or *Hamiltonian cycle* [47]. Second, a connection was assigned from state 0 to state 0, and third, a random permutation of 255 elements was selected and used to assign an additional connection from each of the remaining 255 states. This procedure produced a uniform in-degree 2 out-degree 2 finite-state machine which is strongly connected and which possesses a synchronization sequence, but which also contains a good deal of random structure. Figure 3.13 shows a 16 state version of this type of decoder structure.

Table 3.13 shows the results of this series of tests. For each type of decoder, and for the eight different initial codebooks, the initial performance, performance after training but within the training sequence, and the performance after training

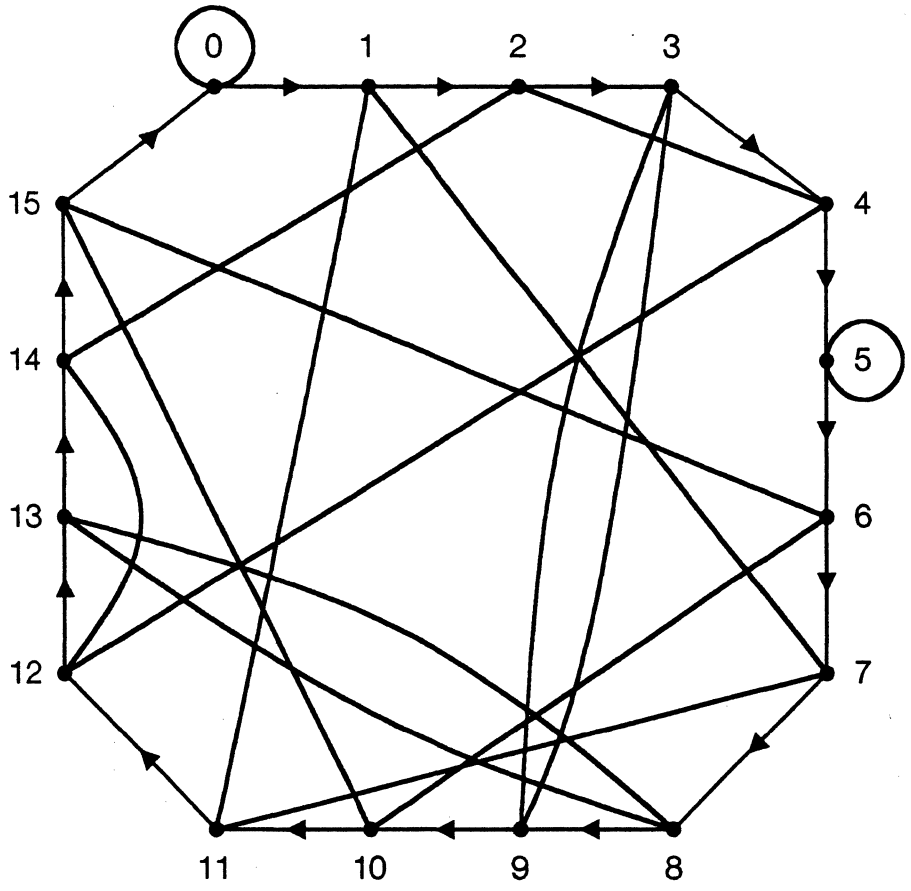


Figure 3.13 *Sixteen state random finite-state machine*

Table 3.13
Random finite-state decoders
Table entries are mean squared error

Shift register FSM			Random FSM		
initial	best	outside	initial	best	outside
0.328	0.302	0.308	0.329	0.303	0.315
0.329	0.300	0.312	0.334	0.303	0.313
0.327	0.296	0.310	0.322	0.302	0.315
0.331	0.302	0.311	0.330	0.304	0.313
0.329	0.305	0.312	0.333	0.303	0.314
0.332	0.304	0.311	0.330	0.305	0.315
0.346	0.303	0.312	0.339	0.305	0.313
0.337	0.300	0.313	0.334	0.303	0.312

and outside the training sequence are listed. The average performance levels of the random finite-state machines are generally slightly less than those of the shift register decoders.

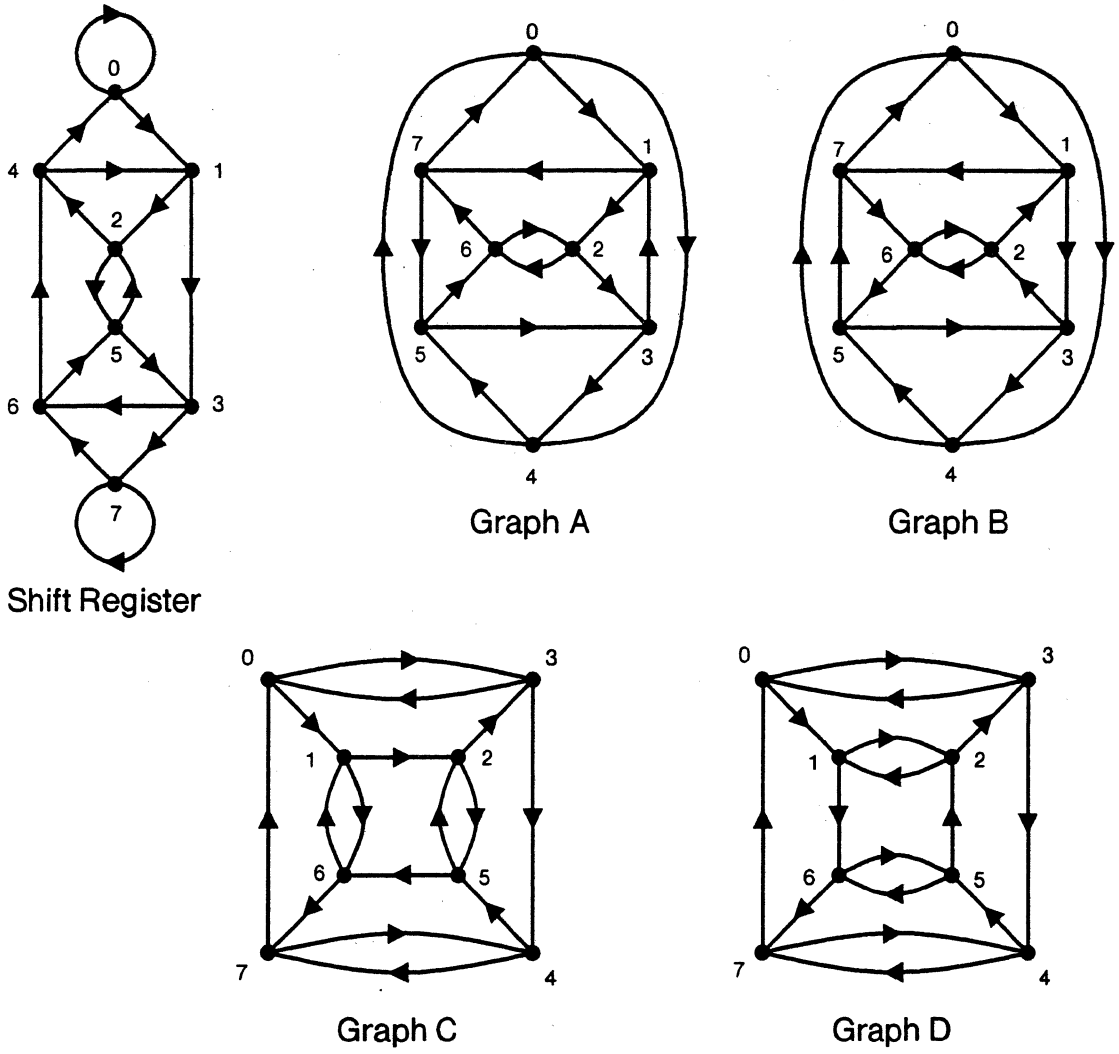


Figure 3.14 *Eight node minimum average path length graphs*

3.4.4.3 Tightly connected decoders

In the previous section, we approached the problem of designing finite-state machine decoders for memoryless sources by selecting a random finite-state machine. In this section, we argue that a decoder finite-state machine with good connectivity to *both* local and distant states should be well matched to the jumpy sequences typical of uncorrelated sources.

In a uniform out-degree two graph, initialized to a certain state, it should be possible to reach distinct states a and b in one step. In one more step, four more distinct states: c, d, e, and f should be accessible, and so on. If this structure were independent of the initial state, then the graph would be as highly connected as possible. In particular, the number of state transitions required to reach state i

Table 3.14
Minimum average path length decoders
Table entries are mean squared error

Random Codebook	Shift Register	Graph A	Graph B	Graph C	Graph D
1	0.331	0.322	0.337	0.354	0.348
2	0.336	0.335	0.340	0.350	0.348
3	0.324	0.354	0.348	0.355	0.357
4	0.315	0.338	0.342	0.352	0.356
5	0.341	0.328	0.328	0.344	0.331
6	0.328	0.334	0.335	0.333	0.323
7	0.341	0.325	0.335	0.334	0.349
8	0.335	0.329	0.339	0.350	0.333

from state j, averaged over all i and j, should be a minimum. In the special case of eight state decoders, the minimum possible average path length is two – two states reachable in one step, four more states in two steps, and the final two states requiring three steps. In contrast, the eight state (three bit) shift register has an average path length of 2.25.

An exhaustive search was conducted for such *minimum average path length* graphs of eight states, with the intent of comparing their performance to that of the eight state shift register. Only uniform in-degree 2, uniform out-degree 2 graphs were considered. The search discovered four topologically distinct graphs, which are shown, along with the shift register graph, in Figure 3.14. Each decoder was trained for five iterations on a 10,000 sample IID Gaussian sequence, then tested on a different 10,000 sample sequence from the same source. This sequence of tests was repeated eight times, with different initial codebooks. The eight random codebooks were drawn from a $\mathcal{N}(0, 0.75)$ Gaussian source. The results of these tests are shown in Table 3.14, which shows the per symbol squared error achieved outside the training sequence.

3.4.5 Conclusions

At least for the problem of encoding a highly correlated source, an alternative decoder structure – delta modulation – was able to outperform a decoder based on the shift register. While we have no constructive method of selecting finite-state machines that are well matched to particular sources – but see [14],[22], and [84] – this experience provides evidence supporting the idea that traditional coding schemes may provide a fruitful source not only of initial codebooks, but of decoder structures as well.

4. CODING SPEECH SOURCES

Chapter 2 presented the basic components of the trellis code design algorithms, and Chapter 3, in the context of coding random sources, examined some of the many operational considerations of using the algorithms to design real systems. In this chapter, we turn to the problem of designing tree and trellis encoding systems for speech. Following some historical review and general information on distortion measures, the three main sections of the chapter discuss trellis codes for the original speech waveform, trellis codes for the residual signal of linear predictive coding systems, and a tree encoding system using a hybrid decoder for the speech waveform. The results of the hybrid tree codes have been very encouraging, as these codes can provide good quality speech at rates of one bit per sample through an automatic design procedure.

4.1 Existing Systems

The literature on speech coding is vast, but some good surveys are [16], [24] and [27]. For the lower rate speech codes, [30] and [62] are good places to start. Section 4.1.3 will add some further discussion of existing tree and trellis codes for speech and section 4.3 provides some additional information on work with linear predictive coding (LPC) and LPC derived systems.

4.1.1 Spectrum of speech compression

Figure 4.1 shows a partial listing of speech compression techniques in rough correspondence to their associated data rates. The coding methods shown may be grouped into two general classes. Above about one bit per sample, most coding systems seek to reproduce the original speech waveform. Below this rate, most systems seek to produce a signal that *sounds* like the original speech, although the waveform produced may not be at all similar. These systems represent attempts to reduce the data rate required for communications by taking advantage of various characteristics of speech. The waveform codes consist mainly of linear and nonlinear quantizers and their predictive and adaptive variations. The non-linear quantization schemes, for example, recognize that speech has a wide dynamic range and provide quantization levels placed to handle with equal fidelity both small signals and larger ones. An equivalent linear quantizer would need many more levels to achieve the same results. The predictive and adaptive coders use knowledge of the recent history of a speech waveform to reduce the dynamic range which must be quantized.

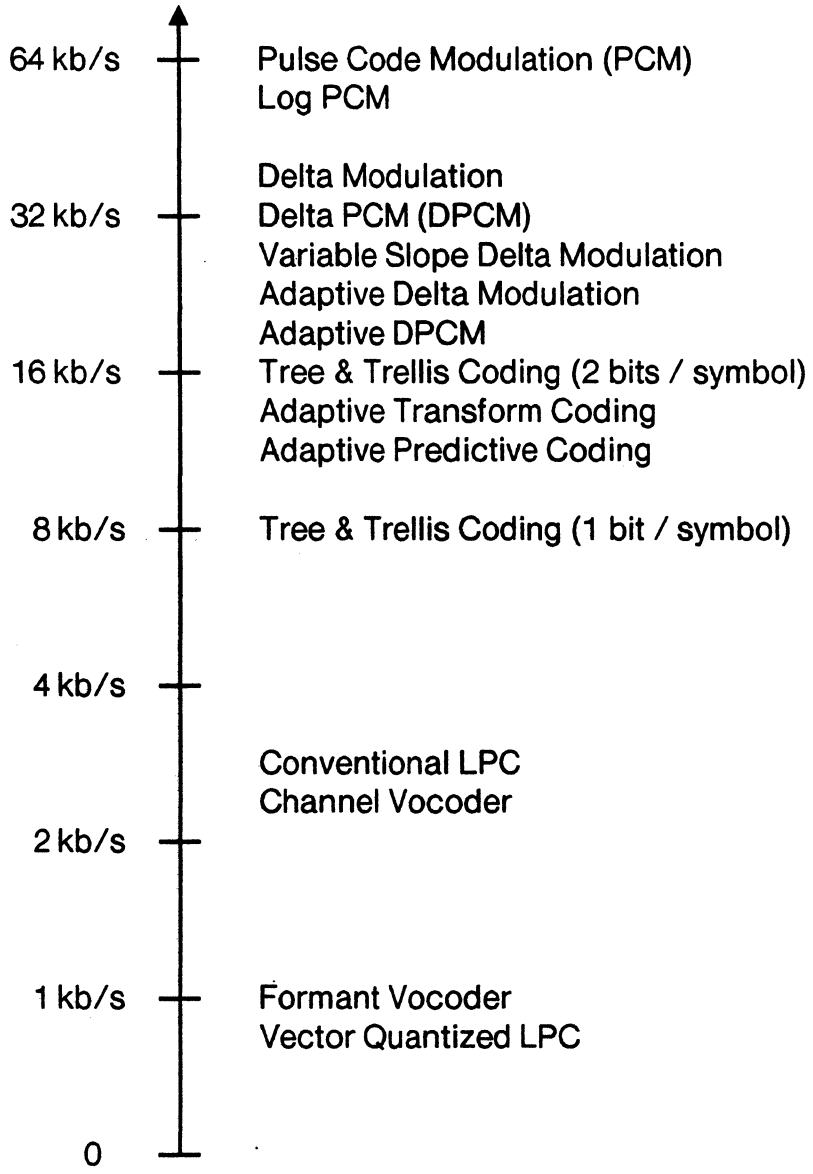


Figure 4.1 *Channel bit rates for various speech coding techniques*

The lower rate speech coding systems, represented in Figure 4.1 by linear predictive coding systems and by the channel and formant vocoders are fundamentally different. They operate by constructing a model of a segment of speech and then transmitting information about the model. At the receiver, the model is reconstructed and driven by a locally produced driving process to recreate the original speech sounds. These systems can successfully avoid encoding the exact waveform since the ear is insensitive to many of its details. These systems are usually referred to as *speech* or *process* coders to distinguish them from the waveform coders.

4.1.2 Distortion measures for speech coding

Both the trellis code design algorithm and the actual process of encoding a sequence with a tree or trellis encoder require a single symbol additive distortion measure. In previous sections, either the distortion measure has been left in its general functional form $d(x, y)$ or a squared error distortion measure has been used. When coding speech, at best the proper distortion measure is a matter of some debate. At worst, it is claimed that a subjective listening test is the only method holding any validity. If the algorithms discussed earlier are to be used for speech at all, the later position is untenable. Because our purpose in attempting to code speech is more to demonstrate at least some practical application for tree and trellis coding than necessarily to produce the best possible speech coding system, we can to a degree sidestep this question. We adopt the following positions: that the squared error distortion measure is at least roughly correlated with perceived speech quality, and that variations of the squared error measure may yield somewhat better results. In spite of these positions, we acknowledge that the perception of the listener must be the ultimate judge.

In order to provide a basis for the discussion of the variations of squared error, we first briefly describe some of the existing distortion measures for speech and their associated arguments [35],[39]. It is generally accepted that fidelity of the short term spectra of the reproduced speech signal is of great importance [24]. While tree and trellis coders operate in the time domain, connections can be made between the time domain distortion measures – which operate on the difference between the original speech waveform and the reproduction – and the spectrum of the error signal.

Minimization of average squared error in the time domain corresponds in the frequency domain to the production of a uniform error spectrum since the distortion measure places equal emphasis on error signal energy at all frequencies. However, this may not be entirely appropriate. The long term power spectrum of speech is not flat, but rather falls off with increasing frequency [58],[69]. In addition, there is an auditory property called masking: the hearing process tends to ignore noise in frequency bands containing considerable speech energy, but is more sensitive to noise in regions of the audible spectrum containing little speech energy. Variations of the squared error distortion measure can take both of these properties into account. The time invariant approach, taken by encoding systems utilizing fixed prediction, is to apply a fixed frequency weighting to the error signal, so that high frequency portions of the error signal are suppressed more than low frequency portions. A time varying approach is taken by most encoding systems utilizing adaptive prediction.

In such systems, the spectrum of incoming speech is whitened by a linear filter, and coding based on squared error distortion is applied to minimize the power of the resulting *residual* signal. At the decoder, the driving process representing the residuals is passed through the inverse spectral shaping filter. If the prediction filter is time varying, then the noise (error) power spectra tend to the same form as the short term speech spectra, thus taking advantage of the masking effect.

One additional auditory property, although not as pronounced or as well accepted as the frequency sensitivities discussed above, is that the ear seems to be less sensitive to noise during low amplitude speech segments than during high amplitude segments. Rather than maintain a uniform signal to noise ratio for all amplitudes, it may be desirable to expend relatively more effort to achieve high SNR during speech segments of higher amplitude.

While the trellis codes discussed in this chapter all use a form of the squared error distortion measure, various modifications are introduced. Section 4.2 describes a frequency weighted error measure for multiple path searched codes and section 4.4 describes a gain weighted squared error measure. Section 4.5, in discussing the hybrid tree codes, returns to squared error, but describes some possible modifications both to the code design algorithm and to the distortion measure which may be fruitful avenues for future research.

4.1.3 Previous tree and trellis codes for speech

The use of tree and trellis techniques for speech coding has quite a long history [19]. Most of the early work was in the area we have called *plagiarized decoders* – the application of a tree search algorithm to the encoding problem in a standard coding system. In the literature, this technique is usually referred to as *delayed decision* [12],[32],[77],[86]. By using a tree encoder, which delays the encoding process by a number of samples in order to observe the consequences of particular encodings, delayed decision systems are able to achieve improved performance. The general concensus has been that such methods can yield improvements in signal to noise ratio, but that the improvements are not usually audible [28]. A second class of approaches to tree coding of speech has been the design of tree decoders based on short or long term correlation functions of speech [7],[17],[43],[58],[65],[69],[82],[83]. In these systems, finite or infinite impulse response digital filters are used as decoders, with the intent of matching the spectral properties of speech or of populating the code tree with values having a desired distribution. These approaches have a good deal in common with the fake process ideas of Linde and Gray. Systems both with fixed and with adaptive decoders have been built [61],[63],[81]. More recently, a

third approach has been the use of an optimization algorithm to improve an initial decoder [10]. The technique uses a variational approach and a training sequence to adapt the tap weights of a transversal filter. This approach is similar in spirit to the techniques of this thesis, but considers only linear decoder structures.

4.2 Speech Coding System

The codes designed in this chapter are for actual speech. This section describes the source data, design algorithm parameters, and other constant factors used later in the chapter.

4.2.1 Speech data

Roughly one hundred seconds of digital speech data were available, consisting of 640,000 samples of speech sampled at 6500 Hz using a 12 bit linear PCM encoding. The speech data were made available by Signal Technology, Inc. of Santa Barbara, California. Since the residual excited and hybrid coding systems discussed in sections 4.5 and 4.6 are associated with linear predictive coding techniques, we will also refer to the speech data as broken into 5000 LPC frames of 128 samples each – corresponding to a rate of about 50 frames per second. The speech data include segments from five different male speakers. While the speech segments were not phonetically balanced or otherwise selected, they were carefully recorded in a low noise environment and with good gain control.

Of the available data, the code design efforts discussed later used two segments of 600 frames or about 12 seconds each. The first segment used was "Okay, um, I could talk about a trip, ah, that, ah, I made quite a few years ago across Canada. It was just marvelous. ah." This segment was used as the training sequence for the trellis code design algorithm. The second segment, by a different speaker, was "My, ah, most recent extended travel, I guess, was the trip I made to Sweden. I went to Stockholm and to Gunnar Fant's lab." This segment was used as a test sequence, to check the performance of designed decoders on different data.

Figure 4.2 shows the waveform for slightly less than 1/4 second of speech from the training data. Each line contains 384 samples, or the equivalent of 3 LPC frames. Segments representative of both voiced and unvoiced speech are shown.

4.2.2 Encoding algorithm and design parameters

While the results of section 3.1.1 indicate that the optimal Viterbi encoding algorithm holds a considerable performance advantage over the M,L Algorithm, the costs

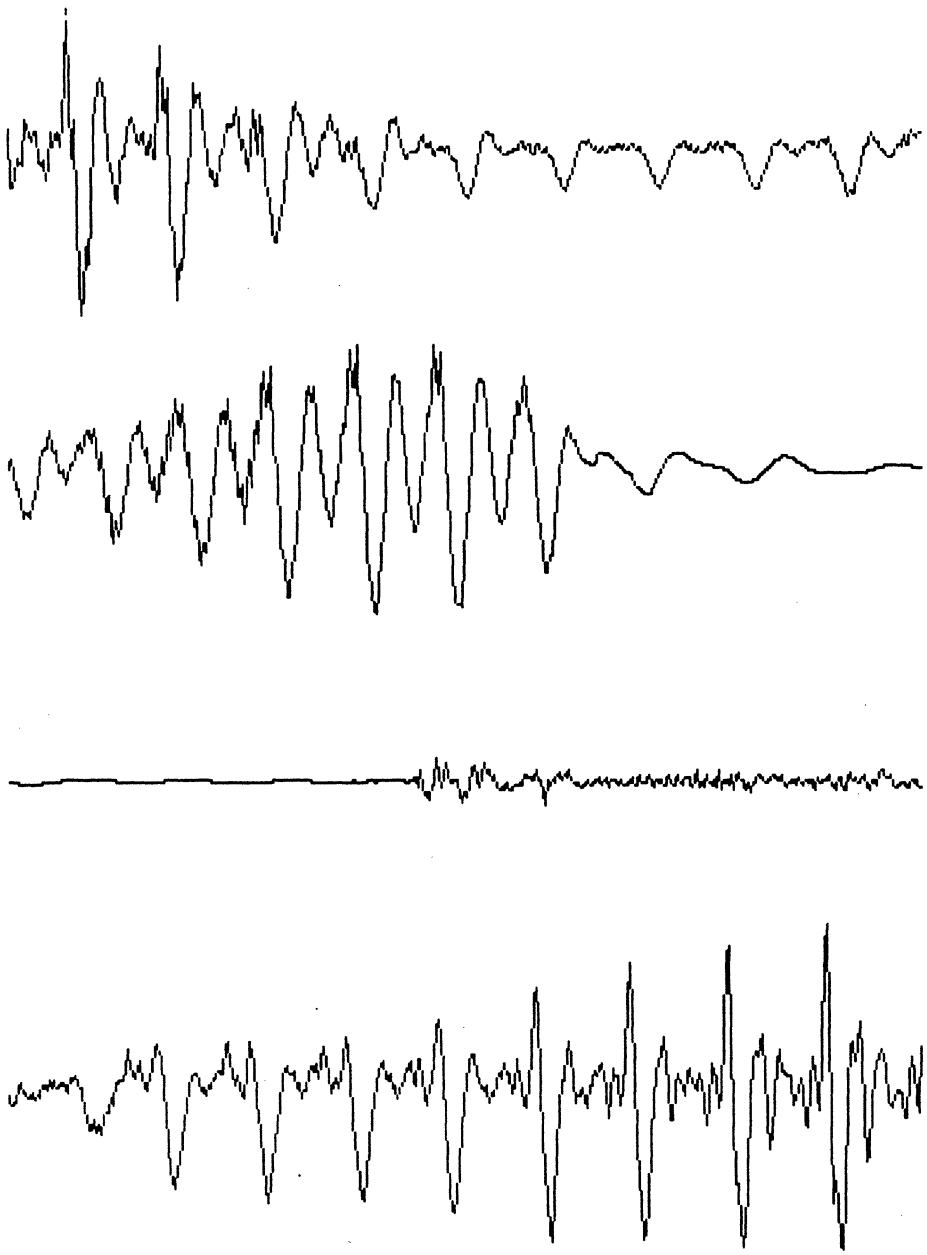


Figure 4.2 Original speech waveform (12 bit linear PCM). The figure contains 384 speech samples per line. The sampling rate was 6500 Hz. The third line is characteristic of unvoiced speech, the others are more typical of voiced speech.

of using the Viterbi Algorithm grow exponentially with the constraint length of the decoder. While a very expensive encoding algorithm could be tolerated during the *design* of a trellis code – since the design is done off line and need not be completed in real time – section 3.1.1 also notes that a trellis code designed using one encoding algorithm may not work well when used with another algorithm. Table 3.3 indicates that the design algorithm adjusts the codebook not only to the characteristics of

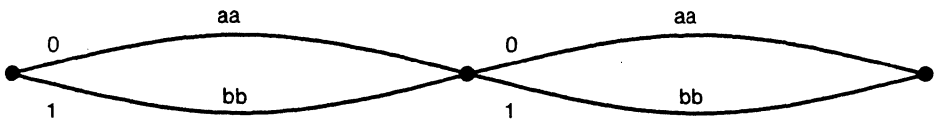
the training sequence, but to the characteristics of the encoding algorithm as well. In view of these results, the speech coding systems discussed in this chapter all use the M,L Algorithm for both code design and for code testing outside the training sequence. In all cases, the algorithm parameters were set to $M = 20$ and $L = 31$. The search algorithm thus maintained 20 path sequences as possible encodings, and incorporated an encoding delay of 31 symbols. It is widely held [28],[77] that less intensive search is fully adequate to achieve most of the benefits of delayed decision encoding, but the primary purpose of the work described here is the *design* of speech codes, rather than a search for the most efficient encoding algorithm. (Indeed, other encoding algorithms, such as the depth or metric first types [4], are usually less expensive even than the M,L Algorithm.)

To facilitate comparison of the codes described later, all code designs were made using the same initial codebooks and the same design algorithm parameters. All the codes discussed are shift register codes designed using extension from small initial codebooks. At each constraint length, the design algorithm was run for six iterations or until the change in signal to noise ratio measured in decibels fell below one percent. Section 4.2.4 will describe the initial codebooks.

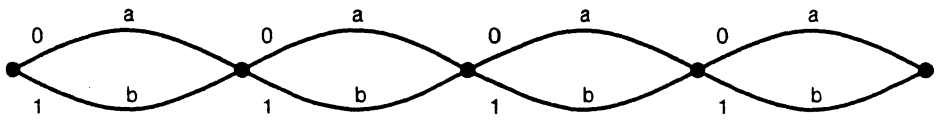
4.2.3 Code structures

Each of the speech coding systems described in this chapter – waveform coding, RELP coding, and hybrid coding – includes a trellis code. In each case trellis decoders were designed for a variety of constraint lengths and for data rates of 1/2 bit per speech sample, 1 bit per sample, and 2 bits per sample. At rate 1 bit per sample, both the rate 1/1 and rate 2/2 code types described in section 3.4.1 were tested. Because of this variety, the relationships among code rate, constraint length, and codebook size become quite complex. Figures 4.3 and 4.4 and Table 4.1 are included here both to try and organize this complexity and for later reference. Figure 4.3 is essentially the same as Figure 3.11. It shows the trellis structures for degenerate (one state) decoders of each of the types used in the speech coding work. Since these constraint length 1 trellis structures have only one state, – they do not remember any information – they are equivalent to block codes. Figure 4.4 shows schematic diagrams of longer constraint length codes of each type. The codes of Figure 4.4 a) and b), operating at 1/2 bit per sample (or per source symbol) and 1 bit per sample, respectively, accept one bit from the channel and produce a codeword composed either of two reproduction values in the case of the 1/2 bit per sample code, or of one reproduction value. The codes of Figure 4.4 c) and d), operating at 1 and 2 bits per source symbol, accept two bits at a time from the channel and

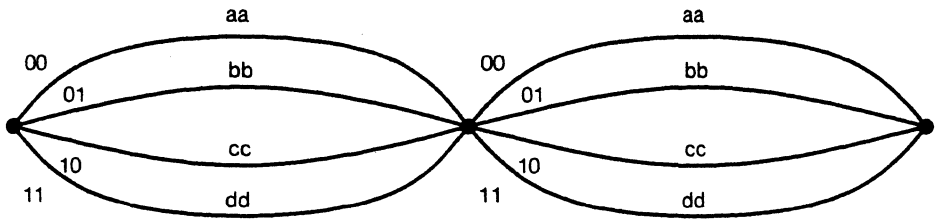
a) 1/2 code -- 1/2 bit/sample, 2 way branching, 2 samples/branch



b) 1/1 code -- 1 bit/sample, 2 way branching, 1 sample/branch



c) 2/2 code -- 1 bit/sample, 4 way branching, 2 samples/branch



d) 2/1 code -- 2 bits/sample, 4 way branching, 1 sample/branch

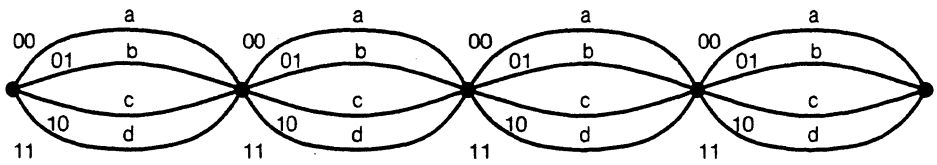


Figure 4.3 Alternative code structures for shift register decoders. All codes shown are for the degenerate constraint-length 1 case.

produce codewords composed either of two or of one reproduction symbol. Table 4.1 relates these code types in terms of their codebook sizes and constraint lengths measured in both source samples and channel symbols. The first column of the table, N, is the number of reproduction values in the codebook – the size of the codebook measured in samples. The second column, lg N, is the base two logarithm of this number. (In the rest of this chapter, different codes will be compared for equivalent N.) The columns labelled k_c refer to code constraint lengths measured in channel symbols (which may consist of either one or two bits), the columns labelled k_s refer to code constraint lengths measured in speech samples, and the columns labelled N_c refer to the number of codewords (where a codeword may consist of either one or two reproduction values).

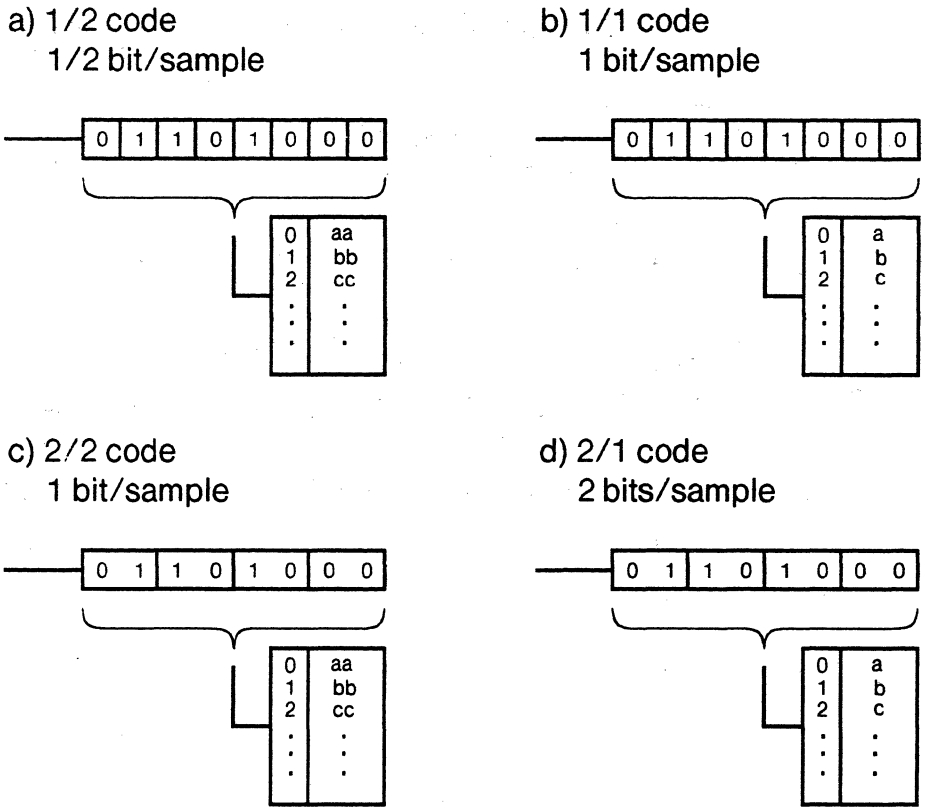


Figure 4.4 Shift register and table-lookup representations for alternative shift register code structures

4.2.4 Initial codebooks

The initial codebook selected for the extension method is not too critical, since the constraint length 1 design is equivalent to the block code case. All the code design experiments described later used the same initial codebooks:

Rate 1/2, 2 codewords: {(1e—4,—7e—4),(4e—4,—4e—4)}

Rate 1/1, 2 codewords: {1e—3,—1e—3}

Rate 2/2, 4 codewords: {(1e—5,1e—5),(—1e—5,—1e—5),(1e—5,—1e—5),(—1e—5,1e—5)}

Rate 2/1, 4 codewords: {1e—4,—7e—4,4e—4,—4e—4}.

4.3 Speech Waveform Trellis Codes

The most straightforward way to apply trellis encoding to speech is simply to build

Table 4.1
Codebook size *vs.* constraint length

		1/2 code			1/1 code		
N	lgN	k_c	k_s	N_c	k_c	k_s	N_c
2	1	-	-	-	1	1	2
4	2	1	2	2	2	2	4
8	3	2	4	8	3	3	8
16	4	3	6	8	4	4	16
32	5	4	8	16	5	5	32
64	6	5	10	32	6	6	64
128	7	6	12	64	7	7	128
256	8	7	14	128	8	8	256
512	9	8	16	256	9	9	512
1024	10	9	18	512	10	10	1024
2048	11	10	20	1024	11	11	2048
4096	12	11	22	2048	12	12	4096

		2/2 code			2/1 code		
N	lgN	k_c	k_s	N_c	k_c	k_s	N_c
2	1	-	-	-	-	-	-
4	2	-	-	-	1	1	4
8	3	1	2	4	-	-	-
16	4	-	-	-	2	2	16
32	5	2	4	16	-	-	-
64	6	-	-	-	3	3	64
128	7	3	6	64	-	-	-
256	8	-	-	-	4	4	256
512	9	4	8	256	-	-	-
1024	10	-	-	-	5	5	1024
2048	11	5	10	1024	-	-	-

a trellis decoder for the original speech waveform. A block diagram of this system is shown in Figure 4.5. The trellis decoders used were all designed by extension, using the codebook design algorithm. As described in the previous section, systems operating at 1/2 bit per speech sample, 1 bit per speech sample and 2 bits per speech sample were constructed. At rate 1 bit per sample, both the 1/1 and 2/2 trellis code variants were tried. Since the original speech data were sampled at 6500 Hertz, 3250 bits/second were required for the rate 1/2 code, 6500 bits/second were required for each of the rate 1 codes, and 13,000 bits/second were required for the 2 bits per sample code.

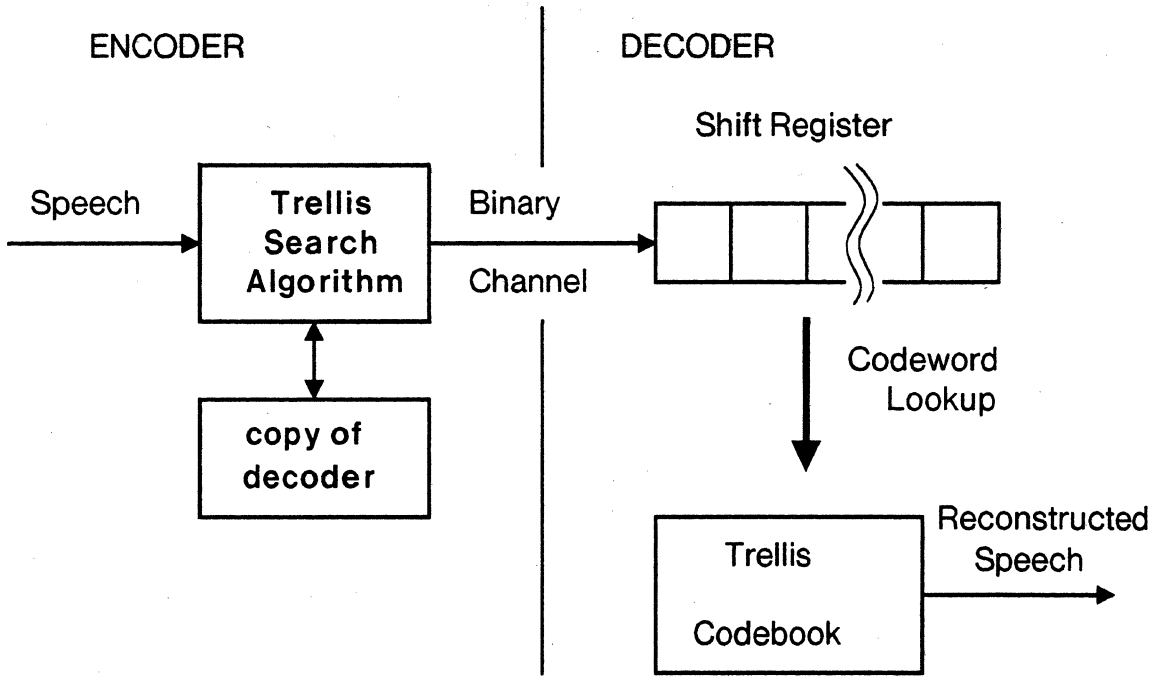


Figure 4.5 *Speech waveform trellis encoding system*

4.3.1 Speech codes using mean squared error

Trellis coding of speech waveforms has been tried before, particularly plagiarized decoders and multiple path searched linear filters [10],[17],[29]. The work of Anderson and Law [10], mentioned in section 4.1, is interesting because it represents a variational approach to the design of linear trellis decoders. The class of table-lookup constraint-length k shift register trellis decoders, such as are created by the trellis code design algorithm, includes as a special case the class of all linear time invariant finite impulse response digital filters with impulse response duration less than k. In principle, even if the extension method should fail to produce good decoders, any of the previously used decoders could only be improved by the algorithm.

Assuming that the M,L Algorithm, or another algorithm whose complexity does not depend on code constraint length, is used as the encoder in a trellis system, the chief implementation cost of a sufficiently large decoder lies almost entirely in the table-lookup codebook. For this reason, we take the position that the proper comparison between competitive codes should be made between codes of equal codebook size, rather than between codes of equal constraint length. This situation in particular arises with the 1/1 and 2/2 trellis codes, both of which operate at one bit per speech sample.

Table 4.2
Speech waveform trellis codes
Table entries in dB SNR

	1/2 Code		1/1 Code		2/2 Code		2/1 Code	
lg N	Inside	Outside	Inside	Outside	Inside	Outside	Inside	Outside
1	1.96	1.85	2.12	2.06	-	-	-	-
2	2.57	2.28	2.91	2.74	-	-	6.04	6.30
3	3.28	3.23	3.29	3.01	4.81	4.93	-	-
4	5.28	4.75	6.70	6.41	-	-	9.44	9.50
5	6.20	5.30	8.23	7.35	7.50	7.09	-	-
6	7.00	5.69	9.08	7.98	-	-	12.45	12.30
7	7.73	6.07	9.56	8.29	9.38	8.39	-	-
8	8.35	6.27	10.16	8.67	-	-	14.25	13.40
9	9.06	6.09	10.83	8.74	10.77	8.96	-	-
10	9.64	5.86	11.10	8.78	-	-	15.97	13.47
11	9.93	5.41	12.05	8.54	12.17	8.70	-	-

Table 4.2 contains signal to noise ratios in dB for the various speech waveform trellis codes designed. Figures are included both for the training data itself (*Inside*), and for the test sequence (*Outside*). Figure 4.6 presents the same information. Both the table and figure present their information as a function of the base two logarithm of the size of the associated codebook. This value is equivalent to the number of address bits required by a memory implementing the codebook. The results for the 1/1 code and the 2/2 code are particularly interesting because they both represent the same overall bit rate. As is obvious from Figure 4.6, the 1/1 and 2/2 codes produce equivalent results for equivalent codebook size, although such codes have different constraint lengths.

Only the available training data, the amount of memory available to the code design computer program, and the amount of time one is willing to spend really limit the constraint lengths of these kinds of decoders. As can be seen from the decrease in performance of both the rate 1/2 code and the rate 1 codes for large codebooks outside the training sequence, probably the 76,800 sample training sequence is of insufficient length for the design of codebooks with more than about 512 codewords. Audibly, as might be expected, the 3250 bit per second system is very noisy, although it is intelligible. The rate 1 systems are an improvement, although still quite noisy, and the rate 2 systems, at 13,000 bits per second are still sufficiently noisy to be classed as "communications quality" – not up to the standards of long distance telephony. The reproduced speech contains only the kind of quantization noise typical of other waveform coding systems, it does not contain the speech related artifacts typical of of LPC or other speech coders.

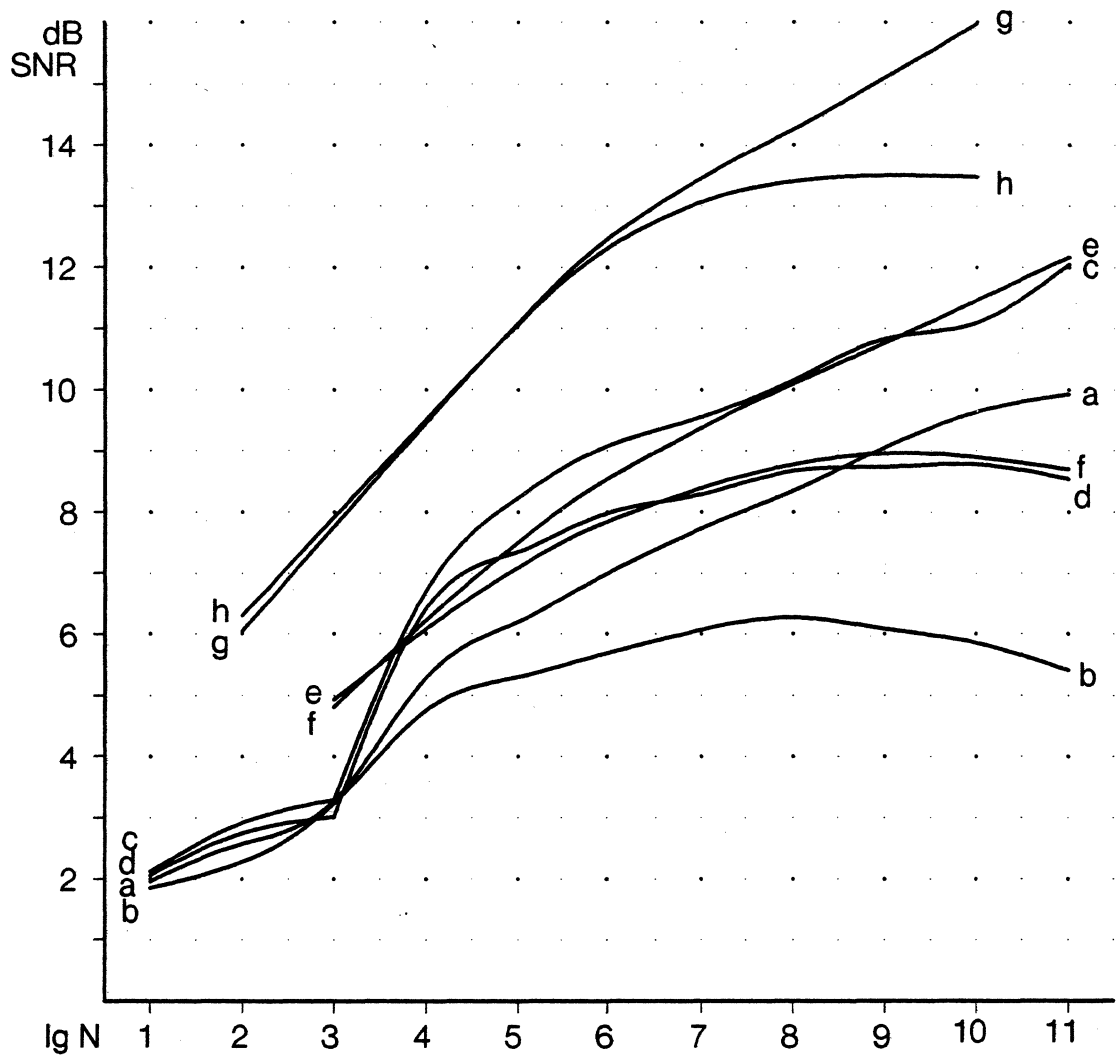


Figure 4.6 *Speech waveform trellis code performance. Curves* **a** *and* **b** *are for the* 1/2 *code (*1/2 *bit per sample) inside and outside the training sequence, respectively. Curves* **c** *and* **d** *are for the* 1/1 *code, curves* **e** *and* **f** *are for the* 2/2 *code (both 1 bit per sample), and curves* **g** *and* **h** *are for the* 2/1 *code (2 bits per speech sample).*

4.3.2 Frequency weighted error measures

While no attempts have yet been made, the directions to take in trying to improve the performance of trellis encoding systems operating on the original speech waveform are fairly clear. The existence of auditory masking is well understood, so the replacement of the squared error distortion measure used above by one which to a greater degree matches the frequency sensitivity of the ear is an obvious step. Use of a frequency weighted squared error measure by a tree or trellis encoding algorithm is straightforward, but its effect on the code design algorithm is not.

Wilson has incorporated frequency weighted squared error measures in a plagiarized decoder tree coding system by using a short length finite impulse response filter to weight the error sequences associated with possible encodings of the source [82],[83]. Associated with each path sequence in the code tree there is an error sequence composed of the differences between the decoder output sequence along that path and the sequence of source samples. By accumulating the sum of the squares of values of the filtered error sequence, a frequency weighted estimate is made of the error signal power along the path. These estimates are then used by the encoding algorithm as a basis for retaining or discarding the potential encodings. The estimate is not entirely accurate because it ignores effects that the last step in the path may have on the encoding of future samples – but this effect is little different than in the single symbol distortion measure case, as none of the efficient tree search algorithms look infinitely far ahead.

The effect that the use of such a weighted error measure might have on the trellis code design algorithm is more difficult to understand. The codebook update step of the algorithm (section 2.2.4) requires that the new codeword for a particular codebook entry have the value minimizing the distortion over those samples of the training sequence which were encoded by the corresponding old codeword. When the distortion measure is not a single symbol distortion measure, this minimization is no longer straightforward. It is true that the centroid of a *sequence* of values is independent of any normalized frequency weighting, since the centroid (for squared error at least) is just the Fourier transform of the sequence evaluated at zero frequency. However, the elements of a partition cell of the design algorithm are not sequentially drawn from the training sequence. It may be possible to perform a frequency weighted version of the update algorithm, but as yet no method is known.

It is entirely possible to design a trellis code using a frequency weighted error measure, but use the simple average for codeword update. If the frequency weighting is not too non-uniform, the results should be quite similar. In any event, frequency weighting during eventual operation of a trellis waveform code may produce better perceptual results.

4.4 Linear Predictive Coding and Vector Quantization

Linear predictive coding of speech by systems such as the one shown in Figure 4.7 has been very successful [60],[62],[71]. The basic principle of the method is a decomposition of the speech signal into an *excitation function*, e.g., the vocal cords, and an *all-pole linear filter* model of the vocal tract. LPC systems operate by breaking up the speech signal into segments or frames, estimating the filter and

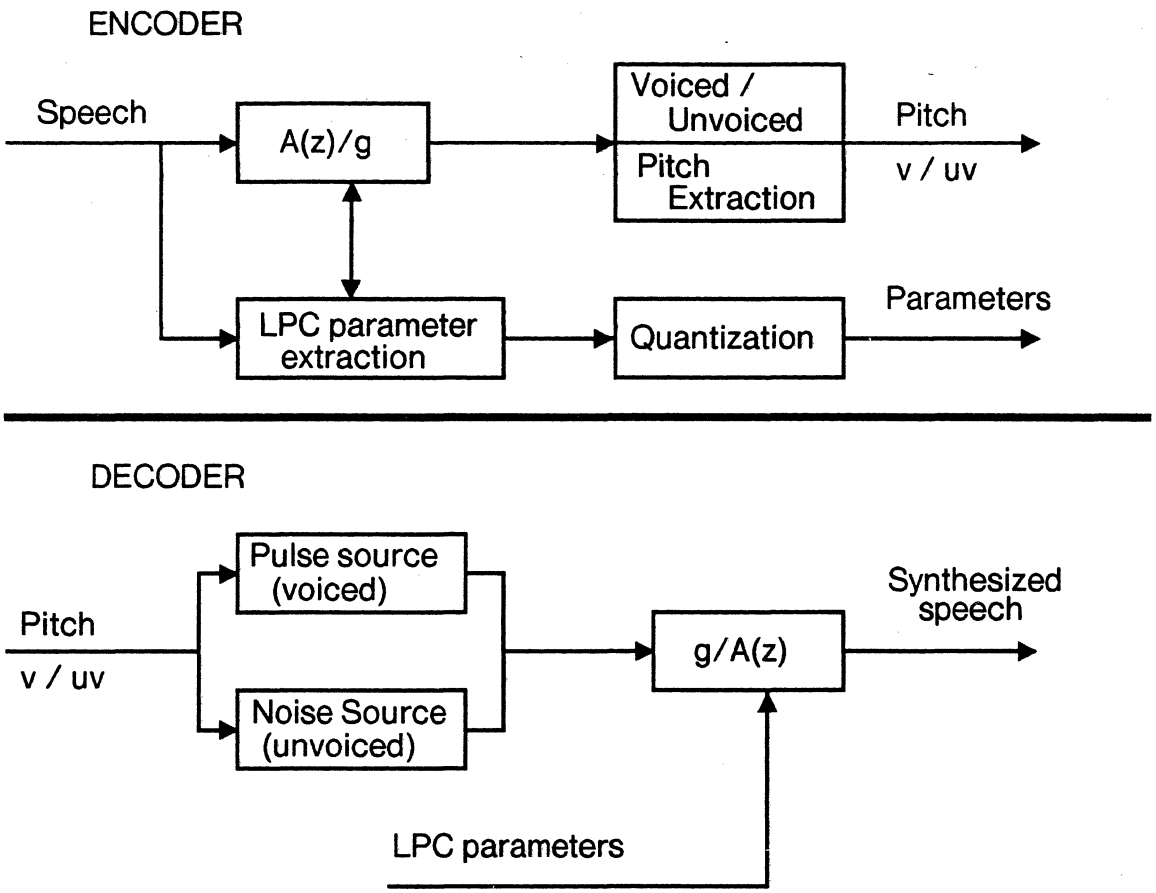


Figure 4.7 Traditional Linear Predictive Coding (LPC) system for speech

excitation parameters for each frame, and transmitting quantized versions of the parameters to the decoder, which uses them to operate a synthesizer. The length of a speech segment, typically 10 to 50 milliseconds, is chosen so that the model parameters remain fairly constant over the segment. While there are many methods for estimating the filter parameters, in one way or another all seek to minimize the power of the error or residual signal [37]. The speech residuals, the result of passing the original speech waveform through the LPC analysis filter $A(z)$ or $A(z)/\sigma$, play a role in the various methods for estimating the exitation function and, in some systems, are themselves quantized and transmitted.

The parameters transmitted by an LPC system include the average power σ, of the residual signal over the frame (the *gain*), the parameters describing the filter $1/A(z)$ (the *model*), whether the frame contains voiced or unvoiced speech (v/uv), and, if voiced, the pitch. (Since the selection of the filter affects the power of the residuals, LPC filters are usually restricted, so that the polynomial representation is *monic* – $a_0 = 1$.) There are many equivalent ways to describe the filter, including

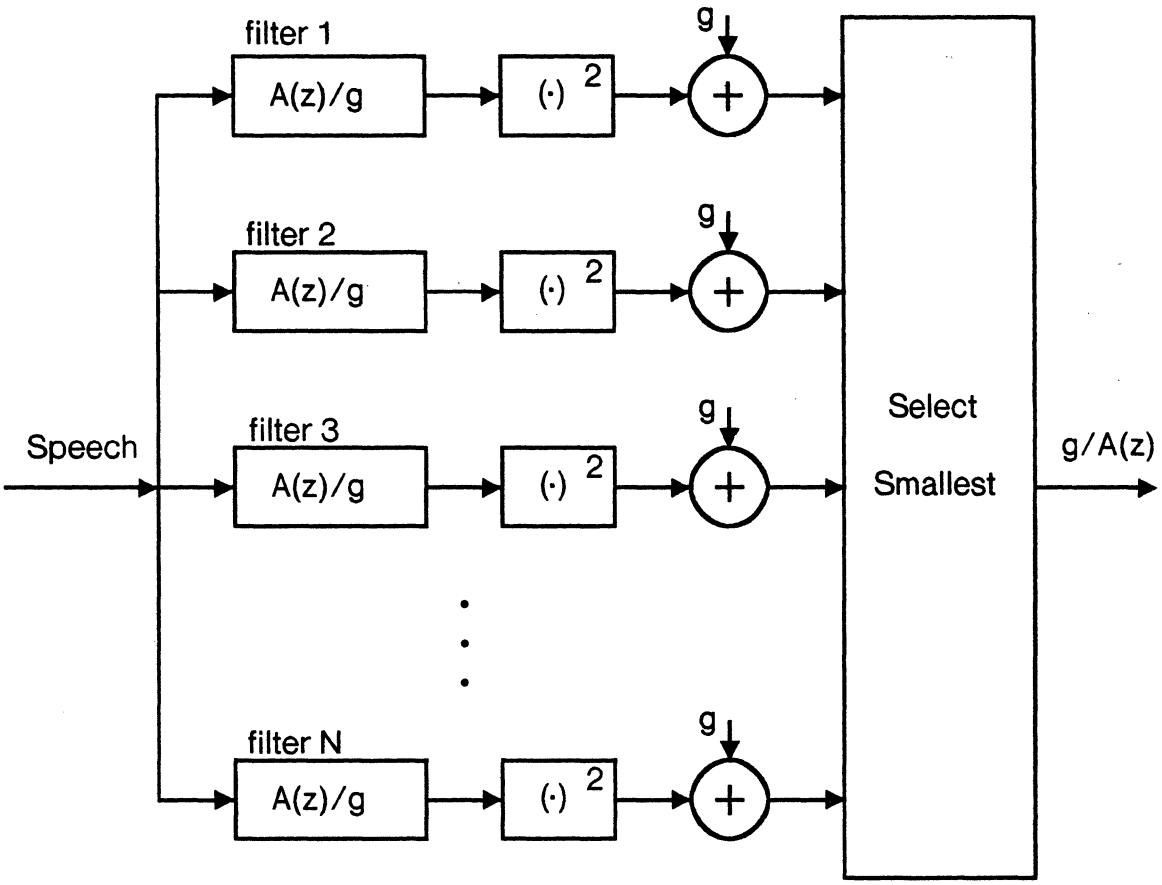


Figure 4.8 Vector quantization of LPC

the speech autocorrelation coefficients (the r's), the filter prediction coefficients (the a's), and the *reflection coefficients* of a ladder form filter implementation (the k's). The reflection coefficients are of particular interest, because $|k_i| < 1$ are necessary and sufficient conditions that the filter $1/A(z)$ be stable.

There has been much work done on the best way to encode the LPC parameters for transmission [34],[36]. Recent research into applications of the block quantizer design algorithm have led to vector quantization methods of encoding the gain and model parameters simultaneously [18],[85],[73]. This method is illustrated by Figure 4.8. During the design phase of the vector quantizer, a finite collection $\{\sigma_i/A_i(z), i = 1, \ldots, N\}$ of gain/model combinations typical of speech are selected by using the block code design algorithm together with a distortion measure between the waveform segment and the filter coefficients. During operation, for each frame a parallel test of each filter is made in search for the one giving the minimum residual energy. The *index* of the filter is then transmitted. This approach also avoids debate about the most appropriate filter representation, as the coefficients can be stored to arbitrary precision at the receiver. While maintaining a fixed rate constraint on

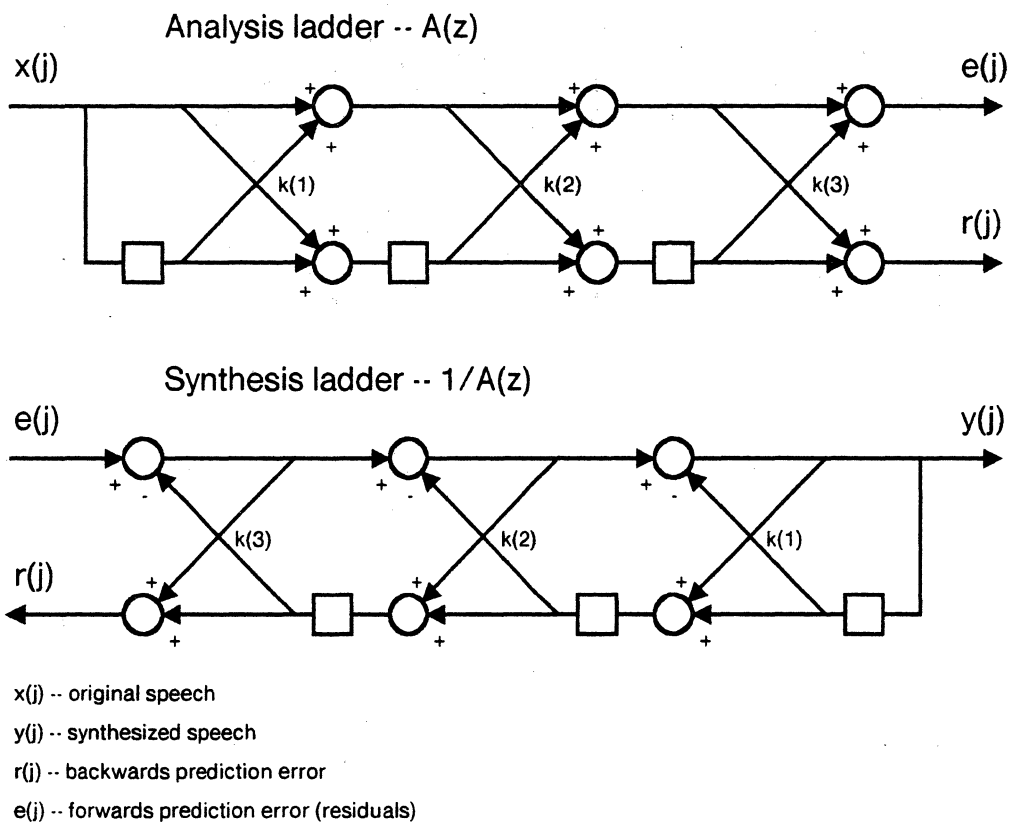


Figure 4.9 Ladder form analysis and synthesis filters for LPC systems

transmission of the gain and model parameters, the vector quantization approach offers a means of avoiding the introduction of unknown distortions in the filter which might occur through independent quantization of the parameters.

Because the original speech signal is broken into frames which are independently analyzed, much attention has been paid to ensuring a smooth transition across frame boundaries where the coefficients may change. Two approaches are to filter the coefficients, or to interpolate them in step with the pitch pulses of voiced speech. In the event a ladder form or lattice filter (Figure 4.9) is used at the synthesizer, another approach is to reduce the filter to a first order predictor at frame boundaries when the coefficients change, and then build up the order of the lattice filter stage by stage until the desired order is reached. This procedure avoids the non-stationarities in the filter output which might occur if the reflection coefficients were changed without consideration of the contents of the filter delay elements [51].

Historically, the most difficult part of LPC encoding has been the estimation of the exitation parameters, which determine whether a speech frame is voiced or unvoiced and, if voiced, determine the pitch. While there are a plethora of methods,

such as peak picking, cepstral analysis, and likelihood ratios [62],[67],[68] none are completely satisfactory. Part of the difficulty is that there are some speech sounds, the *voiced fricatives*, like 'v', that are partly voiced and partly unvoiced, and part is due to the desire that a speech coding system continue to operate in a reasonable manner in the presence of non-speech background noise.

In response to the problems of estimating the excitation part of the LPC decomposition, several systems have been proposed which avoid the issue by encoding and transmitting either the actual LPC residual signal or some associated signal. Adaptive predictive coding (APC), the voice excited vocoder (VEV), and Residual excited LPC (RELP) systems all fall into this category [11],[12],[33],[79]. The major difficulty in each of these systems lies in the encoding of the spectrally flat residual signal while retaining a low transmission rate. A typical method used is to bandlimit the residuals to perhaps 1000 Hz and then to use some form of adaptive PCM to transmit the resulting signal. At the receiver, a nonlinearity is used to restore some energy to frequency regions above 1000 Hz and the resulting signal is used to excite a standard LPC synthesis filter. Tree and trellis encoding offers the potential of transmitting the residual signal at low rates *without* resorting to these downsampling and spectral extension techniques.

Section 4.5 will discuss the application of trellis encoding to RELP systems, and section 4.6 will describe a hybrid coding system which combines characteristics of APC and universal coding.

4.5 LPC with Trellis Encoded Residuals

Transmission of a coded form of the LPC residuals, combined with the model or model and gain portions of an LPC system, should have several advantages. First, since standard LPC systems quite successfully use either white noise or an impulse train as excitation for the synthesis filter, it is evident that errors in the details of the residual waveform or synthesizer driving process may not result in much perceptual deterioration of the speech at the synthesizer output. Second, because the LPC model filter $1/A(z)$ applies a frequency weighting according to the short-term spectral characteristics of speech, residual coding takes advantage of the auditory masking effect. If the residual coding technique results in a white error spectrum, then, at the synthesizer output, the spectrum of the error signal will match the spectrum of the speech signal. Finally, coding of the actual residuals should tend to reduce the effects of failures of the voiced/unvoiced model and the effects of background noise. Since a residual excited system depends less on the

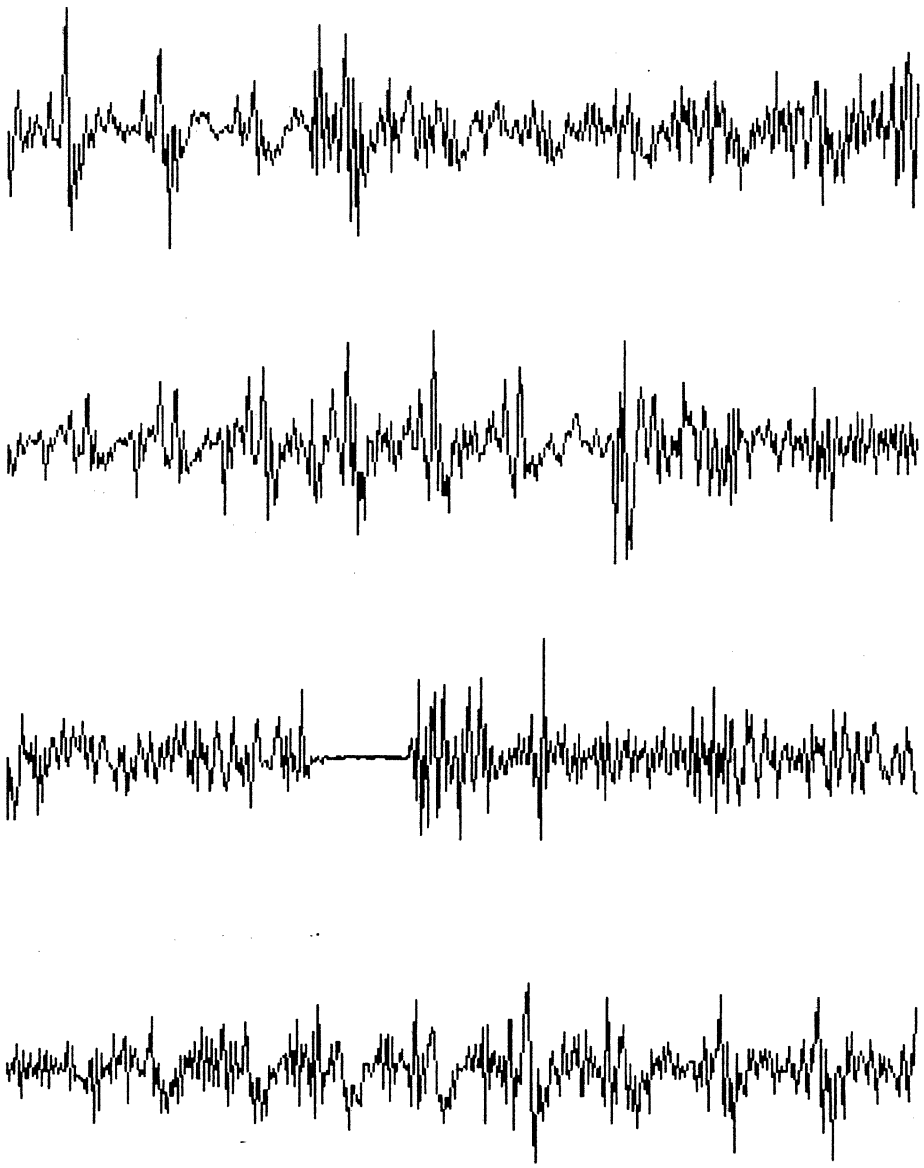


Figure 4.10 *Speech residuals generated by passing the original speech waveform through the filter $A(z)/\sigma$. This figure shows the same section of data as does Figure 4.2. Each line contains 384 samples, or 3 LPC frames.*

accuracy of the voice excitation model used by LPC, it should be able to better handle non-speech sounds.

The appearance of speech residuals differs greatly from that of the original waveform. Figure 4.10 shows the 12 frames of LPC residuals corresponding to the segment of speech shown in Figure 4.2.

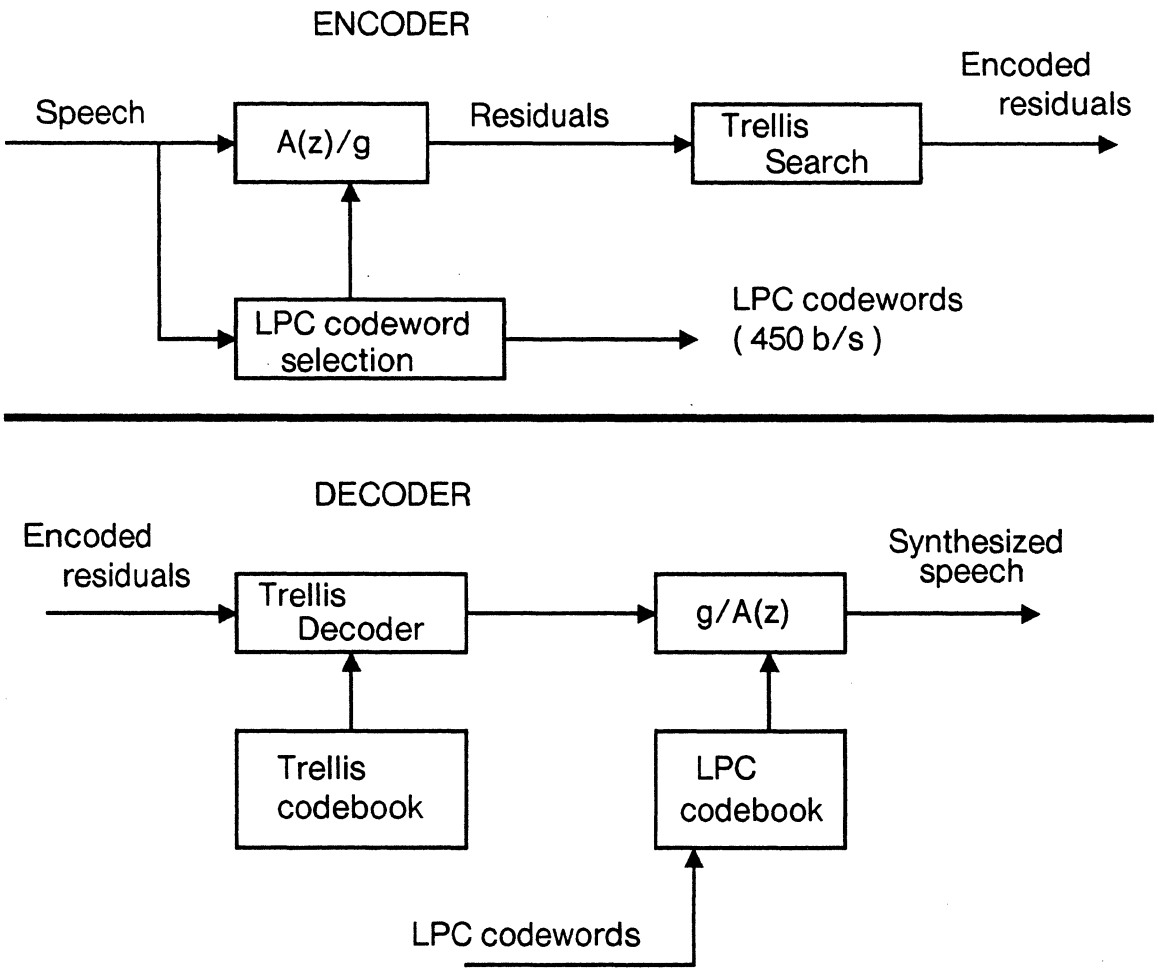


Figure 4.11 *Trellis encoding of speech residuals – Trellis RELP*

4.5.1 RELP coding results

In order to investigate the applicability of trellis coding methods to RELP systems, trellis decoders operating at 1/2 bit per residual sample (one bit for every two residual samples), 1 bit per sample, and 2 bits per sample were designed using the methods described in section 4.2. Figure 4.11 contains a block diagram of these systems. For these experiments, a vector quantized LPC system was used, operating with a codebook of 512 tenth order filters for a rate of 9 bits per LPC frame or about 450 bits per second. The LPC system supplied a combined gain and model, $\sigma/A(z)$, so that the residual signal, generated by passing the original speech through the filter $A(z)/\sigma$, was normalized by the LPC gain. In order to avoid filter state difficulties, a ladder form LPC filter implementation was used with reduction to order 1 at frame boundaries. This technique does result in a transient in the residual signal at frame boundaries, due to the poor matching of the prediction

Table 4.3
Speech residual trellis codes
Table entries in dB SNR for the residuals

	1/2 Code		1/1 Code		2/2 Code		2/1 Code	
lg N	Inside	Outside	Inside	Outside	Inside	Outside	Inside	Outside
1	1.14	1.24	2.89	2.94	-	-	-	-
2	1.30	1.40	3.12	3.12	-	-	6.82	7.13
3	1.77	1.74	3.39	3.48	3.13	3.13	-	-
4	2.02	2.00	3.82	3.94	-	-	8.01	8.24
5	2.26	2.23	4.15	4.27	3.76	3.87	-	-
6	2.57	2.52	4.45	4.59	-	-	8.71	8.91
7	2.76	2.71	4.74	4.85	4.43	4.50	-	-
8	2.94	2.77	4.96	4.98	-	-	9.54	9.47
9	3.13	2.77	5.16	5.02	5.06	4.92	-	-
10	3.33	2.74	5.42	5.05	-	-	10.31	9.76
11	3.73	2.72	5.72	5.03	5.82	5.07	-	-

filter to the actual speech, but since the filter coefficients were not interpolated during the course of each frame, the prediction fit would tend to vary anyway. The LPC filter set was designed by G. Rebolledo using the block code design algorithm and the Itakura-Saito distortion measure [39],[72],[73].

Using 450 bits per second of side information for the gain and model parameters combined with an original sampling rate of 6500 Hz, trellis RELP speech coding systems were produced operating at 3700 bps, 6950 bps, and 13,450 bps for the rate 1/2, 1, and 2 bits per symbol cases.

As in the case of the waveform coding experiments, the trellis decoders were designed using the extension method, starting from constraint-length 1 decoders. The distortion measure used was the simple squared error measure. Table 4.3 and Figure 4.12 contain the signal to noise ratios in decibels achieved for the various encodings of the residual signal, both inside and outside the training data. Since the residuals are produced from the speech waveform by a filter that produces a generally flat spectrum, the signal to noise ratios obtained are below those for the original speech waveform codes of the previous section. They are perhaps better compared with the rate distortion limits for white Gaussian noise of 3 dB at 1/2 bit per symbol, 6 dB at 1 bit/symbol and 12 dB at 2 bits per symbol. (Of course the residuals are *not* white, as they still contain the speech pitch pulses and other information not modelled by the LPC filters.)

It may be somewhat surprising that the extension method of trellis code design should work on speech residuals. As was remarked in section 3.2.3, the extension

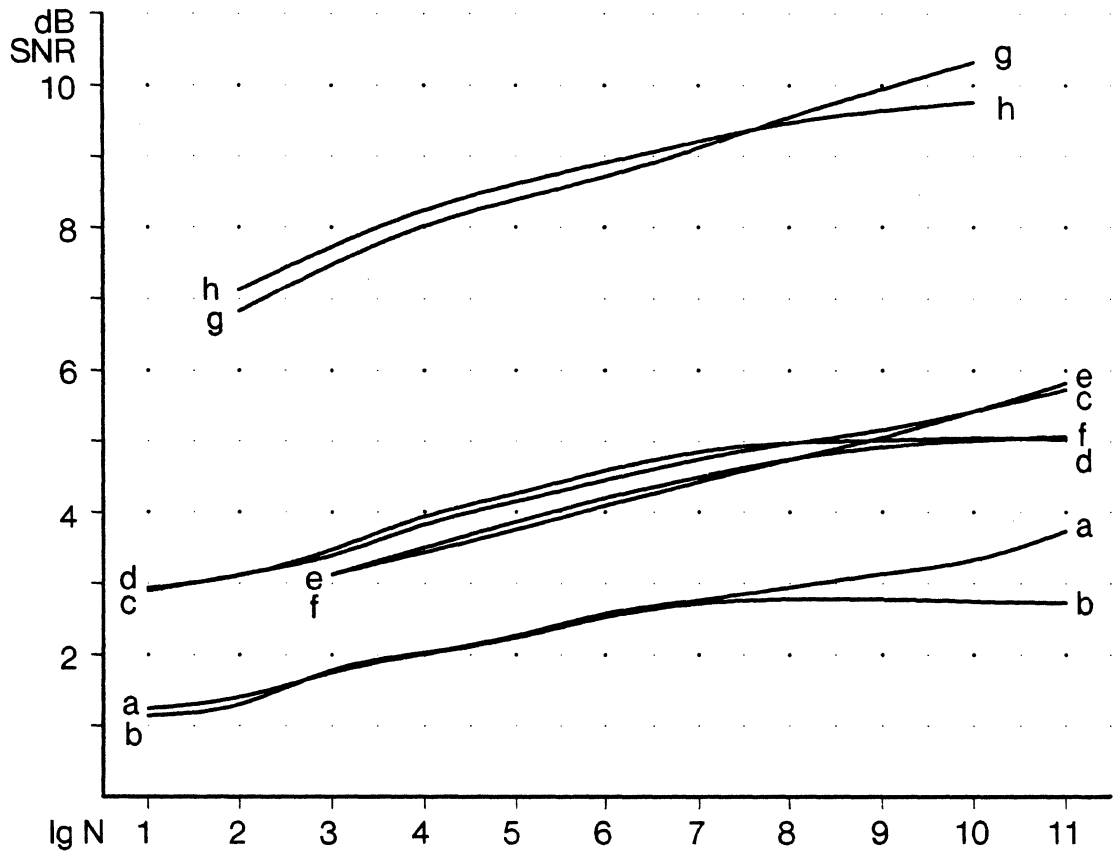


Figure 4.12 *Trellis RELP code performance. Figures are decibels signal-to-noise ratio for the residuals. Curves* **a** *and* **b** *are for the* 1/2 *code inside and outside the training sequence, respectively. Curves* **c** *and* **d** *are for the* 1/1 *code, curves* **e** *and* **f** *are for the* 2/2 *code, and curves* **g** *and* **h** *are for the* 2/1 *code.*

method does not work very well for memoryless sources and the LPC analysis filter $A(z)/\sigma$ acts to remove correlation from the speech waveform. There are several possible answers: first, the all-pole model of speech presumed by LPC does not adequately represent the memory inherent in the speech signal, second, use of the vector quantized LPC system supplies filters which do not exactly match the correlation due to the all-pole model, and third, the presence of pitch pulses in the residual signal provides some non-uniformities which may aid the trellis design algorithm in designing the decoder.

4.5.2 Gain normalized residuals

Many signals of interest, including speech, have a wide dynamic range – their short term power varies widely. In speech, this effect arises for two reasons. First, from

moment to moment, the general envelope of the speech signal rises and falls as the voice becomes louder or softer. Second, the driving process for voiced speech is of generally higher energy than the driving process for unvoiced speech. In traditional waveform coding systems, wide dynamic range is dealt with by such means as companding or log PCM, and at lower rates by adaptive step size in predictive coders [27]. In LPC systems, dynamic range effects are absorbed by the inclusion of the gain term σ.

The speech waveform trellis coding systems described in section 4.3 took no special efforts to handle a wide dynamic range. As a result, system performance was compromised by the requirement that paths through the trellis exist which were suitable both for waveform segments of small amplitude and for segments of relatively larger amplitude. In order to avoid this difficulty in the design of trellis codes for the speech residuals, the residuals were generated from the original speech by the combined gain and model LPC filter $A(z)/\sigma$ rather than by the spectral flattening filter $A(z)$ alone. Use of these *gain normalized residuals* allows the design algorithm to adapt the trellis code for a signal of fairly narrow dynamic range.

Use of the gain normalized residuals has an interaction with use of the mean squared error distortion measure. When mean squared error is used to encode the unnormalized residuals, the encoding algorithm places relatively more emphasis on reducing large errors than on reducing small ones. Since large errors are more likely to occur in regions of large signal amplitude, we see that mean squared error matches the notion that the ear is more sensitive to noise at higher amplitude levels than at low. Squared error used as a distortion measure for residuals thus takes advantage of both the auditory masking effect and the amplitude sensitivity effect. On the other hand, when mean squared error is used to encode the gain normalized residuals, equal emphasis is placed on errors in (ultimately) high amplitude speech segments and on errors in low amplitude segments. While gain normalization may permit the decoder to handle a smaller dynamic range, modifications to the distortion measure may also be in order. The simplest way to change the distortion measure is to incorporate the gain term:

$$d(x, y) = \sigma^2(x - y)^2.$$

This formulation restores the original squared error distortion on the unnormalized residuals while retaining the benefits of code design for a narrow dynamic range. Since there is some argument about the exact nature of the ear's sensitivity to signal to noise ratio as a function of amplitude and since σ represents the power of the residuals, rather than that of the original speech, we point out that any function of the gain, say $f(\sigma)$, can replace the σ^2 term.

A tree search encoder, rather than sequentially selecting the minimum distortion path symbol (single path search), looks ahead down several tree paths to a depth of several symbols, and selects the one having minimum *average* distortion. The possible encodings of a particular symbol are played off against the possible encodings of the preceding and succeeding symbols in order to choose one which is best overall. It is generally impossible to tell how far the effects of encoding a particular symbol in a certain way will propagate, but usually the effects are localized. As a rule of thumb, the encodings of symbols separated by much more than the constraint length of the decoder are unlikely to interact. This effect has an interesting interaction with the gain-modified distortion measures just discussed. Since the gain term, supplied by the LPC machinery, remains constant during the duration of a speech frame and since the frames are considerably longer (128 samples) than any constraint length used, we see that the addition of the gain term to the distortion measure has essentially no effect on the encoding process during the course of a frame. It is only at frame boundaries where the gain changes that the additional term has effect. Consider the case wherein a frame with little speech energy (small σ) follows a frame containing relatively more speech energy. As the encoding algorithm considers the samples within (more or less) a constraint length of the frame boundary, the gain-modified distortion measure will instruct it to devote relatively more effort to closely following the residual waveform on the large σ side of the boundary than on the small σ side.

Utilization of the gain-modified distortion measure $\sigma^2(x - y)^2$ also requires modifications to the code design algorithm. Recall that the codeword update portion of the algorithm consists of setting the updated version of a particular codeword to the value minimizing the distortion over the training sequence samples in the corresponding partition cell:

$$y_i^{m+1} = y : \sum_{j \in S_i^m} d(x_j, y) \leq \sum_{j \in S_i^m} d(x_j, y'); \quad \text{for all } y'.$$

For mean squared error, this centroid computation reduces to the arithmetic mean:

$$y_i^{m+1} = \frac{1}{\|S_i^m\|} \sum_{j \in S_i^m} x_j.$$

For the gain-modified squared error measure, the corresponding modification to the centroid computation is straightforward:

$$y_i^{m+1} = \frac{\sum_{j \in S_i^m} \sigma_j x_j}{\sum_{j \in S_i^m} \sigma_j},$$

Table 4.4
Gain-modified Trellis RELP
Table entries in dB Signal-to-Distortion (arbitrary scale)

	1/2 Code		1/1 Code		2/2 Code		2/1 Code	
lg N	Inside	Outside	Inside	Outside	Inside	Outside	Inside	Outside
1	1.36	-1.22	2.66	0.03	-	-	-	-
2	1.38	-1.22	2.79	0.14	-	-	6.90	4.59
3	1.88	-0.55	3.16	0.70	2.98	0.31	-	-
4	3.10	0.38	4.86	2.43	-	-	8.64	6.19
5	3.85	1.02	5.65	2.99	4.60	2.00	-	-
6	4.44	1.29	6.08	3.42	-	-	9.84	7.29
7	4.96	1.48	6.58	3.69	6.18	3.29	-	-
8	5.39	1.53	7.09	3.76	-	-	11.14	8.02
9	5.80	1.53	7.48	3.77	7.36	3.69	-	-
10	6.29	1.56	8.15	3.86	-	-	12.52	8.35
11	6.94	1.45	8.78	3.76	8.62	3.85	-	-

where σ_j is the gain associated with the LPC frame containing sample j. Not only does the modified distortion measure place more emphasis on encoding samples from high amplitude frames, but the code design algorithm now places more emphasis on adjusting the codewords to produce good results in the high gain frames. In spite of these complications, the original goal of using the gain normalized residuals is still met. The trellis codebook contains values appropriate for a signal with a fairly narrow dynamic range.

A sequence of code design experiments using the modified distortion measure and design algorithm were run, with somewhat encouraging results. Table 4.4 and Figure 4.13 contain ratios of signal power to distortion in decibels for the gain-modified tests. Since the distortion measure is unusual, these values cannot be directly compared with the previous RELP results, but the general trends of code performance are of some interest. While the characteristics of the resulting speech are very similar to those of the original RELP coded speech, the modified encodings are of better perceptual quality.

4.6 Hybrid Tree Codes

Reproduction of the original speech waveform is not a goal of standard LPC, nor is it a goal of the traditional residual excited LPC systems. It is well known that the ear is insensitive to phase information and LPC makes the further assumption that the ear is actually sensitive only to the short term spectrum of speech and to some aspects of the excitation process. By modelling the excitation process as a

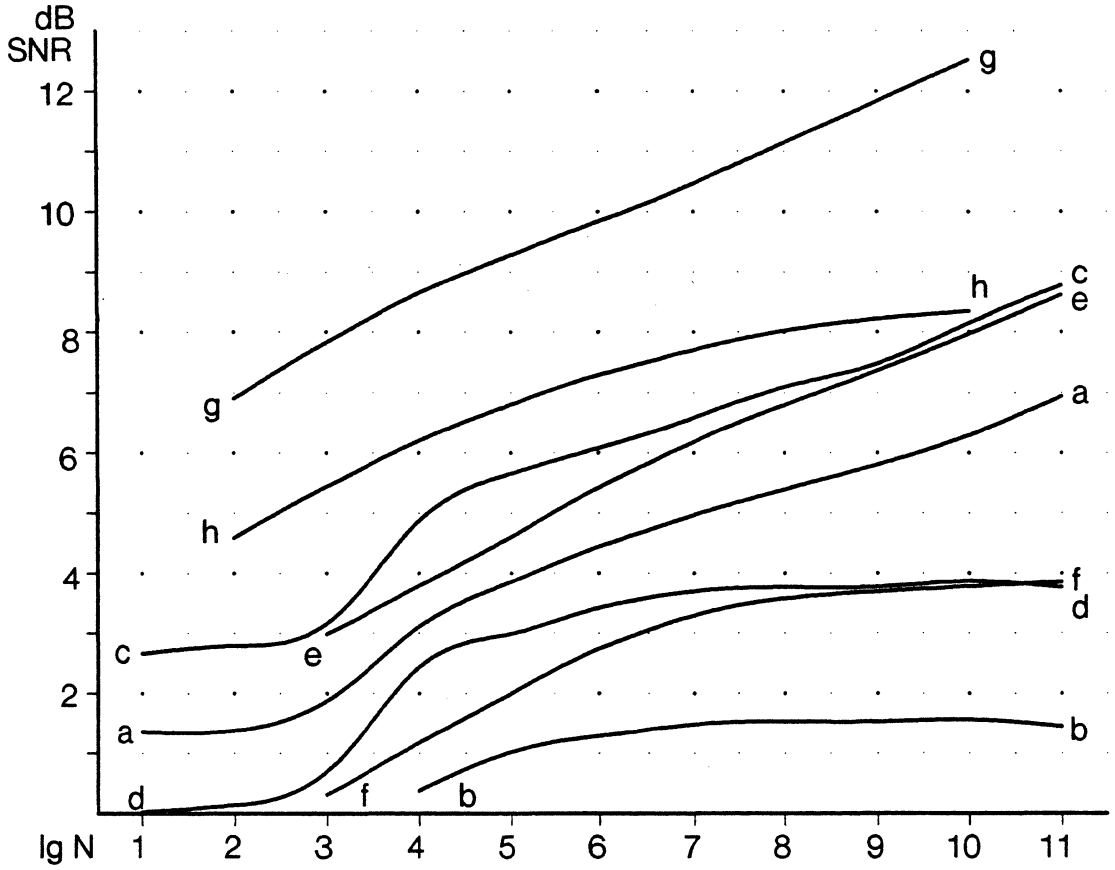


Figure 4.13 *Gain-modified trellis RELP code performance. Figures are decibels signal-to-distortion for the residuals, using an arbitrary scale. Curves* **a** *and* **b** *are for the* 1/2 *code, inside and outside the training sequence, respectively. Curves* **c** *and* **d** *are for the* 1/1 *code, curves* **e** *and* **f** *are for the* 2/2 *code, and curves* **g** *and* **h** *are for the* 2/1 *code.*

pulse train or noise source, LPC abandons the original waveform while retaining those characteristics of the signal thought to be important. The residual excited systems try to encode the actual residual signal, but, by using techniques such as downsampling and nonlinearities at the receiver, also lose track of the original waveform. The trellis coded RELP systems discussed in the previous section were no exception – the rate-constrained residual signal is reproduced according to the trellis code distortion measure without regard to the eventual waveform after the receiver passes the decoded residuals through the LPC synthesis filter $\sigma/A(z)$. It was with some surprise, therefore, that we learned that output waveform of the RELP system was in fact quite close to that of the original speech signal. Figure 4.14 contains three short waveform plots: the original speech, the trellis RELP 1 bit/symbol speech, and the difference between them. The signal to noise ratios for the waveforms produced by the various RELP systems are shown in Table 4.5.

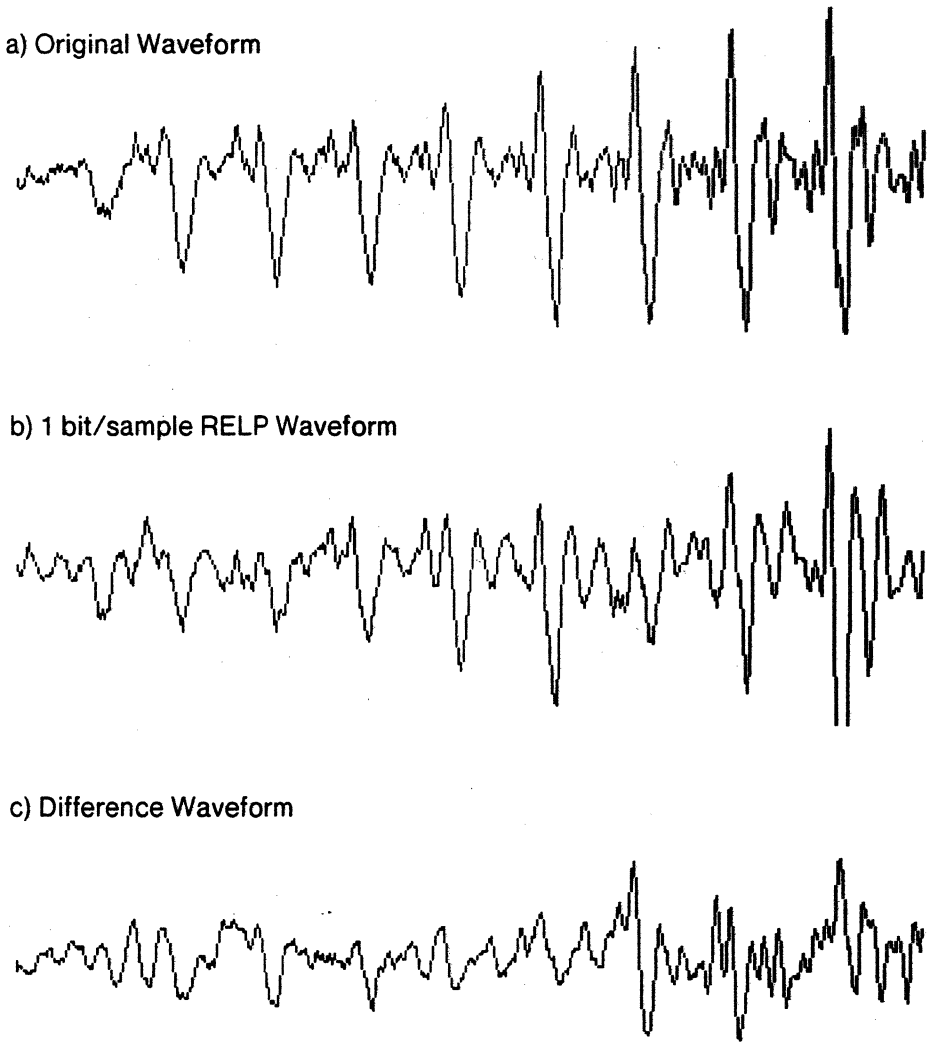


Figure 4.14 *a) Original speech b) Trellis RELP 1 bit/sample c) Difference waveform*

Table 4.5
Trellis RELP reproduction of original speech waveform
Table entries in dB SNR

Code	bit rate	Inside	Outside
1/2	3,700	5.15	2.46
1/1	6,950	6.64	4.25
2/2	6,950	6.20	4.29
2/1	13,450	9.94	8.50

As a result of this observation – that the trellis RELP *could* reproduce the original waveform – we decided to test a composite decoder consisting of the $\sigma/A(z)$ LPC synthesis filter combined with a trellis decoder front-end. This "Hybrid Tree

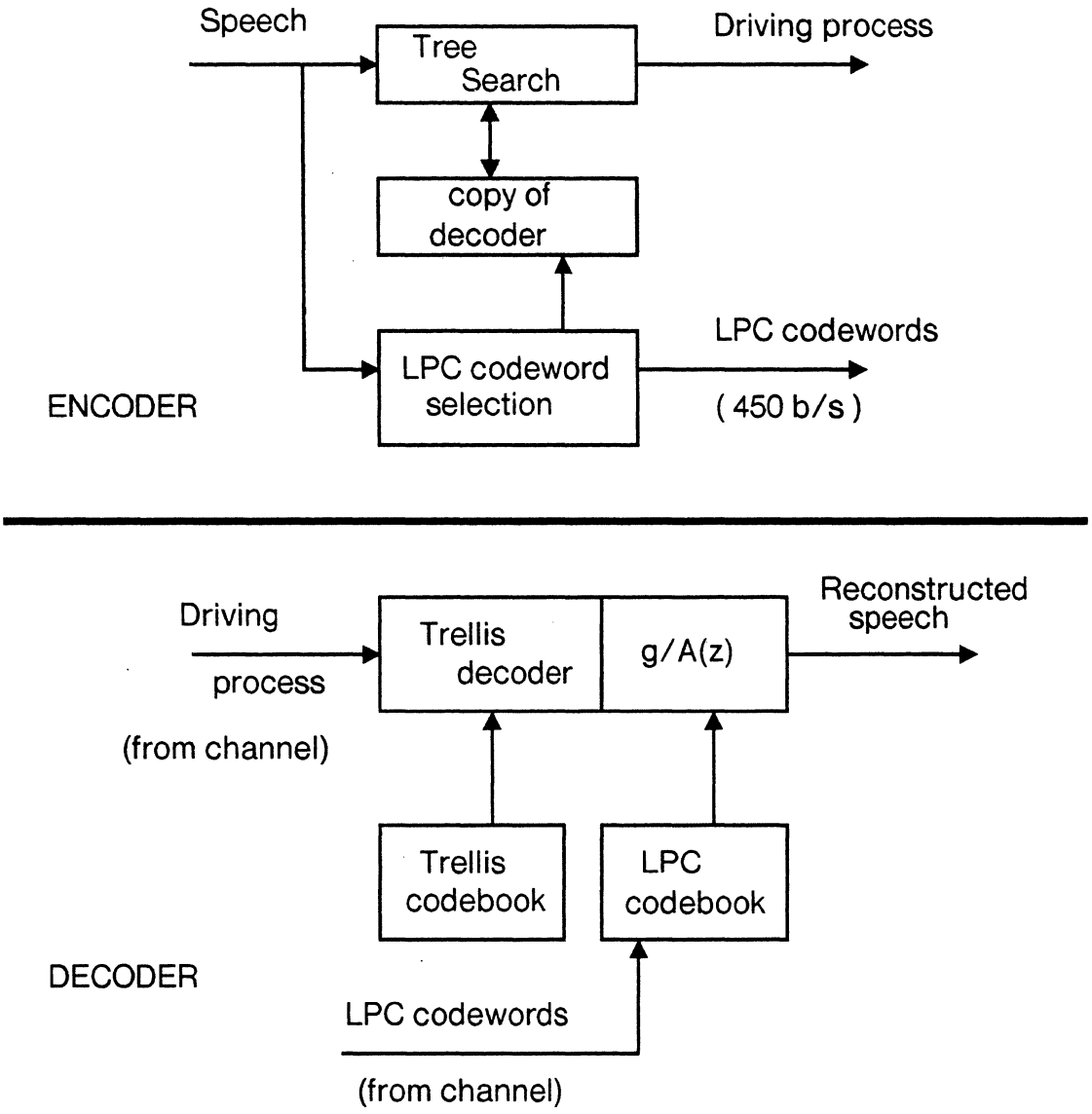


Figure 4.15 Hybrid tree coding system

Code" system is shown in Figure 4.15. It is interesting that the only difference between the RELP systems of the previous section and this new system is that the hybrid system seeks a channel sequence which will, at the decoder output, match the original speech waveform while the RELP system seeks a channel sequence which will, at the output of the trellis decoder, match the speech residuals.

This hybrid encoding system has elements in common with adaptive predictive coding. Hybrid systems with very similar structures have been proposed and, with some variations, used by Matsuyama and Gray [65],[64] and independently by Adoul et al. [1]. These systems will be discussed in section 4.6.1.

Table 4.6 and Figure 4.16 give performance data for hybrid tree encoding

Table 4.6
Hybrid tree codes
Table entries in dB SNR

	1/2 Code		1/1 Code		2/2 Code		2/1 Code	
lg N	Inside	Outside	Inside	Outside	Inside	Outside	Inside	Outside
1	6.16	5.19	8.77	7.61	-	-	-	-
2	4.97	4.28	6.83	5.72	-	-	16.62	14.44
3	4.96	4.35	5.25	4.95	7.36	6.21	-	-
4	6.50	4.92	6.68	6.60	-	-	17.29	16.16
5	6.89	5.27	8.11	7.87	9.55	8.38	-	-
6	7.48	5.74	9.58	9.01	-	-	18.35	16.65
7	7.87	6.37	11.23	9.64	11.32	9.80	-	-
8	8.41	6.51	11.90	9.97	-	-	19.15	17.01
9	8.34	6.44	12.31	9.96	12.57	10.68	-	-
10	8.70	6.50	12.43	10.11	-	-	19.54	17.45
11	8.57	6.56	12.57	10.17	13.03	10.85	-	-

systems operating at 1/2, 1, and 2 bits per sample for the driving process, plus 450 bits per second for the LPC portion of the system. As before, the table entries represent signal to noise ratio in decibels using a squared error distortion measure. The combined channel rates for these codes are 3700, 6950, or 13,450 bits/second. Although these rates are the same as for the RELP system, the hybrid tree system produces considerably better perceptual quality.

4.6.1 Hybrid tree code structure and performance

The hybrid tree system has elements in common with traditional adaptive predictive coders [11],[12],[33]. Figure 4.17 shows an encoder and decoder for an adaptive predictive coding (APC) system. The decoder consists of a time varying predictor driven by a variable step size channel decoder. The standard APC encoder consists of these same elements inside a feedback loop. The APC system is thus a predictive quantization system with a variable predictor and a variable step size quantizer. More recent APC systems have incorporated delayed decision by replacing the APC encoder by a tree search algorithm and still more closely resemble the systems of [1] and [81].

Referring to figures 4.15 and 4.17, we see that the LPC filter portion of the hybrid tree decoder plays the role of the adaptive predictor of the APC system and that the variable step size channel decoder of the APC system has been replaced by a time invariant trellis decoder together with the LPC gain term σ. As before, the LPC portion of the decoder consists of one of 512 tenth order filters, $\sigma_i/A_i(z)$,

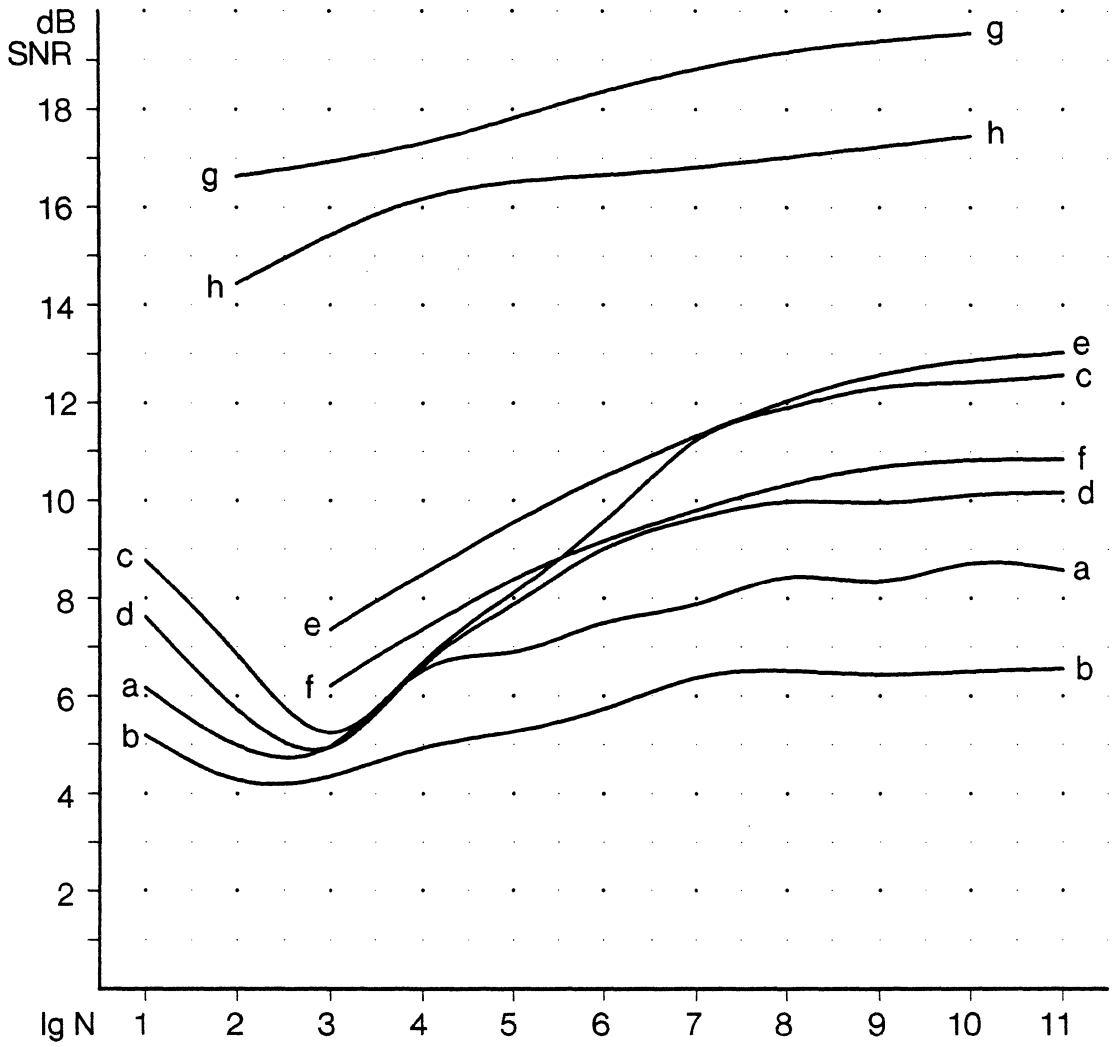


Figure 4.16 *Hybrid tree code performance. Performance figures are in dB SNR for the original speech waveform. Curves* **a** *and* **b** *are for the* $1/2$ *code, inside and outside the training sequence, respectively. Curves* **c** *and* **d** *are for the* $1/1$ *code, curves* **e** *and* **f** *are for the* $2/2$ *code, and curves* **g** *and* **h** *are for the* $2/1$ *code.*

selected by a vector quantizer. The filter selection is made independently on each frame of speech – 50 times per second. The trellis portion of the hybrid decoder is in fact the same decoder as was used in the RELP systems – it was designed using the trellis code design algorithm on the speech residuals from the training sequence.

In the usual APC systems, the prediction filter is driven by a variable step size quantizer, which is not a particularly good model of speech residuals. In the hybrid tree coder, the trellis front end has been specifically designed to be a good mimic (or *fake process*) of residuals. Roughly speaking, the trellis decoder is chosen in such

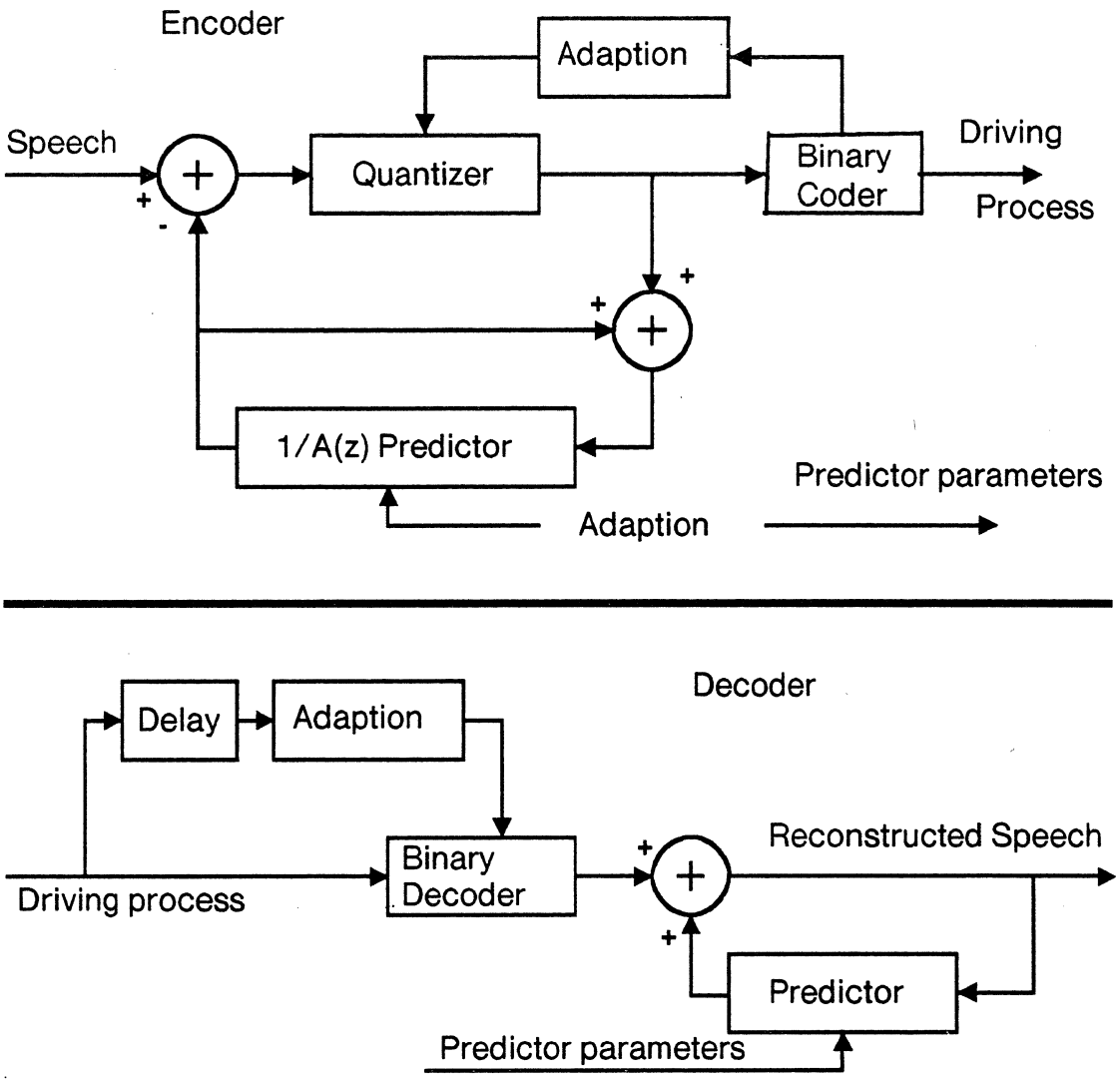


Figure 4.17 *Adaptive predictive coding system*

a way that when driven by random coin tosses, it produces residual-like sequences. It makes good intuitive sense that if the driving process of an APC like system is "close" to residuals, then the overall system will perform better.

It is less clear why the hybrid coder sounds better than the RELP system. Because the hybrid coder uses the squared error distortion measure on the original waveform, it is not taking the same advantage of the auditory masking effect as RELP. However, the most evident artifact in the RELP speech is a growl-like sound at the LPC frame rate and no such artifact is audible in the output of the hybrid coder. This implies an interaction between the trellis encoding of the residuals and the time varying LPC synthesis filter that can be compensated in the hybrid system by placing both components under the control of the tree search.

In [65], a collection of hybrid decoders consisting of LPC filters with a fixed trellis front end are used for universal coding of a speech waveform. The system utilized 16 LPC filters designed by the block code design algorithm together with a fixed trellis decoder of the scrambling function fake process type. The 16 resulting tree decoders were searched in parallel to encode a frame of speech, and the path sequence for and decoder index of the decoder giving the lowest distortion transmitted to the receiver. This system differs from the hybrid system described in this section in two ways: only 16 LPC filters are used, rather than the much larger set of 512 filters considered here, and a fake process trellis decoder matched to a Laplacian distribution is used, rather than a trellis decoder designed explicitly for speech residuals. Matsuyama and Gray do suggest in [65] the use of a spectral distortion measure for filter selection to replace the parallel search of the universal coding technique and this approach is taken in [64]. The same encoding method is used in [1], although the decoders considered do not include a trellis component and the encoder is a single path search such as in standard APC.

4.6.2 Major points of the hybrid code

At this point, we can draw together several threads, and describe the overall structure of the hybrid tree code. First, the relatively poor performance of the trellis waveform coders of section 4.3, together with the observed waveform tracking ability of the "open loop" RELP systems, suggest that it is more advantageous to design a composite, or hybrid, decoder which takes advantage of a model of speech than either to encode the speech waveform directly or to implement separately the source model and driving process (residual) encoding. In the speech case, the source model takes the form of an all pole time varying filter which spectrally flattens the original speech signal to produce the residuals. In the systems we have built, the particular model used is an LPC vector quantizer, which on a frame by frame basis, selects the gain and filter combination from a finite set of such combinations according to a spectral distortion measure [18],[35]. The residual signal, although not itself used (or even generated) during normal operation of the system, is used during system design as training data for the trellis code design algorithm. The design algorithm is used to construct a trellis decoder front end for the hybrid decoder. The purpose of the trellis portion of the decoder is to translate the digital channel sequence into a discrete time process resembling speech residuals which then drives the LPC synthesis filter. In actual operation, the time invariant trellis decoder together with the time varying source model serve as the code generator for a classic tree encoding system.

In a subjective comparison, the hybrid tree encoding system described in this section has, for a given channel rate, given speech quality significantly improved over that of the similar system of [64]. The system reported here differs from that of [64] by using a much larger set of LPC synthesis filters and by using the trellis code design algorithm for the design of the trellis portion of the hybrid decoder.

4.6.3 Modifications of the hybrid tree codes

Although we have claimed that the LPC model is used in the hybrid coding system as a whitening filter rather than as a vocal tract model, the fact that the constraint length extension method of trellis code design was successfully used on speech residuals suggests that the all-pole LPC model is not entirely adequate. If more accurate whitening filters are used in similar systems, the extension method may fail, as indeed it does for truly memoryless sources. In such a case, other methods of selecting initial trellis decoders should still work. Populating the initial codebook with values randomly drawn from the training sequence would be a good candidate.

The idea for the preceding hybrid tree code design grew from the observation that coded residuals driving the LPC inverse filter actually did reproduce the original waveform (although the same decoder organization was used earlier, in [65]). The fidelity of the reproduced speech waveform and the perceptual quality of the speech were improved by using a tree search to close the loop around the composite decoder. Since the trellis front end and the LPC synthesis filter (and thus the entire decoder structure) are identical in the RELP and hybrid systems, it is evident that the differing results of the two systems are entirely due to the selection of a different path through the trellis part of the decoder. This should be viewed as a consequence of using a different distortion measure in the two cases. Rather than using either straight squared error on the residuals or a gain weighted measure, the hybrid code uses a fairly complicated time-varying frequency weighted measure on the residual signal – it observes the trellis decoder output sequence through the time-varying LPC synthesis filter. (This complicated frequency weighted error measure on the residuals is exactly equal to a mean squared error distortion measure on the original speech waveform.)

However, it should be evident that there is a particular sequence which, at the input of the LPC inverse filter, will *exactly* reproduce the original speech waveform. This signal is just the residuals. The process of filtering original speech with the filter $A(z)/\sigma$ and then filtering the result with the inverse filter $\sigma/A(z)$ is a simple exercise. As we have seen, first encoding the residuals and then running them through the LPC synthesis filter does not work as well as the hybrid decoder. This

Table 4.7
Trellis RELP reproduction of the original speech waveform
Table entries in dB SNR

		Squared Error		Gain-modified	
Code	bit rate	Inside	Outside	Inside	Outside
1/2	3,700	5.15	2.46	6.57	2.97
1/1	6,950	6.64	4.25	7.92	4.86
2/2	6,950	6.20	4.29	7.98	4.77
2/1	13,450	9.94	8.50	11.14	8.81

means only that the squared error measure used on the residuals is not the most appropriate error measure for speech. This fact is an indication that designing the trellis code portion of the hybrid decoder by using squared error on the speech residuals is also suboptimal.

It may be possible to further improve the hybrid tree codes by changing the design procedure for the front end trellis decoder.

One way to improve the design might be to view the LPC portion of the hybrid coder as a time varying distortion measure. This approach is the logical extension of the gain-weighted distortion measure discussed in section 4.5. This procedure would operate as follows:

(1) Encode the training sequence using a hybrid decoder consisting of the LPC back end and an initial guess for the trellis front end.

(2) Use this encoding to generate a partition of the *residuals*. The updated trellis codewords would then be the centroids of the respective partition cells.

The problem with this scheme is just that discussed in section 4.3 when frequency weighted error measures were first considered. The centroid computation is no longer just the *average* of the values in the partition cell. While the centroid of a frequency weighted signal is the same as the centroid of the original signal, we wish to compute the centroid of widely separated samples of the frequency weighted sequence. The exact solution to this problem is unknown, but it may be that the arithmetic average, perhaps modified by the gain of the associated LPC frame, is sufficient to produce an improved trellis code.

4.6.4 Gain weighted hybrid codes

To check these ideas, another set of hybrid tree decoders were constructed by retaining the LPC portion of the system but substituting the RELP trellis decoders de-

Table 4.8
Gain-modified hybrid tree codes
Table entries in dB SNR

	1/2 Code		1/1 Code		2/2 Code		2/1 Code	
lg N	Inside	Outside	Inside	Outside	Inside	Outside	Inside	Outside
1	6.23	5.01	8.71	7.07	-	-	-	-
2	6.16	4.94	7.51	5.92	-	-	17.20	14.98
3	6.74	5.77	7.38	6.35	7.56	6.03	-	-
4	7.96	6.14	10.35	9.16	-	-	18.32	16.92
5	8.45	6.30	11.61	9.69	11.38	9.75	-	-
6	8.71	6.68	12.31	10.32	-	-	19.04	17.39
7	8.98	6.53	12.40	10.54	12.71	10.86	-	-
8	9.25	6.77	13.06	11.05	-	-	19.52	17.75
9	9.22	6.91	12.96	10.93	13.31	11.46	-	-
10	9.41	6.92	13.28	11.00	-	-	20.16	18.01
11	9.66	6.84	13.25	11.11	13.75	11.55	-	-

signed using gain-weighted squared error for those designed using ordinary squared error. In support of this idea is the information shown in Table 4.7, which extends Table 4.5; use of the gain-modified RELP system produced higher signal to noise ratios for the eventual reconstructed waveforms. The results for the gain-modified hybrid tree codes are collected in Table 4.8 and Figure 4.18. The performance of the 2 bits per speech sample code outside the training sequence improved about 1/2 dB, from 17.45 dB to 18.01 dB. At rate one bit per sample, both the 1/1 code and the 2/2 code improved by nearly one dB. At 1/2 bit per sample, performance is again slightly enhanced by the modification.

Finally, Figure 4.19 combines some of the contents of Figures 4.16 and 4.18 in order to better display the differences wrought by using the gain-modified distortion measure in the design of the trellis front end. Figure 4.19 displays results from both types of hybrid codes, but outside the training sequence only.

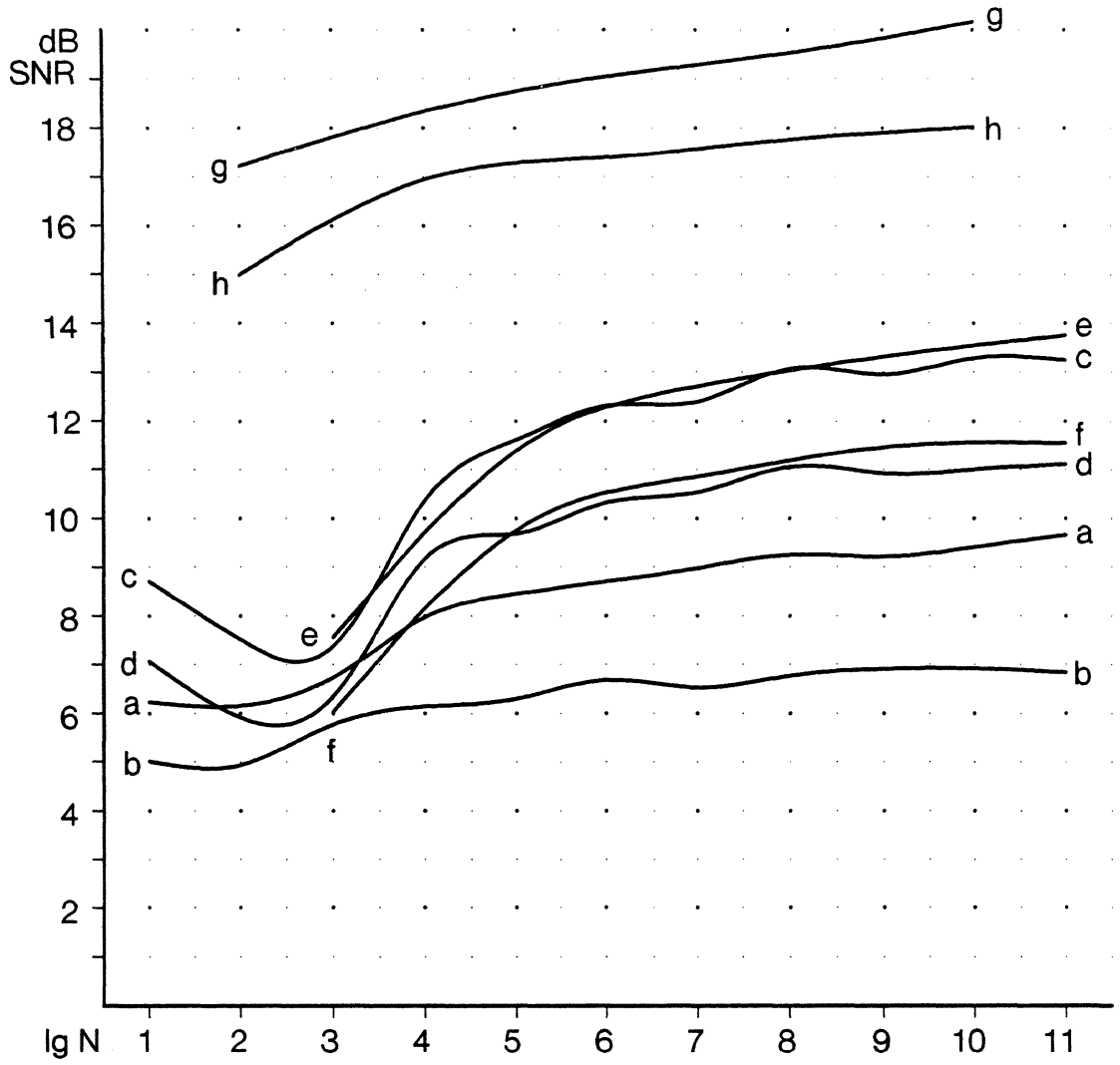


Figure 4.18 *Hybrid tree code performance using gain-modified trellis. Performance figures are in dB SNR for the original speech waveform. Curves* **a** *and* **b** *are for the* $1/2$ *code, inside and outside the training sequence, respectively. Curves* **c** *and* **d** *are for the* $1/1$ *code, curves* **e** *and* **f** *are for the* $2/2$ *code, and curves* **g** *and* **h** *are for the* $2/1$ *code.*

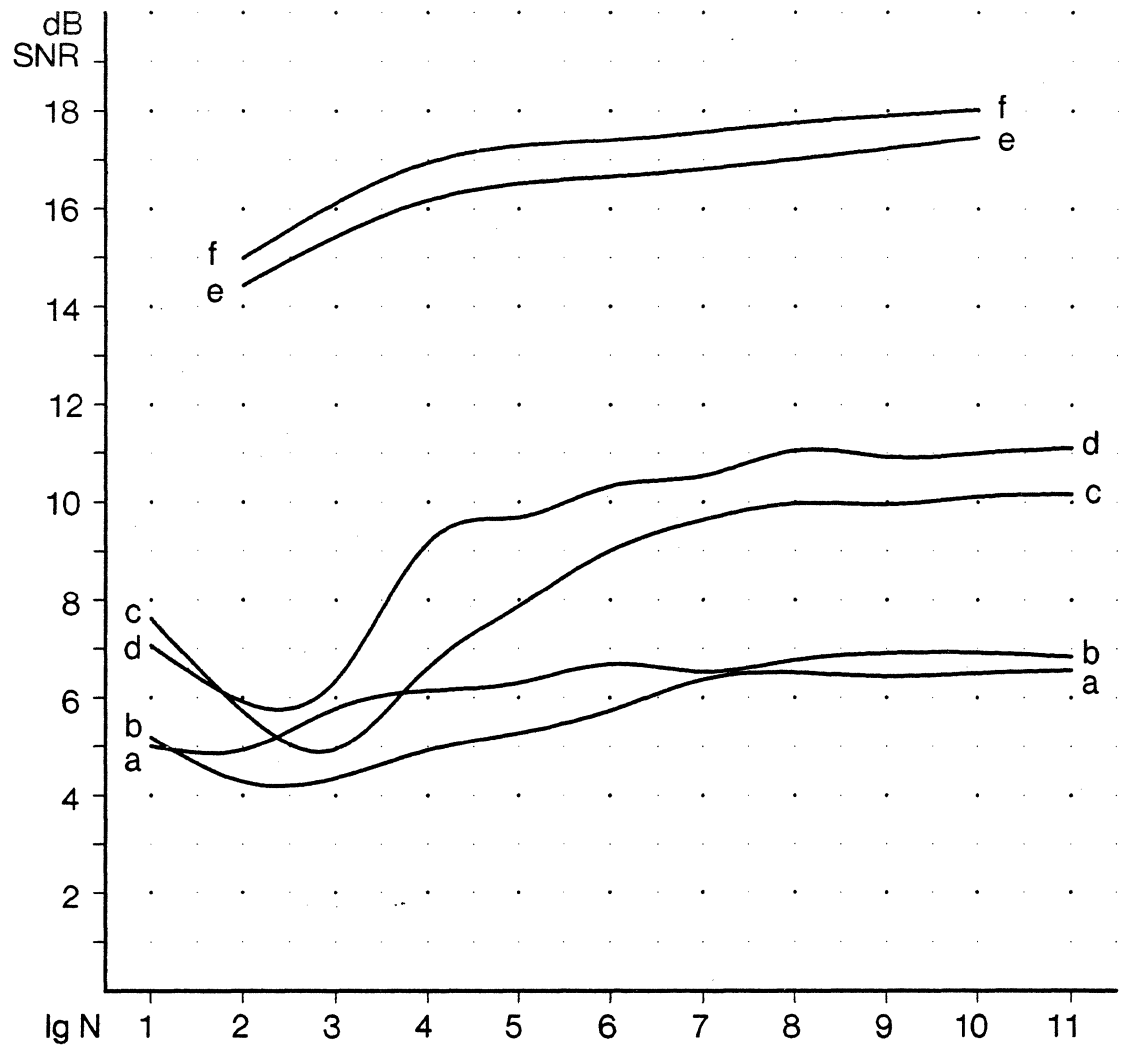


Figure 4.19 *Hybrid tree code performance. Comparison of squared error trellis and gain-modified trellis. Curves* **a** *and* **b** *are for the* $1/2$ *codes outside the training sequence, for the squared error and modified trellises respectively. Curves* **c** *and* **d** *are for the* $1/1$ *code, and curves* **e** *and* **f** *are for the* $2/1$ *code.*

5. SUMMARY

This thesis has described a methodology for the automatic design of trellis encoding data compression systems for discrete time sources. The first algorithm involved in the system is an iterative procedure for improving a given trellis decoder. The algorithm uses the encoding of a training sequence of actual data from the source to select improved codewords for a table-lookup trellis decoder. The second algorithm, given a shift register trellis decoder, returns a decoder of longer constraint length which performs at least as well as the given decoder. These algorithms have been extensively tested on random sources, yielding both improvements to standard data compression systems and performance close to the theoretical bounds.

In the applications area of speech coding, the trellis code design algorithms have been used to design three classes of low rate speech compression systems operating at 1/2, 1, and 2 bits per speech sample. The first, trellis encoding of the original speech waveform, provides intelligible, but noisy speech – this result is in agreement with those of other researchers. The second class of system, a residual excited linear predictive speech coder, uses 450 bits per second of side information in order to transmit the specification of a sequence of LPC synthesis filters to the decoder. This system offers quality improved over that of the waveform coder, but marred by some artifacts of the LPC block structure. The third class of speech coding system consists of a hydrid decoder together with a tree search encoder. In subjective listening tests, it offers good quality speech at a rate of 1 bit per sample.

The problem of the design of trellis encoding systems, however, is by no means fully solved. Results for random sources indicate that in some cases the traditional shift register decoder for trellis systems is not optimal. At present, only heuristic methods are available to suggest alternative decoder structures but, given a finite-state decoder structure, the iterative design algorithm referred to above is able to improve the associated output codebook. In addition, further study of the *ways* in which the design algorithms alter given trellis decoders for particular sources may offer improved insight into the operation of trellis encoding systems.

In the speech coding area, this thesis has used the squared error distortion measure, which is felt by most researchers to be an inadequate measure of speech quality. More complex distortion measures can be used by tree and trellis encoders, but the incorporation of non single-symbol distortion measures into the code design algorithms remains an unsolved problem.

A. SYMBOLOGY

a	Gauss-Markov feedback coefficient
b	scale factor of TPQD decoder
$d(x, y)$	distortion measure
$f(\vec{x})$	quantizer function (block quantizer)
i	codeword index (subscript)
j	time index (subscript)
k	length of a shift register decoder (constraint length)
k_i	reflection coefficient
k_c	constraint length in channel symbols
k_s	constraint length in source symbols
$\lg a$	base two logarithm of a
m	iteration index (superscript)
m/n	shift register code structure
n	length of training sequence
q	cardinality of communications channel
r	finite-state machine state
u	channel symbol
u_j^k	length-k string of channel symbols ending at time j
x	training sequence symbol
$\vec{x}$	vector source symbol
$\hat{x}$	reproduction of x
$\{x_j\}$	training sequence
y	reproduction symbol (decoder output)
$\{y_i\}$	codebook
A^m	block quantizer codebook
$A(z)$	LPC analysis filter
$1/A(z)$	LPC synthesis filter
C^m	trellis codebook
$\|\|C\|\|$	cardinality of codebook C
D	Distortion (of Rate-Distortion)
L	M,L Algorithm search depth
M	number of paths considered by M,L Algorithm
N	codebook size
N_c	codebook size in codewords
P	partition
R	Rate (of Rate-Distortion)

S_i	partition cell
V	channel alphabet
$[V, W, Z, \nu, \varsigma]$	finite-state machine description
W	decoder state alphabet
X	input alphabet
Y	finite-state machine output codebook
Z	decoder output alphabet
$\mathcal{A}$	block quantizer function
$\mathcal{C}$	codebook update function
$\mathcal{N}(a, b)$	Gaussian distribution with mean a and variance b
$\mathcal{P}$	partition function
α	string of base-q digits
$\alpha\beta$	concatenated strings
$(\alpha)_q$	string of digits interpreted as a number in base-q
α^k	string of digits of length k
$\|\alpha\|$	length of string α
Δ	sample average distortion
Δ'	average distortion using updated codebook
ϵ	iteration limit
ν	next state function
σ	variance or LPC gain term
ς	output function
ς'	output codebook index function

B. Tree and Trellis Search Algorithms

In this section, we will briefly discuss the operation of the Viterbi trellis search algorithm and the M,L Algorithm tree search algorithm [25],[46],[80]. The Viterbi Algorithm, first developed for the problem of decoding convolutional channel codes, is a form of dynamic programming. The M,L Algorithm is best thought of as a breadth first tree search. There are many other algorithms for encoding tree or trellis codes. The reader is referred to the surveys [4] and [5] and to the original references for some of the algorithms [6],[9],[26],[44, sec. 10.4],[46].

B.1. The Viterbi Algorithm

Because the encoding of a particular source symbol affects and is affected by the encodings of both past and future symbols, any optimal encoding algorithm such as the Viterbi Algorithm must necessarily inspect an entire sequence of source symbols before deciding on the encoding of any part of the sequence. This is usually impractical for continuously operating sources or in the presence of real-time encoding requirements, so often the source data is given an artificial block structure and the blocks encoded separately, although there may be special treatment of the block boundaries. Since the complexity of encoding grows only linearly with block length, the blocks may be quite long – perhaps several thousand symbols.

The key to understanding the Viterbi Algorithm lies in the following observation: suppose that we are somewhere in the middle – time j – of encoding a source block. Since the code has a trellis structure, it must be that the path of the optimal encoding passes through a particular state at time j, although we may not know which one. The algorithm proceeds by remaining vague about the choice of encoding until the end of the source block is reached, but at each time computing the distortions associated with paths ending in each possible state. Then, at the end of the block, the algorithm transmits the path which has accumulated the lowest distortion.

With the aid of Figure B.1, we can easily describe the kernel of the Viterbi algorithm as specialized for a one bit per symbol system with distortion measure $d(x, x')$. Suppose we associate with each state i and time j a value $D_i(j)$ which represents the cumulative (up to time j) distortion associated with the best path which passes through state i at time j. By inspection of Figure B.1 we see that there are only two ways to reach a certain state from preceding states, thus (for example)

$$D_{01}(j+1) = \min(D_{00}(j) + d(x_j, y_{001}), D_{10}(j) + d(x_j, y_{101}))$$

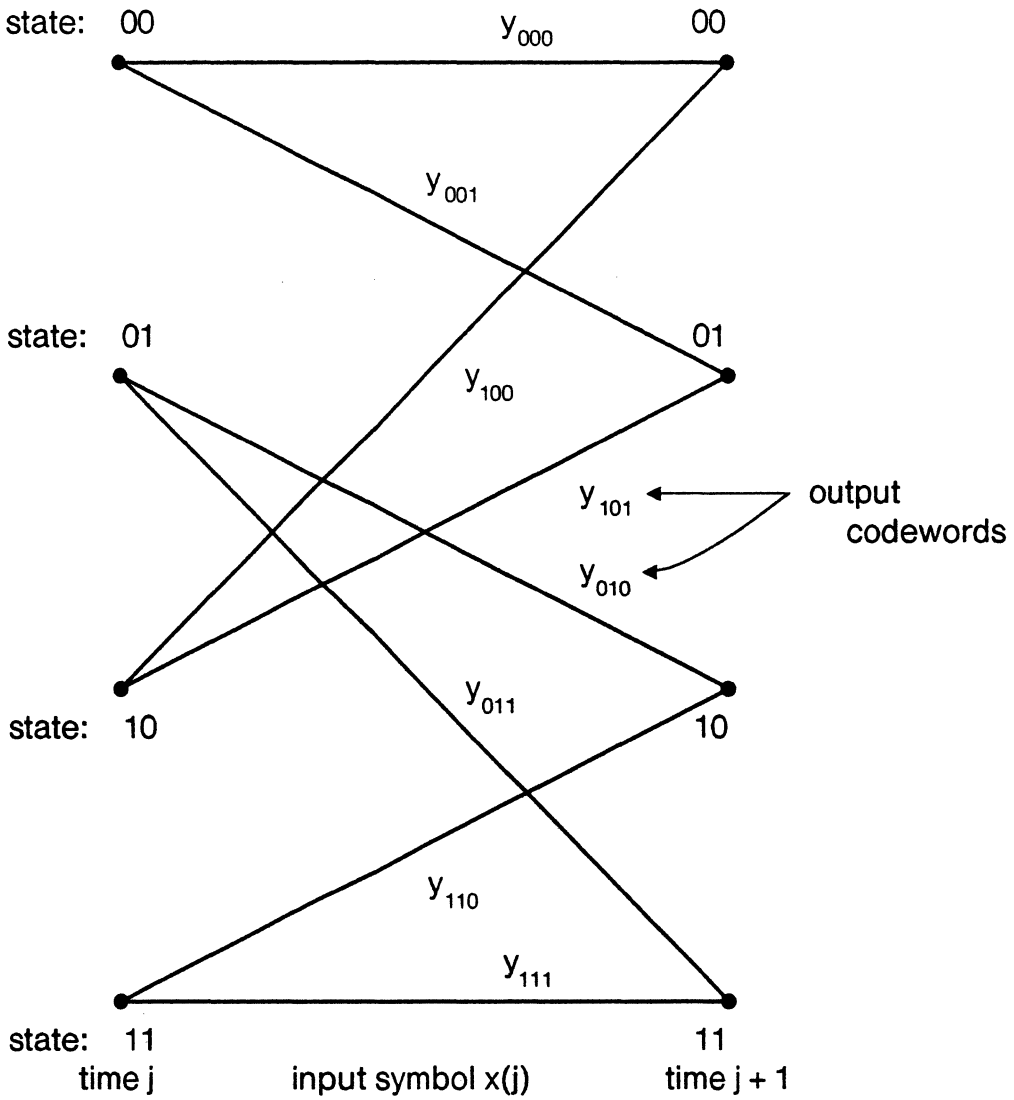


Figure B.1 *Trellis section for a 1 bit/symbol ($q = 2$) constraint length 3 shift register decoder. Subscripts are the shift register contents associated with the selection of a particular codeword.*

where we have used base-2 subscripts for states and for decoder codebook indices. This expression provides a recurrence relation among the $D_i(j)$. The *min* operation selects the most desirable path along which to enter a state. The encoding of a length-n block proceeds as follows:

(1) Start in some state r at time 0 by setting $D_r(0) = 0$ and $D_i(0) = \infty; i \neq r$.

(2) Repeat step (2a) for $j = 0, \ldots, n-1$.

(2a) Given x_j, the source symbol for time j, and $D_i(j)$ for all i, evaluate $D_i(j+1)$ for all i.

(3) Select and transmit the path which ends in the state with the minimum value of $D_i(n)$.

Step (3) is moderately tricky to implement. One way to do it is to store, for each state-time pair, information about whether the state was entered by the upper or lower path. When the end of the block is reached, it is a simple matter to follow these links back to the beginning of the block and then to run forward again, releasing path symbols to the channel.

We mentioned that the Viterbi Algorithm's ability to select an optimal encoding depends on its inspection of the *entire* source sequence before releasing any symbols to the communications channel. In situations where this would cause excessive delay, the source sequence is often broken up into blocks which are then encoded independently. If blocking is done, the problem arises of specifying the state of the decoder at the beginning of a block. In this thesis, we have ignored this problem by setting the initial decoder state for a block to the final state of the previous block. This approach permits the decoder to operate without knowledge of the encoding algorithm but tends to introduce above average distortions near block boundaries – because the possible decoder states have been constrained. One perhaps better method is to set $D_i(0) = 0$ for all states i rather than for just one of them. This has the effect of leaving the specification of the initial state open until the algorithm has chosen the encoding for the entire block. Before the selected path sequence is released to the channel, the state number of the initial state can be transmitted. This procedure introduces some synchronization difficulties into the system, but avoids a distortion glitch at the beginning of each block. A similar effect may be obtained by slightly overlapping the blocks.

The Viterbi Algorithm as described above requires a computation to be performed for each possible decoder state as each source symbol is considered, thus the algorithm complexity grows linearly with the number of decoder states (exponentially with the constraint-length of a shift register decoder) and linearly with the length of the sequence to be encoded. The required storage is also proportional to the number of decoder states and, subject to argument about the best data structure for storing the path information, at least proportional to the block length.

B.2. M,L Algorithm

The M,L Algorithm, sometimes called the M Algorithm, is a breadth first tree search, determined by the fact that it examines all code tree branches at time j that it well ever examine before moving on to time $j + 1$. The algorithm is useful

with both tree and trellis codes, but does not distinguish between them. The M,L name derives from the fact that the algorithm maintains M potential paths through the code tree and searches to depth L.

As a symbol arrives from the source at time $j + L$, the algorithm has already released to the channel all code symbols up to time j. M paths, all stemming from a common root in the code tree at time j and extending to time $j + L$, are under consideration as possible encodings. The algorithm consists of four steps, executed in sequence for each source symbol.

(1) Extend paths. With the arrival of a new source symbol, extend each path currently under consideration in each of the possible ways. In a one bit per symbol system the code tree branches two ways at each level, so each path would be extended in two ways. As part of this step, the algorithm calculates the distortions that would be incurred if any of the extended paths were actually to be used.

(2) Release a channel symbol. Locate the path offering the minimum average distortion over the next L source symbols and release to the channel its first symbol. (There are other possible symbol release procedures, [29],[38], but they offer only minor differences.)

(3) Discard unreachable paths. Discard all paths whose first symbol does not agree with the symbol just released. These paths can no longer be considered for further extension because they no longer stem from the common tree root.

(4) Retain the M best paths. If there are more than M paths remaining after step (3), recursively discard the highest distortion path until only M paths remain.

The M,L Algorithm offers a fixed encoding delay of L symbols and fixed computational effort per symbol. The computational cost of the algorithm grows linearly with M and is generally little effected by the value of L. Storage costs are proportional to the product of M and L. While steps (2) and (4) above seem to require sorting the paths in order of their distortion, in fact it is only necessary to locate the best and M-th best paths. This simpler problem, known as a *median* or *percentile* search, is computable in linear time [49].

One major advantage of the M,L algorithm is the extent to which its level of performance may be varied by changing the values of M and L. While most of the benefits of a tree search may be obtained with modest values of M, perhaps 4 to 10, larger values usually provide some additional performance [17],[82]. Values of L somewhat greater than the constraint length of the decoder seem generally sufficient.

We stated earlier that in the case of searching trees generated by finite-state decoders, the M,L Algorithm takes no advantage of the folded tree, or trellis, structure of the decoder. If the M,L Algorithm were used to exhaustively search a code tree, M would grow exponentially with L, independent of the decoder structure. The Viterbi Algorithm, by taking advantage of the trellis structure of finite-state decoders, is able to limit the number of paths considered to the number of states of the decoder – independent of search depth.

This fact is not merely of interest in the limit of large M and L. Even for relatively modest search effort, if two or more of the paths under consideration by the M,L Algorithm rejoin one another (by reaching the same decoder state by different routes) they may both be further extended. This event will result in some duplicate computation and a reduction in the effective value of M. One way to avoid this kind of difficulty is to add a special test between steps (3) and (4) above.

(3a) Check for duplicates. If there are any paths which terminate in the same decoder state, discard all but the best.

This additional step has the effect of locating paths which separate in the code tree, but later rejoin one another by reaching an identical decoder state by different routes. As M becomes greater than the total number of decoder states, this modification will limit the number of retained paths to the number of decoder states.

C. Random Number Generators

Computer aided experiments with codes for random sources cannot be pursued without test data. Since true random numbers, perhaps obtained by measuring the periods between emission of particles by a substance undergoing radioactive decay, are difficult to obtain and cumbersome to store, recourse is usually made to "random" number generators. The use of such pseudorandom generators may provide some benefit, as the sequences obtained can be easily reproduced. An excellent discussion of these matters may be found in [48].

The experiments described in Chapter 3 of this thesis were conducted with the aid of the additive random number generator

$$y_j = (y_{j-55} - y_{j-24}) \pmod{2^{31} - 1}.$$

This generator is described in [74] and generates numbers uniformly distributed between 0 and $2^{31} - 1$. Although it requires a table of the previous 55 outputs, the generator is extremely fast, since it does not require any multiplications. On computers capable of 32-bit arithmetic, the modulus operation is also fast, requiring only a masking operation and, roughly half the time, a single subtraction.

An alternative number generator,

$$y_j = (1327217885 y_{j-1}) \pmod{2^{31} - 1},$$

requiring less storage, but including a multiplication, was used for the construction of random initial decoders. This generator is of the classic linear congruential type and also produces numbers distributed uniformly between 0 and $2^{31} - 1$.

Once uniform random numbers are obtained, there are various algorithms available for the production of numbers having other distributions. For the Gaussian sources used in Chapter 3, an extension of Forsythe's method was used – Algorithm FL5 in [3]. This procedure is quite complex, but extremely fast. A simpler, but much slower method may be found in [31].

$$\omega_j = \sqrt{2 \ln \frac{1}{y_j}} \sin 2\pi y_{j+1}$$

where ω is distributed $\mathcal{N}(0,1)$, and y is uniform in $[0,1)$.

D. SPEECH DATA

Attached to this report is a soundsheet containing most of the speech results discussed in chapter 4. The soundsheet includes representatives of waveform trellis encoding, trellis RELP encoding, and hybrid tree encoding. Chapter 4 discussed alternative trellis structures for use at one bit per speech sample; since the 1/1 and the 2/2 code structures produce very similar results, only the 1/1 results are included here. Chapter 4 also discussed alternative versions of the RELP and hybrid encoding systems using the modified distortion measure $\sigma(x-y)^2$. Since the modifications resulted in improved signal to noise ratios, the modified results are presented here.

The first three bands include segments of the various encoding methods at 1/2 bit per speech sample, 1 bit per speech sample, and 2 bits per speech sample. All these segments are from outside the training sequence. The fourth band includes the original speech data: first the test segment and then the training segment.

Band 1: Waveform trellis encoding

Segment	Code type	Bits/sample	Bits/second
1	1/2	1/2	3,250
2	1/1	1	6,500
3	2/1	2	13,000

Band 2: Trellis RELP encoding

Bits/second entries include 450 bps LPC side information

4	1/2	1/2	3,700
5	1/1	1	6,950
6	2/1	2	13,450

Band 3: Hybrid tree encoding

Bits/second entries include 450 bps LPC side information

7	1/2	1/2	3,700
8	1/1	1	6,950
9	2/1	2	13,450

Band 4: Original PCM speech

Segment 10 is the original test sequence

Segment 11 is the original training sequence

10	12	78,000
11	12	78,000

REFERENCES

[1] J.-P. Adoul, J. L. Debray, and D. Dalle: "Spectral Distance Measure Applied to the Design of DPCM Coders with L Predictors," *Conf. Record* 1980 IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing, Denver, CO, pp. 512–515, April 1980.

[2] A. Aho, J. Hopcroft, and J. Ullman: *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA 1976.

[3] J. H. Ahrens and U. Dieter: "Extensions of Forsythe's Method for Random Sampling from the Normal Distribution," *Mathematics of Computation*, Vol. 27, No. 124, pp. 927–937, October 1973.

[4] J. B. Anderson: "Effectiveness of Sequential Tree Search Algorithms in Speech Digitization," *Conf. Record,* 1979 International Conference on Communications, Boston, MA, pp. 8.1.1–8.1.6, June 1979.

[5] J. B. Anderson: "Recent Advances in Sequential Encoding of Analog Waveforms," *Conf. Record* 1978 National Telecommunications Conference, Birmingham, AL, pp. 19.4.1–19.4.5, December 1978.

[6] J. B. Anderson: "A Stack Algorithm for Source Coding with a Fidelity Criterion," *IEEE Trans. on Information Theory*, Vol. IT-20, No. 2, pp. 211-226, March 1974.

[7] J. B. Anderson and J. B. Bodie: "Tree Encoding of Speech," *IEEE Trans. on Information Theory*, Vol. IT-21, No. 4, pp. 379-387, July 1975.

[8] J. B. Anderson, C. W. P. Ho: "Architecture and Construction of a Hardware Sequential Encoder for Speech," *IEEE Trans. on Communications*, Vol. COM-25, No. 7, pp. 703-707, July 1977.

[9] J. B. Anderson and F. Jelinek: "A 2-Cycle Algorithm for Source Coding with a Fidelity Criterion," *IEEE Trans. on Information Theory*, Vol. IT-19, No. 1, pp. 77–92, January 1973.

[10] J. B. Anderson and C. W. Law: "Real-Number Convolutional Codes for Speech-Like Quasi-Stationary Sources," *IEEE Trans. on Information Theory*, Vol. IT-23, No. 6, pp. 778–782, November 1977.

[11] B. S. Atal and M. R. Schroeder: "Adaptive Predictive Coding of Speech Signals," *Bell. Syst. Tech. J.*, Vol. 45, No. 7, pp. 1973–1986, October 1970.

[12] D. W. Becker and A. J. Viterbi: "Speech Digitization and Compression by Adaptive Predictive Coding with Delayed Decision," *Conf. Record,* 1975 IEEE

National Telecommunications Conf., pp. 46/18–46/23, New Orleans, LA 1975.

[13] T. Berger: *Rate Distortion Theory, A Mathematical Basis for Data Compression*, Prentice-Hall, Englewood Cliffs, NJ 1971.

[14] A. W. Biermann and J. A. Feldman: "On the Synthesis of Finite-State Machines," *A.I. Memo No. 114*, Computer Science Department, Stanford University, April 1970.

[15] G. Birkhoff and T. C. Bartee: *Modern Applied Algebra*, McGraw-Hill, New York 1970.

[16] B. A. Blesser: "Digitization of Audio," *J. Audio Engineering Society*, Vol. 26, No. 10, pp. 739–771, October 1978.

[17] J. B. Bodie: "Multi-Path Tree Encoding for Analog Data Sources," *Report CRL-20, Communications Research Laboratory*, McMaster University, Hamilton, Ontario, Canada. June 1974.

[18] A. Buzo, A. H. Gray Jr., R. M. Gray, and J. D. Markel: "Speech Coding Based on Vector Quantization," *IEEE ASSP*, Vol. ASSP-28, No. 5, pp. 562–574, October 1980.

[19] C. C. Cutler: "Delayed Encoding: Stabilizer for Adaptive Coders," *IEEE Trans. Comm. Technology*, Vol COM-19, No. 6, Dec. 1971, pp. 898-904.

[20] C. R. Davis, M. E. Hellman: "On Tree Coding with a Fidelity Criterion," *IEEE Trans. on Information Theory*, Vol. IT-21, No. 5, pp. 373–378, July 1975.

[21] R. A. Donnan and J. R. Kersey: "Synchronous Data Link Control: A Perspective," *IBM Systems Journal*, May 1974.

[22] J. G. Dunham: "An Iterative Theory for Code Design," *Submitted to IEEE Trans. on Information Theory*, 1980.

[23] W. A. Finamore and W. A. Pearlman: "Optimal Encoding of Discrete-Time Continuous-Amplitude Memoryless Sources with Finite Output Alphabets," *IEEE Trans. on Information Theory*, Vol. IT-26, No. 2, pp. 144–155, March 1980.

[24] J. L. Flanagan, M. R. Schroeder, B. S. Atal, R. E. Crochiere, N. S. Jayant, and J. M. Tribolet: "Speech Coding," *IEEE Trans. on Communications*, Vol. COM-27, No. 4, pp. 710–737, April 1979.

[25] G. D. Forney, Jr.: "The Viterbi Algorithm," *Proceedings of the IEEE*, Vol. 61, No. 3, pp. 268–278, March 1973.

[26] R. G. Gallager: "Tree Encoding for Symmetric Sources with a Distortion

Measure," *IEEE Trans. on Information Theory*, Vol. IT-20, No. 1, pp. 65–76, January 1974.

[27] J. D. Gibson: "Adaptive Prediction in Speech Differential Encoding Systems," *Proceedings of the IEEE*, Vol. 68, No. 4 April 1980, pp. 488–525.

[28] J. D. Gibson: "Tree Coding versus Differential Encoding of Speech," 1981 International Conf. on Information Theory, Santa Monica, CA, February 1981.

[29] J. D. Gibson and A. C. Goris: "Incremental and Variable-Length Tree Coding of Speech," *Conf. Record,* 1979 International Conference on Communications, Boston, MA, 8.5.1–8.5.5, June 1979.

[30] B. Gold and C. M. Rader: "The Channel Vocoder," *IEEE Trans. on Audio and Electroacoustics*, Vol. AU-15, No. 4, pp. 148–160, December 1967.

[31] B. Gold and C. Rader: *Digital Processing of Signals*, McGraw-Hill, New York 1969.

[32] A. J. Goldberg: "Predictive Coding with Delayed Decision," *Conf. Record,* 1977 IEEE Int. Conf. Acoustics, Speech, and Signal Processing, Hartford, CT, pp. 405–408, May 1977.

[33] A. J. Goldberg: "A Real-Time Adaptive Predictive Coder using Small Computers," *IEEE Trans. on Communications*, Vol. COM-23, No. 12, pp. 1443–1451, December 1975.

[34] A. H. Gray Jr., R. M. Gray, and J. D. Markel: "Comparison of Optimal Quantizations of Speech Reflection Coefficients," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, Vol. ASSP-25, No. 1, pp. 9–23, February 1977.

[35] A. H. Gray Jr. and J. D. Markel: "Distance Measures for Speech Processing," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, Vol. ASSP-24, No. 5, pp. 380–391, October 1976.

[36] A. H. Gray Jr. and J. D. Markel: "Quantization and Bit Allocation in Speech Processing," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, Vol. ASSP-24, No. 6, pp. 459–473, December 1976.

[37] A. H. Gray and D. Y. Wong: "The Burg Algorithm for LPC Speech Analysis/Synthesis," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, Vol. ASSP-28, No. 6, pp. 609–615, December 1980.

[38] R. M. Gray: "Time-Invariant Trellis Encoding of Ergodic Discrete-Time Sources with a Fidelity Criterion," *IEEE Trans. on Information Theory*, Vol. IT-23, No. 1, pp. 71–83, January 1977.

[39] R. M. Gray, A. Buzo, A. H. Gray Jr., Y. Matsuyama: "Distortion Measures for

Speech Processing," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, Vol. ASSP-28, No. 4, pp. 367–376, August 1980 .

[40] R. M. Gray, J. C. Kieffer, and Y. Linde: "Locally Optimal Block Quantizer Design," *Information and Control*, Vol. 45, No. 2, pp. 178–198, May 1980.

[41] R. M. Gray and Y. Linde: "Vector Quantization and Predictive Quantization for Gauss-Markov Sources," *Submitted to IEEE Trans. on Communications*, 1981.

[42] R. M. Gray, D. L. Neuhoff, and D. S. Ornstein: "Nonblock Source Coding with a Fidelity Criteria," *Annals of Probability*, Vol. 3, pp. 478–491, April 1975.

[43] N. S. Jayant and S. A. Christensen: "Tree-Encoding of Speech Using the (M, L)-Algorithm and Adaptive Quantization," *IEEE Trans. on Communications*, Vol. COM-26, No. 9, pp. 1376–1379, September 1978.

[44] F. Jelinek: *Probabilistic Information Theory*, McGraw-Hill, New York 1968.

[45] F. Jelinek: "Tree Encoding of Memoryless Discrete Time Sources with a Fidelity Criterion," *IEEE Trans. Information Theory*, Vol. IT-5, No. 5, pp. 584–590, September 1969.

[46] F. Jelinek and J. B. Anderson: "Instrumentable Tree Encoding of Information Sources," *IEEE Trans. Information Theory*, Vol. IT-17, No. 1, pp. 118–119, January 1971.

[47] D. E. Knuth: *The Art of Computer Programming,* Vol. 1, *Fundamental Algorithms*, Addison-Wesley, Reading, MA 1973.

[48] D. E. Knuth: *The Art of Computer Programming,* Vol. 2, *Seminumerical Algorithms*, Addison-Wesley, Reading, MA 1973.

[49] D. E. Knuth: *The Art of Computer Programming,* Vol. 3, *Sorting and Searching*, Addison-Wesley, Reading, MA 1973.

[50] A. N. Kolmogorov and S. V. Fomin: *Introductory Real Analysis*, Dover Publications, New York 1970.

[51] D. T. L. Lee: "Ladder Form Filters," *Ph. D. Dissertation*, Information Systems Laboratory, Stanford University, August 1980.

[52] Y. Linde: "The Design of Tree and Trellis Data Compression Systems," *Ph.D. Dissertation*, Information Systems Laboratory, Stanford University, December 1977.

[53] Y. Linde, A. Buzo, R. M. Gray: "An Algorithm for Vector Quantizer Design,"

IEEE Trans. on Communications, Vol. COM-28, No. 1, pp. 84–95, January 1980.

[54] Y. Linde and R. M. Gray: "The Design of Tree and Trellis Data Compression Systems," *Information Systems Laboratory Report*, Stanford University, SEL Technical Report No. 6504-2, February 1978.

[55] Y. Linde and R. M. Gray: "A Fake Process Approach to Data Compression," *IEEE Trans. on Communications*, Vol. COM-26, No. 6, pp. 840–847, June 1978.

[56] S. P. Lloyd: "Least Squares Quantization in PCM's," *Bell Telephone Laboratories Paper*, Murray Hill, NJ, 1957.

[57] D. G. Luenberger: *Introduction to Linear and Nonlinear Programming*, Addison-Wesley, Reading, MA 1973.

[58] R. A. MacDonald: "Signal-to-Noise and Idle Channel Performance of Differential PCM Systems," *Bell Sys. Tech. J.*, pp. 1123-1151, September 1966.

[59] J. MacQueen: "Some Methods for Classification and Analysis of Multivariate Observations," *Proc. Fifth Berkeley Symp. on Math. Statistics and Probability*, Vol. 1, pp 281–296, 1967.

[60] J. Makhoul: "Linear Prediction: A Tutorial Review," *Proceedings of the IEEE*, Vol. 63, No. 4, pp. 561–580, April 1975.

[61] J. W. Mark: "Adaptive Trellis Encoding of Discrete-Time Sources with a Distortion Measure," *IEEE Trans. on Communications*, Vol. COM-25, No. 4, pp 408–417, April 1977.

[62] J. D. Markel and A. H. Gray: *Linear Prediction of Speech*, Springer-Verlag, New York 1976.

[63] Y. Matsuyama: "Process Distortion Measures and Signal Processing," 1981 International Conf. on Information Theory, Santa Monica, CA, February 1981.

[64] Y. Matsuyama: "Speech Compression Systems Using a Set of Inverse Filters," *Ph. D. Dissertation*, Information Systems Laboratory, Stanford University, August 1978.

[65] Y. Matsuyama and R. M. Gray: "Universal Tree Coding for Speech," *IEEE Trans. on Information Theory*, Vol. IT-27, No. 1, pp. 31–40, January 1981.

[66] J. Max: "Quantizing for Minimum Distortion," *IRE Trans. on Information Theory*, Vol. IT-6, No. 2, pp. 7–12, March 1960.

[67] M. Morf and D. T. L. Lee: "Fast Algorithms for Speech Modeling," *Technical Report*, Information Systems Laboratory No. M308-1, Stanford University,

December 1978.

[68] A. M. Noll: "Cepstrum Pitch Determination," *J. Acoustic Soc. Amer.*, Vol. 41, pp. 293–309, February 1967.

[69] J. B. O'Neal, Jr. and R. W. Stroh: "Differential PCM for Speech and Data Signals," *IEEE Trans. on Communications*, Vol. COM-20, No. 10, pp. 900–912, October 1972.

[70] W. A. Pearlman: "A Sliding Block Code for Small User Alphabets with Performance near the Rate-Distortion Limit," 1981 International Conf. on Information Theory, Santa Monica, CA, February 1981.

[71] L. R. Rabiner and R. W. Schafer: *Digital Processing of Speech Signals*, Prentice-Hall, Englewood Cliffs, NJ 1978.

[72] G. Rebolledo: "Vector Quantization Applied to Speech Coding," 1981 International Conf. on Information Theory, Santa Monica, CA, February 1981.

[73] G. Rebolledo: "Vector Quantization Applied to Speech Coding," *Ph. D. Dissertation*, Information Systems Laboratory, Stanford University, June 1981.

[74] J. Reiser: "The Analysis of Additive Random Number Generators," *Stanford University Report*, STAN-CS-77-601, March 1977.

[75] C. E. Shannon: "Coding Theorems for a Discrete Source with a Fidelity Criterion," *IRE National Convention Record, part 4*, pp. 142–163, 1959.

[76] C. E. Shannon: "A Mathematical Theory of Communication," *Bell. Syst. Tech. J.*, Vol. 27, pp. 379–423, July 1948 and pp. 623-656, October 1948.

[77] J. Uddenfeldt and L. H. Zetterberg: "Algorithms for Delayed Encoding in Delta Modulation with Speech-Like Signals," *IEEE Trans. on Communications*, Vol. COM-24, No. 6, pp. 652-658, June 1976.

[78] Chong Kwan Un and Hwang Soo Lee: "A Comparative Study of Adaptive Delta Modulation Systems," *Conf. Record,* 1979 International Conference on Communications, Boston, MA, pp. 8.6.1–8.6.5, June 1979.

[79] Chong Kwan Un and D. Thomas Magill: "The Residual-Excited Linear Prediction Vocoder with Transmission Rate Below 9.6 kbits/s," *IEEE Trans. on Communications*, Vol. COM-23, No. 12, pp. 1466–1474, December 1975.

[80] A. J. Viterbi and J. K. Omura: "Trellis Encoding of Memoryless Discrete-Time Sources with a Fidelity Criterion," *IEEE Trans. on Information Theory*, Vol. IT-20, No. 3, pp. 325–332, May 1974.

[81] S. G. Wilson: "Adaptive Tree Encoding of Discrete-Time Sources with Speech

Applications," *Conf. Record,* 1978 IEEE National Telecommunications Conf., Birmingham, AL, pp. 19.5.1–19.5.5, December 1978.

[82] S. G. Wilson: "Adaptive Tree Encoding of Speech at Low Bit Rates with Weighted Error Criteria," *Communication Systems Laboratory Report*, University of Virginia, No. UVA/526171/EE78/101, November 1978.

[83] S. G. Wilson and S. Husain: "Adaptive Tree Encoding of Speech at 8000 bits/s with a Frequency Weighted Error Criterion," *IEEE Trans. on Communications*, Vol. COM-27, No. 1, pp. 165–170, January 1979.

[84] H. S. Witsenhausen: "On the Structure of Real Time Source Coders," *Bell Syst. Tech. J.*, Vol. 58, No. 6, pp. 1437–1451, July 1979.

[85] D. Y. Wong, F. B. Juang, and A. H. Gray: "An 800 bps Speech Compression System Based on Vector Quantization," 1981 International Conf. on Information Theory, Santa Monica, CA, February 1981.

[86] L. H. Zetterberg and J. Uddenfeldt: "Adaptive Delta Modulation with Delayed Decision," *IEEE Trans. on Communications*, Vol. COM-22, No. 9, pp. 1195–1198, September 1974.

XEROX

Xerox Corporation
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California 94304

XEROX® is a trademark of XEROX CORPORATION Printed in U.S.A.